ADNANE EL KABBAL

UN SYSTEME DE TYPE POUR L'ANALYSE DES PARE-FEUX

Mémoire présenté
à l'Université du Québec en Outaouais
pour l'obtention
de grade de Maître (es) Sciences (M.Sc.)

Directeur de recherche : Dr. Kamel Adi

Département d'informatique et d'ingénierie
Université du Québec en Outaouais
Gatineau

MARS 2005

Dédicace

À mes parents et ma famille,

À tous ceux que j'aime et qui compte pour moi.

Remerciements

JE tiens à remercier tout particulièrement Dr. Kamel Adi d'avoir accepté de diriger ce travail et d'avoir pris cette charge très à coeur. Toujours disponible, Kamel a su, grâce à son intuition et une confiance étonnantes, inciter un étudiant parfois dérouté à surmonter chaque difficulté.

A son contact, je pense avoir appris à envisager les choses non plus d'un point de vue purement technique, mais avec recul, en formulant d'abord chaque problème pour tirer les questions essentielles, avant d'y répondre. En bref, c'est grâce à son expérience que j'ai pu acquérir, je l'espère, la démarche d'un chercheur.

Je remercie également l'ensemble des membres du jury d'avoir accepté la charge de valider ce rapport et pour l'intérêt et le temps qu'ils ont bien voulu accorder à ce travail.

Enfin, que soient également remerciés :

- Toute l'équipe du Laboratoire de Recherche en Sécurité Informatique (LRSI), en particulier : Livui Pene et Luke Sullivan, pour leur précieuse collaboration.

- L'Université du Québec en Outaouais de m'avoir fourni, au cours de la période de ma maîtrise, un cadre de travail remarquable tant par son haut niveau scientifique que par son ambiance détendue.

- Tous mes professeurs, spécialement : Karim El Guemhioui, Luigi Logrippo, Michal Iglewski, Andrej Pelc, Rokia Missaoui et Alain Charbonneau, pour le savoir qu'ils m'ont enseigné ainsi que leurs conseils et encouragements.

- Dr. Mohamed Mejri de l'université de Laval pour ses précieuses idées.

- Tous mes amis : Mohamed Bougataya, Hicham Chaoui, Meguader Aissa, Mohammed Derkaoui et Hassan Amani, qui étaient toujours à mes côtés pour me soutenir et me remonter le moral.

– Tous ceux qui ont participé de près ou de loin, à rendre ce travail tel qu'il est aujourd'hui.

Je vous suis redevable.

Adnane El Kabbal
le 11 mars 2005

Résumé

Ce travail de recherche vise à proposer des réponses à certaines questions relatives aux domaines de la spécification et de l'analyse des pare-feux. Pour ce faire, nous avons consacré la première partie de ce mémoire pour la présentation d'un langage qui peut être utilisé pour exprimer, d'une façon simple, des politiques de sécurité, et nous avons illustré sa syntaxe à travers un exemple de spécification. La deuxième partie de ce travail, nous a permis de ramener la détection d'anomalies dans les pare-feux à un problème de typage où le type représente les conflits potentiels entre les règles d'un pare-feu. Le système de types que nous proposons apporte une contribution novatrice par rapport aux techniques existantes au moins sur les points suivants :

- notre technique est plus précise dans le sens qu'elle permet de détecter des anomalies pouvant impliquer plus que deux règles, comme c'est le cas dans les travaux existants. En effet, notre analyse peut arriver à la conclusion qu'un ensemble de règles E_1 est en contradiction avec un ensemble de règles E_2 par exemple.
- l'algorithme d'inférence de types qui mécanise le système de types est très efficace puisqu'il minimise le nombre de comparaisons entre les règles. Ce qui se traduit par une nette diminution des coûts d'analyse des pare-feux.

Table des matières

1 Introduction **1**

2 Eléments de base **3**

2.1 Introduction 3

2.2 Menaces informatiques 4

2.3 Attaques sur les systèmes informatiques 4

2.3.1 Vol de mot de passe 5

2.3.2 Ingénierie sociale 5

2.3.3 Porte de derrière 5

2.3.4 Reniflement 6

2.3.5 Usurpation d'adresse IP 6

2.3.6 Déni de service 7

2.3.7 Virus et vers 8

2.3.8 Cheveaux de Troie et bombes logiques 8

2.3.9 Hoax 9

2.4 Sécurité informatique 9

2.4.1 Sécurité et politique de sécurité 10

2.5 Intrusions 11
2.6 Systèmes de détection d'intrusions 12
2.7 Systèmes de contrôle d'accès par pare-feux 13
2.7.1 Structure d'un système de pare-feu 14
2.7.2 Pare-feu filtre de paquets 14
2.7.3 Pare-feu proxy 15
2.8 Conclusion 15

3 Etat de l'art 16

3.1 Introduction 16
3.2 Renforcements locaux des politiques globales 18
3.2.1 Forme logique des objectifs de sécurité 18
3.2.2 Problème de la localisation 19
3.2.3 Vérification de posture 19
3.2.4 Génération de posture 20
3.2.5 Implémentation 20
3.3 Analyse des pare-feux par les diagrammes de décision binaire 20
3.3.1 Diagramme de décision binaire 20
3.3.2 Règles de filtrage 21
3.3.3 Conversion des règles en expressions booléennes 22
3.3.4 Résultats 24
3.3.5 Validation automatique 26
3.4 Analyse des pare-feux à l'aide du calcul ambiant 26
3.4.1 Calcul ambiant 27

3.4.2 Syntaxe . . . 27
3.4.3 Sémantique . . . 28
3.4.4 Exemple de spécification d'un pare-feu . . . 29
3.5 Langage de spécification des autorisations (ASL) . . . 31
3.5.1 Hiérarchie d'utilisateurs . . . 31
3.5.2 Syntaxe . . . 32
3.5.3 Règles ASL . . . 33
3.6 Politiques par défaut . . . 36
3.7 Critique . . . 37
3.8 Conclusion . . . 38

4 Spécification des pare-feux 39

4.1 Introduction . . . 39
4.2 Motivation . . . 40
4.3 Langage de spécification des politiques des pare-feux . . . 40
4.3.1 Topologie de réseau . . . 41
4.3.2 Syntaxe . . . 42
4.3.3 Sémantique . . . 44
4.4 Exemple de spécification . . . 46
4.4.1 Spécification de la politique globale . . . 48
4.4.2 Spécificaion des politiques locales . . . 48
4.5 Conclusion . . . 48

5 Détection d'anomalies dans les pare-feux 52

5.1 Introduction . . . 52

5.2 Motivation . . . 53

5.3 Modélisation des anomalies des pare-feux . . . 54

5.3.1 Relations entre les règles . . . 55

5.3.2 Ombrage . . . 58

5.3.3 Généralisation . . . 59

5.3.4 Corrélation . . . 59

5.3.5 Redondance . . . 60

5.4 Système de types . . . 60

5.4.1 Algèbre de types . . . 61

5.4.2 Règles de typage . . . 61

5.4.3 Algorithme d'inférence de types . . . 64

5.5 Étude de cas . . . 65

5.6 Conclusion . . . 67

6 Conclusion générale 68

Annexe A 69

A Exemples de spécification de politiques locales 70

Bibliographie 73

Table des figures

2.1 Attaque de type usurpation d'adresse IP 7

2.2 Protection intégrée à multi-niveaux 10

2.3 Environnement dans lequel les intrusions prennent place 11

2.4 Modèle de système de détection d'intrusions 12

2.5 Architecture à base de pare-feu 13

3.1 Diagramme de décision binaire de $(x_1 \vee x_2) \wedge x_3$ 21

3.2 Représentation réduite de $(x_1 \vee x_2) \wedge x_3$ 21

3.3 Exemple de hiérarchie d'utilisateurs 32

4.1 Exemple de protection d'une organisation 47

Liste des tableaux

3.1 Définition d'une règle CISCO 22

3.2 Syntaxe abstraite du calcul ambiant 27

3.3 Congruence structurelle 29

3.4 Relations de réduction 30

4.1 Section de déclaration 42

4.2 Sémantique d'une séquence de règles 46

4.3 Sémantique d'un pare-feu 46

4.4 Spécification de la politique globale de l'exemple 50

4.5 Politique locale du pare-feu Engineering-Finance 51

5.1 Fonctions de projection 61

5.2 Règles de typage 62

5.3 Spécification d'un pare-feu 65

5.4 Algorithme d'inférence de types 66

A.1 Politique locale du pare-feu Engineering-Periphery 70

A.2 Politique locale du pare-feu Allied-Engineeing 71

A.3 Politique du pare-feu Periphery-External 71

A.4 Politique du pare-feu Finance-Periphery 72

A.5 Politique du pare-feu Allied-External 72

Chapitre 1

Introduction

De nos jours, la sécurité informatique représente bien plus ce que la mise en place de codes d'accès ou l'utilisation de techniques de chiffrement. Elle est devenue une question de premier ordre aussi bien pour les particuliers que pour les organisations. Les appels pour une protection efficace des systèmes informatiques et des réseaux ne cessent de se multiplier avec la multiplication des attaques et leurs diversifications. De ce fait, plusieurs techniques de protection des systèmes informatiques sont proposées. Parmi ces techniques, les protections par les systèmes de pare-feux occupent une place de choix.

Les pare-feux sont des dispositifs logiciels ou matériels qui contrôlent l'accès aux différentes composantes d'un réseau en autorisant uniquement la circulation du trafic qui ne viole pas la politique de sécurité mise en place. Les pare-feux sont configurés en utilisant une liste de règles de filtrage implantant une politique de sécurité. Quand un paquet est reçu, la liste des règles est parcourue du début à la fin et une décision est prise quant à l'acceptation ou le rejet du paquet.

Cependant, même s'il est maintenant bien établi que les pare-feux apportent une contribution significative à l'amélioration de la sécurité des réseaux informatiques, il reste que ces dispositifs accusent encore plusieurs lacunes reliées principalement à la manière dont ils sont configurés. En effet, la spécification des règles de filtrage est une tâche délicate, subtile et complexe à la fois. La sécurité assurée par le pare-feu dépend en grande partie de la façon dont ces règles sont spécifiées. Effectivement, un pare-feu peut contenir des anomalies résultantes de situations conflictuelles entre les différentes règles de filtrage qui le spécifient. Ces anomalies constituent des failles de sécurité qui peuvent être exploitées dans un but malicieux.

Les règles d'un pare-feu se développent continuellement et évoluent suivant les changements des besoins de sécurité. De plus, le pare-feu peut contenir plusieurs milliers de règles, souvent

écrites par différents administrateurs à différents moments. Ces constatations nous amènent donc à penser qu'il est très difficile d'éviter d'introduire des anomalies en spécifiant les règles du pare-feu. En conséquence, il est primordial de se doter d'outils formels afin de vérifier si ces règles sont correctement spécifiées et de pouvoir les corriger le cas échéant.

Dans ce mémoire, nous nous proposons d'établir les fondements de base d'une nouvelle technique de vérification formelle des pare-feux. Ainsi, nous présentons un système de types capable de détecter différents types d'anomalies dans les pare-feux. Ce système offre la possibilité de comparer des regroupements de règles et de détecter des anomalies impliquant plus que deux règles. Cette particularité nous distingue de la plupart des techniques de vérification des pare-feux traitées dans la littérature qui, souvent, réalisent des comparaisons systématiques règle à règle. Ainsi, notre technique offre au moins trois avantages :

1. une procédure de vérification efficace en optimisant le nombre de comparaisons des règles. En effet, l'efficacité dans la vérification est cruciale pour pouvoir analyser en pratique des configurations de pare-feux qui peuvent contenir jusqu'à plusieurs milliers de règles.
2. une plus grande flexibilité dans le processus de vérification. En effet, si une règle est ajoutée à un pare-feu déjà vérifié, seules deux comparaisons sont nécessaires pour vérifier la nouvelle configuration : projeter l'ancienne configuration en deux règles (acceptation et refus) et les comparer avec la nouvelle règle.
3. une meilleure précision de l'analyse. En effet, notre système permet de détecter des anomalies entre des séquences de règles plutôt qu'uniquement entre règles individuelles comme c'est le cas des techniques actuelles. Ainsi, notre technique permet des réponses de type : les règles R_1 et R_2 ensemble provoquent une anomalie d'ombrage avec la règle R_3.

Le reste de ce mémoire est organisé comme suit. Au chapitre 2, nous passons au travers des notions de base de la sécurité informatique. Nous décrivons dans le chapitre 3, l'état de l'art en matière de renforcement de la sécurité des réseaux à travers les pare-feux. Au chapitre 4, nous proposons un langage formel, nommé FPSL, dont le but et d'exprimer les politiques des pare-feux. Au chapitre 5, nous présentons un système de types capable de détécter les anomalies dans les pare-feux. À la fin, nous concluons ce mémoire en rapellant nos principales contributions et soulignons quelques développements futures.

Chapitre 2

Eléments de base

Résumé

Ce chapitre est un bref survol des notions de base en matière de sécurité informatique. Nous présentons ici les dangers qui guettent les réseaux et les systèmes informatiques et les moyens mis en place afin de renforcer leur sécurité.

2.1 Introduction

La sécurité informatique consiste à s'assurer que les ressources d'un ordinateur ou d'un réseau sont uniquement utilisées par les personnes autorisées et dans le cadre où il est prévu qu'elles le soient. Cela n'est pas toujours facile à réaliser vu qu'actuellement, de plus en plus d'ordinateurs et de réseaux sont reliés entre eux ou à Internet. Cette connectivité a facilité, certes, l'échange entre ces composants, mais elle a augmenté en même temps les risques d'attaques contre ces réseaux.

Multiples sont les dangers qui guettent les réseaux informatiques : intrusions, virus, piratage, etc. Quoique ces dangers ont des techniques différentes pour nuire à un système, leur approche reste la même : contourner le dispositif de sécurité mis en place dans le réseau. Un tel dispositif est généralement composé de plusieurs éléments tels que les pare-feux, les systèmes de détection d'intrusion, les serveurs proxy, etc.

Nous présentons dans cette partie de notre mémoire quelques éléments de référence afin que le lecteur puisse mieux comprendre ce qu'est la sécurité informatique et l'environnement dans lequel cette activité doit être menée. Le présent chapitre est organisé comme suit : la

section 2.2 donne une définition des menaces internes et externes. La section 2.3 recense les types d'attaques contre un réseau. La section 2.4 définit ce qu'est la sécurité informatique. La section 2.5 donne une idée générale à propos des intrusions. La section 2.6 présente l'idée derrière les systèmes de détection d'intrusion. La section 2.7 discute le contrôle de l'accès à travers les pare-feux. Finalement, la section 2.8 conclut ce chapitre.

2.2 Menaces informatiques

Une menace informatique est tout événement potentiel et appréhendé, de probabilité non nulle, susceptible de porter atteinte à la sécurité d'un système d'information. Une telle menace se présente sous l'une des deux formes suivantes : interne ou externe.

Les menaces externes sont considérées comme étant les points d'attaques potentiels du réseau par des utilisateurs non autorisés localisés à l'extérieur du réseau à protéger. Nous distiguons deux types d'attaques : actives ou passives. Les attaques actives visent à modifier ou à détruire des données transmises à l'extérieur du réseau ou à s'introduire dans le réseau. Les attaques passives ont pour but d'espionner le réseau en écoutant des informations confidentielles transmises à l'extérieur du réseau.

Les menaces internes sont des points d'attaques par des utilisateurs situés au coeur du réseau. Les attaques peuvent aussi être actives ou passives. Les attaques actives visent la modification ou la destruction des données, l'utilisation des applications du réseau interne ou la congestion du réseau. Les attaques passives ont pour but la collecte ou la lecture d'informations confidentielles.

Dans la section suivante, nous nous intéressons aux différents types d'attaques dont les réseaux informatiques sont les principales victimes. Nous essayons de déterminer les causes et les conséquences de telles attaques et évoquons certaines techniques existantes afin de s'en prémunir.

2.3 Attaques sur les systèmes informatiques

Les attaques informatiques sont des dangers qui guettent les réseaux informatiques. Leurs conséquences vont du simple espionnage du réseau ou collecte d'informations sans pour autant causer son disfonctionnement, au détournement du réseau au profit du pirate. De nos jours, les fréquences des attaques informatiques se sont accrues avec la forte connexion des réseaux des différents organismes entre eux ou à l'Internet. Les formes des attaques ont aussi évolués dépendamment des caractéristiques du réseau et des dispostifs assurant sa sécurité.

Dans ce qui suit, nous nous intéressons à dresser la liste de certaines de ces attaques. L'ordre dans lequel elles apparaissent peut, d'une certaine manière, exhiber leur importance ainsi que leur complexité. Nous allons ensuite proposer quelques solutions et techniques pour lutter contre ces attaques.

2.3.1 Vol de mot de passe

Le chemin [9] le plus simple pour accéder à un ordinateur est généralement à travers sa commande de *login*. Cet accès est basé sur la saisie d'un mot de passe correct en un certain nombre d'essais. Les mots de passe d'un système sont encryptés à l'aide d'une fonction de hachage et ensuite stockés dans un fichier sur le disque. Pour vérifier si un mot de passe est valide, le système encode ce mot et le compare à celui stocké.

Un pirate qui veut découvrir un mot de passe, utilise soit un dictionnaire de mots ou va essayer tous les mots possibles jusqu'à l'obtention du bon mot de passe. Afin d'éviter la découverte des mots de passe, il est judicieux de laisser un temps d'attente entre deux essais et de limiter leur nombre avant le blocage du compte. De cette manière, une telle attaque est quasiment impossible car elle est beaucoup trop longue à réaliser. En plus, les utilisateurs qui utilisent des mots de passe du genre : date de naissance, numéro de téléphone etc., facilitent beaucoup la réussite de ce type d'attaque. Une solution simple sera donc de choisir des mots de passe longs et difficiles à deviner.

2.3.2 Ingénierie sociale

L'ingénierie sociale n'est pas vraiment une attaque informatique, elle est plutôt une méthode utilisée pour soutirer des informations, à propos d'un système ou des mots de passe, à quelqu'un sans qu'il s'en rende compte. Elle consiste surtout à se faire passer pour quelqu'un que l'on n'est pas, en inventant un quelconque motif, et de demander à sa victime des informations personnelles (comme son *login* ou son mot de passe). Cette attaque se fait soit au moyen d'une simple communication téléphonique et quelques mots, soit par courriel.

Pour éviter ce type d'attaques, il est conseillé de ne jamais communiquer des données confidentielles, par téléphone ou par courriel, à un individu dont l'identité n'a pas été vérifiée.

2.3.3 Porte de derrière

Lorsqu'un pirate réussit à accéder à un serveur, il souhaiterait y retourner sans avoir à relever une autre fois les défis posés par le système de sécurité en place. Pour cela, il laisse des portes de derrière (*backdoors*) qui lui permettront de reprendre facilement le contrôle

du système. Les intrus procèdent de deux manières : soit, ils mettent une seule porte bien cachée ou ils en mettent beaucoup en espérant qu'une ne sera pas détectée. Les portes de derrière sont le résultat de :

- L'ajout d'un nouveau compte sur le serveur avec un mot de passe choisi.
- La modification de la configuration du pare-feu pour qu'il accepte une adresse IP définie.
- La création d'un compte ftp.
- L'ouverture de l'application telnet.
- L'ouverture des ports, etc.

Les portes de derrière sont très dures à détecter. Cependant, nous pouvons minimiser leur impact en surveillant les ports ouverts et en s'inquiétant de tout comportement suspect du système.

2.3.4 Reniflement

Le reniflement (*sniffing*) est une technique utilisée pour dérober des mots de passe. En effet, lors d'une connexion à un réseau qui utilise la diffusion (*broadcasting*), les données transitant dans ce réseau arrivent à toutes les cartes réseau des ordinateurs connectés. En temps normal, seules les trames destinées à une machine donnée sont lues par cette machine, les autres sont ignorées.

Grâce à un logiciel appelé "sniffer", un pirate peut intercepter les trames qui circulent sur le segment de réseau. Si quelqu'un se connecte à ce moment-là et envoi son mot de passe en claire sur le réseau (comme avec le protocole Telnet), il sera possible de le lire. De même, le pirate peut savoir à tout moment quelles pages web consultent les personnes connectées au réseau, les sessions ftp en cours et les courriels en envoi ou en réception.

Cette technique n'est pas forcement illégale, car elle permet aussi de diagnostiquer un réseau, mais mal utilisée, elle devient la source d'une attaque. Elle est cependant limitée, car l'intrus doit se trouver sur le même segment de réseau que la machine qu'il veut pirater. Il est conseillé donc de limiter la taille des sous-réseaux internes et de les séparer par des ponts ou un matériel qui n'utilise pas la diffusion : routeur, pare-feu, etc.

2.3.5 Usurpation d'adresse IP

L'usurpation d'adresse (*spoofing*) consiste à se faire passer pour une autre machine en falsifiant son adresse IP. Nous pouvons résumer cette attaque comme suit : tout d'abord l'intrus

doit choisir le serveur qu'il veut attaquer et obtenir le maximum d'informations à propos d'une machine (ou des machines) autorisée à se connecter à ce serveur afin de la rendre iopérente. Ensuite, l'intrus falsifie son adresse IP en la remplaçant par celle de la machine rendue inopérante. Finalement, l'intrus initie une session de communication avec le serveur.

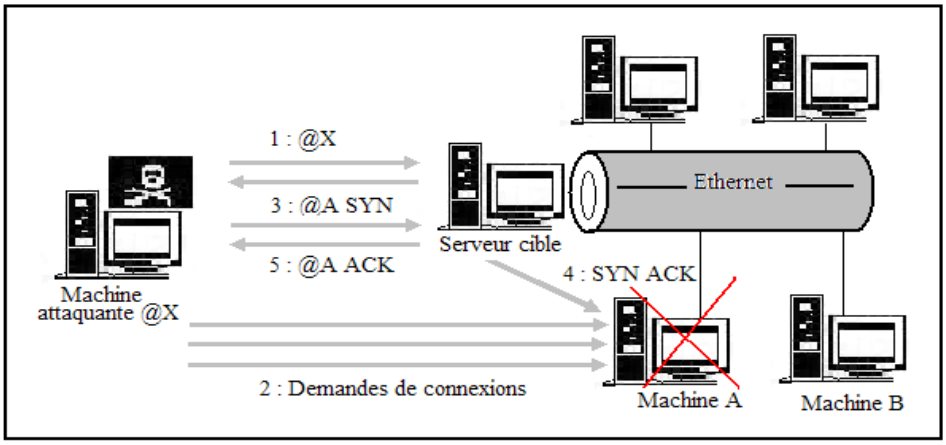

FIG. 2.1 – *Attaque de type usurpation d'adresse IP*

Cette attaque est très dure à réaliser, car elle se passe en aveugle. En effet, l'attaquant ne reçoit pas les données que le serveur envoie et il doit maîtriser parfaitement les protocoles de manière à savoir ce qu'attend le serveur à tout moment. Une manière de pallier à ce problème sera d'indiquer la route que le paquet doit prendre pour atteindre la machine de destination, mais beaucoup de serveurs ne les autorisent plus.

2.3.6 Déni de service

Le déni de service (*Denial of Service*) est une attaque qui vise à générer des arrêts de service et donc d'empêcher le bon fonctionnement d'un système. Cette attaque ne permet pas en elle-même d'avoir accès à des données. En général, le déni de service va exploiter les faiblesses de l'architecture d'un réseau ou d'un protocole. Différents types de dénis de services existent : le "flooding", le "TCP-SYN flooding", le "smurf", le débordement de tampon, etc. Ces attaques causent la diminution de la bande passante et la déconnexion ou le blocage de la machine cible.

- **TCP-SYN flooding :** le "flooding" consiste à inonder une machine par de nombreux paquets IP de grosses tailles causant ainsi son blocage et sa déconnexion du réseau. Le "TCP-SYN flooding", une variante du "flooding", s'appuie sur le protocole TCP. En effet, un grand nombre de demandes de connexion au serveur sont envoyées à partir

de plusieurs machines ou d'une seule qui falsifie son adresse IP. Le serveur va répondre en envoyant un grand nombre de paquets SYN-ACK et va attendre en réponse un ACK qui ne viendra jamais. Si les paquets sont envoyés plus vite que le *timeout* des demi-connexions, le serveur se sature et finit par se déconnecter.

- **Smurf :** le "smurf" est une ancienne attaque basée sur le ping et les serveurs de diffusion. Sa technique est la suivante : le pirate falsifie d'abord son adresse IP pour se faire passer pour la machine cible à attaquer. Il envoie alors un ping sur un serveur de diffusion. Toutes les machines connectées renverront chacune un pong au serveur qui le fera suivre à la machine cible. Celle-ci sera alors inondée sous les paquets et finira par se déconnecter.
- **Débordement de tampon :** cette attaque a pour origine une faille du protocole IP. Un pirate envoie à une machine cible un paquet d'une taille supérieure à la taille maximale admise pour les paquets. Celui-ci sera alors fractionné pour l'envoi et rassemblé par la machine cible. A ce moment, il y aura débordement des variables internes de la machine. Suite à ce débordement, plusieurs cas se présentent : soit que la machine se bloque, redémarre ou ce qui est plus grave, écrase du code en mémoire.

2.3.7 Virus et vers

Nous pouvons définir un virus comme étant un programme caché dans un autre et qui peut s'exécuter et se reproduire en infectant d'autres programmes ou d'autres ordinateurs. Les dégâts causés par un virus vont du simple programme qui affiche un message à l'écran, au programme qui formate le disque dur après s'être multiplié.

Les virus sont très difficiles à détecter parce qu'il en existe de nouveaux à tous les jours. De plus, certains virus sont mutants, c'est-à-dire qu'ils sont des variantes de virus déjà connus et que d'autres sont polymorphes et peuvent modifier leur signature. Le seul moyen de s'en prémunir est d'avoir un bon anti-virus régulièrement mis à jour.

Un vers est un programme autonome qui peut se reproduire et se déplacer sur le réseau sans l'intervention de l'utilisateur. Il se développe surtout grâce à la messagerie sous la forme d'une pièce jointe. Pour s'en prémunir, il suffit de ne pas ouvrir les fichiers joints en cas de doute.

2.3.8 Cheveaux de Troie et bombes logiques

Le cheval de Troie (*trojan*) est un programme informatique dont le but est de créer une porte de derrière afin de pirater une machine. Il peut aussi voler des mots de passe, copier des données ou exécuter des actions nuisibles.

La détection d'un cheval de Troie est difficile, car il faut savoir si l'action effectuée par le programme est un comportement normal ou malicieux. Cependant, il existe certains symptômes qui peuvent nous révéler la présence d'un cheval de Troie : la machine est connectée au réseau et échange des données alors que l'on ne fait rien, réactions curieuses de la souris, ouverture automatique de programmes, plantages à répétition, etc.

Un moyen simple de détecter les cheveaux de Troie est de vérifier les ports ouverts. Un port ouvert en permanence alors qu'aucun programme ne tourne signifie qu'un cheval de Troie est présent sur la machine.

Les bombes logiques sont des dispositifs programmés dont le déclenchement s'effectue à un moment déterminé en exploitant la date du système, le lancement d'une commande ou un appel au système. Elles sont en général invisibles tant que la condition n'a pas été remplie. Leurs actions peuvent variées : la consommation des ressources, la destruction des fichiers, la création des failles systèmes, la récupération des numéros de licences, etc.

2.3.9 Hoax

Les Hoax sont des canulars envoyés par courriel. Ils ne font aucune action nuisible sur la machine, mais ils engorgent les réseaux suite à leur propagation massive. De plus, ils surchargent les boîtes à lettres et propagent la désinformation. En général, ils propagent une nouvelle d'un nouveau virus ou d'une chaîne de courriel et demandent sa diffusion.

Dans la section suivante, nous nous intéressons au problème de la sécurité. Nous définission ce qu'est la sécurité informatique et comment la concrétiser à travers des systèmes dont l'objectif est de protéger les réseaux contre les attaques.

2.4 Sécurité informatique

La sécurité informatique consiste à s'assurer que les ressources matérielles et logicielles sont uniquement utilisées dans le cadre où il est prévu qu'elles le soient. Elle doit garantir les propriétés suivantes :

- **Intégrité** : les données reçues sont bel et bien celles qui ont été envoyées.
- **Confidentialité** : seules les personnes disposant des privilèges d'accès sont autorisées à accéder aux ressources.
- **Disponibilité** : le bon fonctionnement du système est assuré.

La sécurité informatique ne peut être traitée sans une définition claire et précise des objectifs à atteindre et des outils à mettre en place. Ces objectifs déterminent la politique de sécurité à instaurer et à suivre.

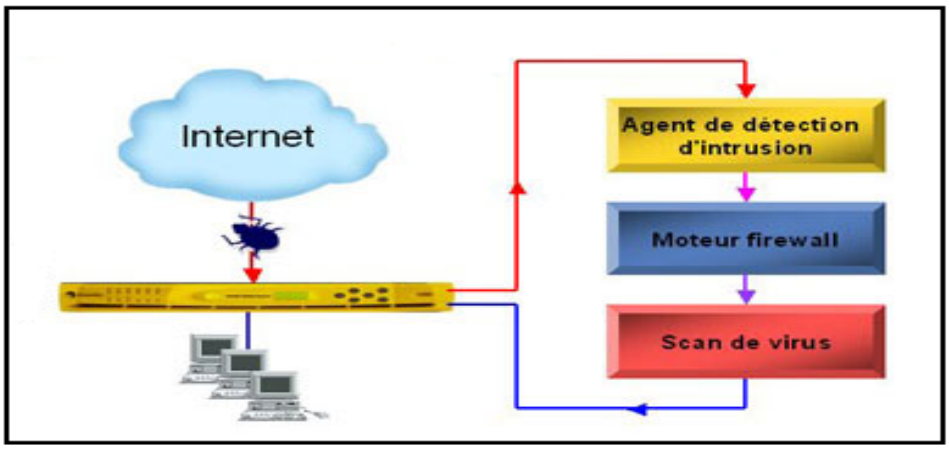

FIG. 2.2 – *Protection intégrée à multi-niveaux*

2.4.1 Sécurité et politique de sécurité

La sécurité des systèmes informatiques se résume généralement à garantir les droits d'accès aux données et aux ressources d'un système en mettant en place des mécanismes d'authentification et de contrôle. Ces mécanismes permettent d'assurer que les utilisateurs des dites ressources possèdent uniquement les droits qui leur ont été octroyés. Toutefois, la sécurité informatique doit être étudiée de telle manière à ne pas empêcher les utilisateurs de développer les usages qui leur sont nécessaires et de faire en sorte qu'ils puissent utiliser le système d'information en toute confiance. C'est la raison pour laquelle il est nécessaire de définir une politique de sécurité.

Une politique de sécurité est l'ensemble des orientations suivies par une organisation en matière de sécurité. Elle a pour finalités de :

- Élaborer des règles et des procédures à mettre en oeuvre dans les différents services de l'organisation.
- Définir les actions à entreprendre et les personnes à contacter dans le cas d'une attaque informatique.
- Sensibiliser les utilisateurs aux problèmes liés à la sécurité des systèmes d'information.

La sécurité d'un système informatique fait souvent l'objet de métaphores. Nous la comparons régulièrement à une chaîne en expliquant que le niveau de sécurité d'un système est caractérisé par le niveau de sécurité du maillon le plus faible. Cela signifie que la sécurité doit être abordée dans un contexte global visant :

- La sensibilisation des utilisateurs aux problèmes de sécurité.

- La sécurité logique et en d'autres termes la sécurité des données.
- La sécurité des télécommunications.
- La sécurité des applications.
- La sécurité physique, soit la sécurité au niveau des infrastructures matérielles.

Dans ce qui suit, nous définissons ce que sont les intrusions et leurs formes. Nous présentons, par la suite, deux dispositifs permettant de protéger les réseaux informatiques contre ces instrusions à savoir : les systèmes de détection d'intrusions et les pare-feux, .

2.5 Intrusions

Qualifiée d'intrusion, toute action qui a pour fin de compromettre la disponibilité, l'utilité, l'intégrité, l'authenticité, la confidentialité et/ou la possession d'un système d'information [9]. Généralement, une intrusion se manifeste sous l'une des formes suivantes [23] :

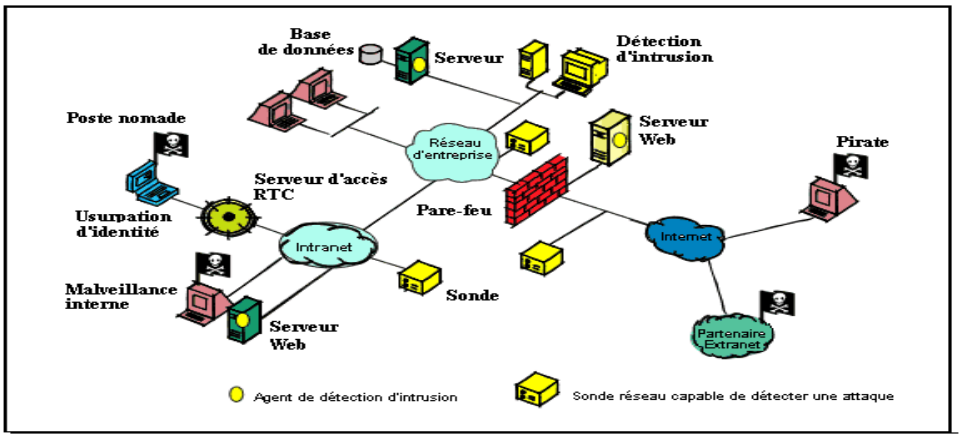

FIG. 2.3 – *Environnement dans lequel les intrusions prennent place*

- Lire des informations protégées : ces informations peuvent être les données internes d'une compagnie, des numéros de cartes de crédit, des données financières ou des fichiers de mots de passe.
- Changer des informations protégées : modifier ou supprimer des informations protégées d'un système.
- Utiliser les ressources protégées des ordinateurs : le temps CPU, l'espace disque, les imprimantes, etc.
- Arrêter un service : empêcher le fonctionnement d'une machine ou d'un service.

- Usurper l'identité d'un ordinateur pour attaquer d'autres systèmes derrière un système de pare-feu.

2.6 Systèmes de détection d'intrusions

La détection d'intrusions est définie comme étant toute action dont le but est de détecter une tentative d'intrusion [23]. Tout système qui remplit cette propriété est alors considéré comme étant un système de détection d'intrusions.

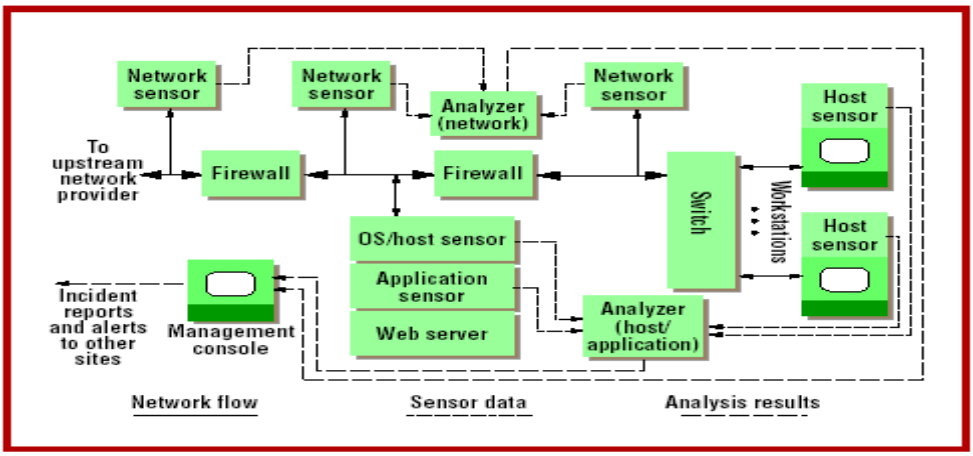

FIG. 2.4 – *Modèle de système de détection d'intrusions*

Un système de détection d'intrusions (ou SDI) est un composant de sécurité ayant pour but de surveiller un système informatique afin d'essayer de détecter tout comportement malicieux. Ce dispositif utilise plusieurs techniques [13] pour le faire :

- **Détection d'intrusions d'anomalie** : dans ce cas, le SDI est à la recherche de toute activité qui a un comportement anormal. Le SDI connaît le comportement normal et essaie de détecter une possible intrusion basée sur la présence d'un comportement anormal. Ce type de système est capable de s'adapter aux nouvelles méthodes d'intrusion, mais il peut faillir parce que toutes les tentatives d'intrusion ne sont pas nécessairement des activités malicieuses.
- **Détection d'intrusions d'abus** : le SDI utilise des patrons d'attaques connues et essaie de détecter ces patrons. Ce type de système est très efficace contre les attaques dont les comportements ressemblent aux patrons, mais ils sont absolument inefficaces dans le cas contraire.
- **Détection d'intrusions combinée** : un SDI qui utilise les deux techniques énumérées précédemment.

Plusieurs systèmes SDI ont été développés par différentes entreprises, mais aucun d'eux ne peut se prétendre d'être parfait. En effet, il y a deux types d'erreurs de fonctionnement de ces systèmes qui peuvent apparaître. La première, la plus sérieuse, est baptisée fausse négative (*false negative*) et survient quand un SDI ne détecte pas une action d'intrusion. La seconde nommée fausse positive (*false positive*) émerge quand le SDI détecte une fausse intrusion. La première erreur est très sérieuse car elle compromet tout le système. La seconde peut devenir sérieuse dans le cas où elle apparaît souvent et qu'elle est ignorée.

2.7 Systèmes de contrôle d'accès par pare-feux

Dans la vie courante, un pare-feu (*firewall*) est une barrière qui stoppe le feu dans les deux directions. En informatique, un pare-feu est un filtre de paquets à l'entrée d'un réseau. Le trafic, par conséquent, doit être capable de passer à travers cette barrière, mais il doit être minutieusement examiné avant.

Les pare-feux sont une technologie informatique récente très répandue de nos jours. A travers leur brève histoire, ils ont évolués considérablement. Les premiers pare-feu étaient de simples filtres de paquets pour le contrôle d'accès entre les réseaux. Aujourd'hui, ils fournissent toute une panoplie de fonctionnalités de sécurité [21] :

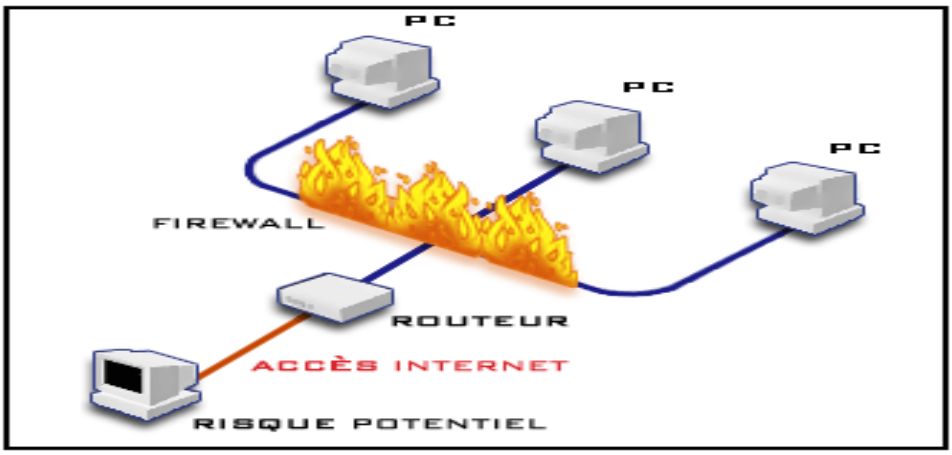

FIG. 2.5 – *Architecture à base de pare-feu*

- Authentifier les utilisateurs en utilisant des jetons ou des mots de passe jetables (*one-time password*).
- Encrypter et décrypter le trafic traversant le réseau.

- Créer des tunnels privés et sécurisés, et des réseaux virtuels à travers des réseaux peu sûrs.
- Protéger le réseau contre les virus.
- S'assurer que sa propre sécurité n'est pas compromise.

Un pare-feu est un composant logiciel ou matériel ou un ensemble de composants qui restreignent l'accès entre un réseau protégé et Internet ou d'autres réseaux [8]. Le système de pare-feu contrôle l'accès en se basant sur des règles de filtrage. Ces règles sont conçues pour renforcer la politique de sécurité d'une organisation. Un système à base de pare-feu assure que :

- L'organisation concentre son effort de sécurité à chaque point de connexion.
- Tout le trafic entre Internet et le réseau interne peut être réglé et contrôlé afin de s'assurer que la politique de sécurité est atteinte.

Cependant, aucun système incluant un pare-feu proprement configuré n'est sûr à cent pour cent. Les systèmes à base de pare-feu sont souvent des configurations de logiciel ou matériel trop grandes et complexes, et il ne serait pas sage de compter complètement sur des systèmes qui sont conçus, la plus part du temps, de composants non sûrs.

2.7.1 Structure d'un système de pare-feu

Un système de pare-feu doit être conçu avec deux objectifs : se protéger lui-même et protéger le réseau derrière lui des attaques. Pour fournir une pareille protection, les systèmes de pare-feu sont conçus selon le principe de la défense en profondeur (*defense in depth principle*). En effet, plusieurs mécanismes sont utilisés dans une sorte de lignes arrières qui si une ligne de défense tombe, une autre assure la défense et ainsi de suite. Pour plus d'information sur la stratégie de la défense en profondeur, il est conseillé de se référer aux travaux de Chapman [8].

Nous abordons dans ce qui suit, deux principales classes de pare-feux [18] à savoir le pare-feu filtre de paquets et le pare-feu proxy.

2.7.2 Pare-feu filtre de paquets

Un filtre de paquet est un routeur spécial ou un programme qui examine les paquets IP reçus et décide si ces paquets passent ou non, suite à la vérification d'un ensemble de règles qui spécifient ce qu'il faut faire avec ces paquets. La règle par défaut est généralement de rejeter tout ce qui n'est pas explicitement spécifié pour être autorisé. Elle peut être aussi d'autoriser tout ce qui n'est pas explicitement spécifié pour être rejeté.

L'avantage de ce type de filtre est qu'il est transparent à l'utilisateur et aux applications. Tous les logiciels existants dans un réseau ne doivent subir aucune modification. Par contre l'administrateur du pare-feu doit avoir une bonne connaissance des logiciels installés afin de mettre à jour les règles du filtre pour garantir le fonctionnement des dits logiciels.

2.7.3 Pare-feu proxy

Un pare-feu proxy est un serveur intermédiaire avec qui les deux machines hôtes interne et externe communiquent. L'utilisateur croit qu'il utilise directement une machine hôte externe, mais tout passe par une machine proxy qui peut demander un mot de passe pour permettre l'accès. Dans ce type de système, tous les logiciels existants doivent être modifiés si une machine veut être capable de rejoindre l'autre rive de la barrière.

2.8 Conclusion

La sécurité informatique est une question qui ne cesse de se développer avec la prolifération des réseaux et des menaces informatiques. Elle devient un objectif urgent et vital pour protéger les systèmes informatiques contre les accès non autorisés et abusifs.

Des dispositifs de détection d'intrusions et de protection des réseaux, tels que les SDI et les pare-feux, sont développés et utilisés afin de renforcer cette idée de sécurité. Cependant, aucun d'entre eux ne peut garantir une protection totale contre les différentes attaques informatiques, quelles soient internes ou externes.

En l'absence d'une définition claire et concise des objectifs de sécurité à atteindre et des sites à protéger, les outils de protection actuels échouent à fournir une protection fiable. Cela est dû au fait que la sécurité informatique est traitée en l'absence d'une vision globale mettant en relief l'ensemble des orientations suivies par une organisation en matière de sécurité. Ces orientations, qui visent à définir et analyser les failles d'un système d'information et à combler ces failles par la mise en place de dispositifs de protection contre les attaques informatiques, définissent ce que nous appellons une politique de sécurité.

Chapitre 3

Etat de l'art

Résumé

Les pare-feux sont des dispositifs utilisés pour contrôler le trafic et améliorer la protection du réseau en adoptant un comportement de filtrage vis à vis des paquets qui y transitent. Dans ce chapitre, nous adressons l'état des recherches dans le domaine de la sécurité des réseaux à travers les techniques de spécification, d'analyse et d'implémentation des pare-feux et des règles de filtrage des paquets.

3.1 Introduction

Les pare-feux jouent un rôle important dans la gestion du trafic dans un réseau. En acceptant ou en rejetant les paquets, ils permettent de renforcer, à la fois, la sécurité et la performance du réseau. Pour cela, ils sont dotés de mécanismes qui permettent de spécifier le comportement à adopter envers les paquets circulant dans le réseau. Ces mécanismes sont souvent implantés par un ensemble de règles dites «règles de filtrage».

Les règles expriment la politique de sécurité définie sur le trafic entrant ou sortant d'un réseau. Les paquets formant ce trafic doivent satisfaire à cette politique pour qu'ils puissent être autorisés par le pare-feu. Les règles de filtrage sont complexes et sont souvent spécifiées manuellement. En pratique, ces règles évoluent suivant les besoins et donnent lieu à deux problèmes majeurs :

1. Plus la liste des règles est complexe, moins elle est facile à comprendre. En plus, l'ajout ou la suppression d'une ou plusieurs règles peut induire à un changement de la

sémantique de cette liste. En particulier, certaines règles inconsistantes peuvent ouvrir une brêche de sécurité dans le pare-feu.

2. La recherche dans une liste de règles peut devenir très coûteuse et particulièrement pour les routeurs, causant d'avantage la lenteur du réseau.

Dans ce chapitre, nous nous intéressons aux travaux de recherche de J. D. Guttman, de S. Hazelhurst, de F. Neilson et *al.* et de S. Jajodia et *al.* Dans ces travaux, les auteurs ont développé des techniques de spécification et d'analyse des pare-feux.

Dans [16], Guttman présente un langage simple pour exprimer des politiques de sécurité globales et propose deux algorithmes. Le premier algorithme génère automatiquement des règles de filtrage à partir de la politique de sécurité globale et de la topologie du réseau décrite dans un certain langage. Le deuxième compare un ensemble de filtres avec la politique globale pour relever toute violation de cette politique.

La technique proposée par S. Hazelhurst [17] convertit les règles d'un pare-feu en un format logique qui peut être représenté par un diagramme de décision binaire. Il propose ensuite l'utilisation de ce diagramme pour l'exploration de l'ensemble de ces règles afin de répondre à des questions relatives à leur validation.

F. Neilson et *al.* [22] utilisent le calcul ambiant [6] pour spécifier des pare-feux. Les pare-feux et les agents qui veulent les accéder sont considérés comme des processus ambiants et sont, en conséquence, spécifiés à l'aide de ce calcul. La validation de la spécification consiste à prouver qu'un agent ne peut passer le pare-feu que s'il possède les clés appropriées.

S. Jajodia et *al.* [20] présentent un langage logique, dont le nom est ASL *(Authorization Specification Language)*, pour spécifier des autorisations d'accès à des objets par des sujets dans un système simple et unifié. Pour arriver à cela, ils commencent par définir un ensemble de prédicats et de règles pour schématiser les autorisations d'accès et proposent, ensuite, une méthode pour vérifier la consistance des règles spécifiées.

Ce chapitre est organisé de la manière suivante : la section 3.2 résume les techniques formelles développées pour résoudre le problème de protection des paquets dans un réseau. La section 3.3 présente une technique d'encodage des règles d'un pare-feu sous la forme d'un diagramme de décision binaire et l'utilisation de ce diagramme pour la vérification des pare-feux. La section 3.4 considère la spécification et la validation des pare-feux à l'aide du calcul ambiant. La section 3.5 présente un aperçu des éléments essentiels du langage ASL. La section 3.6 présente les différentes formes de politiques pouvant être appliquées par défaut. La section 3.7 donne une brève critique de l'état de l'art. Finalement la section 3.8 conclut ce chapitre.

3.2 Renforcements locaux des politiques globales

Les pare-feux contrôlent l'accès au réseau à l'aide des mécanismes de filtrage. Le filtrage permet d'empêcher l'envoi ou la réception des paquets jugés nuisibles ou inappropriés. Ces mécanismes doivent être en mesure de décider du comportement à adopter vis à vis d'un paquet en examinant simplement l'information relative à son entête.

Dans [16], Guttman aborde des questions relatives aux politiques de contrôle d'accès au réseau et comment les renforcer. Son objectif est d'utiliser les services de sécurité fournis par les routeurs de filtrage de paquets afin d'assurer une protection fiable dans les réseaux complexes. Il propose, dans un premier temps, un langage proche de Lisp pour exprimer des politiques de sécurité globales. Ensuite, il présente deux algorithmes, le premier permet de calculer des filtres locaux à partir d'une politique de sécurité globale et de la topologie du réseau et le deuxième permet de comparer un ensemble de filtres avec la politique globale afin de détecter toute violation de cette politique.

3.2.1 Forme logique des objectifs de sécurité

Un objectif de sécurité permet de dresser la liste des menaces potentielles à prévenir et des points du réseau à protéger. Ceci permet d'empêcher les paquets non autorisés d'atteindre les zones du réseau qu'il faut protéger. Ces zones sont appelées des points de renforcement et un comportement de filtrage leurs est affecté.

Deux types d'informations peuvent contribuer à implémenter des objectifs de sécurité à savoir le contenu de l'entête du paquet et du chemin qu'il a pris à travers le réseau. L'entête d'un paquet permet de connaître son adresse source, son adresse destination, le chemin qu'il va suivre pour atteindre sa destination, le type de protocole utilisé, etc. Un objectif de sécurité concerne en général deux zones du réseau a et a' et peut avoir la forme suivante :

Si p était dans a et plus tard il a rejoint a',
Alors l'entête de p satisfait φ.

Ceci est interprété comme suit : si un paquet p part de la zone a, traverse un routeur r et atteint la zone a', son entête doit nécessairement satisfaire une propriété φ.

L'ensemble des objectifs de sécurité pour chaque paire de zones (a, a') constitue la politique de sécurité pour ces zones. Donc, une politique de sécurité est une fonction π définie comme suit :

$$\pi : A \times A \rightarrow B$$

où A est un ensemble de zones du réseau et B est une collection d'ensembles de paquets.

3.2.2 Problème de la localisation

Les propriétés de sécurité sont des propriétés globales qu'il faut renforcer localement. Les mécanismes de renforcement consistent à ce que chaque routeur prenne lui-même ses propres décisions par rapport aux paquets à filtrer. Le routeur dispose seulement d'informations locales à propos des interfaces sur lesquelles il reçoit ou envoie les paquets. La localisation est donc le problème de réalisation d'objectifs globaux en utilisant des informations locales. Son but principal est de définir, à la fois, des objectifs de sécurité au niveau global et une déclaration de filtrage au niveau local. Cette déclaration porte le nom de posture.

Une posture définit une spécification d'un comportement de filtrage qui est appliqué à un filtre situé au niveau de chaque paire (routeur, direction) sur le réseau. C'est une fonction :

$$f : R \times A \times D \longrightarrow B$$

où R est l'ensemble des routeurs, A est l'ensemble des zones, D est l'ensemble des deux directions *in* et *out* et B est un ensemble de paquets.

Un filtre est représenté par une contrainte Φ qui indique que le filtre autorise un paquet p à passer dans le cas où $p \in \Phi$. Deux filtres sont associés à chaque interface d'un routeur : un va examiner les paquets entrants (*inbound*), l'autre ceux sortants (*outbound*). Nous interprétons Φ comme étant l'ensemble des paquets abstraits qui ont la permission de passer un filtre situé sur une interface et dans une direction indiquée.

Guttman propose deux algorithmes qui ont pour objectif de résoudre deux problèmes. Le premier part d'une politique de sécurité globale et d'une posture de filtrage et détermine si cette dernière renforce la politique globale. Le deuxième construit une posture de filtrage locale partant d'une politique de sécurité globale. Ces algorithmes sont dits de vérification de posture et de génération de posture.

3.2.3 Vérification de posture

Nous appelons un ensemble de faisabilité $\mathcal{F}$, l'ensemble de tous les paquets qui franchissent tous les filtres lors de leur passage par un chemin reliant deux zones du réseau. Pour vérifier si une posture f renforce une politique $\pi(a_0, a_i)$, chaque chemin entre deux zones a_0 et a_i du réseau est vérifié afin de s'assurer que l'ensemble de faisabilité $\mathcal{F}$ pour ce chemin est inclus dans $\pi(a_0, a_i)$. En d'autres termes, si σ est le chemin qui commence à la zone a_0 et se termine à la zone a_i, il faut vérifier que $\mathcal{F}(\sigma) \subset \pi(a_0, a_i)$. Si c'est le cas, alors $\mathcal{F}$ renforce la politique $\pi(a_0, a_i)$. Dans le cas contraire, les violations sont exhibées sous la forme $\mathcal{F}(\sigma) \backslash \pi(a_0, a_i)$.

3.2.4 Génération de posture

La génération de posture déduit une posture f à partir d'une politique de sécurité $\pi(a_0, a_i)$. L'idée de cet algorithme est la suivante : nous commençons par une simple posture f_0 et nous la corrigeons successivement pour éliminer toutes les violations. En utilisant l'approximation courante f_i, nous calculons $\mathcal{F}(p)$. Si $\mathcal{F}(p) \subset \pi(a_0, a_n)$, alors $f_{i+1} = f_i$. Sinon, nous corrigeons les violations appartenant à V= $\mathcal{F}(p) \backslash \pi(a_0, a_n)$.

3.2.5 Implémentation

Une implémentation [15], sous le nom de NPT *(Network Policy Tool)*, des deux algorithmes précédents a été réalisée. Dans cette implémentation, un utilisateur peut éditer une posture produite par l'algorithme de génération de posture pour opérer des améliorations locales. L'algorithme de vérification de posture peut alors être utilisé pour assurer que ces améliorations n'ont pas causé de violations.

3.3 Analyse des pare-feux par les diagrammes de décision binaire

Dans [17], Hazelhurst considère une méthode qui exprime un ensemble de règles d'un pare-feu sous la forme : **si** <condition> **alors** <action> et propose un encodage de ces règles sous un format logique à l'aide d'un diagramme de décision binaire. Son principal avantage est qu'elle permet l'exploration rapide d'un ensemble de règles d'une taille raisonnable.

3.3.1 Diagramme de décision binaire

Un digramme de décision binaire est un arbre binaire dans lequel une expression booléenne est représentée. L'évaluation de l'expression $(x_1 \vee x_2) \wedge x_3$ de la Figure 3.1 se fait à partir de la racine de l'arbre. Ensuite, selon la valeur de la variable, nous choisissons la direction à explorer. Dans [5], Bruyant introduit le concept de diagramme de décision binaire réduit et ordonné respectant les contraintes suivantes :

- Tous les états terminaux dupliqués sont supprimés. Deux états terminaux 0 et 1 resteront à la fin.
- Tous les états non terminaux dupliqués sont supprimés.
- Tous les tests redondants sont supprimés.

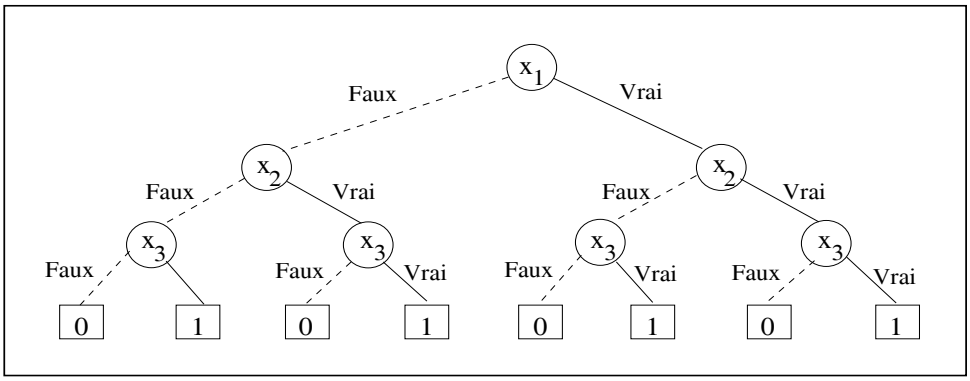

FIG. 3.1 – *Diagramme de décision binaire de* $(x_1 \vee x_2) \wedge x_3$

Les diagrammes de décision binaire ont deux propriétés : ils sont des représentations compactes d'expressions booléennes et pour un ordre donné, la représentation d'une expression booléenne est canonique, c'est-à-dire que les diagrammes pour les expressions $\neg(a \wedge (b \vee c))$ et $(\neg a \vee \neg b) \wedge (\neg a \vee \neg c)$ sont les mêmes. Le diagramme réduit de l'expression $(x_1 \vee x_2) \wedge x_3$ est donné par la Figure 3.2.

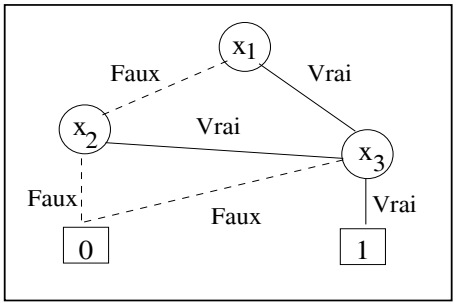

FIG. 3.2 – *Représentation réduite de* $(x_1 \vee x_2) \wedge x_3$

Finalement, puisqu'un diagramme de décision est une représentation réduite d'une expression booléenne, sa taille dépend de l'ordonnancement choisi des variables de cette expression. Trouver un ordonnancement optimal à ces variables est un problème NP-complet [4].

3.3.2 Règles de filtrage

Un ensemble de règles est une liste de règles qui implémente la politique de sécurité du pare-feu. Une règle prend le format : **si** <condition> **alors** <action> où l'action est soit accepter

ou rejeter un paquet. La Table 3.1, donne une description générique de la définition d'une règle dans une liste d'accès pour un routeur CISCO. Cette description signifie que n'importe quel paquet TCP venant de l'adresse IP 20.9.17.8 et destiné à l'adresse 121.11.127.20 est à accepter si l'adresse du port de destination est dans la plage [23..27].

```
access-list  101  permit  tcp     20.9.17.8  0.0.0.0
                              121.11.127.20  0.0.0.0
                              range 23  27
```

TAB. 3.1 – *Définition d'une règle CISCO*

Dans une liste d'accès, les règles sont analysées une par une pour vérifier si la condition correspond au paquet reçu. Si oui, le paquet est alors soit accepté, soit rejeté selon l'action spécifiée. Sinon la condition ne correspond pas et la recherche continue avec la règle suivante et ainsi de suite jusqu'à la fin de la liste. Si aucune règle ne correspond, le paquet est rejeté. Toutefois, il est à noter que l'ordre dans lequel les règles sont spécifiées est très important, car ces règles sont vérifiées dans un ordre donné. Le changement de cet ordre peut changer le comportement du pare-feu vis à vis d'un paquet : changer une acceptation en un rejet et vice-versa.

Une règle d'accès CISCO [19], respecte la forme de la Table 3.1. Les composantes de cette règle ont la signification suivante :

- *Permit* ou *reject* : indique que les paquets correspondant à la règle sont à accepter, sinon à rejeter.
- Le protocole du paquet : TCP dans ce cas, UDP et ICMP dans d'autres.
- L'adresse source : quatre segments, chacun d'eux est un nombre entre 0 et 255.
- Le masque de l'adresse source : quatre segments aussi.
- L'adresse destination : similaire à l'adresse source.
- Le masque de destination : similaire au masque source.
- La plage des adresses du port.

3.3.3 Conversion des règles en expressions booléennes

La clé de la technique proposée est la suivante : les nombres peuvent être représentés comme des vecteurs-bit. Au bas niveau, un segment d'adresse est un vecteur de 8 bits. En utilisant un diagramme de décision binaire, nous pouvons représenter symboliquement les ensembles des nombres et opérer sur eux plusieurs opérations logiques. Par exemple, pour représenter le nombre x sur 8 bits, l'auteur introduit le vecteur-bit $< x_7, ..., x_0 >$ où chaque x_i est une

variable booléenne du diagramme de décision binaire. La condition pour que le vecteur de x soit égal à 3 est $<x_7, ..., x_0> = <$**f, f, f, f, f, f, t, t**$>$. En d'autres termes nous obtenons l'expression booléenne $x'_7x'_6x'_5x'_4x'_3x'_2x_1x_0$ où x_0 *et* x_1 valent vrai, et x'_2, x'_3, x'_4, x'_5, x'_6 *et* x'_7 valent faux.

Cette technique introduit un certain nombre de variables et d'expressions pour représenter l'information contenue dans une règle. Nous assignons à chaque protocole un nombre dans la plage 0, ..., $n_p - 1$, et nous introduisons m_p ($m_p = log_2 n_p$) variables dont les valeurs sont dans la plage π_0, ..., π_{m_p-1} pour encoder le protocole utilisé. Si la règle fait référence à un paquet utilisant le protocole TCP, dont la valeur est égale à 3 par exemple, alors ce dernier est représenté par l'expression $<\pi_2, \pi_1, \pi_0> = <$**f, t, t**$>$ où tout simplement $\pi'_2\pi_1\pi_0$. Par conséquent, les composantes d'une règle peuvent être exprimées de la manière suivante :

- Pour chaque segment de l'adresse source, l'auteur introduit 8 variables de la forme sax[0], ..., sax[7] où x est le numéro de segment. Par exemple si le segment 2 de l'adresse source fait référence au nombre 141, ce dernier est codé comme :
$$sa2[7]sa2'[6]sa2'[5]sa2'[4]sa2[3]sa2'[2]sa2[1]sa2[0].$$
- Pour chaque segment de l'adresse destination, l'auteur introduit 8 variables de la forme dax[0], ..., dax[7]. L'encodage des adresses destination s'effectue de la même manière que celui des adresses source.
- Nous introduisons 16 variables booléennes $p[15]$, ..., $p[0]$ pour encoder les numéros de port.

Prenons par exemple l'adresse source 20.9.17.8. Cette adresse peut être encodée de la manière suivante :

$$\begin{array}{l} sa1'[7]sa1'[6]sa1'[5]sa1[4]sa1'[3]sa1[2]sa1'[1]sa1'[0] \wedge \\ sa2'[7]sa2'[6]sa2'[5]sa2'[4]sa2[3]sa2'[2]sa2'[1]sa2[0] \wedge \\ sa3'[7]sa3'[6]sa3'[5]sa3[4]sa3'[3]sa3'[2]sa3'[1]sa3[0] \wedge \\ sa4'[7]sa4'[6]sa4'[5]sa4'[4]sa4[3]sa4'[2]sa4'[1]sa4'[0] \end{array}$$

L'adresse destination peut être encodée de la même manière. La représentation du port par contre est différente, soit port $= <p[15], ..., p[0]>$. Les conditions peuvent être exprimées à l'aide d'opérations booléennes. De manière similaire aux autres parties de la règle, la condition que le port par exemple soit égale à 25 est la suivante : **port** $=$ *int2bv* 25, où *int2bv* est une fonction qui convertit un nombre en sa représentation vecteur-bit.

Les adresses source et destination, dans une règle, ont deux composantes : une adresse de base et un masque. Le masque procure de la flexibilité pour spécifier un certain nombre de correspondances possibles en une seule règle. Il indique quel bit de l'adresse de base doit être utilisé dans le processus de correspondance et lequel doit être ignoré. Si une adresse de

base donnée dans une règle est $s_1.s_2.s_3.s_4$ et le masque est $m_1.m_2.m_3.m_4$, alors le paquet $a_1.a_2.a_3.a_4$ correspond exactement quand :

$$(s_1 \textbf{ or } m_1 = a_1 \textbf{ or } m_1) \wedge (s_2 \textbf{ or } m_2 = a_2 \textbf{ or } m_2) \wedge$$
$$(s_3 \textbf{ or } m_3 = a_3 \textbf{ or } m_3) \wedge (s_4 \textbf{ or } m_4 = a_4 \textbf{ or } m_4)$$

où **or** est le "ou" logique sur les bits des deux vecteurs.

Les segments du masque sont soit 0 ou 255. Par exemple :

- 146.141.27.66 0.0.0.0 : signifie que le paquet doit correspondre exactement à celui venant d'une machine bien spécifique.
- 146.141.27.66 0.0.255.255 : signifie que le paquet doit venir de quelque part d'un domaine. L'identité de la machine source n'est pas connue.

Un ensemble des règles peut être représenté à l'aide d'une expression booléenne et peut être définit récursivement comme suit :

- Si l'ensemble des règles est vide, alors aucun paquet ne peut être accepté et l'expression booléenne correspondante est **f** (faux).
- Si la première règle est une règle d'acceptation, alors le paquet va être accepté s'il correspond à la règle ou s'il est accepté par le reste des règles de l'ensemble. L'expression booléenne correspondante est la disjonction de l'expression booléenne représentant la première règle et l'expression booléenne représentant le reste des règles.
- Si la première règle est une règle de rejet, alors le paquet va être accepté s'il ne correspond pas à la première règle et s'il est accepté par le reste des règles de l'ensemble. L'expression booléenne correspondante est la conjonction de la négation de l'expression booléenne représentant la première règle et l'expression booléenne représentant le reste des règles.

3.3.4 Résultats

L'algorithme de conversion des règles décrit précédemment a été implémenté par l'auteur et testé sur un ensemble de 430 règles. Le temps mis pour produire le diagramme de décision binaire a été de 20s sur une station Sun Ultra 4, et la taille du diagramme résultant a été de 1.1K (le fichier texte de la liste d'accès était de 32K).

Dans ce qui suit, nous expliquons comment des représentations de diagramme de décision binaire peuvent être utilisées pour analyser les changements effectués dans un ensemble de règles.

L'outil d'analyse posposé dans [8], dispose d'un algorithme qui affiche une expression booléenne sous une forme compréhensible par un utilisateur. L'exemple suivant présente un ensemble de deux règles. Les lignes qui commencent par des «:» sont des entrées données par l'utilisateur à l'outil prototype.

```
: sc [Proto, Port] cond;
:

Proto  Ports  Src1  Src2  Src3  Src4  Dst1  Dst2  Dst3  Dst4

  1   0-65535 0-255 0-255 0-255 0-255 0-255 0-255 0-255 0-255
  3        80 0-255 0-255 0-255 0-255   120    17   112   100
```

L'exemple affiche la condition *cond* montrant toutes les valeurs des adresses source, destination, port et protocole dont les paquets sont autorisés. Le premier argument de *sc* donne, en ordre, les deux premières colonnes qui doivent être choisies. La routine *sc* a un ordre par défaut, mais l'utilisateur peut spécifier n'importe quel ordre en utilisant le premier argument. En changeant l'ordre, l'utilisateur peut voir les règles sous différents angles.

Pour valider un ensemble de règles, il faut répondre à des questions telles que :

- Acceptons-nous des paquets sur le port 25, si oui quel type de paquets ?
- Sur quel port acceptons-nous des paquets TCP ?
- Quels paquets acceptons-nous d'une adresse y ?
- Quel type de paquets allons-nous autoriser pour qu'ils soient envoyé à l'adresse y ?

Nous dressons ici des exemples de réponses à quelques questions de ce genre en terme de combinaison de conditions booléennes :

- Quel type de paquet UDP acceptons-nous ?

  ```
  : sc [Port, Proto] ([Proto <- udp] ::: cond);

  Ports Proto  Src4  Src3  Src2  Src1  Dst4  Dst3  Dst2  Dst1

   53       2  0-255 0-255 0-255 0-255 0-255 0-255 0-255 0-255
  ```

- Quels paquets ne sont pas acceptés ?

  ```
  : sc [Proto, Port, Dst1, Dst2, Dst3, Dst4]

  ([Port range (80, 90), Proto <- gre] ::: not_allowed);
  ```

- Quels paquets, que nous acceptons et qui ont le premier segment de l'adresse de destination à 121 et qui ne sont pas des paquets ICMP ?

```
: sc [Port, Proto] ([Dst1 <- 120, NOT (Proto <- icmp)] ::: cond);

Ports Proto   Src4   Src3  Src2  Src1  Dst4  Dst3  Dst2  Dst1

0-19     1   0-255  0-255 0-255 0-255 0-255 0-255 0-255   120
20-21    1   0-255  0-255 0-255 0-255 0-255 0-255 0-255   120
         3   0-255  0-255 0-255 0-255     3   112    17   120
   22    1   0-255  0-255 0-255 0-255 0-255 0-255 0-255   120
         3       9      0    20   120 0-255 0-255 0-255   120
23-24    1   0-255  0-255 0-255 0-255 0-255 0-255 0-255   120
```

3.3.5 Validation automatique

Un algorithme de validation automatique a été implémenté. Le but de cet algorithme est de valider les changements effectués dans un ensemble de règles. L'algorithme fonctionne comme suit : un passage à travers la liste des règles détecte les règles redondantes, c'est-à-dire l'apparition plus d'une fois de la même règle ou les valeurs d'un masque qui couvrent des règles subséquentes. Si une règle redondante est détectée, elle est présentée à l'utilisateur.

Dans la section suivante, nous résumons les techniques développées par Neilson *et al.* [9]. Dans ce travail de recherche, les auteurs utilisent le calcul ambiant pour spécifier les pare-feux et les agents qui veulent les traverser.

3.4 Analyse des pare-feux à l'aide du calcul ambiant

Le calcul ambiant est un calcul basé sur l'algèbre des processus. Ce calcul possède la faculté de permettre à des processus actifs de se déplacer entre des sites représentant des domaines administratifs différents. Par conséquent, il étend la notion de mobilité traitée par Java [14] où seulement un code passif peut se déplacer entre les sites. Le déplacement des processus ambiants est gouverné par les capacités qu'ils possèdent.

Dans ce qui suit, nous allons donner un bref aperçu du calcul ambiant ainsi que sa syntaxe et sa sémantique. Nous allons ensuite donner un exemple de spécification des pare-feux et des agents qui les traversent à l'aide de ce calcul.

3.4.1 Calcul ambiant

Un ambiant est un processus opérant à l'intérieur d'un domaine nommé. C'est un espace délimité ayant un nom, un intérieur et un extérieur. Le mouvement des processus ambiants est régenté par les capacités dont ils disposent et inclue la possibilité pour un processus ambiant de se déplacer à l'extérieur où à l'intérieur d'un autre processus ambiant. Les entrées et les sorties des ambiants se font de la même manière qu'en π-calcul à l'exception que le canal est définie implicitement et intégré à l'intérieur du processus ambiant. Le calcul ambiant a été proposé par Luca Cardelli et Andrew Gordon [6]. Dans ce calcul, les processus et les sites sont modélisés sous la forme d'ambiants et leurs aptitudes d'aller d'un site vers un autre est gouvernée par les capacités qu'ils possèdent. Pour de plus amples informations sur le calcul ambiant, le lecteur est invité à se référer à [6, 7].

3.4.2 Syntaxe

La présentation du calcul ambiant donnée en [6], définit un ensemble de primitives que nous résumons dans la Table 3.2. La syntaxe du calcul ambiant donnée est basée sur trois catégories : une classe de processus définie par $P \in \mathbf{Proc}$, une classe de capacités définie par $M \in \mathbf{Cap}$, et une classe d'attributions ou de noms définie par $N \in \mathbf{Nam}$. Ci dessous, la signification de chaque élément de la syntaxe du calcul ambiant.

P	::=	$(\nu n)P$	restriction	M	::=	in N	enter N
	|	0	inactivity		|	out N	exit N
	|	$P \mid P'$	composition		|	open N	open N
	|	$!P$	replication		|	x	variable
	|	$N[P]$	ambient		|	ϵ	null
	|	$M.P$	movement		|	$M.M'$	path
	|	$\langle M \rangle$	out of capability				
	|	$\langle\langle N \rangle\rangle$	output of name	N	::=	n	name
	|	$(x).P$	input of capability		|	u	variable
	|	$((u)).P$	input of name				

TAB. 3.2 – *Syntaxe abstraite du calcul ambiant*

Les processus :

- restriction : comme dans le π-calcul, la portée locale est gérée en utilisant l'opérateur de restriction.

- inactivity : indique qu'un processus est inactif.
- composition : deux processus qui s'exécutent en parallèle. $P.Q$ denote le calcul séquentiel de P suivi de Q et $P \mid Q$ denote le calcul parallèle de P et Q.
- replication : permet la génération de plusieurs copies d'un processus.
- ambient : est un processus opérant à l'intérieur d'un domaine nommé. Dans la notation $w[\ldots]$, w denote le nom d'un processus ou d'un domaine, et le calcul qu'effectue le processus et ayant lieu dans un domaine est écrit entre les "[ ]".
- mouvement : le mouvement des processus ambiants dépend des capacités dont ils disposent et inclue la possibilité pour un processus ambiant de se déplacer à l'extérieur d'un ambiant où il est inclu ou à l'intérieur d'un processus ambiant voisin.
- out of capability et out of name : se font de la même manière qu'en π-calcul à l'exception que le canal est définie implicitement et intégré à l'intérieur de l'ambiant.
- input of capability et input of name : se font de la même façon que pour les sorties des capacités et noms.

Les capacités :

- enter N : la capacité *in* permet à un processus ambiant d'entrer dans un processus ambiant voisin nommé N.
- exit N : la capacité *out* permet à un processus ambiant de quitter un processus ambiant parent nomé N.
- open N : la capacité *open* dissout les barrières autour d'un processus ambiant voisin nommé N.
- variable : permet de gérer les capacités.
- null : définit une capacité nulle.
- path : la composition séquentielle des capacités permet de décrire un chemin.

3.4.3 Sémantique

La sémantique du calcul ambiant est définie par une relation de congruence structurelle, notée "$\equiv$", et par une relation de réduction, notée "$\rightarrow$". La Table 3.3 résume cette sémantique.

La relation de congruence est inductivement définie par les axiomes et les règles de la Table 3.3. Les axiomes et les règles dans la partie gauche de la table assurent que la relation est une relation d'équivalence, que la composition parallèle est commutative et associative avec le processus inactif. Ils décrivent aussi le comportement de la replication.

Les axiomes et les règles dans la partie droite de la Table 3.3, permettent d'intervertir des opérateurs de restriction. $P\{n \leftarrow m\}$ est le processus dans lequel n est remplacé par m. Les deux derniers axiomes contrôlent l'expansion des capacités.

$P \equiv P$	$(\nu n)(\nu m)P \equiv (\nu m)(\nu n)P$
$P \equiv Q \wedge Q \equiv R \Rightarrow P \equiv R$	
$P \equiv Q \Rightarrow Q \equiv P$	$(\nu n)(P \mid Q) \equiv P \mid (\nu n)Q$
$P \equiv Q \Rightarrow (\nu n)P \equiv (\nu n)Q$	
$P \equiv Q \Rightarrow P \mid R \equiv Q \mid R$	$(\nu n)(m[P]) \equiv m[(\nu n)P]$
$P \equiv Q \Rightarrow !P \equiv !Q$	
$P \equiv Q \Rightarrow N[P] \equiv N[Q]$	$(\nu n)P \equiv (\nu m)(P\{n \leftarrow m\})$
$P \equiv Q \Rightarrow M.P \equiv M.Q$	
$P \equiv Q \Rightarrow (x).P \equiv (x).Q$	$(x).P \equiv (x').(P\{x \leftarrow x'\})$
$P \equiv Q \Rightarrow ((\mu)).P \equiv ((\mu)).Q$	
$P \mid Q \equiv Q \mid P$	$((u)).P \equiv ((u')).(P\{u \leftarrow u'\})$
$(P \mid Q) \mid R \equiv P \mid (Q \mid R)$	
$P \mid 0 \equiv P$	$(M.M').P \equiv M.M'.P$
$!P \equiv P \mid !P$	
$!0 \equiv 0$	$\epsilon.P \equiv P$
$(\nu n)0 \equiv 0$	

TAB. 3.3 – *Congruence structurelle*

La relation de réduction est inductivement définie par les axiomes et les règles de la Table 3.4. Ces règles stipulent que la réduction peut concerner les restrictions, les processus ambiants et les compositions parallèles, et que la congruence structurelle peut être utilisée pour réarranger les expressions de processus ambiants.

La sous-section suivante présente un exemple de spécification d'un pare-feu à l'aide du calcul ambiant ainsi que sa réduction en utilisant les règles de la Table 3.4.

3.4.4 Exemple de spécification d'un pare-feu

Nous utilisons ici le calcul ambiant pour modéliser un pare-feu et un agent qui a la persmission de passer le pare-feu. L'exemple est tiré de Cardelli et Gordon [17].

$$\begin{aligned} \mathit{Firewall}: &\quad w[k[\mathit{out}\ w.\mathit{in}\ k'.\mathit{in}\ w] \mid \mathit{open}\ k'.\mathit{open}\ k''.P] \\ \mathit{Agent}: &\quad k'[\mathit{open}\ k.k''[Q]] \end{aligned}$$

Les processus modélisant le pare-feu et l'agent sont d'abord mis en parallèle comme suit :

$$n[(\nu w)(\mathit{Firewall} \mid \mathit{Agent})]$$

$P \to Q \Rightarrow (\nu n)P \to (\nu n)Q$	$n[\mathsf{in}\ m.P \mid Q] \mid m[R] \to m[n[P \mid Q] \mid R]$
$P \to Q \Rightarrow n[P] \to n[Q]$	$m[n[\mathsf{out}\ m.P \mid Q] \mid R] \to n[P \mid Q] \mid m[R]$
$P \to Q \Rightarrow P \mid R \to Q \mid R$	$\mathsf{open}\ m.P \mid m[Q] \to P \mid Q$
$P \equiv P' \wedge P' \to Q' \wedge Q \equiv Q' \Rightarrow P \to Q$	

TAB. 3.4 – *Relations de réduction*

Ici, le symbole ν est employé pour marquer la portée locale et w est le nom du pare-feu. n peut être vu comme étant le nom de tout le système. La réduction de cette expression en utilisant la sémantique définie précédemment est comme suit :

$$
\begin{aligned}
& n[(\nu w)(w[\ k[\underline{out\ w}.in\ k'.in\ w] \mid open\ k'.open\ k''.P\ ] \mid k'[open\ k.k''[Q]])] && (3.1)\\
\to\ & n[(\nu w)(k[\underline{in\ k'}.in\ w] \mid w[open\ k'.open\ k''.P] \mid k'[open\ k.k''[Q]])] && (3.2)\\
\to\ & n[(\nu w)(w[open\ k'.open\ k''.P] \mid k'[k[in\ w] \mid \underline{open\ k}.k''[Q]])] && (3.3)\\
\to\ & n[(\nu w)(w[open\ k'.open\ k''.P] \mid k'[\underline{in\ w} \mid k''[Q]])] && (3.4)\\
\to\ & n[(\nu w)(w[\underline{open\ k'}.open\ k''.P \mid k'[k''[Q]]])] && (3.5)\\
\to\ & n[(\nu w)(w[\underline{open\ k''}.P \mid k''[Q]])] && (3.6)\\
\to\ & n[(\nu w)(w[P \mid Q])] && (3.7)
\end{aligned}
$$

Chaque ligne du programme représente une étape d'exécution et la commande qui est exécutée à chaque étape est soulignée. La commande $(out\ w)$ dans la ligne (3.1) envoie un processus "pilote" nommé k à l'extérieur du pare-feu w. Son but est de guider n'importe quel agent qui connait les mots de passe correctes à l'intérieur du pare-feu. À la ligne (3.2), nous avons trois processus qui s'exécutent en parallèle : le processus pilote, le processus contenu dans le pare-feu et finalement l'agent. Notons la commande $(open\ k)$ à l'intérieur de l'agent. L'agent doit connaître le nom du processus pilote afin d'y accéder. À ce stade, le "nom" du processus agent k' correspond à la clé dans la commande $(in\ k')$ du pilote, ce qui veut dire que l'agent a la clé correcte pour laisser le processus pilote devenir une partie du processus agent (ce qui permet au pilote de conduire l'agent à l'intérieur du pare-feu). Le résultat est la ligne (3.3), où la partie restante du processus pilote $k[in\ w]$ s'exécute en parallèle avec le code source de l'agent (commençant avec la commande *open*). Le but de cette commande (*open*) soulignée est d'ouvrir le processus pilote de sorte que ce ne soit plus un processus séparé à l'intérieur de l'agent, mais à la place son code $(in\ w)$ s'exécute en parallèle directetemt avec le reste du code de l'agent $k''[Q]$ comme il est montré à la ligne (3.4). Le code $(in\ w)$ dans cette ligne conduit maintenant le code agent/pilote à l'intérieur

du pare-feu, ce qui mène au résultat de la ligne (3.5). À l'intérieur du pare-feu, le processus agent k' peut maintenant être ouvert par l'intérmédiaire de la construction (*open*) dans la ligne (3.5). Le troisième mot de passe k'' est utilisé pour isoler le processus Q, qui est le code de l'agent, et l'empêcher d'interférer avec le protocole du pare-feu P. La construction (*open*) dans la ligne (3.6) réunit ces deux morceaux de code ensemble de sorte que dans (3.7), ils s'exécutent en parallèle, ce qui signifie que l'agent a été entièrement admis et permis d'exécuter son code à l'intérieur du pare-feu.

Dans la section suivante, nous donnons un résumé du langage des autorisations ASL. Dans un premier temps, nous définissons sa syntaxe. Ensuite, nous présentons les règles que nous pouvons exprimer avec ce langage. Finalement, nous montrons comment ASL gère les conflits entre les règles spécifiées.

3.5 Langage de spécification des autorisations (ASL)

ASL [20] est un langage logique permettant aux utilisateurs de spécifier des autorisations et des politiques d'accès pouvant être appliquées pour contrôler l'exécution d'actions spécifiques sur des objets donnés dans un système unique. Il est basé sur deux éléments : un objet "O", qui peut être un fichier ou un répertoire d'un système d'exploitation ou une table d'une base de données, et une entité autorisée pouvant être un utilisateur "U", un groupe "G" ou un rôle "R". De ce fait, une politique d'autorisation en ASL est une correpondance entre un quadruplet (o, u, R, <sign> a) consistant respectivement en un objet, un utilisateur, un rôle et une action, et un ensemble de signes $\{+, -\}$ où $+a$ définit une autorisation positive et $-a$ une autorisation négative.

3.5.1 Hiérarchie d'utilisateurs

ASL permet d'exprimer des autorisations aussi bien pour des utilisateurs individuels que pour des groupes d'utilisateurs. Un groupe lui-même peut être formé d'utilisateurs individuels et d'autres groupes d'utilisateurs. Ces groupes forment une hiérarchie appelée "hiérarchie d'utilisateurs". Pour formaliser cette structure, nous commençons par définir un système de données, noté DS, comme étant un quadruplet (Obj, T, S, A) où:

- Obj est l'ensemble des objets;
- T est l'ensemble des types;
- S=U $\cup$ G est l'ensemble des sujets où U est l'ensemble des utilsateurs et G est l'ensemble des groupes;
- A est l'ensemble des modes d'autorisation ou d'actions.

Définition 3.5.1 (Hiérarchie d'utilisateurs) Supposons que DS=(Obj, T, S, A) est un système de données. DS est une hiérarchie d'utilisateurs si et seulement s'il existe un ensemble partiellement ordonné fini $(G, \leq)$ tel que :

$$x \text{ est l'élément minimal de } G \Leftrightarrow x \in U.$$

Notons qu'un sujet peut appartenir à plusieurs groupes et par conséquent, les groupes n'ont pas besoin d'être disjoints. Nous disons que l'adhésion d'un sujet s à un groupe G est directe, si le sujet est défini comme étant un membre du groupe. Elle est indirecte, si le sujet s est un membre directe d'un sous groupe G' de G, mais s n'est pas un membre directe de G.

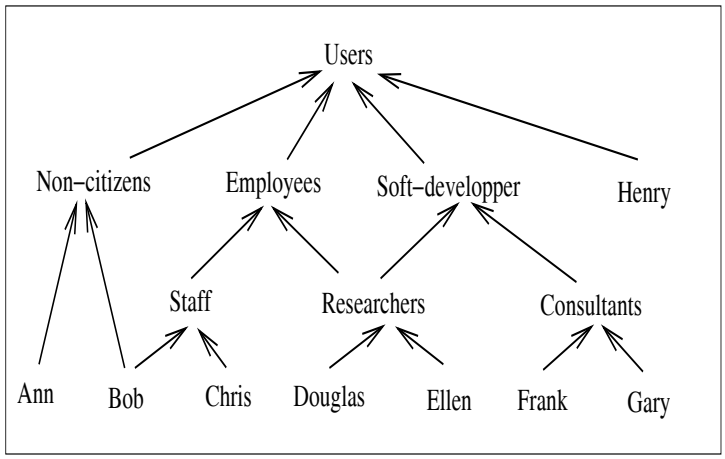

FIG. 3.3 – *Exemple de hiérarchie d'utilisateurs*

Dans la Figure 3.3, les éléments minimums de la hiérachie sont les utilisateurs : Ann, Bob, Chris, Douglas, Ellen, Frank, Gary et Henry.

3.5.2 Syntaxe

Le langage ASL a été conçu pour spécifier des règles qui contrôlent l'accès à des objets par des sujets dans une même et unique machine. Sa syntaxe est définie par un ensemble de règles construites principalement à partir des prédicats suivants :

- **cando**$(o, s, < sign > a)$ **:** est un prédicat ternaire explicitement spécifié par l'administrateur du système. Ce prédicat prend comme arguments : un objet, un sujet et une autorisations signée. La présence d'un prédicat de la forme $cando(o, s, +a)$ ne signifie pas que le sujet s ou ses membres, dans le cas ou s est un groupe, va être autorisé à

exécuter une action a sur un objet o. Il signifie tout simplement que l'administrateur du système veut que s soit capable d'exécuter l'action a sur un objet o.

- **decando**(o, s, $< sign > a$) **:** est un prédicat ternaire representant les autorisations dérivées par le système en utilisant des règles logiques d'inférence. Il a les mêmes arguments que *cando* et aussi le fait d'avoir $decando(o,\ s,\ +a)$ ne signifie pas que le sujet s ou ses membres, dans le cas ou s est un groupe, va être autorisé à exécuter l'action a sur l'objet o.
- **do**(o, s, $< sign > a$) **:** est un prédicat ternaire avec les mêmes arguments que *cando* et *dercando*. Ce pédicat représente les accès à autoriser ou à refuser à chaque sujet sur chaque objet. Si $do(o,\ u,\ +a)$ est vrai, alors l'utilisateur u est autorisé à exécuter l'action a sur l'objet o. Similiarement, si $do(o,\ u, -a)$ est vrai, l'utilisateur u n'est pas autorisé à exécuter l'action a sur l'objet o.
- **done**(o, s, a) **:** est un prédicat ternaire qui admet les mêmes arguments que les précédents. Intuitivement, si $done(o,\ u,\ a)$ est vrai, alors l'utilisateur u a déjà exécuté l'action a sur l'objet o.
- **error**(o, s, a) **:** est un prédicat qui a aussi les mêmes arguments que les autres prédicats déjà cités. Si $error(o,\ u,\ a)$ est vrai, alors une erreur s'est produite lorsque le sujet s a essayé d'exécuter l'action a sur l'objet o.
- **dirin**(s_1, s_2) **:** ce prédicat binaire admet comme arguments deux sujets s_1 et s_2. Ce prédicat capture une relation d'adhésion directe entre des sujets : utilisateurs et groupes. Si **dirin**(s_1, s_2) est vrai, alors le sujet s_1 appartient directement à s_2.
- **in**(s_1, s_2) **:** ce prédicat binaire admet lui aussi comme arguments deux sujets s_1 et s_2 et capture une relation d'adhésion indirecte entre des sujets : utilisateurs et groupes.
- **typeof**(o, t) **:** est un prédicat binaire admettant comme arguments un objet o et un type d'objet t. Intuitivement, ce prédicat capture une relation de regroupement entre des objets.
- **owner**(o, s) **:** ce prédicat binaire admet comme premier argument un objet o et comme second argument un sujet s. Ce prédicat associe un utilisateur unique, pas un groupe, avec chaque objet et est appelé le propriétaire de cet objet.

3.5.3 Règles ASL

Une règles ASL est composée d'une entête représentant le type d'autorisation et d'un corps repésentant les conditions à satisfaire pour que l'autorisation soit accordée ou non. Ces conditions sont données par des littéraux. Un littéral dénote un prédicat ou sa négation. Nous présentons, dans ce qui suit, les règles qu'il est possible d'exprimer avec ce langage.

- **Règle de décision :** une règle de décision *done* est une règle de la forme:

$$done(o,\ s,\ a) \leftarrow .$$

où o, s, a sont respectivement des éléments de Obj, S et A.

Les règles de décison *done* sont des faits, par conséquent leurs corps est toujours vide. Elles représentent les accès exécutés par les utilisateurs et peuvent être spécifiées par le système. Elles sont aussi très utiles pour implémenter des politiques dans lesquelles les accès futurs des utilisateurs sont basés sur les accès qui ont déjà été effectués par ces mêmes utilisateurs.

- **Règle d'autorisation :** une règle d'autorization *cando* est une règle de la forme :

$$cando(o,\ s,\ <sign>\ a) \leftarrow L_1\ \&\ ...\ \&\ L_n$$

où $n \geq 0$, $<sign>$ est soit $+$ ou - et pour chaque $0 < i \leq n$, L_i est un littéral qui peut être un des prédicats suivants : *in, dirin, typeof* ou *owner*.

Les règles d'autorisation sont spécifiées par l'administrateur du système pour permettre ou refuser l'accès à des sujets.

Exemple 3.5.1 : considérons les règles suivantes:

$$\begin{array}{lcl} cando(o,\ Ann,\ -write) & \leftarrow & typeof(o,\ lettres) \\ cando(file1,\ s,\ +read) & \leftarrow & in(s,\ Employees)\ \ \&\ \ \neg in(s,\ Developers) \end{array}$$

La première règle stipule que Ann ne peut pas écrire des objets de type *lettres*. La deuxième règle dit que les employées qui ne sont pas des développeurs sont autorisés à lire $file1$.

- **Règle de dérivation :** l'admnistrateur système peut exprimer des autorisations à l'aide des règles *cando*. À partir de ces autorisations, le système peut dériver d'autres autorisations. Afin de distinguer les autorisations explicitement spécifiées par un administrateur de celles dérivées par le système, nous utilisons le prédicat *dercando* pour représenter les autorisations dérivées. Une règle de dérivation *dercando* est une règle de la forme:

$$dercando(o,\ s,\ <sign>\ a) \leftarrow L_1\ \&\ ...\ \&\ L_n$$

où $<sign>$ est soit $+$ ou - et pour chaque $0 < i \leq n$, L_i est un littéral qui peut être un des prédicats suivants : *cando, dercando, done, in, dirin, typeof* ou *owner*.

Exemple 3.5.2 : considérons les règles suivantes:

$$\begin{array}{lcl} dercando(file1,\ s,\ -read) & \leftarrow & dercando(file2,\ s',\ read)\ \&\ in(s,\ s")\ \&\ in(s',\ s") \\ dercando(o,\ s,\ -write) & \leftarrow & done(o',\ s,\ read)\ \&\ typeof(o,\ Exams)\ \& \\ & & typeof(o',\ Solutions) \end{array}$$

La première règle dérive une autorisation négative pour le sujet s afin de lire $file1$ sous condition qu'ils existent un autre sujet s' et un groupe s" tels que : s, $s' \in s$" et que s' soit autorisé à lire $file1$. La seconde règle dérive une autorization négative pour un utilisateur à écrire un objet de type *Exams* si l'utilisateur a déjà lu un objet de type *Solutions*. Intuitivement un étudiant ne peut pas passer un examen s'il a dèjà lu les solutions des exercices de l'examen.

- **Règle de décision :** une règle de décision *do* est une règle de la forme :

$$do(o,\ s,\ <sign>\ a) \leftarrow L_1\ \&\ ...\ \&\ L_n$$

où les L_i sont les littéraux suivants : *cando, dercando, done, in, dirin, typeof* ou *owner*, et toutes les variables présentes dans chaque L_i doivent être présentes également dans l'entête de la règle.

Exemple 3.5.3 : considérons les règles suivantes:

$$\begin{array}{lcl} do(file2,\ s,\ +a) & \leftarrow & dercando(file2,\ s,\ +a)\ \ \&\ \ \neg dercando(file2,\ s,\ -a) \\ do(o,\ s,\ +read) & \leftarrow & \neg dercando(o,\ s,\ +read)\ \ \&\ \ \neg dercando(o,\ s,\ -read)\ \ \& \\ & & typeof(o,\ Pblc-docs) \end{array}$$

La première règle dit qu'un sujet peut accéder à $file2$ s'il dispose d'une autorisation positive pour le faire et si aucune autorisation négative n'a été spécifiée pour refuser son accès. La seconde règle stipule que si aucune autorisation, qu'elle soit positive ou négative, n'a été spécifiée pour qu'un sujet accède à un objet de type $Pblc-docs$, le sujet peut lire l'objet.

En plus des règles précédentes, ASL propose un autre type de règle appelé "règle d'intégrité" afin de contôler des erreurs de spécification de base. Les règles d'intégrité peuvent être spécifiées à l'aide de tous les littéraux et leurs combinaisons et sont formellement définies comme suit:

- **Régle d'intégrité :** une règle d'intégrité *error* est une règle de la forme:

$$error(o,\ s,\ a) \leftarrow L_1\ \&\ ...\ \&\ L_n$$

où L_1, ..., L_n sont les littéraux suivants : *cando, dercando, done, do, in, dirin, typeof* et *owner*. La règle d'intégrité précédente dérive une erreur à chaque fois que les conditions du corps de la règle sont satisfaites.

Exemple 3.5.4 : considérons les règles suivantes:

$$\begin{array}{lcl} error(o,\ s,\ a) & \leftarrow & cando(o,\ s,\ +a)\ \ \&\ \ cando(o,\ s,\ -a) \\ error(o,\ s,\ a) & \leftarrow & do(o,\ s,\ write)\ \ \&\ \ do(o,\ s,\ evaluate)\ \ \&\ \ typeof(o,\ Tech-reports) \end{array}$$

La première règle dit qu'une autorisation pour qu'un sujet s accède à un objet o ne peut être à la fois positive et négative. La seconde règle retourne une erreur si un sujet est autorisé à rédiger et à évaluer des rapports techniques.

3.6 Politiques par défaut

Du point de vue spécification des autorisations, nous pouvons identifier trois catégories de politiques : politique fermée, ouverte et hybride [20].

- **Politique fermée** : dans ce type de politique, seules des autorisations positives peuvent être spécifiées. Un utilisateur, dans ce cas, est autorisé seulement si une autorisation positive est accordée à lui ou à un des groupes auxquels il appartient.
- **Politique ouverte** : ici, seules des autorisations négatives peuvent être spécifiées. Dans ce cas, un utilisateur est autorisé seulement si aucune autorisation négative n'est accordée ni à lui ni à aucun des groupes auxquels il appartient.
- **Politique hybride** : les autorisations positives et négatives peuvent être spécifiées en même temps.

Généralement, nous avons besoin de spécifier des politiques à la fois positives et négatives. Malheureusement, ce choix peut être générateur d'un certain nombre de problèmes. Par exemple dans le cas d'ASL, si un utilisateur n'est pas autorisé à accéder à un objet et que le groupe à qui il appartient a une autorisation positive pour le faire, alors quoi faire?

Pour gérer ce type de conflits, ASL propose trois solutions décrites comme suit :

- **Aucun conflit n'est alloué :** la présence à la fois d'autrisations positives et négatives est considéré une inconcistence et par conséquent est refusée.
- **Priorité pour les autorisations positives :** les autorisations positives sont prioritaires par rapport à celles négatives et dans ce cas l'accés est autorisé.
- **Priorité pour les autorisations négatives :** les autorisations négatives sont prioritaires et dans ce cas l'accés est refusé.

Nous pensons que la technique de vérification, proposée dans les lignes précédentes, est rudimentaire et ne peut être efficace en particulier pour vérifier des politiques de sécurité plus complexes. Par conséquence, il est très important de proposer des techniques de spécification et d'analyse de politiques de sécurité plus élaborées.

3.7 Critique

Les travaux de recherche que nous avons évoqués dans ce chapitre traitent des problèmes reliés aux domaines de la spécification et de l'analyse des politiques de sécurité et chacuns d'eux proposent des façons de raisonner et des techniques capables de résoudre une partie de ces problèmes. Cependant, les solutions abordées dans chaque travail accusent encore plusieurs lacunes que nous résumons dans les lignes qui suivent :

- Dans [16], Guttman présente une technique de génération des configurations locales des pare-feux à partir d'une politique de sécurité globale. Ce travail, quoi que très ambitieux et propose des solutions pour résoudre le problème de protection des réseaux à l'aide des systèmes à base de pare-feux, souffre malheureusement de plusieurs limitations. Par exemple les configurations locales générées par l'approche présentée sont très complexes et nécessitent une intervention humaine dans le but d'introduire des optimisations locales souhaitées. Cette génération n'est donc pas complètement automatique, ce qui la rend inefficace dans le cas des réseaux de grandes dimensions. Une autre lacune de cette technique réside dans le fait que l'approche adoptée ne permet pas d'assurer la protection du pare-feu lui-même, elle protègent seulement les machines derières lui.
- Dans [17], Hazelhurst considère un codage des règles de pare-feu dans les arbres binaires de décision. Cette représentation en BDD du pare-feu est utilisée alors pour obtenir d'une manière efficace des réponses à certaines questions de type "est-ce que des paquets à destination du port 25 sont acceptés par le pare-feu ou non", etc. Cette technique peut être utile seulement pour aider un administrateur à détecter manuellement des anomalies dans un pare-feu. Ici aussi le côté automatique de l'approche pour la détection automatique des conflits potentiels entre les règles de filtrage laisse à désirer. Une autre limitation de cette technique concerne la taille du BDD servant à coder les règles en expressions booléennes. En effet, le BDD peut être d'une taille démesurée rendant ainsi sa gestion très coûteuse.
- La méthode présentée dans [22] utilise la notion d'ambiants pour représenter les pare-feux. Le cadre formel de cette approche est trop vaste la rendant très complexe à comprendre et à appliquer. D'autant plus que peu d'exemples sont proposés afin d'appuyer les hypothèses énoncées par les auteurs du travail. La technique considérée n'utilise aucun des langages des pare-feux existants mais présente un nouvel formalisme pour en exprimer des comportements opérationnels. La plus importante des limitations de cette apparoche reste l'absence d'une implémentation automatique du modèle, ce qui laisse en suspend la question suivante : l'approche proposée est-elle fonctionnelle pour des configurations réelles de pare-feux, CISCO par exemple ?
- Le langage ASL abordé dans [20] est un langage qui adopte une forme syntaxique simple pour exprimer des règles d'autorisation d'accès à des objets appartenant à une même machine. Cependant, afin de pouvoir utiliser cette syntaxe pour exprimer des

configurations de pare-feu, il est nécessaire d'introduire des modifications majeures. Ces modifications doivent prendre en considération la particularité du problème de spécification des pare-feux dans une architecture distribuée. Cependant, même si l'idée derrière le langage ASL est très intéressante, ASL n'offre malheureusement aucun cadre de travail formel pour la détection d'anomalies entre les règles, ni même de ptototype automatique pour la spécification et l'analyse des règles exprimées.

3.8 Conclusion

Ce chapitre est un bref survol des travaux de recherche récents dans le domaine du contrôl d'accès. Nous avons principalement mis l'emphase sur quatre travaux représentatifs de la littérature. De ce fait, nous avons résumé ces travaux et avons émis des critiques suite à une analyse rigoureuse de ces derniers. Cette analyse critique va nous permettre de proposer des solutions qui tiennent compte des limites évoquées dans ce chapitre.

Chapitre 4

Spécification des pare-feux

Résumé

Dans ce chapitre, nous développons un langage dédié à la spécification des politiques des pare-feux. Le langage en question est doté d'une syntaxe et sémantique formelles et permet de capturer la topologie d'un réseau ainsi que les règles de filtrage des pare-feux qui protègent ce réseau. À l'aide de ce langage, nous pouvons spécifier aussi bien des politiques globales impliquant plusieurs réseaux que des politiques locales.

4.1 Introduction

Plusieurs langages de spécification des politiques de sécurité sont proposés dans la littétature. Des exemples de tels langages sont : RPSL (*Routing Policy Specification Language*) [2], SPSL (*Security Policy Specification Language*) [10], KeyNote [3] et Ponder [11]. Ces langages proposent un ensemble de notations logiques et algébriques permettant d'exprimer des configurations de pare-feux. Cependant, la nature de ces langages fait qu'ils sont parfois difficiles à utiliser pour plusieurs raisons : certains sont de bas niveau ce qui nécessite une connaissance parfaite des couches réseau, d'autres offrent des notations complexes et difficiles à comprendre pour un non expert.

Dans ce chapitre, nous proposons un langage compacte, simple et facile à utiliser, que nous avons baptisé FPSL (*Firewall Policy Specification Language*), pour la spécification des politiques dans les pare-feux. Pour y parvenir, nous nous sommes inspirés du langage ASL, que nous avons décrit dans le chapitre précédent, afin de bâtir un langage dont les règles sont des expressions logiques basées sur des prédicats. Ces derniers sont capables de capturer toutes

les informations que véhicule l'entête d'un paquet : adresse source, adresse destination, ports, protocol, etc. Nous avons, dans un premier temps, introduit une section déclaration permettant de représenter la topologie du réseau. Ensuite, nous avons doté notre langage d'une syntaxe pouvant représenter des règles d'un pare-feu.

Ce chapitre est organisé comme suit. À la section 4.2, nous motivons notre choix pour développer notre langage. À la section 4.3, nous décrivons la syntaxe et la sémantique du langage de spécification des politiques de sécurité des pare-feux FPSL. À la section 4.4, nous prouvons que notre langage est capable de spécifier des politiques de sécurité à travers un exemple tiré de [16]. La section 4.5 conclut ce chapitre.

4.2 Motivation

L'objectif principal de ce chapitre est de développer un langage de spécification qui se veut simple et expressif. Par expressif nous sous-entendons un langage capable d'exprimer aussi bien des politiques locales que globales. Pour ce faire, nous nous sommes intéressés à étudier plusieurs types de langages tels que ceux décrit dans [20, 2, 10, 3, 11]. Un de ces langages, ASL, a sucité notre intérêt en particulier parce qu'il permet d'exprimer des règles d'accès de haut niveau en utilisant une syntaxe simple et compacte. Cependant, ASL est loin d'être un langage idéal pour spécifier des politiques de sécurité dans les pare-feux. Notre motivation pour développer un langage répondant aux besoins spécifiques de sécurité dans les pare-feux trouve sa justification dans les éléments suivants :

- Nous cherchons à développer un langage pour exprimer des politiques qui renforcent l'accès à plusieurs machines réparties entre plusieurs réseaux.
- Nous cherchons à développer un langage dédié à la spécification des pare-feux prenant en compte les particularités du domaine des pare-feux.
- Pouvoir capturer aussi bien les règles de filtrage que la topologie du réseau.
- Faciliter le développement de techniques d'analyse.

Les points précédents donnent une forme préliminaire aux problématiques que notre langage doit adresser afin d'être capable de spécifier les politiques de sécurité désirées. Dans la section suivante, nous donnons une description complète du langage FPSL.

4.3 Langage de spécification des politiques des pare-feux

FPSL (*Firewall Policy Specification Language*) est un langage dédié à la spécification des politiques de sécurité qui doivent être renforcées dans les pare-feux. Afin de définir les com-

posantes de ce langage, nous avons, dans un premier temps, développé une section de déclaration dont la fonction et de capturer la topolgie du réseau. Ensuite, pour pouvoir exprimer les règles des pare-feux, nous avons introduit des prédicats capables de représenter toutes les informations contenues dans l'entête d'un paquet : adresses de source et de destination, ports, protocol, etc. et donné la syntaxe d'une règle de filtrage.

4.3.1 Topologie de réseau

Une relation forte lie une politique de sécurité à la topologie du réseau dans lequel elle doit être renforcée. En effet, un réseau cohabite souvent avec plusieurs autres réseaux et plusieurs pare-feux qui connectent ces réseaux. Chaque pare-feu définit une politique d'accès locale entre deux réseaux. La section de déclaration de notre langage permet de dresser le schéma du réseau en terme de connexions entre les sous-réseaux et les pare-feux. Nous avons aussi ajouté, à cette section, la possibilité de déclarer des noms de domaines et des services. Nous proposons dans la Table 4.1 une description détaillée d'une telle structure.

La section de déclaration de la topologie du réseau proposée est bâtie autours de trois blocs respectivement pour exprimer les domaines du réseau, les routeurs qui connectent les domaines et les services offerts :

- Domains : dans ce bloc, nous donnons la définition de tous les domaines (zones) qui composent le réseau. Un domaine est défini par un nom et peut être constitué d'un ou plusieurs sous-domaines dans lesquels nous distinguons les machines hôtes par leurs noms ou adresses ip respectives. Le but de différencier les machines hôtes des autres trouve son inspiration du fait que ces machines ont un statut différent des autres, et par conséquent, un traitement spécial leurs est réservé de la part des pare-feux qui les contrôlent.
- Routers : sert à lister tous les routeurs ainsi que les zones qu'ils connectent. Par exemple le routeur "dept_info-dept_eco" connecte le réseau du département d'informatique au réseau du département d'économie. Ce bloc est très important du fait qu'il nous permet de déduire aussi bien les zones adjacentes que celles qui ne le sont pas et sera exploité plus tard pour déduire des politiques locales.
- Services : ce bloc permet de définir les services autorisés par les pare-feux. Par exemple le pare-feu autorise le service *telnet*.

Une fois que l'architecture du réseau est capturée par la section de déclaration, nous sommes en mesure d'entamer la spécification des comportements de filtrage que nous voulons que chaque pare-feu adopte vis à vis des paquets qui le travesent. Cette spécification est réalisée à l'aide du langage FPSL dont la syntaxe et la sémantique sont décrites dans ce qui suit.

Declaration	::=	**Domains**{Domain_value} **Routers** {Router_value} **Services** {Service_value}
Domain_value	::=	Name {Subdomain} Domain_value | Name {Subdomain}
Subdomain	::=	Name {Hosts} Subdomain | Name {Hosts}
Hosts	::=	Host_name = ip_address , Hosts | Host_name , Hosts | ip_address , Hosts | ip_address_range , Hosts | ϵ
Router_value	::=	Router_name Name – Name
Service_value	::=	Service_name Service_value | Service_name
ip_address	::=	Val.Val.Val.Val
ip_address_range	::=	ip_address ... ip_address
Name	::=	string
Host_name	::=	string
Router_name	::=	string
Service_name	::=	string
Val	::=	Num | $*$

TAB. 4.1 – *Section de déclaration*

4.3.2 Syntaxe

Un pare-feu est spécifié par une séquence de règles de filtrage. Une règle est de la forme : *Si* (*condition*) *alors* (*action*). Nous décrivons formellement une règle de filtrage par la grammaire qui suit :

Rule	::=	Action ← Conditions
Action	::=	$a \mid d$
Conditions	::=	Predicate[∧ Conditions]

Condition d'une règle

Chaque règle de filtrage est composée d'une condition représentée par un ou la conjonction de plusieurs prédicats et d'une action. Un prédicat permet de spécifier des conditions sur les

attributs des paquets réseau comme l'adresse IP, le port, le service, etc. Plus formellement, la grammaire qui suit décrit les prédicats qui composent la condition d'une règle :

Predicate	::=	from(Address)
		| to(Address)
		| using(ValPort, ValPort)
		| typeofprot(NameProt)
Address	::=	AdrVal [... AdrVal]
ValPort	::=	DecVal [... DecVal] | any
NameProt	::=	tcp | udp | icmp | http | ftp | smtp | any | ...
AdrVal	::=	Val.Val.Val.Val
Val	::=	Num | *
DecVal	::=	Num

où Address est utilisé pour spécifier une adresse ou une plage d'adresses IP source ou destination. ValPort est utilisé pour spécifier un port ou une plage de ports source et destination. NameProt sert à identifier le type du service par le biais du protocole utilisé. Num est une valeur numérique (sur 8 bits dans le cas de IP version 4) et DecVal prend une valeur dans la plage des valeurs de ports permises. Le symbole "*" est utilisé pour spécifier l'ensemble des valeurs permises entre 0 et 255 . Des exemples de tels arguments peuvent être : from(132.213.10.12), to(132.210.1.12...132.210.1.254) et typeofprot(tcp).

Nous attribuons aux précédents prédicats, les significations suivantes :

- from(Address) : est un prédicat unaire dont l'argument est une adresse IP. Ce prédicat définit l'origine d'un paquet.
- to(Address) : est un prédicat unaire dont l'argument est aussi une adresse IP. Ce prédicat capture la destination que le paquet désire atteindre.
- using(ValPort, ValPort) : est un prédicat binaire dont les deux arguments sont des ports. Ce prédicat représente les ports de source et de destination sur lesquels, respectivement, un paquet est envoyé et reçu.
- typeofprot(NameProt) : est un prédicat unaire dont l'argument est le nom du protocole. Ce prédicat définit le type de service qu'un paquet désire effectué dans la zone de destination spécifiée dans le prédicat to.

Cette grammaire n'est pas limitative et peut être enrichie au besoin avec d'autres types de prédicats, d'autres protocoles, etc. suivant que de nouveaux besoins de spécification émergent.

Action d'une règle

L'autre composante d'une règle est l'action. L'action d'une règle peut être soit l'acceptation a ou le refus d du paquet. Nous proposons l'exemple suivant pour montrer comment nous

exprimons une règle de filtrage à l'aide du langage FPSL.

$$\mathsf{d} \leftarrow \mathsf{from}(132.213.10.12) \wedge \mathsf{to}(211.121.10.3) \wedge \mathsf{typeofprot}(\mathsf{tcp})$$

Cette règle n'autorise pas le passage de tout paquet à destination de 211.121.10.3 si l'adresse IP source est 132.213.10.12 et si son protocole est tcp.

Forme normale d'une règle

Nous avons besoin de définir une manière de disposer les prédicats d'une règle. Cette forme est très importante pour deux raisons : pour standardiser la représentation des règles des pare-feux et pour faciliter la comparaison entre les prédicats du même type de plusieurs règles de filtrage. Nous appelons cette façon d'organiser les prédicats "la forme normale d'une règle".

Une règle est dite sous forme normale, si la disposition des prédicats dans la règle suit par exemple la forme unique : $\mathsf{from}(...) \wedge \mathsf{to}(...) \wedge \mathsf{using}(...) \wedge \mathsf{typeofprot}(...)$. Si R et R' sont deux règles tel que : $R : \varphi \leftarrow \wedge_{i=1}^{n} p_i$ et $R' : \varphi' \leftarrow \wedge_{i=1}^{n} p'_i$, alors $\forall\, i$, $type(p_i) = type(p'_i)$. Avec $type$ est une fonction qui retourne le type d'un prédicat. Pour simplifier les notations, dans la suite de ce chapitre, nous supposons que toutes les règles de filtrage sont sous forme normale.

Pare-feu

Un pare-feu est une suite de règles ordonnées, séparées par un opérateur de séquencement noté ";". Nous décrivons formellement un pare-feu par la grammaire qui suit :

$$\mathsf{Firewall} ::= \epsilon \mid \mathsf{Rule} \mid \mathsf{Firewall}\ ;\ \mathsf{Firewall}$$

où Rule est une règle comme définie précédemment, ϵ dénote une séquence vide et " ; " est l'opérateur de séquencement. Nous supposons que $R; \epsilon = \epsilon; R = R$. Pour le reste de ce chapitre, nous considérerons que : $\mathcal{P}$ dénote l'ensemble des prédicats, $\mathcal{R}$ dénote l'ensemble des règles, $\mathcal{C}$ dénote l'ensemble $\{a, d\}$ des actions des règles, $\mathcal{F}$ dénote l'ensemble des pare-feux, $\mathcal{T}$ dénote l'ensemble des paquets qui peuvent circuler sur le réseau et finalement $\mathcal{S}$ dénote l'ensemble des séquences de règles.

4.3.3 Sémantique

Dans cette section, nous définissons la sémantique d'un prédicat, d'une règle, d'une séquence de règles et d'un pare-feu. Il faut noter que nous faisons une distinction au niveau sémantique entre une séquence de règles et un pare-feu, car un pare-feu, en plus de la séquence de règles qui le compose, contient une règle cachée qui est l'action par défaut de ce dernier.

Prédicat

La sémantique d'un prédicat p, notée $[\![\_]\!]^P$, est l'ensemble de tous les paquets qui satisfont le prédicat p. Plus formellement, cette sémantique est définie par ce qui suit :

$$\begin{aligned}[\![\_]\!]^P : \quad & \mathcal{P} \to 2^{\mathcal{T}} \\ & p \mapsto \{t \in \mathcal{T} \ : \ p(t) = true\}\end{aligned}$$

Si p est un prédicat et t un paquet, alors $p(t) = true$ si les attributs du paquet (adresse IP, port, etc.) correspondent à ceux spécifiés par le prédicat p. Si p et p' sont deux prédicats, alors nous définissons les relations $\sqsubset$, $\sqsubseteq$ et $\sim$ sur les prédicats de la manière suivante :

$$\begin{array}{lcl} p \sqsubset p' & \Leftrightarrow & [\![p]\!]^P \subset [\![p']\!]^P \\ p \sqsubseteq p' & \Leftrightarrow & [\![p]\!]^P \subseteq [\![p']\!]^P \\ p \sim p' & \Leftrightarrow & [\![p]\!]^P = [\![p']\!]^P \\ p \sqsubseteq^{-1} p' & \Leftrightarrow & [\![p']\!]^P \subseteq [\![p]\!]^P \end{array}$$

Condition d'une règle

La sémantique d'une condition c, notée $[\![c]\!]^C$, est l'ensemble de tous les paquets qui satisfont la condition c. Plus formellement :

$$\begin{array}{rcl} [\![p]\!]^C & = & [\![p]\!]^P \\ [\![c_1 \wedge c_2]\!]^C & = & [\![c_1]\!]^C \cap [\![c_2]\!]^C \end{array}$$

Aussi, nous prolongeons les définitions de $\sqsubset$, $\sqsubseteq$ et $\sim$ sur les conditions de la manière suivante :

$$\begin{array}{lcl} c \sqsubset c' & \Leftrightarrow & [\![c]\!]^C \subset [\![c']\!]^C \\ c \sqsubseteq c' & \Leftrightarrow & [\![c]\!]^C \subseteq [\![c']\!]^C \\ c \sim c' & \Leftrightarrow & [\![c]\!]^C = [\![c']\!]^C \end{array}$$

Séquence de règles

Soit S une séquence de règles. La sémantique de S, notée $[\![\_]\!]^S$, est un couple d'ensembles (α, β) tel que α représente l'ensemble des paquets acceptés et β représente l'ensemble des paquets rejetés par la séquence s. Nous définisson formellement la sémantique d'une séquence de règles par :

$$\begin{aligned}[\![\_]\!]^S : \quad & \mathcal{S} \to 2^{\mathcal{T}} \times 2^{\mathcal{T}} \\ & s \mapsto (accept(s), deny(s))\end{aligned}$$

où les fonctions *accept* et *deny* sont définies à la Table 4.2.

$$
\begin{array}{lcl}
accept(\epsilon) & = & \emptyset \\
accept((a \leftarrow \wedge_{i=1}^{n} p_i).s) & = & (\llbracket\ \wedge_{i=1}^{n} p_i\ \rrbracket^C) \cup accept(s) \\
accept((d \leftarrow \wedge_{i=1}^{n} p_i).s) & = & accept(s) \\
 & & \\
deny(\epsilon) & = & \emptyset \\
deny((a \leftarrow \wedge_{i=1}^{n} p_i).s) & = & deny(s) \\
deny((d \leftarrow \wedge_{i=1}^{n} p_i).s) & = & (\wedge_{i=1}^{n} \llbracket\ p_i\ \rrbracket^C) \cup deny(s)
\end{array}
$$

TAB. 4.2 – *Sémantique d'une séquence de règles*

$$\llbracket\ \_\ \rrbracket^F_{\_} : \mathcal{F} \times \mathcal{C} \rightarrow 2^{\mathcal{T}}$$

$$
\begin{array}{rcl@{\qquad}rcl}
\llbracket\ \epsilon\ \rrbracket^F_a & = & \mathcal{T} & \llbracket\ \epsilon\ \rrbracket^F_d & = & \emptyset \\
\llbracket\ a \leftarrow \wedge_{i=1}^{n} p_i\ \rrbracket^F_a & = & \mathcal{T} & \llbracket\ a \leftarrow \wedge_{i=1}^{n} p_i\ \rrbracket^F_d & = & \llbracket\ \wedge_{i=1}^{n} p_i\ \rrbracket^C \\
\llbracket\ d \leftarrow \wedge_{i=1}^{n} p_i\ \rrbracket^F_a & = & \mathcal{T} \setminus \llbracket\ \wedge_{i=1}^{n} p_i\ \rrbracket^C & \llbracket\ d \leftarrow \wedge_{i=1}^{n} p_i\ \rrbracket^F_d & = & \emptyset \\
\llbracket\ F; F'\ \rrbracket^F_a & = & \alpha_1 \cup (\alpha_2 \setminus \beta_1) \cup \mathcal{T} \setminus (\beta_1 \cup \beta_2) & \llbracket\ F; F'\ \rrbracket^F_d & = & \alpha_1 \cup (\alpha_2 \setminus \beta_1) \\
 & & & & & \\
(\alpha_1,\ \beta_1) & = & \llbracket\ F\ \rrbracket^S & (\alpha_2,\ \beta_2) & = & \llbracket\ F'\ \rrbracket^S
\end{array}
$$

TAB. 4.3 – *Sémantique d'un pare-feu*

Pare-feu

La sémantique d'un pare-feu, notée $\llbracket\ \_\ \rrbracket^F_{\_}$, est l'ensemble des paquets qui traversent le pare-feu. Cette sémantique est évidemment paramétrée par la politique par défaut du pare-feu (acceptation ou refus). Formellement, cette sémantique est définie à la Table 4.3.

Dans ce qui suit, nous décrivons un exemple de spécification avec notre langage.

4.4 Exemple de spécification

Dans [16], Guttman présente l'exemple d'une politique de sécurité qu'il faudra renforcer dans le réseau d'une organisation. Le réseau en question, comme l'indique la Figure 4.1, est composé d'un certain nombre de zones reliées par des pare-feux. Les zones du dit réseau

sont définies comme suit :

- une zone Internal composée de deux sous-zones : Finance et Engineering. La sous-zone Finance dispose d'un serveur de messages, quant à Engineering, elle est dotée de deux serveurs : un pour gérer les messages et un autre pour gérer les bases de données.
- une zone Periphery disposant d'un serveur mandataire (proxy).
- une zone Allied.
- une zone External.

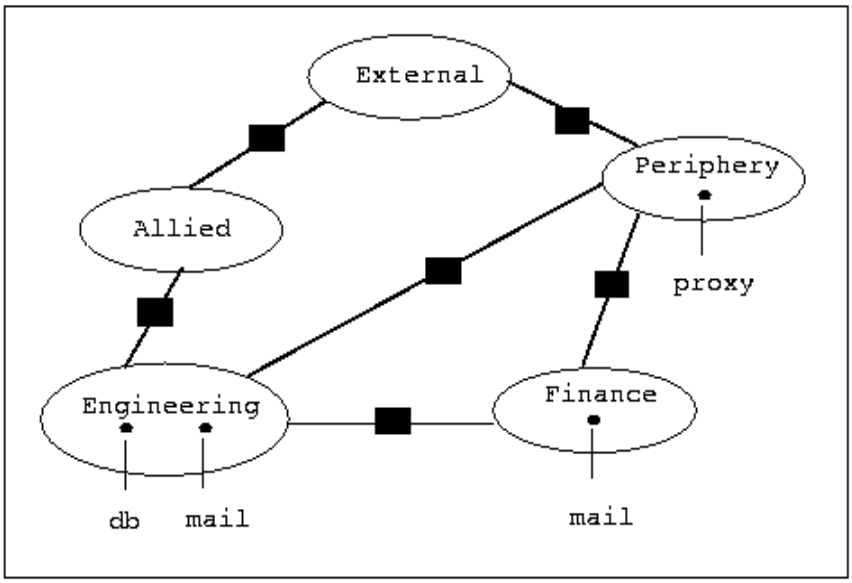

FIG. 4.1 – *Exemple de protection d'une organisation*

Les accès à autoriser par la politique de l'exemple en question sont définis dans les règles qui suivent :

- **Règle 1** : les services *ftp*, *http* et *telnet* de la zone Internal vers la zone External sont acheminés via un serveur mandataire situé dans la zone Periphery.
- **Règle 2** : les connexions *smtp* ne sont pas acheminées via un serveur mandataire, mais sont autorisées seulement si la destination est un serveur de messages.
- **Règle 3** : les requêtes de bases de données *udp* sont autorisées seulement à partir des zones Internal et Allied.
- **Règle 4** : la zone Allied peut acheminer des services *ftp*, *http*, *telnet* à la sous-zone Engineering sans passer pas un serveur mandataire, mais doit acheminer ces services via un serveur mandataire vers la sous-zone Financial.

4.4.1 Spécification de la politique globale

Les règles précédentes définissent d'une manière générale la politique de sécurité de l'exemple de Guttman. Cette politique constitue ce que nous appelons la politique globale du réseau. Malgré que, les adresses de source, de destination et ports ne sont pas définies d'une manière spécifique, nous sommes quand même capables de spécifier cette politique à l'aide de notre langage, en considérant que les adresses de source et de destination sont des utilisateurs appartenant à des groupes (domaines). En ce qui concerne les ports, puisqu'aucune information n'est donnée à leur sujet, nous considérons que tous les ports sont autorisés, et par conséquent, nous les exprimons à l'aide du prédicat using(any, any). Le programme qui capture cette spécification est donné dans la Table 4.4.

4.4.2 Spécificaion des politiques locales

La politique globale que nous avons spécifiée précédemment nécessite d'être implantée localement au niveau de chaque pare-feu. La génération automatique de politiques locales à partir d'une politique globale ne fait pas l'objet de cette recherche. En conséquece, nous allons nous contenter d'une spécification manuelle de la politique de chaque pare-feu de l'exemple précédent. Pour s'y faire, nous procédons comme suit : pour chaque règle R de la politique globale, nous cherchons tous les chemins allant vers la zone de destination D spécifiée dans le prédicat to de la règle R. Ensuite, nous ajoutons la règle R à chaque pare-feu F disposant d'une interface avec D. Par simplification, dans nos exemples, nous ne représentons que les règles de filtrage qui implémentent les pare-feux du réseau. La politique de filtrage gouvernant la communication entre les zones Engineering et Finance est représentée dans la Table 4.5. Les politiques locales des autres pare-feux sont données en annexe.

4.5 Conclusion

Dans ce chapitre, nous avons décrit le noyau d'un langage, que nous avons baptisé FPSL, pour la spécification des politiques de sécurité dans les pare-feux. À l'aide de notre langage nous sommes capables de capturer la topologie du réseau et d'exprimer les composantes d'une règle de filtrage, d'une séquence de règles et finalement d'un pare-feu. La syntaxe que nous avons mis au point, nous a permis de spécifier d'une façon claire et simple l'exemple d'une politique globale ainsi que les politiques locales dérivées qui doivent être renforcées dans le réseau proposé dans [16]. Cette spécification prend la forme d'un programme écrit en FPSL dont l'entête capture la topologie du réseau et dont le coprs contient toutes les règles de filtrage qu'un pare-feu utilisera pour garantir la sécurité des parties du réseau qu'il protège.

Les exemples de spécification que nous avons proposés dans ce chapitre nécessitent d'être vérifiés. Les techniques actuelles d'analyse et de vérification des pare-feux procèdent en deux phases : la première phase permet de convertir les règles de filtrage en expressions booléennes afin de faciliter leur manipulation à l'aide d'un module de vérification. La deuxième phase, consiste à comparer ces expressions par une technique dite "règle à règle". Cette technique vérifie, à chaque fois, la présence ou l'absence d'anomalies entre deux règles.

Dans le prochain chapitre, nous présentons une nouvelle technique de vérification des pare-feux. Cette technique en plus d'être capable de détecter les anomalies entre des règles de filtrage individuelles, permet de détecter aussi les anomalies entre des ensembles de règles. Un des avantages escomptés de cette technique est sa capacité à exhiber les anomalies dans un pare-feu aussi bien que celles provoquées par plusieurs pare-feux.

Domains

{Internal{*Engineering {Eng_db, Eng_mail}*
Finance {Fin_mail}}
Periphery {Per_proxy}
External
Allied}

Routers

{*All_Ext {Allied-External}*
Eng_All {Engineering-Allied}
Eng_Per {Engineering-Periphery}
Eng_Fin {Engineering-Finance}
Fin_Per {Finance-Periphery}
Per_Ext {Periphery-External}}

Services

{*udp, smtp, http, ftp, telnet, proxy*}

d	⟵	from(Internal)	∧	to(External)	∧	using(any, any)	∧	typeofprot(http)
d	⟵	from(Internal)	∧	to(External)	∧	using(any, any)	∧	typeofprot(ftp)
d	⟵	from(Internal)	∧	to(External)	∧	using(any, any)	∧	typeofprot(telnet)
a	⟵	from(Internal)	∧	to(Periphery)	∧	using(any, any)	∧	typeofprot(proxy)
a	⟵	from(Periphery)	∧	to(External)	∧	using(any, any)	∧	typeofprot(http)
a	⟵	from(Periphery)	∧	to(External)	∧	using(any, any)	∧	typeofprot(ftp)
a	⟵	from(Periphery)	∧	to(External)	∧	using(any, any)	∧	typeofprot(telnet)
a	⟵	from(External)	∧	to(Internal)	∧	using(any, any)	∧	typeofprot(smtp)
a	⟵	from(Allied)	∧	to(Internal)	∧	using(any, any)	∧	typeofprot(smtp)
a	⟵	from(Periphery)	∧	to(Internal)	∧	using(any, any)	∧	typeofprot(smtp)
a	⟵	from(Internal)	∧	to(Engineering)	∧	using(any, any)	∧	typeofprot(udp)
a	⟵	from(Allied)	∧	to(Engineering)	∧	using(any, any)	∧	typeofprot(udp)
d	⟵	from(Allied)	∧	to(Finance)	∧	using(any, any)	∧	typeofprot(http)
d	⟵	from(Allied)	∧	to(Finance)	∧	using(any, any)	∧	typeofprot(ftp)
d	⟵	from(Allied)	∧	to(Finance)	∧	using(any, any)	∧	typeofprot(telnet)
a	⟵	from(Allied)	∧	to(Engineering	∧	using(any, any)	∧	typeofprot(http)
a	⟵	from(Allied)	∧	to(Engineering)	∧	using(any, any)	∧	typeofprot(ftp)
a	⟵	from(Allied)	∧	to(Engineering)	∧	using(any, any)	∧	typeofprot(telnet)
a	⟵	from(Allied)	∧	to(Periphery)	∧	using(any, any)	∧	typeofprot(proxy)
a	⟵	from(Periphery)	∧	to(Finance)	∧	using(any, any)	∧	typeofprot(http)
a	⟵	from(Periphery)	∧	to(Finance)	∧	using(any, any)	∧	typeofprot(ftp)
a	⟵	from(Periphery)	∧	to(Finance)	∧	using(any, any)	∧	typeofprot(telnet)

TAB. 4.4 – *Spécification de la politique globale de l'exemple*

a	⟵	from(Finance)	∧	to(Periphery)	∧	using(any, any)	∧	typeofprot(proxy)
a	⟵	from(Engineering)	∧	to(Periphery)	∧	using(any, any)	∧	typeofprot(proxy)
a	⟵	from(External)	∧	to(Engineering)	∧	using(any, any)	∧	typeofprot(smtp)
a	⟵	from(External)	∧	to(Finance)	∧	using(any, any)	∧	typeofprot(smtp)
a	⟵	from(Allied)	∧	to(Engineering)	∧	using(any, any)	∧	typeofprot(smtp)
a	⟵	from(Allied)	∧	to(Finance)	∧	using(any, any)	∧	typeofprot(smtp)
a	⟵	from(Periphery)	∧	to(Engineering)	∧	using(any, any)	∧	typeofprot(smtp)
a	⟵	from(Periphery)	∧	to(Finance)	∧	using(any, any)	∧	typeofprot(smtp)
a	⟵	from(Finance)	∧	to(Engineering)	∧	using(any, any)	∧	typeofprot(udp)
a	⟵	from(Engineering)	∧	to(Engineering)	∧	using(any, any)	∧	typeofprot(udp)
a	⟵	from(Allied)	∧	to(Engineering)	∧	using(any, any)	∧	typeofprot(udp)
a	⟵	from(Allied)	∧	to(Engineering)	∧	using(any, any)	∧	typeofprot(http)
a	⟵	from(Allied)	∧	to(Engineering)	∧	using(any, any)	∧	typeofprot(ftp)
a	⟵	from(Allied)	∧	to(Engineering)	∧	using(any, any)	∧	typeofprot(telnet)
d	⟵	from(Allied)	∧	to(Finance)	∧	using(any, any)	∧	typeofprot(http)
d	⟵	from(Allied)	∧	to(Finance)	∧	using(any, any)	∧	typeofprot(ftp)
d	⟵	from(Allied)	∧	to(Finance)	∧	using(any, any)	∧	typeofprot(telnet)
d	⟵	from(External)	∧	to(Engineering)	∧	using(any, any)	∧	typeofprot(udp)
a	⟵	from(Periphery)	∧	to(Finance)	∧	using(any, any)	∧	typeofprot(http)
a	⟵	from(Periphery)	∧	to(Finance)	∧	using(any, any)	∧	typeofprot(ftp)
a	⟵	from(Periphery)	∧	to(Finance)	∧	using(any, any)	∧	typeofprot(telnet)

TAB. 4.5 – *Politique locale du pare-feu Engineering-Finance*

Chapitre 5

Détection d'anomalies dans les pare-feux

Résumé

Dans ce chapitre, nous présentons un système de types pour la détection d'anomalies dans les pare-feux. Pour ce faire, nous définissons et caractérisons formellement plusieurs anomalies pouvant affecter un pare-feu. Par la suite, nous proposons un système de types qui capture ces anomalies. Ceci nous a permit de réduire la vérification des pare-feux à un problème de typage où les types représentent les conflits potentiels entre les règles du pare-feu. Finalement, nous construisons un algorithme mécanisant le système de types et prouvons sa correction.

5.1 Introduction

Il est clair de nos jours que les pare-feux contribuent énormément à l'amélioration de la sécurité des réseaux informatiques. Cependant, ces dispositifs éprouvent encore de la difficulté à mener à bien leur tâche. Les pare-feux souffrent de plusieurs lacunes reliées principalement à la manière dont ils sont configurés. En effet, la spécification des règles de filtrage qui composent un pare-feu est une tâche complexe et la sécurité assurée par le pare-feu dépend en grande partie de la façon dont ces règles sont spécifiées. Un pare-feu peut contenir des anomalies résultantes de situations conflictuelles entre les différentes règles de filtrage qui le composent. Ces anomalies constituent des failles de sécurité qui peuvent être exploitées par une personne malveillante pour attaquer le système qui est censé être protéger par le pare-feu. Par exemple deux règles peuvent se contredire et dépendamment de l'ordre relatif

de ces règles dans le pare-feu, un paquet peut être accepté dans un cas et être refusé dans l'autre.

Généralement, les règles d'un pare-feu se développent continuellement et évoluent suivant les changements des besoins de sécurité. De plus, le pare-feu peut contenir plusieurs milliers de règles, souvent écrites par différents administrateurs à différents moments. Ces constatations nous amènent donc à penser qu'il est très difficile d'éviter d'introduire des anomalies en spécifiant les règles du pare-feu. Par conséquent, il est primordial de se doter d'outils automatiques permettant d'analyser et de détecter les anomalies d'un pare-feu afin de pouvoir les corriger.

Le présent chapitre est structuré comme suit. La section 5.2 motive notre choix de développer un système de types pour la détection d'anomalies dans les pare-feux. La section 5.3 donne une définition aux anomalies que peut exhiber une liste de règles. La section 5.4 présente un système de types capables de détecter les anomalies dans des pare-feux ainsi que l'algorithme d'inférence de types qui l'implémente. La section 5.5 illustre le fonctionnement de cet algorithme sur un exemple concret de pare-feu. La section **??** fait la preuve de la correction et de la complétude de l'algorithme d'inférence de types. La section 5.6 conclut ce chapitre.

5.2 Motivation

La problématique d'analyser des listes d'accès a été abordée dans différents travaux de recherche (voir le chapitre concernant l'état de l'art pour certains de ces travaux). Une excellente description des anomalies dans les pare-feux est présentée dans [1]. Dans ce travail, Al-Shaer élabore un algorithme permettant de détecter les anomalies dans les pare-feux. L'approche proposée par l'auteur est très interéssante. Malheureusement, ce travail n'est pas supporté par un formalisme théorique rigoureux et nous croyons que l'algorithme de détection sera inefficace si le nombre de règles du pare-feu devient important. Dans [12], l'auteur utilise CLP (un langage logique avec contraintes) pour implémenter un pare-feu en un système expert dans le but d'obtenir des réponses à certaines interrogations relatives à des domaines qui généralement necéssitent une expertise humaine.

La technique que nous proposons dans ce chapitre nous permet de remédier aux lacunes que nous avons pu exhiber à partir de notre critique des travaux citées auparavant. Plus encore, elle propose une nouvelle approche et altérnative au problématique de comparaison des règles de filtrage. En effet, les techniques existantes adoptent une technique séquentielle pour comparer les règles. Chaque règle du pare-feu est comparée avec toutes les autres. De ce fait, le nombre de comparaisons augmente aussi bien que le temps de vérification du moment où la taille de la liste d'accès augmente. Par conséquent, notre technique apporte, comme

nous le présentons ici, une contribution novatrice par rapport aux techniques existantes au moins sur les éléments suivants :

- une procédure de vérification efficace en optimisant le nombre de comparaisons des règles. En effet, l'efficacité dans la vérification est cruciale pour pouvoir analyser en pratique des configurations de pare-feux qui peuvent contenir jusqu'à plusieurs milliers de règles.
- une plus grande flexibilité dans le processus de vérification. En effet, si une règle est ajoutée à un pare-feu déjà vérifié, seules deux comparaisons sont nécessaires pour vérifier la nouvelle configuration : projeter l'ancienne configuration en deux règles (acceptation et refus) et les comparer avec la nouvelle règle.
- une meilleure précision de l'analyse. En effet notre système permet de détecter des anomalies entre des séquences de règles plutôt qu'uniquement entre règles individuelles comme c'est le cas des techniques actuelles. Ainsi, notre technique permet des réponses de type : les règles R_1 et R_2 ensemble provoquent-elles une anomalie avec la règle R_3?

Pour des raisons de clareté de la présentation, dans le reste de ce chapitre, nous modifions légèrement la syntaxe de notre langage en ommettant le nom des prédicats dans la forme normale des règles. Ainsi par exemple la règle :

$$\mathsf{a} \longleftarrow \mathsf{from}(140.192.37.*, \mathsf{Net1}) \wedge \mathsf{to}(161.120.33.40, \mathsf{Net2}) \wedge \mathsf{typeofprot}(\mathsf{tcp}) \wedge \mathsf{using}(40, 80)$$

peut être écrite comme suit :

$$\mathsf{a} \leftarrow 140.192.37.* \wedge 161.120.33.40 \wedge \mathsf{tcp} \wedge 40 \wedge 80$$

D'une maniège générale, nous utiliserons la notation $\varphi \leftarrow \wedge_{i=1}^{n} p_i$ pour décrire une règle avec p_i, $i = 1...n$, conditions et une action φ.

5.3 Modélisation des anomalies des pare-feux

Dans cette section, nous définissons et formalisons différents types d'anomalies que peut contenir un pare-feu. Nous considérons comme anomalie de pare-feu, toute situation conflictuelle entre ses règles ayant pour conséquence l'altération de la sémantique souhaitée de ce pare-feu ou toute spécification de règles qui n'apporte rien de plus à cette sémantique. Par exemple une règle de filtrage peut ne jamais être exécutée parce qu'elle est inhibée par une autre règle ou parce que deux règles sont redondantes, etc. Afin de représenter les anomalies, nous avons besoin de définir d'abord certaines relations entre les règles.

5.3.1 Relations entre les règles

Une attention particulière doit être portée aux relations qui existent entre les règles de filtrage dans le but de prévenir de possibles conflits entre elles. Généralement, deux règles peuvent être disjointes, équivalentes ou une est incluse dans l'autre. Il est également nécessaire de s'assurer de l'ordre dans lequel ces règles sont spécifiées, car cet ordre peut aussi être la source d'anomalies.

Ordre d'apparition d'une règle

L'ordre de disposition des règles d'un pare-feu est très important et sa sémantique en dépend. Nous définissons cet ordre par ce qui suit :

Définition 5.3.1 La relation d'ordre, notée $\preceq_F$ capture l'ordre d'apparition des règles dans un pare-feu F et est définie comme suit :

$$\text{Si } F = x;\ R_1;\ y;\ R_2;\ z \text{ avec } x, y, z \in \mathcal{S} \text{ alors } R_1 \preceq_F R_2.$$

Cette définition dit tout simplement que R_1 précède R_2 dans l'ordre. Pour plus de simplicité si F est le pare-feu par défaut, nous utilisons la notation simplifiée $R_1 \preceq R_2$ pour dénoter la relation d'ordre entre deux règles.

Relation d'appariement

Nous exprimons dans la définition suivante, les types de relations qui peuvent exister entre les conditions des règles d'un pare-feu.

Définition 5.3.2 Soient $R : \varphi \leftarrow \wedge_{i=1}^{n} p_i$ et $R' : \varphi' \leftarrow \wedge_{i=1}^{n} p'_i$, deux règles de filtrage. Nous disons que les règles R et R' sont :

- Complètement appariées, noté $R \equiv R'$, si $[\![\ \wedge_{i=1}^{n} p_i\ ]\!]^C = [\![\ \wedge_{i=1}^{n} p'_i\ ]\!]^C$
- Inclusivement appariées, noté $R \Subset R'$, si $[\![\ \wedge_{i=1}^{n} p_i\ ]\!]^C \subset [\![\ \wedge_{i=1}^{n} p'_i\ ]\!]^C$.
- Partiellement appariées, noté $R \simeq R'$, s'il y a une relation d'inclusion entre quelques (pas tous) prédicats de la règle R et ceux de la règle R' :

$$\exists i,j \in \{1,\ldots,n\} \mid (p_i, p'_i) \in (\sqsubseteq \cup \sqsubseteq^{-1}) \ \wedge\ (p_j, p'_j) \notin (\sqsubseteq \cup \sqsubseteq^{-1})$$

- Corrélées, noté $R \trianglelefteq\trianglerighteq R'$, s'il y a toujours une relation d'inclusion d'un côté ou d'un autre entre les prédicats de R et ceux de R'.

$$\forall i\ \in \{1,\ldots,n\}\ :\ (p_i, p'_i) \in (\sqsubseteq \cup \sqsubseteq^{-1})$$

- Complètement disjoints, noté $R \neq R'$, s'il n'y a aucune relation d'inclusion, de part et d'autre entre les prédicats de R et ceux de R'. Formellement cela est capturé comme suit :

$$\forall i \in \{1, \ldots, n\} \ : \ (p_i, p'_i) \notin (\sqsubseteq \cup \sqsubseteq^{-1})$$

Nous utilisons la notations $R \subsetneqq R'$ pour désigner que $(R \Subset R') \vee (R \equiv R')$.

La proposition qui suit démontre que les relations définies précédemment sont distinctes, i.e. qu'une et une seule relation peut lier à la fois deux règles R et R'

Proposition 1 La relation entre n'importe quelles deux règles doit être une des relations déjà définies.

Preuve : Pour chaque deux règles R et R', la relation qui les lient est basée sur la relation qui lie leurs prédicats. Chaque deux prédicats peuvent être reliés par une relation $\mathcal{R}$ où $\mathcal{R} \in \{=, \subset, \supset, \neq\}$. Si R est composée des prédicats p_1 et p_2 et si R' est composée des prédicats p'_1 et p'_2, alors toutes les relations existant entre les prédicats du même type de R et R' sont les suivantes :

- Si $[\![p_1]\!]^P = [\![p'_1]\!]^P \wedge [\![p_2]\!]^P = [\![p'_2]\!]^P \Rightarrow R \equiv R'$.
- Si $[\![p_1]\!]^P = [\![p'_1]\!]^P \wedge [\![p_2]\!]^P \subset [\![p'_2]\!]^P \Rightarrow R \Subset R'$.
- Si $[\![p_1]\!]^P = [\![p'_1]\!]^P \wedge [\![p_2]\!]^P \supset [\![p'_2]\!]^P \Rightarrow R \Supset R'$.
- Si $[\![p_1]\!]^P = [\![p'_1]\!]^P \wedge [\![p_2]\!]^P \neq [\![p'_2]\!]^P \Rightarrow R \simeq R'$.
- Si $[\![p_1]\!]^P \subset [\![p'_1]\!]^P \wedge [\![p_2]\!]^P = [\![p'_2]\!]^P \Rightarrow R \Subset R'$.
- Si $[\![p_1]\!]^P \subset [\![p'_1]\!]^P \wedge [\![p_2]\!]^P \subset [\![p'_2]\!]^P \Rightarrow R \Subset R'$.
- Si $[\![p_1]\!]^P \subset [\![p'_1]\!]^P \wedge [\![p_2]\!]^P \supset [\![p'_2]\!]^P \Rightarrow R \trianglelefteq\trianglerighteq R'$.
- Si $[\![p_1]\!]^P \subset [\![p'_1]\!]^P \wedge [\![p_2]\!]^P \neq [\![p'_2]\!]^P \Rightarrow R \simeq R'$.
- Si $[\![p_1]\!]^P \supset [\![p'_1]\!]^P \wedge [\![p_2]\!]^P = [\![p'_2]\!]^P \Rightarrow R \Supset R'$.
- Si $[\![p_1]\!]^P \supset [\![p'_1]\!]^P \wedge [\![p_2]\!]^P \subset [\![p'_2]\!]^P \Rightarrow R \trianglelefteq\trianglerighteq R'$.
- Si $[\![p_1]\!]^P \supset [\![p'_1]\!]^P \wedge [\![p_2]\!]^P \supset [\![p'_2]\!]^P \Rightarrow R \Supset R'$.
- Si $[\![p_1]\!]^P \supset [\![p'_1]\!]^P \wedge [\![p_2]\!]^P \neq [\![p'_2]\!]^P \Rightarrow R \simeq R'$.
- Si $[\![p_1]\!]^P \neq [\![p'_1]\!]^P \wedge [\![p_2]\!]^P = [\![p'_2]\!]^P \Rightarrow R \simeq R'$.
- Si $[\![p_1]\!]^P \neq [\![p'_1]\!]^P \wedge [\![p_2]\!]^P \subset [\![p'_2]\!]^P \Rightarrow R \simeq R'$.
- Si $[\![p_1]\!]^P \neq [\![p'_1]\!]^P \wedge [\![p_2]\!]^P \supset [\![p'_2]\!]^P \Rightarrow R \simeq R'$.
- Si $[\![p_1]\!]^P \neq [\![p'_1]\!]^P \wedge [\![p_2]\!]^P \neq [\![p'_2]\!]^P \Rightarrow R \neq R'$.

À partir des conditions que nous venons de présenter, nous constatons que R et R' sont toujours liées par une des relations présentées dans la *Définition 5.3.2.*

Proposition 2 L'ajout d'un ou plusieurs prédicats à n'importe quelles deux règles liées par une des relations déjà définies crée deux nouvelles règles reliées elles aussi par une des mêmes relations.

Preuve : Pour chaque deux règles R et R' composées de k prédicats, si un nouveau prédicat p_{k+1} est ajouté à chacune des règles alors nous obtenons deux nouvelles règles R'' et R'''. Si R et R' sont liées par une des relations définies précédemment, alors toutes les relations possibles existant entre R et R' sont :

- Si $R \equiv R' \wedge [\![\ p_{k+1}\ ]\!]^P = [\![\ p'_{k+1}\ ]\!]^P \Rightarrow R'' \equiv R'''$.
- Si $R \equiv R' \wedge [\![\ p_{k+1}\ ]\!]^P \subset [\![\ p'_{k+1}\ ]\!]^P \Rightarrow R'' \Subset R'''$.
- Si $R \equiv R' \wedge [\![\ p_{k+1}\ ]\!]^P \supset [\![\ p'_{k+1}\ ]\!]^P \Rightarrow R'' \Subset R'''$.
- Si $R \equiv R' \wedge [\![\ p_k + 1\ ]\!]^P \neq [\![\ p'_{k+1}\ ]\!]^P \Rightarrow R'' \simeq R'''$.
- Si $R \Subset R' \wedge [\![\ p_{k+1}\ ]\!]^P = [\![\ p'_{k+1}\ ]\!]^P \Rightarrow R'' \Subset R'''$.
- Si $R \Subset R' \wedge [\![\ p_{k+1}\ ]\!]^P \subset [\![\ p'_{k+1}\ ]\!]^P \Rightarrow R'' \Subset R'''$.
- Si $R \Subset R' \wedge [\![\ p_{k+1}\ ]\!]^P \supset [\![\ p'_{k+1}\ ]\!]^P \Rightarrow R'' \trianglelefteq\trianglerighteq R'''$.
- Si $R \Subset R' \wedge [\![\ p_{k+1}\ ]\!]^P \neq [\![\ p'_{k+1}\ ]\!]^P \Rightarrow R'' \simeq R'''$.
- Si $R \neq R' \wedge [\![\ p_{k+1}\ ]\!]^P = [\![\ p'_{k+1}\ ]\!]^P \Rightarrow R'' \simeq R'''$.
- Si $R \neq R' \wedge [\![\ p_{k+1}\ ]\!]^P \subset [\![\ p'_{k+1}\ ]\!]^P \Rightarrow R'' \simeq R'''$.
- Si $R \neq R' \wedge [\![\ p_{k+1}\ ]\!]^P \supset [\![\ p'_{k+1}\ ]\!]^P \Rightarrow R'' \simeq R'''$.
- Si $R \neq R' \wedge [\![\ p_{k+1}\ ]\!]^P \neq [\![\ p'_{k+1}\ ]\!]^P \Rightarrow R'' \neq R'''$.
- Si $R \simeq R' \wedge [\![\ p_{k+1}\ ]\!]^P = [\![\ p'_{k+1}\ ]\!]^P \Rightarrow R'' \simeq R'''$.
- Si $R \simeq R' \wedge [\![\ p_{k+1}\ ]\!]^P \subset [\![\ p'_{k+1}\ ]\!]^P \Rightarrow R'' \simeq R'''$.
- Si $R \simeq R' \wedge [\![\ p_{k+1}\ ]\!]^P \supset [\![\ p'_{k+1}\ ]\!] \Rightarrow R'' \simeq R'''$.
- Si $R \simeq R' \wedge [\![\ p_{k+1}\ ]\!]^P \neq [\![\ p'_{k+1}\ ]\!]^P \Rightarrow R'' \simeq R'''$.
- Si $R \trianglelefteq\trianglerighteq R' \wedge [\![\ p_{k+1}\ ]\!]^P = [\![\ p'_{k+1}\ ]\!]^P \Rightarrow R'' \trianglelefteq\trianglerighteq R'''$.
- Si $R \trianglelefteq\trianglerighteq R' \wedge [\![\ p_{k+1}\ ]\!]^P \subset [\![\ p'_{k+1}\ ]\!]^P \Rightarrow R'' \trianglelefteq\trianglerighteq R'''$.
- Si $R \trianglelefteq\trianglerighteq R' \wedge [\![\ p_{k+1}\ ]\!]^P \supset [\![\ p'_{k+1}\ ]\!]^P \Rightarrow R'' \trianglelefteq\trianglerighteq R'''$.
- Si $R \trianglelefteq\trianglerighteq R' \wedge [\![\ p_{k+1}\ ]\!]^P \neq [\![\ p'_{k+1}\ ]\!]^P \Rightarrow R'' \simeq R'''$.

À partir des conditions que nous venons de présenter, nous constatons que R'' et R''' sont toujours liées par une des relations présentées dans la *Définition 5.3.2.*

Théorème 1 L'union de ces relations représente l'ensemble des relations existantes entre n'importe quelles deux règles.

Preuve : Nous prouvons au début que toute relation entre n'importe quelles deux règles R et R' est dans $\mathcal{R} = \{\equiv, \Subset, \simeq, \trianglelefteq\trianglerighteq, \neq\}$. Pour ce faire, nous énumérons toutes les permutations de toutes les relations possibles qui relient les prédicats de R et R'. Chaque deux prédicats peuvent être reliés par une relation de $\{=, \subset, \supset, \neq\}$. Par conséquent, il existe 16 combinaisons possibles de relations entre R et R' (exemple $\forall i : p_i = p'_i \Rightarrow R \equiv R'$

). Cela veut dire que n'importe quelle relation entre R et R' doit être une des relations appartenant à $\mathcal{R}$ (*proposition 1*). Ensuite, nous démontrons que l'ajout d'un ou plusieurs prédicats à R ou à R', qui sont des règles liées par une relation, donne naissance à deux nouvelles règles R'' et R''' (*proposition 2*), qui sont elles-même liées par une des relations définies en haut. Nous démontrons cela aussi en énumérant toutes les combinaisons entre R et R' et les prédicats additionés comme suit :

$$R \Subset R', p_{i+1} \supset p'_{i+1} \Rightarrow R''' \doteq R''$$

Définition 5.3.3 Soit R une règle de filtrage d'un pare-feu. Nous dénotons par $Act(R)$, l'action de la règle, i.e. :

$$Si\ R : \varphi \leftarrow \wedge_{i=1}^{n} p_i \Rightarrow Act(R) = \varphi$$

Dans ce qui suit, nous définissons quatre types d'anomalies potentielles pouvant affecter la configuration d'un pare-feu à savoir : l'ombrage, la généralisation, la corrélation et la redondance.

5.3.2 Ombrage

L'ombrage est une anomalie critique qui peut avoir comme conséquence le bloquage des paquets qui sont spécifiés pour être autorisés et vis versa. Une règle est ombragée quand une autre règle qui la précède dans le pare-feu s'applique à tous les paquets qui peuvent s'appliquer aussi à la règle ombragée de telle manière que cette dernière ne sera jamais activée. Formellement, une règle R' est ombragée par une règle R si :

$$R \preceq R' \ \wedge\ R \subseteq R' \ \wedge\ Act(R) \neq Act(R') \tag{5.1}$$

Cette définition stipule qu'une règle R' est ombragée par une autre règle R, s'il existe une relation de type "complètement appariées" ou de type "inclusivement appariées" entre elles et, dans les deux cas, R' suit R dans l'ordre et leurs actions sont différentes. À titre indicatif, nous donnons l'exemple suivant qui montre comment se présente une telle anomalie dans un pare-feu.

Exemple 5.4.1 : Dans la spécification qui suit, la règle 2 est ombragée par la règle 1.

1	a	←	*.*.*.*	∧	161.120.33.40	∧	tcp	∧	any	∧	80
2	d	←	140.192.37.*	∧	161.120.33.40	∧	tcp	∧	any	∧	80

5.3.3 Généralisation

La généralisation est un autre type d'anomalie des pare-feux. Une règle est dite une généralisation d'une autre, si cette règle générale peut être appliquée à tous les paquets auxquelles s'applique une règle spécifique qui la précède dans le pare-feu. Formellement, R' est une généralisation de R si :

$$R \preceq R' \ \wedge \ R' \Subset R \ \wedge \ Act(R) \neq Act(R'). \tag{5.2}$$

Ce qui se traduit par : il existe une relation de type "inclusivement appariées" entre les règles R et R' et, dans les deux cas, R' suit R dans l'ordre et leurs actions sont différentes. Un exemple d'une telle anomalie est donné dans ce qui suit.

Exemple 5.4.2 : Dans la spécification qui suit, la règle 2 est une généralisation de la règle 1.

1	d	←	140.192.37.20	∧	∗.∗.∗.∗	∧	tcp	∧	any	∧	80
2	a	←	140.192.37.∗	∧	∗.∗.∗.∗	∧	tcp	∧	any	∧	80

Cette anomalie n'est pas vraiment critique pour la sécurité du pare-feu. Elle accroit seulement la taille de la listes des règles. Notons toute de même que changer l'ordre des règles causant une généralisation, transforme cette anomalie en une anomalie d'ombrage.

5.3.4 Corrélation

Deux règles sont corrélées si la première règle dans l'ordre s'applique à certains paquets auxquels s'applique la deuxième règle et si la deuxième règle s'applique aux restes des paquets auxquels s'applique la première règle. Formellement, si R et R' sont deux règles, R et R' sont corrélées si :

$$R \trianglelefteq\trianglerighteq R' \ \wedge \ Act(R) \neq Act(R'). \tag{5.3}$$

Un exemple d'anomalie de corrélation est donné par ce qui suit.

Exemple 5.4.3 : Dans la spécification qui suit, la règle 1 et la règle 2 sont corrélées.

1	d	←	140.192.37.20	∧	∗.∗.∗.∗	∧	tcp	∧	any	∧	80
2	a	←	∗.∗.∗.∗	∧	161.120.33.40	∧	tcp	∧	any	∧	80

L'anomalie de corrélation, comme la généralisation, n'est pas critique pour ls sécurité du pare-feu. Elle est juste un avertissement qui dit que l'action des règles corrélées n'est pas

explicitement gérée par le pare-feu. Considérons l'exemple 5.4.3. Dans cet ordre, tout traffic tcp venant de l'adresse 140.192.37.20 et allant à l'adresse 161.120.33.40 est refusé, alors que si nous inversons l'ordre des deux règles, ce même traffic sera accepté.

5.3.5 Redondance

La redondance est une autre anomalie des pare-feux. Une règle est redondante par rapport à une autre règle, si cette dernière peut s'appliquer à tous les paquets de la règle redondante et les actions des deux règles sont similaires. Formellement, si R et R' sont deux règles, R' est dite redondante par rapport à R si :

$$R \Subset R' \wedge Act(R) = Act(R') \tag{5.4}$$

Ce qui se traduit par : R et R' sont complètement appariées ou R est inclusivement appariées à R' ou R' est inclusivement appariées à R et les actions de R et R' sont les mêmes. Dans ce qui suit, nous donnons l'exemple d'une telle anomalie.

Exemple 5.4.4 : Dans la spécification suivante, la règle 2 est redondante à la règle 1.

1	a	←	140.192.37.∗	∧	∗.∗.∗.∗	∧	tcp	∧	any	∧	21
2	a	←	140.192.37.∗	∧	161.120.33.40	∧	tcp	∧	any	∧	21

La redondance n'influence en aucune manière la politique de sécurité d'un pare-feu. En effet, une règle redondante ne contribue pas au processus de filtrage, mais elle accroît la taille et le temps de recherche dans une liste de règles.

5.4 Système de types

Dans cette section, nous présentons un système de types pour la vérification des pare-feux. Ce système est capable de typer un pare-feu. Le type d'un pare-feu représente la séquence d'anomalies qui existent dans celui-ci. Si cette séquence est vide, le pare-feu est dit correct par rapport à la classe d'anomalie considérée. Pour définir un tel système de types, nous avons commencé par construire une algèbre de types afin de représenter tous les types classifiant les différentes anomalies. Nous avons ensuite défini les règles de typage permettant d'inférer le type d'un pare-feu. Enfin, nous avons mécaniser le système de types par un algorithme d'inférence de types.

5.4.1 Algèbre de types

Nous construisons une algèbre de types à partir des types de base suivants :

- *sha* : pour classifier les anomalies d'ombrage.
- *gen* : pour classifier les anomalies de généralisation.
- *cor* : pour classifier les anomalies de corrélation.
- *red* : pour classifier les anomalies de redondance.

et un opérateur de composition, noté ".", permettant de composer des séquences de types. Chaque séquence permet de capturer un ensemble d'anomalies détectées dans un pare-feu. Plus formellement, Un type est décrit par la grammaire qui suit :

$$\tau \ ::= \ sha \mid gen \mid cor \mid red \mid \tau.\tau$$

Nous supposons, dans le reste de ce chapitre, que $\tau.\epsilon = \epsilon.\tau = \tau$, et que $\mathcal{E}$ dénote l'ensemble de tous les types possibles.

5.4.2 Règles de typage

Nous définissons un certain nombre de fonctions auxiliaires nécessaires pour l'élaboration de notre système de types. Dans la Table 5.1, nous définissons deux fonctions de projection, notées $\mathcal{A}$ et $\mathcal{D}$. La fonction $\mathcal{A}$ permet de construire une seule règle à partir de toutes les composantes d'acceptation d'une séquence quelconque de règles de pare-feu. La fonction $\mathcal{D}$ réalise le même traitement sur les composantes de refus. Les opérateurs $\wedge$ et $\vee$ sont repsectivement le "et" et le "ou" logique.

$\mathcal{A}(F)$	$=$	$a \leftarrow \mathcal{A}'(F)$	$\mathcal{D}(F)$	$=$	$d \leftarrow \mathcal{D}'(F)$
$\mathcal{A}'(a \leftarrow \wedge_{i=1}^{n} p_i)$	$=$	$\wedge_{i=1}^{n} p_i$	$\mathcal{D}'(a \leftarrow \wedge_{i=1}^{n} p_i)$	$=$	$false$
$\mathcal{A}'(d \leftarrow \wedge_{i=1}^{n} p_i)$	$=$	$false$	$\mathcal{D}'(d \leftarrow \wedge_{i=1}^{n} p_i)$	$=$	$\wedge_{i=1}^{n} p_i$
$\mathcal{A}'(a \leftarrow \wedge_{i=1}^{n} p_i \ ; \ F)$	$=$	$\wedge_{i=1}^{n} p_i \vee \mathcal{A}'(F)$	$\mathcal{D}'(a \leftarrow \wedge_{i=1}^{n} p_i \ ; \ F)$	$=$	$\mathcal{D}'(F)$
$\mathcal{A}'(d \leftarrow \wedge_{i=1}^{n} p_i \ ; \ F)$	$=$	$\mathcal{A}'(F)$	$\mathcal{D}'(d \leftarrow \wedge_{i=1}^{n} p_i \ ; \ F)$	$=$	$\wedge_{i=1}^{n} p_i \vee \mathcal{D}'(F)$

TAB. 5.1 – *Fonctions de projection*

Intuitivement, si F est un pare-feu alors :

- $\mathcal{A}(F)$ construit une règle $a \leftarrow c_1 \vee c_2 \ \ldots \vee c_n$ où chaque c_i est la condition de la règle R_i de F tel que $Act(R_i) = a$.

- $\mathcal{D}(F)$ construit une règle $d \leftarrow c'_1 \vee c'_2 \ \ldots \vee c'_n$ où chaque c'_i est la condition de la règle R'_i de F tel que $Act(R'_i) = d$.

Exemple : Si F est le pare-feu suivant :

d	←	140.192.37.20	∧	∗.∗.∗.∗	∧	tcp	∧	any	∧	80
a	←	140.192.37.∗	∧	∗.∗.∗.∗	∧	tcp	∧	any	∧	80
a	←	∗.∗.∗.∗	∧	161.120.33.40	∧	tcp	∧	any	∧	80
d	←	140.192.37.∗	∧	161.120.33.40	∧	tcp	∧	any	∧	80

alors :

$\mathcal{A}(F)$ = a ← (140.192.37.∗ ∧ ∗.∗.∗.∗ ∧ tcp ∧ any ∧ 80) ∨
(∗.∗.∗.∗ ∧ 161.120.33.40 ∧ tcp ∧ any ∧ 80)

$\mathcal{D}(F)$ = d ← (140.192.37.20 ∧ ∗.∗.∗.∗ ∧ tcp ∧ any ∧ 80) ∨
(140.192.37.∗ ∧ 161.120.33.40 ∧ tcp ∧ any ∧ 80)

Unit	$\frac{\square}{R : \epsilon}$
Disjoint	$\frac{R \neq R'}{R;R' : \epsilon}$
Shadow	$\frac{R \preceq R' \quad R \subseteq R' \quad Act(R) \neq Act(R')}{R;R' : sha}$
Generalization	$\frac{R \preceq R' \quad R' \Subset R \quad Act(R) \neq Act(R')}{R;R' : gen}$
Correlation	$\frac{R \trianglelefteq \trianglerighteq R' \quad Act(R) \neq Act(R')}{R;R' : cor}$
Redondance	$\frac{(R,\ R') \in (\subseteq \cup \subseteq^{-1}) \quad Act(R) = Act(R')}{R;R' : red}$
Sequence	$\frac{F_1 : \tau_1 \quad F_2 : \tau_2 \quad F_1 \times F_2 : \tau_3}{F_1;F_2 : \tau_1.\tau_2.\tau_3}$
Crossing	$\frac{\mathcal{A}(F_1);\mathcal{A}(F_2) : \tau_1 \quad \mathcal{A}(F_1);\mathcal{D}(F_2) : \tau_2 \quad \mathcal{D}(F_1);\mathcal{A}(F_2) : \tau_3 \quad \mathcal{D}(F_1);\mathcal{D}(F_2) : \tau_4}{F_1 \times F_2 : \tau_1.\tau_2.\tau_3.\tau_4}$

TAB. 5.2 – *Règles de typage*

La Table 5.2 présente notre système de types. La règle Unit spécifie qu'une règle seule ne peut causer d'anomalie dans la classe d'anomalies considérée. La règle Disjoint stipule que si les conditions de deux règles de filtrage sont complètement disjointes, elles ne peuvent pas

engender d'anomalies. La règle Shadow capture l'anomalie d'ombrage. La règle Generalization capture l'anomalie de généralisation. La règle Correlation capture l'anomalie de corrélation. La règle Redondance capture l'anomalies de redondance. La règle Sequence permet de typer un pare-feu. Finalement, la règle Crossing permet de déduire le type résultant de l'interaction de deux séquences de règles. Notons que le système de type présenté ne spécifie pas comment la séquence de règles du pare-feu est décomposée afin de la typer. Cet aspect peut être facilement intégré dans l'algorithme qui mécanisera le système de types. Par ailleurs, il est aisé de modifier le système de type pour que ce dernier garde trace des règles impliquées pendant la génération du type. Cette fonctionnalité peut être très utile dans le processus d'analyse car elle permet de localiser les règles impliquées dans une anomalie.

Définition 5.4.1 (Correction d'un pare-feu) Un pare-feu F est dit correct si aucune des anomalies présentées à la section 5.3 n'est détectée dans F. Plus formellement :

$$F\text{est correct} \Leftrightarrow F : \epsilon \tag{5.5}$$

Notre système de types offre la possibilité de comparer des regroupements de règles et de détecter des anomalies impliquant plus que deux règles. Cette particularité nous distingue de la plupart des techniques de vérification de pare-feux traitées dans la littérature qui, souvent, réalisent des comparaisons systématiques règle à règle. Ainsi, notre technique offre au moins trois avantages :

- une procédure de vérification efficace en optimisant le nombre de comparaisons des règles. Par exemple, considérons un pare-feu avec 6 règles et un algorithme implantant le système de type avec décomposition dichotomique, alors l'algorithme utilisera seulement 10 comparaisons versus 15 pour une technique de comparaison systématique règle à règle, tout en détectant au moins toutes les anomalies détectées par cette dernière. L'efficacité dans la vérification est cruciale pour pouvoir analyser en pratique des configurations de pare-feux qui peuvent contenir jusqu'à plusieurs milliers de règles.
- une plus grande flexibilité dans le processus de vérification. En effet, si une règle est rajoutée à un pare-feu déjà vérifié, seules deux comparaisons sont nécessaires pour vérifier la nouvelle configuration : projeter l'ancienne configuration en deux règles (acceptation et refus) et les comparer avec la nouvelle règle.
- une meilleure précision de l'analyse. En effet notre système permet de détecter des anomalies entre des séquences de règles plutôt qu'uniquement entre règles individuelles comme c'est le cas des techniques actuelles. Ainsi, notre technique permet des réponses de type "les règles R_1 et R_2 ensemble provoquent une anomalie d'ombrage avec la règle R_3".

5.4.3 Algorithme d'inférence de types

Dans cette section, nous élaborons un algorithme d'inférence de types capable de mécaniser notre système de types. Cet algorithme est présenté dans la Table 5.4. L'idée de cet algorithme est d'utiliser une technique de décomposition dichotomique afin de comparer les règles d'un pare-feu F. Nous résumons cette idée dans les points suivants :

1. Décomposer F en séquences de règles F_1 et F_2.
2. Fusionner toutes les règles d'acceptation de F_1 en une seule règle d'acceptation, et toutes les règles de refus de F_1 en une seule règle de refus.
3. Fusionner toutes les règles d'acceptation de F_2 en une seule règle d'acceptation, et toutes les règles de refus de F_2 en une seule règle de refus.
4. Comparer la règle d'acceptation (résultante) de F_1 avec la règle d'acceptation et ensuite avec celle de refus de F_2 et idem avec la règle de refus de F_1 avec celles d'acceptation et de refus de F_2.
5. Répéter les étapes 1, 2, 3 et 4 pour F_1 et F_2.
6. Arrêter la décomposition quand la séquence de règles ne contient qu'une seule règle.

La fonction principale Infer utilise trois fonctions auxiliaires : SubSequenceEffect, ProjAccept et ProjDeny. Infer permet de synthétiser un type pour un pare-feu. Elle prend comme paramètre un pare-feu codé dans une structure de liste et retourne un type annoté. L'annotation permet de garder trace des règles impliquées dans les anomalies. Pour cela, nous utilisons la syntaxe $R.index$ qui retourne le numéros de séquence de la règle R dans le pare-feu.

La fonction SubSequenceEffect permet d'analyser les anomalies résultant de l'interaction de deux séquences de règles F_1 et F_2. Pour ce faire, chaque séquence de règles est projetée par rapport aux deux types d'actions (a et d) et une vérification croisée est faite par rapport aux règles obtenues. Cette projection est réalisée à l'aide des fonctions ProjAccept et ProjectDeny.

ProjAccept prend une séquence de règles et la projette sur la composante d'acceptation, i.e. construit une seule règle avec toutes les règles d'acceptation de la séquence. De la même manière, ProjectDeny construit une seule règle avec toutes les règles d'acceptation de la séquence.

Le principal atout de l'algorithme d'inférence de types est qu'il utilise une technique dichotomique pour la comparaison des règles. Cette technique nous permet d'optimiser le nombre de comparaisons des règles par rapport aux techniques réalisant une comparaison systématique de toutes les règles. Cette optimisation devient cruciale pour traiter de grandes listes contenant plusieurs milliers de règles. En plus, notre algorithme est plus précis par rapport aux algorithmes proposés dans des recherches subséquentes, du fait qu'il nous permet de détecter une nomalie impliquant deux séquences de règles.

N°	action		ip_src		ip_dst		protocole		port_src		port_dst
1	d	←	140.192.37.20	∧	*.*.*.*	∧	tcp	∧	any	∧	80
2	a	←	140.192.37.*	∧	*.*.*.*	∧	tcp	∧	any	∧	80
3	a	←	*.*.*.*	∧	161.120.33.40	∧	tcp	∧	any	∧	80
4	d	←	140.192.37.*	∧	161.120.33.40	∧	tcp	∧	any	∧	80
5	d	←	140.192.37.30	∧	*.*.*.*	∧	tcp	∧	any	∧	21
6	a	←	140.192.37.*	∧	*.*.*.*	∧	tcp	∧	any	∧	21
7	a	←	140.192.37.*	∧	161.120.33.40	∧	tcp	∧	any	∧	21
8	a	←	140.192.37.*	∧	161.120.33.40	∧	tcp	∧	60	∧	80

TAB. 5.3 – *Spécification d'un pare-feu*

5.5 Étude de cas

Dans cette section, nous élaborons un exemple de spécification d'un pare-feu, présenté à la Table 5.3, ainsi que le résultat de son typage en utilisant notre système de type. Le type généré est ensuite interprété pour expliquer les différentes anomalies détectées dans le pare-feu. Pour faire notre analyse, nous supposons que la séquence de règles du pare-feu est décomposée d'une manière dichotomique et que le système de type est annoté afin de pouvoir garder trace des numéros de règles associés aux composantes du type généré.

Dans une première phase, notre algorithme commence par décomposer en séquences de règles le pare-feu F. Il en résulte donc deux séquences de règles $E_1 = 1.2.3.4$ et $E_2 = 5.6.7.8$. Puis, sur E_1 et E_2, il fusionne toutes les règles d'acceptation ensemble en une seule règle d'*accept* et toutes les règles de refus en une seule règle de *deny*. Le résultat est le suivant : les règles 1 et 4 fusionnées en une seule règle de *deny*, les règles 2 et 3 fusionnées en une seule règle d'*accept* et finalement les règles 6, 7 et 8 fusionnées en une seule règle d'*accept*. Notons par 1&4, 2&3 et 6&7&8 les règles résultantes de ces fusions. L'algorithme compare, ensuite, 1&4 avec 5, 1&4 avec 6&7&8, 2&3 avec 5 et finalement 2&3 avec 6&7&8.

Dans une deuxième phase, l'algorithme décompose E_1 en deux ensembles de règles E_3 et E_4 et E_2 en E_5 et E_6, et va ensuite refaire le même traitement de fusion, s'il y'a lieu, et de comparaison entre les règles jusqu'à ce qu'il n'y est plus de possibilité de décomposition dichotomique.

L'exécution de la fonction infer sur les règles composant le pare-feu F de la Table 5.3, donne les résultats suivants :

$$\begin{aligned} \mathsf{infer}(F) \quad = \quad & gen[1;2].sha[3;4].cor[1;3].sha[2;4].gen[5;6].red[6;7].cor[5;7]. \\ & sha[1;4-6;7;8].red[2;3-6;7;8] \end{aligned}$$

$\mathsf{Infer} : \mathcal{F} \rightarrow \mathcal{E}$

$\mathsf{Infer}(F) =$
  case F of
    $R :: nil \Rightarrow \epsilon$
    $R :: R' :: nil \Rightarrow$
      if $(R \neq R')$ then ϵ
      if $(R \equiv R')$ *and* $(Act(R) \neq Act(R'))$ then $sha[R.index - R'.index]$
      if $(R' \Subset R)$ *and* $(Act(R) \neq Act(R'))$ then $sha[R.index - R'.index]$
      if $(R' \Subset R)$ *and* $(Act(R) \neq Act(R'))$ then $gen[R.index - R'.index]$
      if $(R \trianglelefteq\trianglerighteq R')$ *and* $(Act(R) \neq Act(R'))$ then $cor[R.index - R'.index]$
      if $(R \equiv R')$ *and* $(Act(R) = Act(R'))$ then $red[R.index - R'.index]$
      if $(R \Subset R')$ *and* $(Act(R) = Act(R'))$ then $red[R.index - R'.index]$
      if $(R' \Subset R)$ *and* $(Act(R) = Act(R'))$ then $red[R.index - R'.index]$
    let $(F1,F_2) = \mathsf{split}(F) \Rightarrow \mathsf{Infer}(F_1).\mathsf{Infer}(F_2).\mathsf{SubSequenceEffect}(F_1,F_2)$

$\mathsf{SubSequenceEffect} : \mathcal{S} \times \mathcal{S} \rightarrow \mathcal{E}$

$\mathsf{SubSequenceEffect}(F_1, F_2)=$
  let t_1=$\mathsf{Infer}([\mathsf{ProjAccept}(F_1), \mathsf{ProjAccept}(F_2)])$;
    t_2=$\mathsf{Infer}([\mathsf{ProjAccept}(F_1), \mathsf{ProjDeny}(F_2)])$;
    t_3=$\mathrm{Infer}([\mathsf{ProjDeny}(F_1), \mathsf{ProjAccept}(F_2)])$;
    t_4=$\mathrm{Infer}([\mathsf{ProjDeny}(F_1), \mathsf{ProjDeny}(F_2)])$;
  in t_1,t_2,t_3,t_4;

$\mathsf{ProjAccept} : \mathcal{S} \rightarrow \mathcal{R}$

$\mathsf{ProjAccept}(F)=$
let $P =$
  if $(F = nil)$ then false
  if $(F = a \leftarrow \wedge_{i=1}^{n} p_i :: F')$ then $\wedge_{i=1}^{n} p_i \vee \mathsf{ProjAccept}(F')$
  if $(F = d \leftarrow \wedge_{i=1}^{n} p_i :: F')$ then $\mathsf{ProjAccept}(F')$
in $a \leftarrow P$
end

$\mathsf{ProjDeny} : \mathcal{S} \rightarrow \mathcal{R}$

$\mathsf{ProjDeny}(F)=$
let $P =$
  if $(F = nil)$ then $false$
  if $(F = d \leftarrow \wedge_{i=1}^{n} p_i :: F')$ then $\wedge_{i=1}^{n} p_i \vee \mathsf{ProjDeny}(F')$
  if $(F = a \leftarrow \wedge_{i=1}^{n} p_i :: F')$ then $\mathsf{ProjDeny}(F')$
in $d \leftarrow P$
end

TAB. 5.4 – *Algorithme d'inférence de types*

Par conséquent, notre algorithme d'inférence de types détecte les anomalies que nous décrivons ci-après:

- la règle 2 est une généralisation de la règle 1.
- la règle 4 est ombragée par la règle 3.
- la règle 3 est une corrélation de la règle 1.
- la règle 4 est ombragée par la règle 2.
- la règle 6 est une généralisation de la règle 5.
- la règle 7 est une redondance de la règle 6.
- la règle 7 est une corrélation de la règle 5.
- la séquence de règles [6;7;8] provoque une anomalie d'ombrage avec la séquence de règles [1;4].
- la séquence de règles [6;7;8] provoque une anomalie de redondance avec la séquence de règles [2;3].

Un algorithme traditionnel de comparaison des règles aurait effectué 21 comparaisons, même celles inutiles et qui n'exhibent pas de conflits, afin de détecter toutes les anomalies qui existent dans le pare-feu de l'exemple précédent, d'autant plus qu'il n'aurait pas pu exhiber les conflits entre des séquences de règles. Notre algorithme n'utilise que 13 comparaisons pour détecter les anomalies existants dans l'exemple précédent et permet d'exhiber les conflits aussi bien entre des règles individuelles qu'entre des séquences de règles. Le gain est donc considérable en terme de précision et d'efficacité de détection d'anomalies.

5.6 Conclusion

Dans ce chapitre, nous avons élaboré un système de types capable de détecter une classe d'anomalies dans les pare-feux. Cette approche nous a permis d'obtenir une analyse plus fine que celles proposées actuellement dans la littérature, puisque le système de types trouve les anomalies impliquant plus que deux règles à la fois. Nous avons également proposé un algorithme qui mécanise le système de types. Cet algorithme est très efficace puisqu'il évite la comparaison systématique de toutes les règles et propose plutôt une comparaison dichotomique. Par conséquent, il a permis de réduire le nombre de comparaisons effectuées entre les règles. L'algorithme en question retourne une liste d'anomalies qui affecte un pare-feu. Pour chaque anomalie, l'algorithme précise les règles qui la provoquent.

Chapitre 6

Conclusion générale

Ce mémoire est un pas vers le développement de nouvelles techniques d'analyse des pare-feux. Nous avons donné, dans les chapitres précédents, une description claire et intéressante des principales contributions y compris celles de ce document. Naturellement, un résumé des différentes étapes que nous avons entrepris pour mettre à point cette recherche s'avère nécessaire.

Dans un premier temps, nous avons proposé un langage dédié à la spécification des pare-feux. Nous avons baptisé ce langage FPSL et nous l'avons doté d'une syntaxe et d'une sémantique formelle. Notre langage a été conçu de manière à permettre la spécification des ingrédients nécessaires pour capturer des politiques globales aussi bien que locales.

Dans la deuxième partie de cette recherche, nous nous sommes focalisés sur le développement d'un système de types capable de détecter les anomalies dans les pare-feux. Pour ce faire, nous avons tout d'abord défini les relations qui peuvent exister entre les règles d'un pare-feu. La présence de ces relations est à l'origine de tout conflit entre les règles. Nous avons ensuite défini les types d'anomalies résultantes de ces conflits et proposé un système de règles de typage pour capturer formellement ces anomalies. Nous avons, finalement, produit un algorithme d'inférence qui mécanise notre système de types.

La technique que nous avons proposée apporte une contribution majeure par rapport aux autres méthodes d'analyse des pare-feux existantes sur les points suivants :

- une procédure de vérification efficace en optimisant le nombre de comparaisons des règles. Par exemple, considérons un pare-feu avec 6 règles et un algorithme implantant le système de type avec décomposition dichotomique, alors l'algorithme utilisera seulement 10 comparaisons versus 15 pour une technique de comparaison systématique règle à règle, tout en détectant au moins toutes anomalies détectées par cette

dernière. L'efficacité dans la vérification est cruciale pour pouvoir analyser en pratique des configurations de pare-feux qui peuvent contenir jusqu'à plusieurs milliers de règles.

- une plus grande flexibilité dans le processus de vérification. En effet, si une règle est rajoutée à un pare-feu déjà vérifié, seules deux comparaisons sont nécessaires pour vérifier la nouvelle configuration : projeter l'ancienne configuration en deux règles (acceptation et refus) et les comparer avec la nouvelle règle.
- une meilleure précision de l'analyse. En effet notre système permet de détecter des anomalies entre des séquences de règles plutôt qu'uniquement entre règles individuelles comme c'est le cas des techniques actuelles. Ainsi, notre technique permet des réponses de type : les règles R_1 et R_2 ensemble provoquent une anomalie de corrélation avec la règle R_3.

Quelques améliorations peuvent être envisagées :

- La génération automatique des politiques locales à partir des politiques globales.
- La détection d'anomalies dans plusieurs pare-feux distribués sur plusieurs réseaux.
- Paramétrer la décomposition dans l'algorithme d'inférence de type. Ceci permettra de garantir plus de fléxibilité dans le processus de vérification.
- Malheureusement, dans ce travail de recherche, nous n'avons pas traité du problème de validation des pare-feux. Il serait judicieux de développer des techniques d'ajout, de modification et de suppression automatiques des règles de filtrage sans altérer la sémantique du pare-feu. Le cas échéant, il est primordial de produire des techniques de correction automatique de tout conflit provoqué par la mise à jour des listes des pare-feux.

Annexe A

Exemples de spécification de politiques locales

a	⟵	from(Finance)	∧	to(Periphery)	∧	using(any, any)	∧	typeofprot(proxy)
a	⟵	from(Engineering)	∧	to(Periphery)	∧	using(any, any)	∧	typeofprot(proxy)
a	⟵	from(External)	∧	to(Engineering)	∧	using(any, any)	∧	typeofprot(smtp)
a	⟵	from(External)	∧	to(Finance)	∧	using(any, any)	∧	typeofprot(smtp)
a	⟵	from(Allied)	∧	to(Engineering)	∧	using(any, any)	∧	typeofprot(smtp)
a	⟵	from(Allied)	∧	to(Finance)	∧	using(any, any)	∧	typeofprot(smtp)
a	⟵	from(Periphery)	∧	to(Engineering)	∧	using(any, any)	∧	typeofprot(smtp)
a	⟵	from(Periphery)	∧	to(Finance)	∧	using(any, any)	∧	typeofprot(smtp)
a	⟵	from(Finance)	∧	to(Engineering)	∧	using(any, any)	∧	typeofprot(udp)
a	⟵	from(Allied)	∧	to(Engineering)	∧	using(any, any)	∧	typeofprot(udp)
a	⟵	from(Engineering)	∧	to(Engineering)	∧	using(any, any)	∧	typeofprot(udp)
a	⟵	from(Allied)	∧	to(Periphery)	∧	using(any, any)	∧	typeofprot(proxy)
d	⟵	from(External)	∧	to(Engineering)	∧	using(any, any)	∧	typeofprot(udp)
a	⟵	from(Allied)	∧	to(Engineering)	∧	using(any, any)	∧	typeofprot(http)
a	⟵	from(Allied)	∧	to(Engineering)	∧	using(any, any)	∧	typeofprot(ftp)
a	⟵	from(Allied)	∧	to(Engineering)	∧	using(any, any)	∧	typeofprot(telnet)

TAB. A.1 – *Politique locale du pare-feu Engineering-Periphery*

a	⟵	from(External)	∧	to(Engineering)	∧	using(any, any)	∧	typeofprot(smtp)
a	⟵	from(External)	∧	to(Finance)	∧	using(any, any)	∧	typeofprot(smtp)
a	⟵	from(Allied)	∧	to(Engineering)	∧	using(any, any)	∧	typeofprot(smtp)
a	⟵	from(Allied)	∧	to(Finance)	∧	using(any, any)	∧	typeofprot(smtp)
a	⟵	from(Periphery)	∧	to(Engineering)	∧	using(any, any)	∧	typeofprot(smtp)
a	⟵	from(Periphery)	∧	to(Finance)	∧	using(any, any)	∧	typeofprot(smtp)
a	⟵	from(Finance)	∧	to(Engineering)	∧	using(any, any)	∧	typeofprot(udp)
a	⟵	from(Engineering)	∧	to(Engineering)	∧	using(any, any)	∧	typeofprot(udp)
a	⟵	from(Allied)	∧	to(Engineering)	∧	using(any, any)	∧	typeofprot(udp)
a	⟵	from(Allied)	∧	to(Engineering)	∧	using(any, any)	∧	typeofprot(http)
a	⟵	from(Allied)	∧	to(Engineering)	∧	using(any, any)	∧	typeofprot(ftp)
a	⟵	from(Allied)	∧	to(Engineering)	∧	using(any, any)	∧	typeofprot(telnet)
d	⟵	from(External)	∧	to(Engineering)	∧	using(any, any)	∧	typeofprot(udp)

TAB. A.2 – *Politique locale du pare-feu Allied-Engineeing*

a	⟵	from(Periphery)	∧	to(External)	∧	using(any, any)	∧	typeofprot(http)
a	⟵	from(Periphery)	∧	to(External)	∧	using(any, any)	∧	typeofprot(ftp)
a	⟵	from(Periphery)	∧	to(External)	∧	using(any, any)	∧	typeofprot(telnet)
a	⟵	from(Allied)	∧	to(Periphery)	∧	using(any, any)	∧	typeofprot(proxy)
d	⟵	from(Finance)	∧	to(External)	∧	using(any, any)	∧	typeofprot(http)
d	⟵	from(Finance)	∧	to(External)	∧	using(any, any)	∧	typeofprot(ftp)
d	⟵	from(Finance)	∧	to(External)	∧	using(any, any)	∧	typeofprot(telnet)
d	⟵	from(Engineering)	∧	to(External)	∧	using(any, any)	∧	typeofprot(http)
d	⟵	from(Engineering)	∧	to(External)	∧	using(any, any)	∧	typeofprot(ftp)
d	⟵	from(Engineering)	∧	to(External)	∧	using(any, any)	∧	typeofprot(telnet)

TAB. A.3 – *Politique du pare-feu Periphery-External*

a	⟵	from(Finance)	∧	to(Periphery)	∧	using(any, any)	∧ typeofprot(proxy)
a	⟵	from(Engineering)	∧	to(Periphery)	∧	using(any, any)	∧ typeofprot(proxy)
a	⟵	from(External)	∧	to(Engineering)	∧	using(any, any)	∧ typeofprot(smtp)
a	⟵	from(External)	∧	to(Finance)	∧	using(any, any)	∧ typeofprot(smtp)
a	⟵	from(Allied)	∧	to(Engineering)	∧	using(any, any)	∧ typeofprot(smtp)
a	⟵	from(Allied)	∧	to(Finance)	∧	using(any, any)	∧ typeofprot(smtp)
a	⟵	from(Allied)	∧	to(Engineering)	∧	using(any, any)	∧ typeofprot(smtp)
a	⟵	from(Allied)	∧	to(Finance)	∧	using(any, any)	∧ typeofprot(smtp)
d	⟵	from(Allied)	∧	to(Finance)	∧	using(any, any)	∧ typeofprot(http)
d	⟵	from(Allied)	∧	to(Finance)	∧	using(any, any)	∧ typeofprot(ftp)
d	⟵	from(Allied)	∧	to(Finance)	∧	using(any, any)	∧ typeofprot(telnet)
a	⟵	from(Periphery)	∧	to(Finance)	∧	using(any, any)	∧ typeofprot(http)
a	⟵	from(Periphery)	∧	to(Finance)	∧	using(any, any)	∧ typeofprot(ftp)
a	⟵	from(Periphery)	∧	to(Finance)	∧	using(any, any)	∧ typeofprot(telnet)
d	⟵	from(Engineering)	∧	to(Finance)	∧	using(any, any)	∧ typeofprot(http)
d	⟵	from(Engineering)	∧	to(Finance)	∧	using(any, any)	∧ typeofprot(ftp)
d	⟵	from(Engineering)	∧	to(Finance)	∧	using(any, any)	∧ typeofprot(telnet)
a	⟵	from(Allied)	∧	to(periphery)	∧	using(any, any)	∧ typeofprot(proxy)

TAB. A.4 – *Politique du pare-feu Finance-Periphery*

d	⟵	from(Finance)	∧	to(External)	∧	using(any, any)	∧ typeofprot(http)
d	⟵	from(Finance)	∧	to(External)	∧	using(any, any)	∧ typeofprot(ftp)
d	⟵	from(Finance)	∧	to(External)	∧	using(any, any)	∧ typeofprot(telnet)
d	⟵	from(Engineering)	∧	to(External)	∧	using(any, any)	∧ typeofprot(http)
d	⟵	from(Engineering)	∧	to(External)	∧	using(any, any)	∧ typeofprot(ftp)
d	⟵	from(Engineering)	∧	to(External)	∧	using(any, any)	∧ typeofprot(telnet)
a	⟵	from(Periphery)	∧	to(External)	∧	using(any, any)	∧ typeofprot(http)
a	⟵	from(Periphery)	∧	to(External)	∧	using(any, any)	∧ typeofprot(ftp)
a	⟵	from(Periphery)	∧	to(External)	∧	using(any, any)	∧ typeofprot(telnet)

TAB. A.5 – *Politique du pare-feu Allied-External*

Bibliographie

[1] E. S. Al-Shaer and H. H. Hamed. Firewall policy advisor for anomaly discovery and rule editing. In *Proceedings of IEEE/IFIP Integrated Management Conference(IM'2003)*, Mars 2003.

[2] C. Alaettinoglu, C. Villamizar, E. Gerich, D. Kessens, D. Meyer, T. Bates, D. Karrenberg, and M. Terpstra. Routing policy specification language (rpsl). Technical report, UCS/ISI and Cisco Systems and University of Oregon, http://citeseer.ist.psu.edu/alaettinoglu97routing.html, June 1999.

[3] M. Blaze, J. Feigenbaum, and J. Ioannidis. The keynote trust-management system, version 2. Technical Report RFC 2704, ATT Labs - Research and U. of Pennsylvania, Septembre 1999.

[4] B. Bolling and I. Wegener. Improving the variable ordering of ordered binary-decision diagrams is np-complete. In *IEEE Transactions on Computers*, volume 45(9), pages 993–1002, September 1996.

[5] R. Bruyant. Symbolic boolean manipulation with ordered binary-decision diagrams. In *ACM Computing Surveys*, volume 24(3), pages 293–318, September 1992.

[6] L. Cardelli and A. D. Gordon. Mobile ambients. In *Proceeding of FoSSaCS'98, LNCS 1378*, pages 140–155. Springer, 1998.

[7] L. Cardelli and A. D. Gordon. Types for mobile ambients. In *Proceedings of POPL'99*, pages 79–92. ACM Press, 1999.

[8] D. Chapman and E. Zwicky. *Building Internet Firewalls. Second Edition.* O'Reilley and Associates, Inc., 2000.

[9] W. R. Cheswick, S. M. Bellovin, and A. D. Rubin. *Firewalls and Internet Security. Repelling the Wily Hacker.* Addison Wesley, 2003.

[10] M. Condell, C. Lynn, and J. Zao. Security policy specification language (spsl). Technical report, Internet Draft, Network Working Group BBN/GTEI, http://www.ietf.org/proceedings/98dec/I-D/draft-ietf-ipsec-spsl-00.txt, 1999.

[11] N. Dulay, E. Lupu, M. Sloaman, and N. Damianou. A policy deployment model for the ponder language. In *Proceeding of IEEE/IFIP International Symposium on Integrated Network Management (IM'2001)*. IEEE press, May 2001.

[12] P. Eronen and J. Zitting. An expert system for analysing firewall rules. In *Proceedings of the 6th Nordic Workshop on Secure IT Systems (NordSec 2001)*, Novembre 2001.

[13] T. Escamilla. *Intrusion Detection - Network Security Beyond the Firewall.* Wiley and sons, 1998.

[14] L. Gong. *Inside Java 2 Platform Security.* Addison-Wesley, 1999.

[15] J. D. Guttman. Filtering postures : Local enforcement for global policies. In *Proceeding of the IEEE Symposium on Security and Privacy*, pages 120–29, May 1997.

[16] J. D. Guttman. Security goals: Packet trajectories and strand spaces. In *Lecture Notes in Computer Science*, volume 2171, pages 197–263, 2001.

[17] S. Hazelhurst. Algorithms for analyzing firewall and router access lists. Technical Report TR-Wits-CS-1999-5, Department of Computer Science, University of Witwatersrand, South Africa, July 1999.

[18] M. Hendry. *Practical Computer Network Security.* Hendry, Mike, 1995.

[19] Cisco Systems Inc. Configuring ip systems. Technical report, Cisco Systems Inc, http://www.cisco.com/univercd/cc/td/doc/product/software, 1997.

[20] S. Jajodia, P. Samari, V. S. Subrahmanian, and E. Bertino. A unified framework for enforcing multiple access control policies. In *International Conference on Management of Data*, pages 475–485, 1997.

[21] M. Nacht. The spectrum of modern firewalls. *Computers and Security*, 17(1):54–56, 1997.

[22] F. Neilson, H. R. Nielson, and R. R. Hansen. Validating firewalls using flow logics. In *Theoretical Computer Science*, volume 283, pages 381–418, 2002.

[23] M. Vandenwauver, C. Vaduva, J. Classens, and R. Mayer. Why entrepsises need more than firewalls and intrusion detection systems. In *Proceeding of the Eighth IEEE International Workshop On Enabling Technologies : Infrastructure for Collaborative Entreprises*, pages 152–157, Stanford, California, June 1999.