Projet d'introduction à la recherche

Analyse et migration de pare-feux

Sommaire

Remerciements....4
1 Introduction....5
1.1 Fonctionnement d'un pare-feu....5
1.2 Problèmes liés au filtrage....5
1.3 Problématique des outils de migrations....5
2 Le projet....6
2.1 Le sujet du projet....6
2.2 Les enjeux du projet....6
2.3 Les outils utilisés....7
2.4 La structure globale envisagée pour l'application....7
3 L'outil d'analyse....8
3.1 Introduction au problème....8
3.2 Les solutions envisagées....8
3.3 Les solutions mises en place....9
3.3.1 La représentation abstraite....9
3.3.2 L'analyseur....10
3.4 Exemples d'analyse....12
3.4.1 Exemple de conflit....12
3.4.2 Exemple de redondance....13
3.4.3 Exemple de conflit lié à la priorité des règles....13
3.5 Les tests effectués....14
4 L'outil de migration....15
4.1 Introduction au fonctionnement....15
4.2 Les solutions mises en place....15
4.2.1 Traduction de l'entrée (Parsers)....16
4.2.2 Représentation intermédiaire et conversion (Wrappeurs)....16
4.2.3 Traduction vers la sortie (Printers)....17
4.3 Exemples de migrations....17
4.4 Les tests effectués....18
5 Conclusion....19
Bibliographie....20
Glossaire....21
Annexes....22

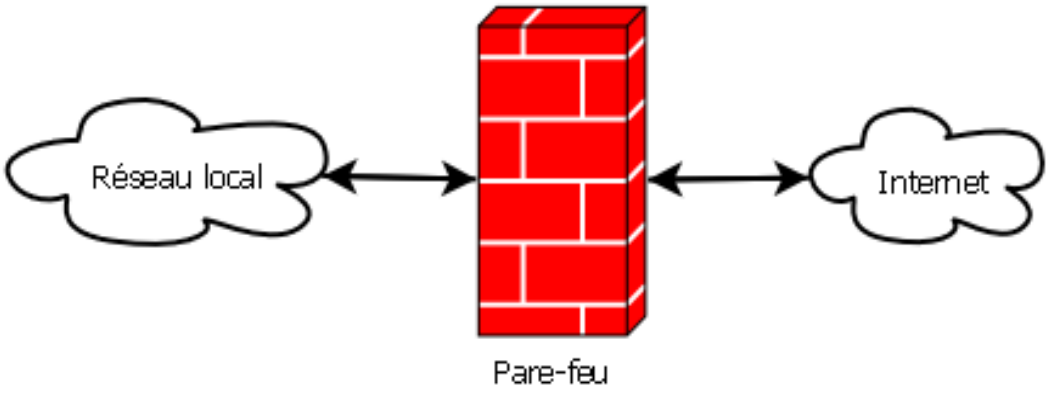
Remerciements

Nous remercions nos encadrants M.Horatio CIRSTEA et M.Pierre-Etienne MOREAU de l'équipe de recherche PAREO (Loria) pour nous avoir fourni toute les supports nécessaires et avoir toujours été disponible pour nous aider durant ce projet.

1 Introduction

À l'aire de l'information et de l'explosion de l'internet, la sécurité occupe de plus en plus les esprits. En effet, pour les particuliers comme pour les entreprise la sécurisation des installations informatique a un enjeu de plus en plus important. Cette sécurisation a une importance primordiale, en particulier pour les entreprises, imaginez que toutes les informations sur les ordinateurs d'une entreprise soient consultable à partir de n'importe quel ordinateur relié à internet, le résultat serait désastreux. Un des moyen les plus efficace de sécuriser les installations liés au réseau est l'utilisation d'un pare-feu. Le pare-feu permet dans une certaine mesure d'isoler plusieurs réseaux, comme par exemple un réseau local de l'internet. Le pare-feu joue le rôle d'un « mur » qui a pour but de filtrer le trafic désiré du trafic non désiré. On comprend donc assez aisément l'intérêt d'un tel type d'outils.

La mise en place d'un pare-feu personnel, par exemple, peut être représentée assez synthétiquement de la manière suivante:

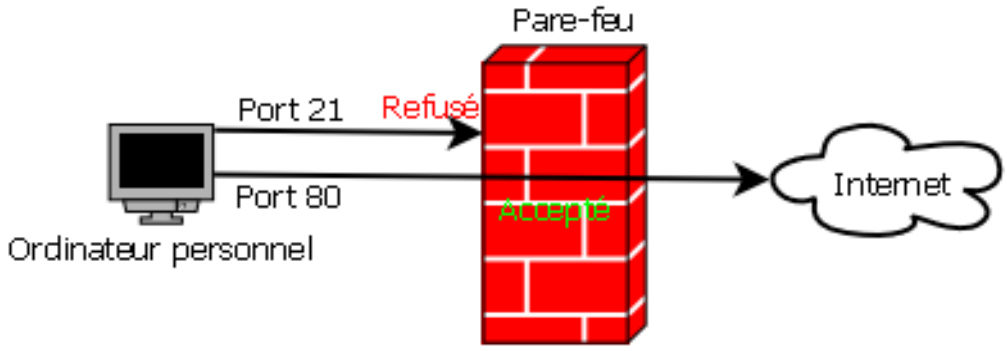


Il existe beaucoup de pare-feux différents, certains libres comme *iptables/Netfilter*, *ipfw*, *packet filter*, d'autres propriétaires développés par de grandes firmes. Dans un soucis d'efficacité (la documentation et l'utilisation des outils propriétaire est souvent payante), durant notre projet nous n'avons travaillé qu'avec des outils libres.

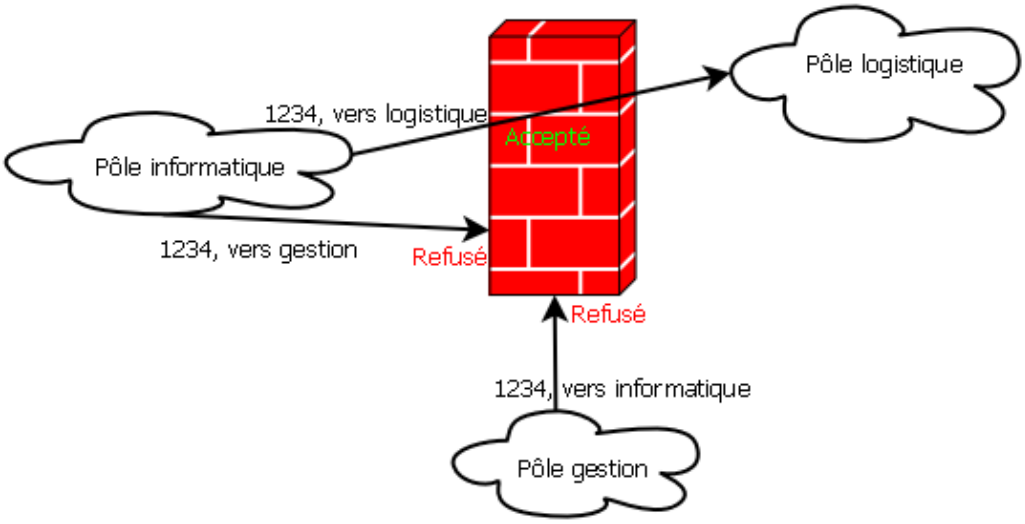
1.1 Fonctionnement d'un pare-feu

Il existe un grand nombre de pare-feux différents qui ont souvent des fonctionnements bien distincts, cependant il existe un certain nombre de points communs entre la plupart d'entre eux. Pour une question d'efficacité et de maintenabilité, la plupart des pare-feux proposent un paramétrage s'appuyant sur l'utilisation d'un ensemble de règles. Chaque règle permet d'identifier un trafic d'un certain type et de donner une action à accomplir lorsque l'on rencontre un trafic de ce type. Les règles sont souvent de la forme « Autoriser un certain type de trafic » ou « Refuser un certain type de trafic », l'efficacité du pare-feu reposant sur le choix des règles les plus adaptées à la configuration et à l'utilisation de l'infrastructure réseau en place. Pour illustrer différents types d'utilisation d'un pare-feu, nous allons présenter quelques exemples d'implantations.

Premier exemple, supposons un réseau sur lequel nous voulons seulement autoriser la navigation sur internet (ce qui correspond à un trafic sur le port 80 dans le protocole TCP), typiquement un ordinateur personnel. Le pare-feux sera paramétré avec deux règles : « Accepter le trafic sur le port 80 » et « Refuser tout le reste du trafic ». Le pare-feux se chargera donc de bloquer tout le trafic qui ne correspondra pas à la navigation sur internet. De façon schématique :



Nous pouvons aussi imaginer des règles un peu plus complexes, par exemples, supposons une entreprise qui possède deux réseaux, le réseau du pôle *informatique* dont l'adresse de toutes les machines va de 192.168.1.0 à 192.168.1.255 et le réseau du pôle *logistique* donc l'adresse de toutes les machines va de 192.168.2.0 à 192.168.2.255. Elle souhaite que les machines de ces deux réseaux ne puissent que s'échanger des informations à travers le système de messagerie de l'entreprise qui utilise le port 1234. Nous aurons règles sur le pare-feu de l'entreprise: « Accepter le trafic allant de 192.168.1.0-192.168.1.255 vers 192.168.2.0-192.168.2.255 sur le port 1234 », « Accepter le trafic allant de 192.168.2.0-192.168.2.255 vers 192.168.1.0-192.168.1.255 sur le port 1234 » et « Refuser tout le reste du trafic ». De façon schématique :



1.2 Problèmes liés au filtrage

L'action de filtrage peut s'avérer plus ou moins problématique. Dans la mesure ou dans la plupart des cas le filtrage est paramétré à l'aide d'un ensemble de règles, il y a des chances que des règles de cet ensemble interfèrent ou soient contradictoires. Tout l'enjeux du filtrage est donc de mettre en place une politique bien précise tout en préservant l'intégrité de l'ensemble des règles.

La détection de ce type de problèmes lié à l'intégrité peut être plus ou moins aisé. Par exemple, on constatera facilement que la règle « Accepter le trafic de 192.168.0.1 » et « Refuser le trafic de 192.168.0.1 » sont contradictoires. Cependant, le conflit sera déjà un plus difficile à détecter si on compare les règles « Accepter le trafic de 192.168.1.0-192.168.1.255 » et « Refuser le trafic de 192.168.1.128-192.168.2.128 ». La vérification manuelle devient quasi-irréalisable à partir d'une vingtaine ou d'une trentaine de règles, sachant que les pare-feux en entreprise peuvent utiliser plusieurs milliers de règles, l'automatisation de cette tâche peut sembler importante.

1.3 Problématique des outils de migrations

Il arrive souvent que des entreprises souhaitent de changer leur pare-feu pour en utiliser un autre plus récent, moins cher, plus performant. Il faut donc pour cela importer dans le nouveau pare-feu toute la politique de filtrage de l'ancien système. Il n'existe malheureusement pas de standards pour l'écriture des règles qui permette de réaliser aisément ce type de migration. Chaque pare-feu a sa propre façon de décrire ses règles et très souvent la migration est réalisée à la main ou l'idée est tout simplement abandonnée. D'autre part, les pare-feux ayant tous une approche du filtrage qui leur est propre, aujourd'hui, il est vraiment très difficile d'élaborer un outil permettant de traduire les règles d'un pare-feu vers un pare-feu d'un autre type de façon automatique. C'est en partie pour cette raison qu'il n'existe pas énormément (voir pas du tout) d'outil de ce type sur le marché.

2 Le projet

2.1 Le sujet du projet

Le sujet initial du projet était de réaliser un outil permettant d'étudier un ensemble de règles *iptables* pour les synthétiser à plus haut niveau dans une grammaire abstraite en utilisant l'outil *Tom* qui permet de réaliser des transformation de façon très simple. Une fois cette transformation opérée, il s'agissait d'analyser la représentation abstraite pour effectuer des optimisations.

Nous nous sommes assez vite rendu compte qu'en utilisant l'outil *Tom*, il serait aisé de synthétiser des règles d'autres types de pare-feux pour les analyser, l'enjeu étant d'avoir une représentation abstraite assez modulaire pour pouvoir gérer des pare-feux ayant une approche bien différente du filtrage. Il nous fallait ainsi être capable de gérer la synthétisation de règles de plusieurs types de pare-feux.

Après discussion avec l'un des principaux intéressé du projet, nous nous sommes rendu compte que l'approche adoptée pour afficher les règles post-analyse était assez proche voir équivalente à celle d'un outil de migration. Notre outil devais désormais être capable de réaliser la migration d'un pare-feu d'un certain type vers un pare-feu d'un autre type.

2.2 Les enjeux du projet

Après avoir recherché des outils du même type, nous nous sommes rendu compte qu'il n'existait que très peu, voir pas, d'outils permettant d'effectuer ce type de migration ainsi que ce type d'analyses sur différents types de pare-feux.

Aussi nous nous sommes rendu compte qu'il existait une réelle demande pour un outil de ce type. Bien souvent, faute de trouver un outil permettant de réaliser une analyse ou une migration de façon automatique, les administrateurs systèmes procèdent à la main, ou au vu de la quantité de données à traiter (pouvant aller jusqu'à plusieurs milliers de règles) abandonnaient tout simplement l'idée d'une analyse et repartaient « de zéro » lors d'une migration. Un tel outil peut aussi être relativement pratique et efficace pour l'optimisation, les pare-feux d'entreprise, suite à des modifications « à la volée », se retrouvant souvent à contenir plusieurs milliers de règles très souvent incohérentes pour la plupart.

2.3 Les outils utilisés

Pour nous permettre d'implémenter les outils de d'analyse et de migration (implémentation qui aurait été fastidieuse en *Java*), nos encadrant de projet nous ont proposé d'utiliser l'outil *Tom* couplé avec d'autres outils plus adaptés aux différentes phases de notre analyse.

Ainsi, nous avons été amenés à utiliser l'outil *Gom* qui permet de créer des structure abstraites de façon assez modulaire et qui permet d'automatiser certaine réécritures de structures.

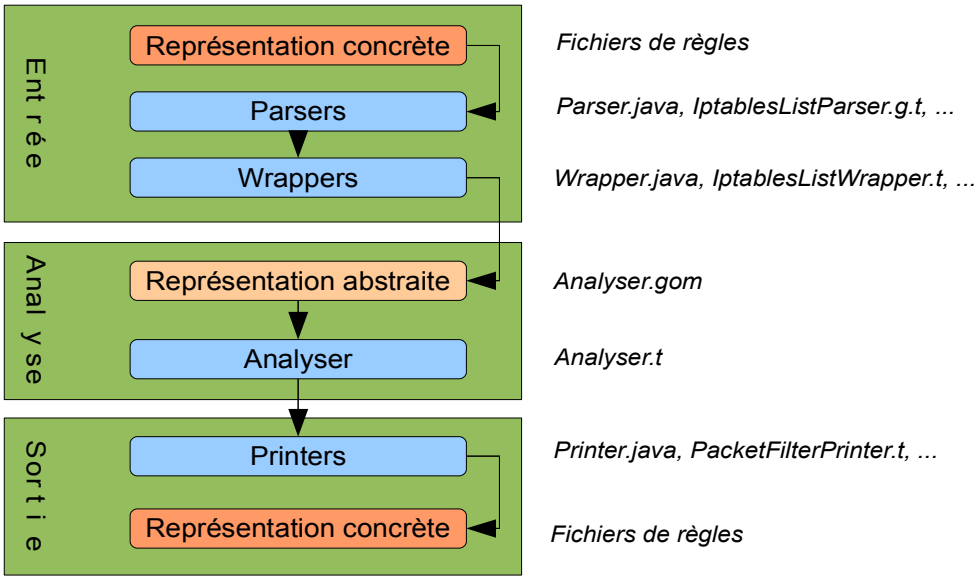
Nous avons aussi utilisé l'outil *Antlr* qui permet d'effectuer une analyse lexicale et syntaxique (avec des possibilités équivalentes à celles de *Lex* et *Yacc*), il nous a été utile pour convertir les ensembles de règles en entrée vers une représentation abstraite de ceux-ci.

Ces deux outils couplés avec l'outil *Tom* (se basant sur *Java*), développé par un de nos encadrant, qui permet d'effectuer des transformations et des traitements sur des données de façon très facile et efficace, nous ont permis de répondre à la problématique proposée de façon très propre, modulaire et maintenable (*Tom* traite des données structurées avec l'outil *Gom*).

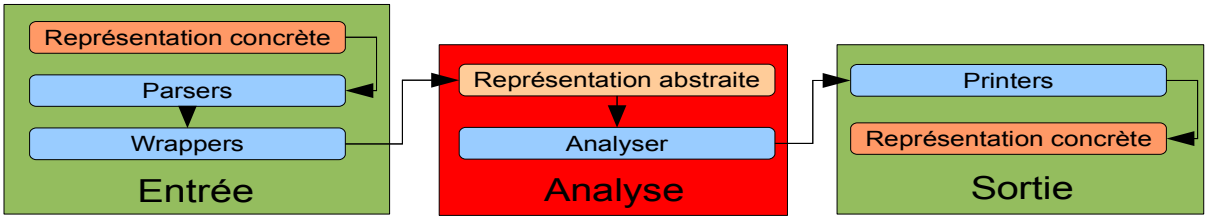
Nous avons aussi appris à utiliser l'outil *Apache Ant* qui nous a permis de faire fonctionner tous ces outils en parallèle en automatisant la compilation de la même façon que le permet l'outil *GNU/Make*.

2.4 La structure globale envisagée pour l'application

Suite au développement de plusieurs versions de l'application, nous somme arrivés à une structure globale qui est la suivante :



3 Analyse et détection



3.1 Introduction au problème

L'idée étant de réaliser les analyses sur des données abstraites d'une représentation commune à tous les pare-feux, le premier enjeu a été de trouver une représentation qui ne soit pas trop proche d'un pare-feu en particulier et qui soit assez modulaire pour qu'il soit possible d'exporter tous les types de représentations concrètes dans cette représentation.

Une fois cette partie terminée, il nous a fallu réfléchir aux différents types d'incohérences et d'optimisations possibles sur un ensemble de règles. Nous avons eu quelques difficultés à trouver tous les types d'incohérences possibles (le principal problème étant d'être certain de les avoir tous traité).

3.2 Les solutions envisagées

Pour la création de la représentation abstraite, nous nous sommes muni d'un ensemble de jeux de règles de plusieurs types de pare-feux bien différents (*iptables*, *ipfw*, *packetfilter*, …) et nous avons essayé de trouver des points qu'ils avaient tous en commun. Nous sommes assez vite arrivés à leur trouver des parties communes, l'idée étant d'arriver à synthétiser les informations dans une structure qui soit aussi adaptée pour l'analyse. Les parties ayant posé le plus de problème ont été la gestion des options et des spécificités de chaque pare-feux, il fallait s'assurer de pouvoir gérer tous les types d'options des pare-feux bien que ceux ci aient tout un jeu d'options bien différentes, le tout étant de garder une structure claire et pratique pour l'analyse.

La seconde partie importante de l'analyseur était la détection d'incohérences. Suite à un peu de réflexion personelle, nous étions arrivé à exhiber et à traiter trois types d'incohérences, une de type conflit et deux de type redondance. Pour y arriver nous avons du implémenter des algorithmes capables de détecter l'inclusion d'un *sous-réseau* dans un autre *sous-réseau* (en se basant sur leurs adresses respectives). Par la suite, nos encadrants nous ont suggéré de lire une publication[1] explicitant un nombre supplémentaire d'incohérences et prouvant leur unicité.

1 « Conflict Classification and Analysis of Distribued Firewall Policies », IEEE Journal, Vol. 23, No. 10, October 2005

Nous nous sommes appuyés sur cette publication pour permettre à notre analyseur de détecter des nouveaux types d'incohérences auxquels nous n'avions pas pensés (de type généralisation et corrélation).

3.3 Les solutions mises en place

3.3.1 La représentation abstraite

Pour obtenir une représentation abstraite cohérente, nous avons du analyser différents types de pare-feux et établir un certain nombre de point communs. Tous ces pare-feux étant géré à l'aide d'un ensemble de règles ordonnées, qui permettent d'appliquer un type de filtrage sur un trafic particulié, il nous a semblé logique de faire apparaître un type **Règle** (*Rule*) dans notre structure abstraite. Il nous a ensuite fallu identifier les parties communes entre les règles des différents types de pare-feux.

En ce qui concerne les règles, sommes arrivé à isoler les différentes parties suivantes (la structure complète de la grammaire abstraite est présente en *Annexe1*) :

- **Action** (*Action*): chacun des pare-feux permettait d'effectuer un type d'action donné (Accépter, Rejeter, Archiver) pour la règle.
- **Interface** (*Iface*): tous les pare-feux permettaient de spécifier sur quelle interface réseau il fallait appliquer la règle de filtrage.
- **Protocole** (*Protocol*): tous les pare-feux permettent de donner un protocole à cibler pour le filtrage (le type de protocole diffère en fonction des pare-feux, nous n'avons mis que les protocoles les plus communs).
- **Cible** (*Target*): chaque pare-feu permettait de préciser sur quel type de trafic appliquer le filtrage (Entrant, Sortant, Retransmis)
- **Addresse** (*Address*): chacun des pare-feux permettait de spécifier une adresse (ou un sous-réseau) *source* et et un de *destination* sur lequel appliquer le filtrage. Une adresse étant composée d'un nombre (correspondant à l'adresse entière Ipv4 ou Ipv6) et d'un masque de sous-réseau.
- **Port** (*Port*): tous les pare-feux permettaient de spécifier un port (ou un groupe de ports) de *source* et de *destination* spécifique.

Pour ce qui est des types de paramètres de filtrages non communs à tous les pare-feux (la gestion de l'état d'une connexion par exemple), nous les avons regroupé dans la structure **Options** (*Options*).

Une politique de filtrage donnée était donc un ensemble de **Règles** (*Rules*) pour lesquelles nous conservions l'ordre.

3.3.2 L'analyseur

Le but de l'analyseur étant de reconnaître et de traiter les incohérences possibles entre les règles de la grammaire abstraite, nous avons du effectuer quelques recherches sur les types d'incohérenecs existant. Le plus grand problème a été de s'assurer que nous prenions bien en compte tous les types d'incohérences.

Première version de l'analyseur

Dans une première partie, nous avions trouvé deux types d'incohérences, des incohérences de types *redondance* et d'autres de type *conflit*.

- Les incohérences de type *conflit* étaient détectées si deux règles possédant des actions différentes étaient appliquées sur le même trafic (le filtrage est effectif sur les mêmes adresses de provenance et de destination). Nous nous étions dit que ce type d'incohérences pouvait seulement subvenir sur des règles s'appliquant sur mêmes adresses (adresses/sous-réseaux identiques). Les deux règles « Accepter 192.168.0.1 → 192.168.0.2 » et « Refuser 192.168.0.1 → 192.168.0.2 » étaient par exemple détectées comme conflictuelles.
- Les incohérences de type *redondance* étaient détectées lorsque une action identique était appliquée sur deux adresses identiques ou deux sous-réseaux imbriqués.[Nous noterons dans la suite de nos exemple les adresses en notation *CIDR* pour plus de lisibilité].Deux sous-réseaux sont dit imbriqués si l'intersection entre les groupes d'adresses leur appartenant est non vide, par exemple, le sous-réseau 192.168.0.0/24 est imbriqué (inclus) dans le sous-réseau 192.168.0.0/16. C'est aussi le cas pour un sous réseau et une adresse particulière, par exemple, 192.168.0.8 est incluse dans le sous-réseau 192.168.0.0/24 (192.168.0.8 peut aussi s'écrire 192.168.0.0/32). Ainsi « Accepter 192.168.0.2 → 192.168.4.0/24 » était détectée comme redondante à la règle « Accepter 192.168.0.0/24 → 192.168.4.0/24 ».

Le principal problème de cette première version étant que nous avions oublié de considérer l'ordre de priorité entre les règles.

Seconde version de l'analyseur

Après nous être rendu compte de l'oubli de prise en charge de la priorité entre les règles et après avoir présenté cette première version à nos encadrants, ceux-ci nous ont proposé de lire une publication[2] exhibant tous les types possibles d'incohérences entre les règles d'un pare-feux. La lecture de cette publication nous a permis de corriger la première version tout en ajoutant la détection de nouvaux types d'incohérences. Les incohérences exhibés dans cette publication sont de quatre types, celles de type *obscursissantes* (shadowing), de type *redondantes*, de type *généralisation* et de type *corrélation*. La publication démontre aussi leur unicité.

- Les incohérences de type *obscursissantes* s'appliquent aux règles qui ne pourront jamais être atteinte suite à un problème de priorité (une partie de l'ancienne incohérence de type *conflit*). C'est le cas si un sous-réseau est inclus dans un autre et que leur action de filtrage est différente, la règle du second sous-réseau inclus n'est jamais atteinte. Supposons deux règles $R_1 =$ « Accepter le trafic vers 192.168.0.0/16 » et $R_2 =$ « Refuser le trafic de « 192.168.0.0/24 » avec $Priorité(R_1) > Priorité(R_2)$. Dans ce cas R_2 ne pourra jamais être atteinte car le par-feu appliquera systématiquement la règle R_1 qui est plus prioritaire. C'est aussi le cas pour deux politiques de filtrage (actions) différentes sur les mêmes adresses. Ainsi si la règle « Accepter le trafic vers 10.0.0.1 » est moins prioritaire que la règle « Refuser le trafic vers 10.0.0.1 », elle ne sera jamais atteinte.

- Les incohérences de type *redondantes* sont les mêmes que celles que nous avions exhibé dans la première version de l'analyseur. La règle redondante est systématiquement supprimée par l'analyseur.

- Les incohérences de type *généralisation* se produisent lorsque tout le trafic filtré par une règle est aussi filtré par une règle de priorité moins haute. Ce type d'incohérence à une implication sur la politique de filtrage de laquelle l'utilisateur ne se rend souvent pas compte. Soit deux règles $R_1 =$ « Refuser le trafic vers 192.168.2.0/24 » et $R_2 =$ « Accepter le trafic de « 192.168.0.0/16 » avec $Priorité(R_1) > Priorité(R_2)$. L'implication de ces deux règles est que le trafic n'est accepté que sur les adresses de R_2 moins leur intersection avec celles de R_1 , autrement dit sur les adresses allant de 192.168.0.0 à 192.168.1.255 et celles allant de 192.168.3.0 à 192.168.255.255.

2 « Conflict Classification and Analysis of Distribued Firewall Policies », IEEE Journal, Vol. 23, No. 10, October 2005

- Les incohérences de type *corrélation* sont détectées lorsque deux règles ont des actions de filtrage différentes et que leur domaine de filtrage se chevauchent (pas de problèmes de priorité ici. Par exemple, supposons les règles R_1= « Accepter le trafic 192.168.2.1 → 192.168.4.0/24 » et R_2= « Refuser le trafic 192.168.2.0/24 → 192.168.4.1 ». On remarque que le domaine source de R_1 est inclus par celui de R_2 et que le domaine de destination de R_2 est inclus dans celui de R_1 ce qui empêche ces deux règles de représenter une incohérence de type *redondance*, *obscurcissement* ou *généralisation.* Cependant nous sommes toujours ici dans le cas d'une incohérence.

L'incohérence *obscursissantes* est considérée comme une erreur grave, tandis que les trois autres ne sont considérées que comme des erreur bénines et donc nous n'affichons qu'un avertissement lors de la détection de ce type d'incohérences à l'analyse.

3.4 Exemples d'analyse

3.4.1 Exemple de conflit

Voici un exemple de l'execution de l'analyseur sur le groupe de règles suivant où les règles « Accépter le trafic de 192.168.0.0/24 » et « Refuser le trafic de 192.168.0.0/24 » sont en conflit:

```
Chain INPUT (policy DROP)
target     prot opt source               destination
ACCEPT      tcp  --  anywhere             192.168.0.0/24
DROP             tcp  --  anywhere             192.168.0.0/24
```

Résultat de l'execution :

```
error[shadowing]:
---
'ACCEPT     tcp  --  anywhere             192.168.0.0/24'
      <in conflict with>
'DROP       tcp  --  anywhere             192.168.0.0/24'
---
[1] ACCEPT     tcp  --  anywhere             192.168.0.0/24
[2] DROP       tcp  --  anywhere             192.168.0.0/24
[0] None
Which rule do you want to delete [0/1/2] ?
```

L'analyseur a bien détecté le conflit et comme il ne sais pas quelle décision prendre, un dialogue permet à l'utilisateur de choisir laquelle des deux règles supprimer.

3.4.2 Exemple de redondance

Un autre exemple, de redondance cette fois, la règle « Accepter le trafic de 192.168.0.0/24 » travaille sur un sous réseau déjà traité par la règle « Accepter le trafic de 192.168.0.0/16 »:

```
Chain INPUT (policy DROP)
target     prot opt source               destination
ACCEPT      tcp  --  anywhere                   192.168.0.0/24
ACCEPT      tcp  --  anywhere             192.168.0.0/16
```

Résultat de l'execution:

```
warning[redundancy]
---
'ACCEPT     tcp  --  anywhere       192.168.0.0/24'
      <included in>
'ACCEPT     tcp  --  anywhere       192.168.0.0/16'
---
```

Ici l'analyseur est en mesure de prendre une décision et il supprime « Accepter le trafic de 192.168.0.0/24 » qui est redondante.

3.4.3 Exemple de conflit lié à la priorité des règles

Un dernier exemple, qui concerne deux règles ayant un conflit lié à leur priorité. Supposons deux règles R_1= « Accepter le trafic vers 10.0.0.0/16 » et R_2= « Refuser le trafic de « 10.0.0.0/24 » avec $Priorité(R_1) > Priorité(R_2)$. C'est une erreur de type *obscurcissement* (shadowing),en effet, R_2 ne pourra jamais être atteinte car le par-feu appliquera systématiquement la règle R_1 qui est plus prioritaire.

```
Chain INPUT (policy DROP)
target     prot opt source               destination
ACCEPT      tcp  --  anywhere        10.0.0.0/16
DROP             tcp  --  anywhere        10.0.0.0/24
```

Résultat de l'execution:

```
error[shadowing]:
---
'ACCEPT     tcp  --  anywhere       10.0.0.0/16'
      <shadows>
'DROP       tcp  --  anywhere       10.0.0.0/24'
---
[1] ACCEPT     tcp  --  anywhere          10.0.0.0/16
[2] DROP       tcp  --  anywhere          10.0.0.0/24
[0] None
Which rule do you want to delete [0/1/2] ?
```

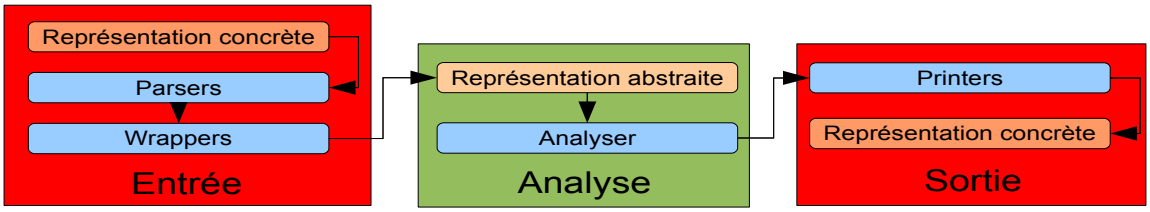
L'analyseur détecte deux types d'incohérences, dont une conflictuelle pour laquelle il demande à l'utilisateur la décision à prendre.

3.5 Les tests effectués

Dans la publication figuraient quelque tests caractéristique des différents types d'incohérences reconnus par l'analyseur. Nous avons lancé ces tests et ils fonctionnent.

Nous avons également testé l'analyseur sur des fichiers un peu plus conséquents (une trentaine de règles), ceux-ci se sont déroulé sans problèmes.

4 L'outil de migration



4.1 Introduction au fonctionnement

L'outil de migration comporte principalement deux parties. Une première partie a pour rôle de convertir un ensemble de règles dans un langage d'entrée (*iptables*, *Ipfirewall*, *Packet Filter*) en une structure abstraite reconnue par l'analyseur pour permettre l'analyse. La seconde partie a pour rôle de convertir les règles en sortie de l'analyseur (structure abstraite) dans un langage de sortie, qui peut être différent de celui d'entrée.

Un des principaux enjeux a été de créer une interface de traduction qui soit compréhensible, lisible et maintenable pour permettre facilement la prise en charge de nouveaux langages. Il fallait donc que les étapes de traduction soit séparées de façon claire pour faciliter la compréhension et éventuellement la correction de bogues.

4.2 Les solutions mises en place

Nous avons choisi de départager les deux phases de traduction (*entrée* → analyse → *sortie*) en deux parties pour la phase de traduction de l'entrée et une partie pour la traduction vers le langage de sortie. Les deux parties pouvant être réalisées de façon indépendante (on peut prendre en langage en charge seulement en entrée ou en sortie).

En ce qui concerne la traduction du langage d'entrée, nous avons choisi de traduire tout d'abord les règles concrète dans une représentation intermédiaire (pour rendre l'analyse syntaxique plus claire et plus pratique). La seconde partie se charge de convertir la représentation intermédiaire en représentation abstraite reconnue par l'analyseur. La phase de conversion comme celle de traduction étant deux phases importante et conséquentes, il nous a semblé indispensable de les séparer pour plus de lisibilité.

La partie de traduction vers le langage de sortie pouvant s'effectuer directement à partir de la représentation abstraite, nous n'avons pas jugé utile de la séparer en plusieurs phases. Il nous suffit pour cette partie de parcourir la représentation abstraite en sortie de l'analyseur et de traduire en conséquent.

En procédant de cette façon, le mécanisme de traduction de sortie reste lisible et maintenable.

Pour conserver une certaine harmonie dans le code, nous avons adopté une convention de nommage pour les fichiers relatifs à l'outil de migration (*Nom.gom*, *NomParser.g.t*, *NomWrapper.t*, *NomPrinter.t*)

4.2.1 Traduction de l'entrée (Parsers)

Pour réaliser la traduction des langages d'entrées, un de nos encadrants nous a présenté l'outil *Antlr* qui permet d'utiliser une grammaire pour reconnaître les langage d'entrée et les convertir dans une structure intermédiaire. Nous avons du établir une grammaire pour chaque langage d'entrée. Cette traduction est réalisée par les classes de type « Parser » (un exemple de grammaire est disponible en *Annexe2*).

Chaque pare-feux permettant d'effectuer des actions très précise en ce qui concerne le filtrage, nous n'avons bien entendu pas eu le temps de créer des grammaires exhaustives pour chaque langage d'entrée. Nous nous sommes contentés de gérer les opérations les plus importantes de chaque type de pare-feu. Le langage permettant de gérer les grammaires étant assez facile à prendre en main, il sera toujours possible pour les futurs mainteneurs du programme de compléter ces grammaires.

Nous avons créer une interface *Java* (*Annexe3*) permettant de contraindre les futurs mainteneurs à utiliser un certain formalisme, cela permet de conserver une certaine harmonie dans le code.

4.2.2 Représentation intermédiaire et conversion (Wrappeurs)

Représentation intermédiaire

Nous avons du pour chaque langage d'entrée créer une représentation abstraite proche du langage d'entrée (pour faciliter la phase de traduction se basant sur des gramaires) en utilisant l'outil *Gom*. Chacune de ces représentation se devait assez exhaustive pour pouvoir gérer intégralement la grammaire du traducteur d'entrée (voir exemple de représentation en *Annexe4*). Le traducteur d'entrée après l'analyse lexicale et syntaxique du code retourne une structure abstraite de ce type.

Conversion de la représentation intermédiaire en représentation abstraite

La conversion « représentation intermédiaire → représentation abstraite » est faite grâce à l'outil *Tom* et à des méthodes récursives en *Java*. Nous parcourons la représentation intermédiaire et générons la représentation abstraite en conséquent. Cette conversion est réalisée par les classes de type « Wrapper » (voir *Annexe5*). Encore une fois, il y a un convertisseur par langage d'entrée.

Pour être certain que les convertisseurs fonctionnent convenablement, nous avons du nous assurer que l'intégralité de la représentation intermédiaire était prise en charge par charge convertisseur. Encore une fois, pour nous assurer de conserver une certaine harmonie dans le code, nous avons eu recours à une interface *Java* pour obliger les futurs mainteneurs à conserver le formalisme établi (voir *Annexe6*).

4.2.3 Traduction vers la sortie (Printers)

Une fois la représentation abstraite analysée et corrigée, il ne reste plus qu'à traduire (afficher) cette représentation sous forme de règle compréhensibles par le pare-feux désiré. Nous avons pour cela recours à l'outil *Tom* qui nous permet de parcourir la structure abstraite (en *Gom*) et de réaliser les affichages en fonction. Pour chaque langage de sortie, nous avons un traducteur. La traduction est réalisée par les classes de type « Printer » (voir *Annexe7*).

Afin de s'assurer du bon fonctionnement des traducteurs, nous avons du nous assurer que l'intégralité de la représentation abstraite était prise en charge par charge traducteurs. Encore une fois, pour nous assurer de conserver une certaine harmonie dans le code, nous avons eu recours à une interface *Java* pour obliger les futurs mainteneurs à conserver le formalisme établi (voir *Annexe8*).

4.3 Exemples de migrations

Voici un exemple de migration d'*iptables* vers *Packet Filter* pour le fichier d'entrée suivant:

```
Chain INPUT (policy DROP)
target     prot opt source          destination
ACCEPT      all   --  localhost       localhost
ACCEPT      all  --  anywhere         anywhere  state
RELATED,ESTABLISHED
ACCEPT      tcp  --  192.168.0.0/24   anywhere  state NEW

Chain OUTPUT (policy DROP)
target     prot opt source      destination
ACCEPT      all  --  localhost   localhost     tcp dpt:80
ACCEPT      all  --  anywhere    anywhere       state
RELATED,ESTABLISHED
ACCEPT      tcp  --  anywhere    192.168.0.0/24 state NEW
```

Résultat de l'execution :

```
$ java iptables.Main -i iptables-simple -l packetfilter
---
0 Anomalie detected.
Shadowing:        0%
Redundancy:       0%
Generalization:   0%
Correlation:      0%
---
```

Fichier de sortie généré:

```
block drop in all
pass in inet from 127.0.0.1/32 to 127.0.0.1/32
pass in from any to any keep state
pass in inet proto tcp from 192.168.0.0/24 to any keep state
block drop out all
pass out inet from 127.0.0.1/32 to 127.0.0.1/32 port 80
pass out from any to any keep state
pass out inet proto tcp from any to 192.168.0.0/24 keep state
```

On peut constater ici que les règles générées pour *Packet Filter* correspondent bien à celles en entrée (*iptables*).

4.4 Les tests effectués

Nous avons effectué plusieurs tests pour vérifier que la migration était faite de façon correcte.

Le premier test consistait à passer en entrée *un langage d'un certain type* ne contenant aucun problème d'intégrité et nous avons vérifié que la sortie dans même langage nous retournait bien le même résultat (test de l'identité).

Un second test nous a permit de vérifier que le passage en entrée de *deux langage équivalents de types différents* sans problèmes d'intégrité nous donnait bien la même représentation abstraite et donc la même sortie dans un langage particulier.

5 Conclusion

5.1 Les apports

Nous avons appris beaucoup de choses dans le cadre du projet. Dans une première partie, nous avons appris à nous servir de nouveaux outils informatiques très pratiques et efficaces (*Tom*,*Antlr*,*Ant*). Nous avons aussi pu acquérir beaucoup de nouvelles connaissances dans le domaine de la sécurité et des pare-feux. Enfin nous avons découvert une partie du travail des équipes de recherches et avons découvert une facette du domaine de la recherche que nous n'avions jamais envisagé.

5.2 Le projet

Nous sommes très satisfait de l'outil développé durant le projet, il est fonctionnel et efficace. Pour le moment il ne gère que deux ou trois types de pare-feux, mais la structure de base de l'application étant en place, il est relativement aisé d'y rajouter le support de nouveaux types. Nous avons été très heureux d'apprendre que l'outil serait à terme utilisé « en production » et que le développement d'une version graphique était prévue.

Bibliographie

- *Tom's Documentation*, 2009, *http://tom.loria.fr/wiki/index.php5/Documentation*
- Terence Parr, *The definitive ANTLR Reference*, 2005
- Ehab Al-Shaer, Hazem Hamed, Raouf Boutaba, Masum Hasan, *Conflict Classification and Analysis of Distributed Firewall Policies*, IEEE Journal Vol. 23 No. 10, 2005
- Herve Eychenne, *Iptables GNU/Linux manual*, 2009
- *Packet Filter FAQ*, 2009, *http://www.openbsd.org/faq/pf/index.html*
- Joseph J. Barbish, *FreeBSD Handbook*,2009, *http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/firewalls.html*
- *IPFW HOWTO*, 2009, *http://www.freebsd-howto.com/HOWTO/Ipfw-HOWTO*
- *IPFW MaxOSX 13 manual*, 2007
- *Wikipedia Firewall article*, 2009, *http://en.wikipedia.org/wiki/Firewall*
- *Apache Ant user manual*, 2009, http://ant.apache.org/manual/index.html

Glossaire

Iptables: Outil permettant de paramètrer (ajout, modification de règles) le pare-feu de GNU/Linux (Netfilter).

Java: Langage de programmation orienté objet réalisé par la société *Oracle/Sun*.

Tom: Outil développé initialement par Pierre-Etienne Moraux permettant d'effectuer des transformation sur des données de façon très aisée et efficace.

Gom: Outil permettant de créer les structures de données abstraites traitées avec *Tom*

Antlr: Outil permettant de réaliser une analyse lexicale et syntaxique à partir d'une grammaire

Apache Ant: Outil permettant l'automatisation de la construction de projets (*Java* principalement)

Notation CIDR: La notation CIDR d'une adresse IP permet d'abréger la notation classique « Adresse Ip + Masque de sous-réseaux » pour décrire un sous réseau. Par exemple l'adressage du réseau « 192.168.2.0 + masque:255.255.255.0 » s'écrit « 192.168.0.0/24 » en notation CIDR et cible les machines ayant des adresses variant entre 192.168.2.0 à 192.168.2.255.

Annexes

Table des Annexes

Annexe1: Structure de la représentation abstraite (Gom).....23
Annexe2: Fichier de grammaire de traduction de l'entrée « iptables --list » (Antlr):.....25
Annexe3: Interface Wrapper (Java):.....27
Annexe4: Représentation intermédiaire pour l'entrée « iptables --list » (Gom):.....27
Annexe5: Convertisseur représentation intermédiaire → représentation abstraite pour l'entrée « iptables --list » (Tom) :.....28
Annexe6: Interface Wrapper (Java):.....30
Annexe7: Interface Printer (Java):.....30
Annexe8: Traducteur vers la sortie « Packet Filter ».....30

Annexe1: Structure de la représentation abstraite (*Gom*)

```
module iptables.Analyser
imports boolean int long String
abstract syntax

      Rules = Rules(Rule*)
            | RulesL(r:Rule,rs:Rules)
            | RulesR(rs:Rules,r:Rule)
            | RulesA(rsl:Rules,rsr:Rules)

      Rule = Rule(
                  action:Action,
                  iface:Iface,
                  proto:Protocol,
                  target:Target,
                  srcAddr:Address,
                  destAddr:Address,
                  srcPort:Port,
                  destPort:Port,
                  opts:Options,
                  input:String)

      Action =
            Accept()
            | Drop() /* do not process the packet */
            | Reject() /* drop then send an error message to the
sender */
            | Log()

      /* The network interface */
      Iface =
            IfaceAny()
            | Iface(name:String)

      Protocol =
            ProtoAny()
            | TCP()
            | UDP()
            | IPv4()
            | IPv6()
            | ICMP()
            | ARP()
            | RIP()
            | Ethernet()

      Target =
            In()
            | Out()
            | Forward()

      Address =
            AddrAny()
            | Addr4(ip:int,smask:int, str:String)
            /* Only need a subnetmask for the network prefix in
IPv6 */
```

```
            | Addr6(ipnet:long,iphost:long,snetmask:long,
str:String)

      Port =
            PortAny()
            | Port(number:int)


      Options =
            NoOpt()
            | Opt(global:GlobalOptions, proto:ProtocolOptions,
states:States)

      /* Global options */
      GlobalOptions = GlobalOpts(GlobalOption*)

      GlobalOption =
            NoGlobalOpt()
            | LogLevel(level:int)

      /* Protocol options */
      ProtocolOptions = ProtoOpts(ProtocolOption*)

      /* >>> TODO: split by protocol */
      ProtocolOption =
            NoProtoOpt()
            | ICMPxType(id:int)
            | ICMPxTTLZero()
            | ICMPxDestinationUnreachable()
            | TCPxSyn(val:boolean)
            | TCPxAck(val:boolean)

      /* Packet state options */
      States = States(State*)

      State =
            StateAny()
            | New()
            | Related() /* New connection associated with an
existing one */
            | Established() /* packets was seen in both
directions */
            | Invalid() /* packets associated with no known
connection */

module Analyser:rules() {
      RulesL(r@Rule(_,_,_,_,_,_,_,_,_,_),rs@Rules(R*)) ->
Rules(r,R*)
      RulesR(rs@Rules(R*),r@Rule(_,_,_,_,_,_,_,_,_,_)) ->
Rules(r,R*)
      RulesA(rsl@Rules(L*),rsr@Rules(R*)) -> Rules(L*,R*)
}
```

Annexe2: Fichier de grammaire de traduction de l'entrée « iptables --list » (*Antlr*):

```
grammar IptablesListParser;
options {
  output=AST;
  ASTLabelType=Tree;
}

tokens {
      %include
{ iptables/ast/IptablesListParserAstTokenList.txt }
}

@header {
      package iptables;
}
@lexer::header {
      package iptables;
}

file :
      (block)* EOF
            -> ^(FirewallRulesIptablesList ^(IptablesListBlocks
(block)*));

block:'Chain' target '(policy' action ')'
      'target' 'prot' 'opt' 'source' 'destination'
      (rule)* { String str = $rule.text; } -> ^(
            IptablesListBlock target action
                  ^(IptablesListRules (rule)*) STRING[str]
      );

rule: action proto oopt a1=address a2=address opts { String str =
$rule.text; } -> ^(
            IptablesListRule action proto $a1 $a2 opts
STRING[str]
      );

action :
      'ACCEPT'    -> ^(Accept)
      | 'DROP'    -> ^(Drop)
      | 'REJECT'  -> ^(Reject)
      | 'LOG'     -> ^(Log)
      ;

proto :
      'all'       -> ^(ProtoAny)
      | 'tcp' -> ^(TCP)
      | 'udp' -> ^(UDP)
      | 'ip4' -> ^(IPv4)
      | 'ip6' -> ^(IPv6)
      | 'icmp'-> ^(ICMP)
      | 'eth' -> ^(Ethernet)
      ;

target :
```

```
      'INPUT'           -> ^(In)
      | 'OUTPUT'        -> ^(Out)
      | 'FORWARD'       -> ^(Forward)
      ;

address : 'anywhere'    -> ^(AddrAnyRaw)
      | 'localhost'     -> ^(AddrStringDotDecimal4 'localhost')
      | IPV4DOTDEC      -> ^(AddrStringDotDecimal4 IPV4DOTDEC)
      | IPV4CIDR        -> ^(AddrStringCIDR4 IPV4CIDR)
      | IPV6HEX         -> ^(AddrStringHexadecimal6 IPV6HEX)
      | IPV6CIDR        -> ^(AddrStringCIDR6 IPV6CIDR)
      ;

oopt:       '--';

port: 'spt:' INT        -> ^(IptablesListPortSrc INT)
      | 'dpt:' INT      -> ^(IptablesListPortDest INT)
      ;

state : 'NEW'           -> ^(New)
      | 'RELATED'       -> ^(Related)
      | 'ESTABLISHED'   -> ^(Established)
      | 'INVALID'       -> ^(Invalid)
      ;

states:     state stateIter* -> ^(States state stateIter*);

stateIter : ',' state -> ^(state);

opts: opt* -> ^(IptablesListOptions opt*);

opt:  proto port        -> ^(port)
      | 'state' states  -> ^(IptablesListStates states)
      | STRING          -> ^(UnknownOption STRING)
      ;

INT : ('0'..'9')+ ;
INTDOT            : (INT '.');
INTSTARDOT  : ((INT|'*') '.');
IPV4DOTDEC  : (b+=INTSTARDOT)+ { $b.size() <= 3 }? (INT|'*');
IPV4CIDR    : (b+=INTDOT)+ { $b.size() <= 3 }? INT '/' INT;
HEX2COLON   : (('0'..'9') ('0'..'9') ('0'..'9') ('0'..'9') ':');
IPV6HEX           : (b+=HEX2COLON)+ { $b.size() <= 7 }?
(('0'..'9') ('0'..'9') ('0'..'9') ('0'..'9'));
IPV6CIDR    : IPV6HEX '/' INT;

ESC : '\\' ( 'n'| 'r'| 't'| 'b'| 'f'| '"'| '\''| '\\') ;
STRING : '"' (ESC|~('"'|'\\'|'\n'|'\r'))* '"' ;
ID : ('a'..'z'|'A'..'Z'|'_') ('a'..'z'|'A'..'Z'|'_'|'0'..'9')*
('*')?;
WS : (' '|'\t'|'\n')+ { $channel=HIDDEN; } ;

SLCOMMENT : '//' (~('\n'|'\r'))* ('\n'|'\r'('\n')?)?
{ $channel=HIDDEN; } ;
```

Annexe3: Interface Wrapper (*Java*):

```
package iptables;

import iptables.analyser.types.*;
import iptables.firewall.types.*;

public interface Wrapper {
      public Rules wrap(FirewallRules fr);
}
```

Annexe4: Représentation intermédiaire pour l'entrée « iptables --list » (*Gom*):

```
module iptables.IptablesList
imports int String Analyser AddressParser
abstract syntax

IptablesListBlocks = IptablesListBlocks(IptablesListBlock*)

IptablesListBlock = IptablesListBlock(
                  target:Target,
                  action:Action,
                  rules:IptablesListRules,
                  input:String)

IptablesListRules = IptablesListRules(IptablesListRule*)

IptablesListRule = IptablesListRule(
                  action:Action,
                  proto:Protocol,
                  addrsrc: AddressRaw,
                  addrdst: AddressRaw,
                  options: IptablesListOptions,
                  input:String)

IptablesListOptions = IptablesListOptions(IptablesListOption*)

IptablesListOption =
            IptablesListPortSrc(n:int)
            | IptablesListPortDest(n:int)
            | IptablesListStates(s:States)
            | UnknownOption(str:String)
```

Annexe5: Convertisseur représentation intermédiaire → représentation abstraite pour l'entrée « iptables --list » (Tom) :

```
package iptables;

import iptables.iptableslist.types.*;
import iptables.analyser.types.*;
import iptables.addressparser.types.*;
import iptables.firewall.types.*;
import tom.library.sl.*;
import java.util.*;

public class IptablesListWrapper implements Wrapper {
      %include { iptables/firewall/Firewall.tom }
      %include { sl.tom }

      public Rules wrap(FirewallRules fr) {
            %match (fr) {
                  FirewallRulesIptablesList(ilb) -> {
                        return wrapBlocks(`ilb);
                  }
            }
            return null;
      }

      private Rules wrapBlocks(IptablesListBlocks ilbs) {
            %match(ilbs) {
                  IptablesListBlocks(ilb,X*) -> {
                        return `RulesA(
                              wrapBlock(ilb),
                              wrapBlocks(X*));
                  }
            }
            return `Rules();
      }

      private Rules wrapBlock(IptablesListBlock ilb) {
            %match(ilb) {
                  IptablesListBlock(t,a,is,in) -> {
                        return `RulesL(
                              Rule(a,IfaceAny(),ProtoAny(),t,
                                    AddrAny(),AddrAny(),

      PortAny(),PortAny(),NoOpt(),in),
                              wrapRule(is,t)
                        );
                  }
            }
            return `Rules();
      }

      private Rules wrapRule(IptablesListRules ilrs,Target t) {
            %match(ilrs) {
            IptablesListRules(
                  IptablesListRule(
```

```
                        act,
                        p,
                        asrc,
                        adst,
                        o,
                        in),
                  X*
            ) -> {
                  boolean opt = false;

                  Port sport = `PortAny(), dport = `PortAny();

                  Options options = `NoOpt();
                  States states = `States(StateAny());
                  GlobalOptions globalOpt =
`GlobalOpts(NoGlobalOpt());
                  ProtocolOptions protoOpt =
`ProtoOpts(NoProtoOpt());

                  %match(o) {
                        IptablesListOptions(
                              _*,
                              IptablesListPortSrc(ns),
                              _*) -> {
                              sport = `Port(ns);
                        }

                        IptablesListOptions(
                              _*,
                              IptablesListPortDest(nd),
                              _*) -> {
                              dport = `Port(nd);
                        }

      IptablesListOptions(_*,IptablesListStates(s),_*) -> {
                              states = `s;
                              opt = true;
                        }
                  }

                  if (opt)
                        options =
`Opt(globalOpt,protoOpt,states);

                  return `RulesL(Rule(act,IfaceAny(),p,t,
                              AddressParser.addressWrapper(asrc),
                              AddressParser.addressWrapper(adst),
                              sport,dport,options,in),
                        wrapRule(X*,t));
            }
      }
            return `Rules();
      }
}
```

Annexe6: Interface Wrapper (*Java*):

```
package iptables;

import iptables.analyser.types.*;
import iptables.firewall.types.*;

public interface Wrapper {
      public Rules wrap(FirewallRules fr);
}
```

Annexe7: Interface Printer (*Java*):

```
package iptables;

import iptables.analyser.types.*;

public abstract class Printer {
      protected void printOpt(String optname) {
            if (optname.length() > 0)
                  System.out.print(optname + " ");
      }

      protected void print2Opt(String optname, String arg) {
            if ((optname.length() > 0) && (arg.length() > 0))
                  System.out.print(optname + " " + arg + " ");
      }

      protected void printStr(String str) {
            if (str.length() > 0)
                  System.out.print(str);
      }

      protected void printNewLine() { System.out.println(""); }

      protected abstract void prettyPrinter(Rules rs);
}
```

Annexe8: Traducteur vers la sortie « Packet Filter »

```
package iptables;

import iptables.analyser.types.*;
import tom.library.sl.*;
import java.util.*;

public class PacketFilterPrinter extends Printer {
      %include { analyser/Analyser.tom }
      %include { sl.tom }

      private final String
            CMD = "",
            OPT_PROTOCOL = "proto",
            OPT_IFACE = "on",
            OPT_ADDRFAM = "af",
```

```
        OPT_IPSRC = "from",
        OPT_IPDST = "to",
        OPT_PORT = "port",
        OPT_FLAG = "flag",

        ACTION_ACCEPT = "pass",
        ACTION_DROP = "block drop",
        ACTION_REJECT = "block return",
        ACTION_REDIR = "rdr",
        ACTION_LOG = "log",

        DIR_INPUT = "in",
        DIR_OUTPUT = "out",
        DIR_ALL = "all",

        AF_IPV4 = "inet",
        AF_IPV6 = "inet6",

        STATE_KEEP = "keep state",
        STATE_MODULATE = "modulate state",
        STATE_SYNPROXY = "synproxy state",

        ADDR_ANY = "any";

    private String wrapAction(Action a) {
        %match(a) {
            Accept()    -> { return ACTION_ACCEPT; }
            Drop()          -> { return ACTION_DROP; }
            Reject()    -> { return ACTION_REJECT; }
        }
        return "";
    }

    private String wrapTarget(Target t) {
        %match(t) {
            In()        -> { return DIR_INPUT; }
            Out()       -> { return DIR_OUTPUT; }
        }
        return "";
    }

    private String wrapProtocol(Protocol p) {
        String s = OPT_PROTOCOL + " ";
        %match(p) {
            TCP() -> { return s + "tcp"; }
            UDP() -> { return s + "udp"; }
            IPv4()      -> { return s + "ip4"; }
            IPv6()      -> { return s + "ip6"; }
            ICMP()      -> { return s + "icmp"; }
            ARP() -> { return s + "arp"; }
            RIP() -> { return s + "rip"; }
            Ethernet()  -> { return s + "eth"; }
        }
        return "";
    }

    private String wrapAddress(Address a,String opt) {
```

```
            %match(a) {
                  (Addr4|Addr6)[str=str] -> { return opt + " " +
`str; }
                  AddrAny() -> { return opt + " " + ADDR_ANY; }
            }
            return "";
      }

      private String wrapAddressFamily(Address src,Address dest)
{
            %match(src,dest) {
                  Addr4[],Addr4[] -> { return  AF_IPV4; }
                  Addr6[],Addr6[] -> { return  AF_IPV6; }
                  Addr6[],Addr4[] -> { return  AF_IPV6; }
                  Addr4[],Addr6[] -> { return  AF_IPV6; }
                  AddrAny(),Addr6[] -> { return  AF_IPV6; }
                  Addr6[],AddrAny() -> { return  AF_IPV6; }
                  AddrAny(),Addr4[] -> { return  AF_IPV4; }
                  Addr4[],AddrAny() -> { return  AF_IPV4; }
            }
            return "";
      }

      private String wrapPort(Port p) {
            %match(p) {
                  Port(i) -> { return OPT_PORT + " " + `i; }
            }
            return "";
      }

      private String wrapOptGlob(GlobalOptions gopts) {
            %match(gopts) {
                  GlobalOpts(o@!NoGlobalOpt(),X*) -> {
                        String opt = "";
                        %match(o) {
                        }
                        return opt + wrapOptGlob(`X*);
                  }
            }
            return "";
      }

      private String wrapOptProto(ProtocolOptions popts) {
            %match(popts) {
                  ProtoOpts(o@!NoProtoOpt(),X*) -> {
                        String opt = "";
                        %match(o) {
                        }
                        return opt + wrapOptProto(`X*);
                  }
            }
            return "";
      }

      private String wrapOptStates(States states) {
            %match(states) {
                  States(!StateAny(),X*) -> {
```

```
                        return STATE_KEEP + " ";
                  }
            }
            return "";
      }

      private String wrapOptions(Options opts) {
            %match(opts) {
                  Opt(glob,proto,states) -> {
                        return wrapOptGlob(`glob)
                              + wrapOptProto(`proto)
                              + wrapOptStates(`states);
                  }
            }
            return "";
      }

      private void printCmdPolicy(Action a,Target t) {
            printStr( wrapAction(`a) + " " + wrapTarget(`t) + " "
                  + DIR_ALL + "\n");
      }

      public void prettyPrinter(Rules rs) {
            %match(rs) {
                  Rules(Rule(action,IfaceAny(),ProtoAny(),target,

      AddrAny(),AddrAny(),PortAny(),PortAny(),
                              NoOpt(),_),
                        X*
                  ) -> {
                        printCmdPolicy(`action,`target);
                        prettyPrinter(`X*);
                        return;
                  }

      Rules(Rule(action,iface,proto,tar,srcaddr,dstaddr,
                              srcport,dstport,opts,_),
                        X*
                  ) -> {
                        %match(action) {
                              Log() &&
Rules(Rule(action2,iface,proto,

      tar,srcaddr,dstaddr,srcport,dstport,
                              opts,_)) << rs -> {

      printOpt(wrapAction(`action2));
                                    printOpt(wrapTarget(`tar));
                                    printOpt(ACTION_LOG);
                              }
                              !Log() -> {

      printOpt(wrapAction(`action));
                                    printOpt(wrapTarget(`tar));
                              }
                        }
```

```
                        %match(iface) {
                              Iface(name) -> {
                                    print2Opt(OPT_IFACE,`name);
                              }
                        }


      printOpt(wrapAddressFamily(`srcaddr,`dstaddr));
                        printOpt(wrapProtocol(`proto));

      printOpt(wrapAddress(`srcaddr,OPT_IPSRC));
                        printOpt(wrapPort(`srcport));

      printOpt(wrapAddress(`dstaddr,OPT_IPDST));
                        printOpt(wrapPort(`dstport));
                        printOpt(wrapOptions(`opts));
                        printNewLine();
                        prettyPrinter(`X*);
                  }
            }
      }
}
```