Symbolic analysis of network security policies using rewrite systems

Tony Bourdier, Horatiu Cirstea

Symbolic analysis of network security policies using rewrite systems

Tony Bourdier
Inria Nancy & Université Henri Poincaré & Loria – Pareo Team
BP 101, 54602 Villers-lès-Nancy Cedex, France
Tel.: (+33)3.54.95.84.15
Tony.Bourdier@inria.fr

February 2011

Abstract

First designed to enable private networks to be opened up to the outside in a secure way, the growing complexity of organizations make firewalls indispensable to control information flow within a company. Their central role in the security of the organization information make their management a critical task. Firewalls constitute the main component of network security policies, that is why for years many works have focused on checking and analyzing firewalls. The composition of firewalls, taking into account routing rules, has nevertheless often been neglected. In this paper, we propose to specify each component of firewalls, *i.e.* filtering and translation rules, as a rewrite system. We show that such specifications allow to handle usual problems such as comparison, structural analysis and query analysis. We also propose a formal way to describe compositions of firewalls (including routing) in order to constitute a whole network security policy. We show how to extract from rewrite-based specifications of firewalls together with a specification of their composition a rewrite system describing the network traffic under the corresponding security policy. We show that the obtained rewrite system has enough good properties to handle some interesting problems.

Contents

1 Introduction

Security constitutes a crucial concern in modern information systems. Several aspects are involved, such as user authentication (establishing and verifying users' identity), cryptology (changing secrets into unintelligible messages and back to the original secrets after transmission) and security policies (preventing illicit or forbidden accesses from users to information).

Due to the increasing complexity of organizations, network security policies are rarely defined as a single firewall. Most of time, they are made up of numerous firewalls whose composition depends on the network topology and routing rules. That is why it is often difficult to ensure *a priori* that the composition of different local security policies (*i.e.* firewalls) expresses the intended security policy. It is admitted for some years the importance of using formal methods to specify security policies. For example, to achieve high levels of certification (EAL[1] 5, 6, 7), it is necessary to provide a formal specification enabling to obtain mechanized formal proofs, to carry out techniques for test generation, or to perform static analyses ensuring required properties.

Many methods and tools have been developed for analyzing and testing firewall policies. These methods are broken down into two different categories: the active methods and the passive methods. The former consist in sending packets to the network and to make a diagnosis according to the received packets. The main advantage of these methods is that they require no abstract representation of firewalls and thus no error can be introduced between the specification and the implementation. However, such methods have the major drawback of consuming bandwidth, interfering with the traffic and being non exhaustive. That is why we focused on passive methods, that is methods which send no packet and make an offline analysis. Two main categories of passive analysis are investigated in the literature: structural analysis and query analysis. Structural analysis examines the relationships that rules have with other rules within a firewall configuration or across multiple firewalls. A misconfiguration (or conflict) occurs when several rules match the same packet or when a rule can be removed without changing the behavior of the firewall. Query analysis provides a way to ask questions of the form "Which computers in the private network can receive packets from 212.12.30.25 ?". It then consists in defining a language to describe a firewall query and a way to compute its solutions. Some works [CCBGA06, ABR08, BB07, GL04, CCBGA05, Liu08, ASHBH05] looked into structural analysis and others [Haz00, EZ01, LG09] looked into query analysis. Indeed, [CCBGA06, ABR08, BB07, GL04, CCBGA05, Liu08, ASHBH05] focus on defining, detecting and discussing misconfigurations. Some of these works abstract firewall filtering rules as one or two-dimensional ranges of IP, which does not allow to take completely advantage of the obtained results. Others assume that packets are not modified during their network traversal and then do not support network translation address capabilities. Moreover, they often focus on policies based on a single firewall or do not take into account the network topology and routing rules.

In this paper, we introduce a new framework, based on rewrite systems, for specifying security policies. We demonstrate that this approach allows to perform analyses handled by cited works and to go a step further by dealing with network topology and routing rules.

[1] Evaluation Assurance Level

The structure of this paper is as follows. In section 2 we present notions and notations we use throughout this paper. The section 3 is devoted to the presentation of the rewrite-based framework for specifying and analyzing firewalls. In section 4 we address the problem of analyzing compositions of firewalls. Finally, the last section concludes with some perspectives for further work.

2 Technical preliminaries

We suppose the reader to be familiar with notions related to term algebra (terms, substitutions, positions, ...), rewriting systems (reduction relation, confluence, termination,...) [BN98] and tree automata [CDG+08]. In this section we recall some basic notions, present the notations we use throughout this paper and define a new notion (constrained rewrite systems).

Term algebra. We suppose in this paper that there is only one (many-sorted) signature $(\mathcal{F}, \mathcal{S})$. Symbols of $\mathcal{F}$ are denoted by bold characters $\mathbf{f}, \mathbf{g}, \ldots$ and their profile are denoted as follows $\mathbf{f} : \mathsf{s}_1 \times \ldots \times \mathsf{s}_n \to \mathsf{s}$ where $\mathsf{s}_1, \ldots, \mathsf{s}$ are sorts of $\mathcal{S}$ and n is the arity of $\mathbf{f}$. The set of terms of sort s with variables is denoted by $\mathcal{T}_{\mathcal{X}}^{\mathsf{s}}$ and the set of ground terms of sort s is denoted by $\mathcal{T}^{\mathsf{s}}$. For any $t \in \mathcal{T}_{\mathcal{X}} = \cup_{\mathsf{s} \in \mathcal{S}} \mathcal{T}_{\mathcal{X}}^{\mathsf{s}}$, $\mathcal{V}ar(t)$ denotes the variables occurring in t. If any variable of t occurs only once in t, then t is said linear. A position within t is a sequence ω of integers describing the path from the root of t (seen as a finite labeled tree) to the root of the subterm at that position, denoted by $t_{|\omega}$. We use ε for the empty sequence. $|\,\omega\,|$ is the length of the position. $\mathcal{P}os(t)$ denotes the set of positions of t. $t(\omega)$ is the symbol of t at position ω and $t\,[s]_\omega$ the term t with the subterm at position ω replaced by s. A substitution σ is a mapping from $\mathcal{X}$ to $\mathcal{T}_{\mathcal{X}}$ which is the identity except over a finite set of variables (its domain) and which is extended to an endomorphism of $\mathcal{T}_{\mathcal{X}}$. A substitution is said ground if all the variables of its domain are mapped to ground terms. A term t matches a term t' iff $\sigma(t') = t$ for some substitution σ. Two terms t and t' are unifiable iff $\sigma(t') = \sigma(t)$ from some substitution σ.

Relations. Given a binary relation R over $E \times E$, we write $u \twoheadrightarrow_R v$ to denote that $(u, v) \in R$. For any $v \in E$, $R^{-1}(v)$, or equivalently $\twoheadrightarrow_R^{-1}(v)$, is the set $\{u \in E \mid u \twoheadrightarrow_R v\}$. For any $F \subseteq E$, $R^{-1}(F)$ is the set $\{u \in E \mid \exists v \in F, u \twoheadrightarrow_R v\}$. $\twoheadrightarrow_R^*$ is used to denote the reflexive transitive closure of R. A binary relation R is *confluent* iff for any $u, w, v \in E$, if $u \twoheadrightarrow_R^* v$ and $u \twoheadrightarrow_R^* w$, then $v \twoheadrightarrow_R^* t$ and $w \twoheadrightarrow_R^* t$ for some $t \in E$. $u \in E$ is irreducible *w.r.t* R iff there is no v such that $u \twoheadrightarrow_R v$. If $u \twoheadrightarrow_R v$ and v is irreducible *w.r.t* R, then v is a *normal form* of u.

Tree automata. A *tree automaton* is a triple $A = (Q, Q_F, \Delta)$ where Q is a finite set of symbols called *states* disjoint from $\mathcal{F}$, $Q_F \subseteq Q$ is the set of *final states* and Δ is a finite set of transitions of the form $\mathbf{f}(q_1, \ldots, q_n) \twoheadrightarrow q$ where $q_1, \ldots, q_n, q \in Q$ and n is the arity of $\mathbf{f}$. $\twoheadrightarrow_\Delta$ is extended into $\twoheadrightarrow_\Delta^*$ as follows: if $t_i \twoheadrightarrow_\Delta^* q_i$ and $\mathbf{f}(q_1, \ldots, q_n) \twoheadrightarrow_\Delta q$, then $\mathbf{f}(t_1, \ldots, t_n) \twoheadrightarrow_\Delta^* q$. The language recognized by $A = (Q, Q_F, \Delta)$ is $\mathcal{L}(A) = \{t \in \mathcal{T} \mid \exists q \in Q_F, t \twoheadrightarrow_\Delta^* q\}$. A set (or a language) of terms recognized by a tree automaton is said *regular*. A relation R is regular if there exists an automaton recognizing $\{\tilde{t} \mid t \in R\}$ where for any $t = (t_1, \ldots, t_n)$ and

$\omega \in \cup_i \mathcal{P}os(t_i)$, $\tilde{t}(\omega) = (t_1[\omega), \ldots, t_n[\omega))$ where $t_i[\omega) = t_i(\omega)$ if $\omega \in \mathcal{P}os(t_i)$ and Λ otherwise. Boolean operations, Cartesian product, projection and cylindrification preserve regularity.

Rewrite systems. A rewrite rule is a pair of terms $l \twoheadrightarrow r$. The terms l and r are respectively called the *left-hand side* and *right-hand side* of the rule. A rewrite system R is finite set of rewrite rules. Any rewrite system R induces a binary relation over terms denoted by $\twoheadrightarrow_{\mathrm{R}}$ as follows: for any terms t, t', $t \twoheadrightarrow_{\mathrm{R}} t'$ if there exists a rule $l \twoheadrightarrow r$ of R, $\omega \in \mathcal{P}os(t)$ and a substitution σ such that $t_{|\omega} = \sigma(l)$ and $t' = t\,[\sigma(r)]_\omega$. A rewrite rule is linear iff its left-hand side and right-hand side are linear. A rewrite system is linear if all its rules are linear. A growing rewrite system [Jac96] or GRS is a linear rewrite system such that for every rule $l \twoheadrightarrow r$, if $l(\omega) = r(\omega) \in \mathcal{X}$ for some positions ω, ω', then $|\,\omega\,| \leq 1$. An ordered rewrite system is a rewrite system in which rules are ordered. For an ordered rewrite system R, $\twoheadrightarrow_{\mathrm{R}}$ is defined as follows: for any terms t, t', $t \twoheadrightarrow_{\mathrm{R}} t'$ if there exists a rule $l \twoheadrightarrow r$ of R, $\omega \in \mathcal{P}os(t)$ and a substitution σ such that $t_{|\omega} = \sigma(l)$ and $t' = t\,[\sigma(r)]_\omega$ and such that there is no prior rule $l' \twoheadrightarrow r'$ such that $t_{|\omega'} = \sigma'(l')$ for some ω' and σ'.

Definition 1 (CRS). A constrained rewrite system is a rewrite system such that for every rule $l \twoheadrightarrow r$ is associated to a set of membership constraints $x \in A$ where x is in $\mathcal{V}ar(l)$ and A is a regular tree language. If $l \twoheadrightarrow r$ is associated to $\{x_1 \in A_1, \ldots, x_n \in A_n\}$, we write $l \twoheadrightarrow r \;\;\|\;\; x_1 \in A_1, \ldots, x_n \in A_n$.

The binary relation over terms $\twoheadrightarrow_{\mathrm{R}}$ induced by a constrained rewrite system R is such that for any terms t, t', $t \twoheadrightarrow_{\mathrm{R}} t'$ iff there exists a rule $l \twoheadrightarrow r \;\;\|\;\; x_1 \in A_1, \ldots, x_n \in A_n$ of R, $\omega \in \mathcal{P}os(t)$ and a substitution σ such that $t_{|\omega} = \sigma(l)$, $\sigma(x_i) \in A_i$ for every i and $t' = t\,[\sigma(r)]_\omega$.

For any linear (constrained or not) rewrite system R and rule r of R, we denote by $rec(r)$ the regular set of ground terms that are reducible by r and $rec(r/R)$, in the case of ordered rewrite systems, the set of terms that are reducible by r and by no other prior rule.

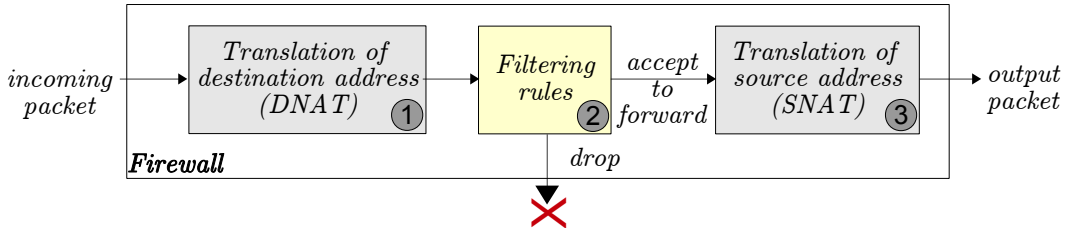
3 Firewall specification and analysis

3.1 Introduction

In a network, when a host wants to transmit a message to another host, data message are encapsulated in a packet. A packet consists of the data that should be transmitted as well as some additional information, called header, used to route it to the appropriate destination. To control packets transmission between different subnetworks, it is common to deploy a network security policy based on a combination of firewalls. A firewall is an application that controls the forwarding of packets which cross it by using a combination of:

- *packet filtering*, which consists in inspecting each packet and either allowing it to continue its traversal or dropping it and
- *network address translation*, which consists in modifying network address information in packet headers.

Firewalls inspect incoming packets and accept or deny to forward them based upon a list of decision rules. These rules map the description of a set of packets to a decision. The most often used criteria [CF02, Rus02] that firewalls use are the packet's source and destination address, its protocol, and, for TCP and UDP traffic, the port number. Moreover, firewalls often offer network address translation (NAT) functionality, which consists in rewriting the source (SNAT) or destination address (DNAT) into another address. The following diagram sums up the behavior of a firewall:



At each step (1, 2 and 3), the packet is compared against a list of rules and the action (translation of destination address, drop or forward and translation of source address) corresponding to the first matched rule is performed.

Example 2. The following figure gives a simple example of firewall:

Filtering:

IP address src	*IP address dest*	*Protocol*	*Port src*	*Port dst*	*Decision*
192.168.20.1/24	121.130.1.1/28	*tcp*	80	*any*	*Accept*
any	*any*	*any*	*any*	*any*	*Drop*

NAT:

Src/Dest	*Address range*	*Port range*	*New address : port*
Dest	192.168.5.128/25	*any*	121.130.1.15:80
Src	192.168.20.1/24	80	121.130.1.1:80

We use the CIDR notation [FL06] to denote subnetworks[2]. Any packet of protocol *tcp* whose source is 192.168.20.1:80 and destination 192.168.5.130:80 (notation address:port) is forwarded by the firewall as a packet whose source is 121.130.1.1:80 and whose destination is 121.130.1.15:80 whereas any packet whose destination is 121.130.1.30:80 is dropped by the firewall.

3.2 Rewrite-based specification of firewalls

In this section, we present a formalization of firewalls based on rewrite systems.

3.2.1 Packets and subnetworks.

In our approach, packets are represented as algebraic terms. For readability reasons, we consider that firewalls inspect only addresses and ports. Other information, such as protocols, tcp flags, states, could be considered without difficulty. The selected symbolic representation of packets is based on the following signature:

$$
\begin{array}{lll}
\mathbf{0},\mathbf{1} & : & \mathsf{Binary} \to \mathsf{Binary} \\
\# & : & \to \mathsf{Binary} \\
\mathbf{ip} & : & \mathsf{Binary} \to \mathsf{IP} \\
\mathbf{port} & : & \mathsf{Binary} \to \mathsf{Port} \\
\mathbf{from} & : & \mathsf{IP} \times \mathsf{Port} \to \mathsf{SrcAddress} \\
\mathbf{dest} & : & \mathsf{IP} \times \mathsf{Port} \to \mathsf{DstAddress} \\
\mathbf{packet} & : & \mathsf{SrcAddress} \times \mathsf{DstAddress} \to \mathsf{Packet}
\end{array}
$$

Let us describe in this section the meaning of the above symbols and defer until section 3.3 the discussion about the consequences of our choices.

IPv4 as well as IPv6 addresses can be equivalently seen as sequences of bits, tuples of hexadecimal numbers or integers. We have thus numerous possibilities to describe addresses as terms. However, we must keep in mind that the basic functionality from which firewalls perform packet inspection is the comparison between addresses and ranges of addresses. From an algebraic point of view, the only one operation we have to compare terms each other is the pattern matching process. Then, testing that an address belong to a given range must ideally correspond to one or several pattern matching steps. Ranges manipulated by firewalls correspond to subnetwork[3] domains. A special feature of subnetwork domains is that all hosts it contains are addressed with a common, identical prefix in their IP address when it is written as a bit vector. This results that a subnetwork is entirely characterized by a bit sequence of variable length. By denoting addresses as words over $\{0,1\}$, or equivalently as terms built from monadic symbols **0** and **1** and a constant #, we obtain that subnetworks are denoted by linear terms built from **0** and **1**. For example, $t = \mathbf{11000000\,10101000\,00010100}(x)$ (we omit parentheses to keep readability) denotes the subnetwork 192.168.20.1/24 whereas $\mathbf{11000000\,10101000\,00010100\,00000001}(\#)$ denotes the IP address 192.168.20.1. For convenience, we will use in this paper the dot-decimal notation for addresses, the decimal notation for ports and the CIDR notation for subnetwork. When a variable occurs in the corresponding term, it will be indicated in brackets. For example t will be denoted by $192.168.20.1/24[x]$. Finally, packets are terms of sort Packet. For example:
$\mathbf{packet}(\mathbf{from}(\mathbf{ip}(\mathit{192.168.1.1}), \mathbf{port}(\mathit{80})), \mathbf{dest}(\mathbf{ip}(\mathit{172.20.3.1}), \mathbf{port}(\mathit{80})))$ and
$\mathbf{packet}(\mathbf{from}(\mathbf{ip}(\mathit{192.168.1.1/24}[x]), y), \mathbf{dest}(\mathbf{ip}(\mathit{172.20.3.1/24}[x']), y'))$ respectively represents a packet and refers to the set of packets whose source address belongs to the subnetwork 192.168.1.1/24 and whose destination address belongs to 172.20.3.1/24.

3.2.2 Firewall rules.

To represent firewall rules, we add to the signature the following symbols:

$$\mathbf{accept}, \mathbf{drop} \quad : \quad \to \mathsf{Decision}$$

From a rewriting point of view, a filtering rule rewrites a packet into **accept** or **drop** whereas a NAT rule rewrites the source or destination address of a packet.

[3] A subnetwork is a logically visible subdivision of a network characterized by an IP ranges (its domain).

Definition 3 (Firewall). A firewall $\mathfrak{f}$ is composed of three ordered rewrite systems $\mathrm{Pre}_\mathfrak{f}, \mathrm{Filter}_\mathfrak{f}$ and $\mathrm{Post}_\mathfrak{f}$ such that:

- rules of $\mathrm{Filter}_\mathfrak{f}$ are of the form $t_{packet} \twoheadrightarrow t_{decision}$ where t_{packet} is a linear term of sort Packet and $t_{decision}$ a term of sort $\mathsf{Decision}$;
- rules of $\mathrm{Pre}_\mathfrak{f}$, resp. $\mathrm{Post}_\mathfrak{f}$, are of the form:

$$\mathbf{dest}(t_{ip}, t_{port}) \twoheadrightarrow \mathbf{dest}(t'_{ip}, t'_{port})$$
$$\textit{resp. } \mathbf{from}(t_{ip}, t_{port}) \twoheadrightarrow \mathbf{from}(t'_{ip}, t'_{port})$$

where t_{ip} and t_{port} are linear whereas t'_{ip} and t'_{port} are ground term.

Example 4. The firewall described in the example 2 can be specified as follows. $\mathrm{Filter}_\mathfrak{f}$ is the following ordered rewrite system:
$\begin{cases} \mathbf{packet}(\mathbf{from}(\mathbf{ip}(\textit{192.168.20.1/24}[x]), \mathbf{port}(\textit{80})), \mathbf{dest}(\mathbf{ip}(\textit{121.130.1.1/28}[y]), z)) \twoheadrightarrow \mathbf{accept} \\ \mathbf{packet}(x, y) \twoheadrightarrow \mathbf{drop} \end{cases}$
while $\mathrm{Pre}_\mathfrak{f}$ and $\mathrm{Post}_\mathfrak{f}$ are respectively made up of the rules:
$\mathbf{dest}(\mathbf{ip}(\textit{162.168.5.128/25}[x]), y) \twoheadrightarrow \mathbf{dest}(\mathbf{ip}(\textit{121.130.1.15}), \mathbf{port}(\textit{80}))$ and
$\mathbf{from}(\mathbf{ip}(\textit{192.168.20.1/24}[x]), \mathbf{port}(\textit{80})) \twoheadrightarrow \mathbf{from}(\mathbf{ip}(\textit{121.130.1.1}), \mathbf{port}(\textit{80}))$.

Definition 5 (Semantics). For any firewall $\mathfrak{f}$, we define the following sets[4] and (functional) relations:

- $[\![\mathfrak{f}]\!]^{\mathrm{accept}} = \left\{ t \in \mathcal{T}^{\mathsf{Packet}} \mid \exists u \in \mathcal{T}^{\mathsf{Packet}}, t \twoheadrightarrow_{\mathrm{Pre}_\mathfrak{f};\{x \twoheadrightarrow x\}} u \twoheadrightarrow_{\mathrm{Filter}_\mathfrak{f}} \mathbf{accept} \right\}$
- $[\![\mathfrak{f}]\!]^{\mathrm{drop}} = \left\{ t \in \mathcal{T}^{\mathsf{Packet}} \mid \exists u \in \mathcal{T}^{\mathsf{Packet}}, t \twoheadrightarrow_{\mathrm{Pre}_\mathfrak{f};\{x \twoheadrightarrow x\}} u \twoheadrightarrow_{\mathrm{Filter}_\mathfrak{f}} \mathbf{drop} \right\}$
- $[\![\mathfrak{f}]\!]^{\mathrm{NAT}} = \left\{ (t, u) \in (\mathcal{T}^{\mathsf{Packet}})^2 \mid \exists v \in \mathcal{T}^{\mathsf{Packet}}, t \twoheadrightarrow_{\mathrm{Pre}_\mathfrak{f};\{x \twoheadrightarrow x\}} v \twoheadrightarrow_{\mathrm{Post}_\mathfrak{f};\{x \twoheadrightarrow x\}} u \right\}$
- $[\![\mathfrak{f}]\!] = \left\{ (t, u) \in [\![\mathfrak{f}]\!]^{\mathrm{NAT}} \mid t \in [\![\mathfrak{f}]\!]^{\mathrm{accept}} \right\} \cup \left([\![\mathfrak{f}]\!]^{\mathrm{drop}} \times \{\mathbf{drop}\} \right)$

From an abstract point of view, a firewall can be seen as a partial or total function which takes as input a packet and returns either another packet (possibly the same) or **drop**. This function is exactly $[\![\mathfrak{f}]\!]$ and is called the semantics of $\mathfrak{f}$.

3.3 Analysis of firewalls

In this section, we show that our formalism allows to automatically show some semantics properties as well as to perform structural and query analysis.

3.3.1 Semantics properties.

If one see a firewall as a decision process which associates to an incoming packet a decision which can be drop or another packet, the following properties are relevant to analyze: *consistency*, which indicates that at most one decision is taken for a given incoming packet, *termination*, which assures that a firewall computes a decision in a finite time and *completeness*, which means that for any incoming packet, the firewall returns a decision. Another important issue is to compare different firewalls.

By construction, every firewall denotes a terminating and consistent decision process. Then, we deal with completeness and comparison. We first give the formal definition of completeness and next define an order to compare firewalls.

Definition 6 (Completeness). We say that a firewall $\mathfrak{f}$ is *complete* iff $[\![\mathfrak{f}]\!]$ is a total function.

In the case of complete firewalls, it can be important to determine if a firewall is less or more permissive than another one. More permissive means allowing at least the same traffic. Such an order is obviously not total.

Definition 7 (Order). We define a partial order over complete firewalls $\preceq$ as follows: for any $\mathfrak{f}$ and $\mathfrak{f}'$, $\mathfrak{f} \preceq \mathfrak{f}'$ ($\mathfrak{f}'$ is more permissive than $\mathfrak{f}$) iff $[\![\mathfrak{f}]\!]^{\text{accept}} \subseteq [\![\mathfrak{f}']\!]^{\text{accept}}$ and for any packet $t_{packet} \in [\![\mathfrak{f}]\!]^{\text{accept}}$, $[\![\mathfrak{f}']\!](t_{packet}) = [\![\mathfrak{f}]\!](t_{packet})$.
We write $\mathfrak{f} \approx \mathfrak{f}'$ iff $\mathfrak{f} \preceq \mathfrak{f}'$ and $\mathfrak{f}' \preceq \mathfrak{f}$. Note that $\mathfrak{f} \approx \mathfrak{f}'$ iff $[\![\mathfrak{f}]\!] = [\![\mathfrak{f}']\!]$.

Now, let us establish the corresponding results:

Proposition 8. Completeness is decidable.

Proposition 9. The order relation $\preceq$ is decidable.

Proof. Both propositions rely on the regularity of sets and relations defined in Definition 5. Indeed, since left-hand sides of all rewrite rules composing a firewall are linear and share no variable with their corresponding right-hand sides, we can easily show that $\twoheadrightarrow_{\text{Pre}_\mathfrak{f}}$ and $\twoheadrightarrow_{\text{Post}_\mathfrak{f}}$ are regular tree relations (some technical manipulations are needed to take into account the order). It follows that $\twoheadrightarrow_{\text{Pre}_\mathfrak{f};\{x \twoheadrightarrow x\}}$ and $\twoheadrightarrow_{\text{Post}_\mathfrak{f};\{x \twoheadrightarrow x\}}$ are also regular. We can also show that $[\![\mathfrak{f}]\!]^{\text{accept}}$ and $[\![\mathfrak{f}]\!]^{\text{drop}}$ are regular tree sets. By composition and restriction, we finally obtain that $[\![\mathfrak{f}]\!]^{\text{NAT}}$ and $[\![\mathfrak{f}]\!]$ are regular tree (functional) relations. The completeness can be tested by checking that $[\![\mathfrak{f}]\!]^{\text{accept}} \cup [\![\mathfrak{f}]\!]^{\text{drop}}$ covers the (regular) set of all possible incoming packets. As for the order, it suffices to test the inclusion $[\![\mathfrak{f}]\!]^{\text{accept}} \subseteq [\![\mathfrak{f}']\!]^{\text{accept}}$ together with an equivalence between the restrictions of $[\![\mathfrak{f}]\!]$ and $[\![\mathfrak{f}']\!]$ to $[\![\mathfrak{f}]\!]^{\text{accept}}$ (which are also regular). □

3.3.2 Structural analysis.

Structural analysis refers to the detection of misconfigurations (or anomalies) in firewalls rules. A complete survey of misconfigurations can be found in [CCBGA06, HAS06]. Examples of anomalies are shadowing, redundancy, correlation, exception,... Such misconfigurations are properties expressed as relationships that rules have with other rules. It would be tedious to show that all these anomalies can be detected using our framework. That is why we only discuss about an example of misconfigurations (shadowing) and claim that other anomalies can be treated in the same way. Let us first recall the definition of the shadowing anomaly: we say that a firewall has *shadowing* iff it contains at least one filtering rule such that all packets it accepts (resp. drops) are dropped (resp. accepted) by a prior rule. In such a case, the concerned rule is said to be *shadowed*. Detection of shadowed rules, as well as the other misconfigurations, is based on the following feature. Each rule r is associated to several sets: $rec(r)$, denoting the set of packets matching r; $rec(r/\text{Filter}_\mathfrak{f})$,

denoting the set of packets matching r which do not match any prior rule of $\text{Filter}_\mathfrak{f}$ (*i.e.* $rec(r) \setminus \bigcup_{r'<r} rec(r')$) and $rec(r/\text{Filter}_\mathfrak{f}[d])$ for any decision d, which denotes the set of packets matching r and that match no other prior rule of $\text{Filter}_\mathfrak{f}$ associated to the decision d. Misconfigurations can be detected using inclusion or emptiness tests. In the case of shadowing, to detect if a rule r is shadowed, it suffices to check the emptiness of $rec(r/\text{Filter}_\mathfrak{f}[\textbf{accept}])$ if r is mapped to **drop** and $rec(r/\text{Filter}_\mathfrak{f}[\textbf{drop}])$ otherwise. Actually, misconfigurations can be detecting using a combination of operations which preserve regularity starting from regular sets ($rec(r)$). Thereby, the real question which arises from structural analysis in our case is to know if the selected symbolic representation of packets provides efficient algorithms to perform these operations. Indeed, it is well-known that operations over tree automata have high complexity in general. We can remark that in our case the complexity of operations strongly relies on the representation of addresses. Therefore, let us focus our discussion over addresses and their representation. We made the choice of describing addresses as words over $\{0, 1\}$ (or equivalently as terms built from monadic symbols **0** and **1** and a constant #). To simplify explanations, we consider word automata (the correspondence with tree automata is straightforward). A good property of manipulated ranges of addresses is that corresponding minimal and deterministic automata has no loop except at their unique final state which loops over itself for any word. Let us call n-prefix (or simply prefix) language any regular language of the form $\alpha_1.\{0, 1\}^* \cup \ldots \cup \alpha_n.\{0, 1\}^*$. The main advantages of such languages are the following:

- boolean operations preserve the prefix property,
- boolean operations can be performed in $O(n)$ (where n is the number of states of the bigger operand) over the minimal deterministic automata and
- the corresponding algorithms directly produce deterministic and minimal automata (which avoid to perform any determinization).

As said before, the sets of addresses of a given subnetwork are 1-prefix. It follows that $rec(r)$, $rec(r/\text{Filter}_\mathfrak{f})$, ..., are prefix languages. Consequently, misconfigurations can be efficiently detected.

3.3.3 Query analysis.

Another main issue addressed about firewalls is query analysis. Query analysis provides a way to assist firewall administrators in understanding the behavior of a firewall by computing the result of user-defined queries such as "Which hosts in the subnetwork 192.168.1.1/22 can receive packets from a host in the subnetwork 172.20.1.1/24 ?". Such analysis only recently emerged with [LGMN05, LG09]. We have previously shown that the semantics of a firewall is a regular relation. Thus, any query expressed as a first order formula built from:

- variables, ground terms or terms whose head is the symbol **packet** and whose subterms are variables or ground terms ;
- membership atom to a set or a relation defined in Definition 5 and
- membership atom to a linear term (which means being a ground instance of)

can be rewritten into a tree automaton recognizing the set of solutions of the query, that is values of free variables making the formula true.

4 Network security policies

A network security policy is generally deployed by using several firewalls. Even if you trust in each firewall occurring in a network, for example after having performed analyses presented in section 3.3, you should not automatically trust in their composition. Indeed, firewall composition introduces a new security element which disturbs the isolated behavior of firewalls : routing. Routing is often neglected in network security policy analyses. Nevertheless, routing rules can generate major security faults. For example, they can make a packet follow a path during which it will not be properly filtered. They can also create a loop in a path, which can lead to a congestion and even a denial of service. The effects induced by routing and interaction between firewalls make the semantics of the global network security policy hard to understand, particularly in large network with a complex topology. We propose in this section to go a step further in network security policies analysis by taking into account the network topology and the routing rules.

4.1 Specification

In order to analyze a network security policy, one should first specify the network topology. More precisely, one must specify subnetworks, location of security hosts and connectivity between subnetworks and security hosts (security hosts refer to nodes in which firewalls, as logical entities, are deployed). A subnetwork is a logical unit consisting of a set of network hosts which can communicate each other without going through a security host. It often corresponds to a particular section of an organization and is usually represented by a symbolic name and by an IP address range (called domain). Security hosts are interconnection nodes in the network. They can be connected to subnetworks (a security host connected to a subnetwork is its gateway[5]) and to other security hosts. The following picture gives an example of topology:

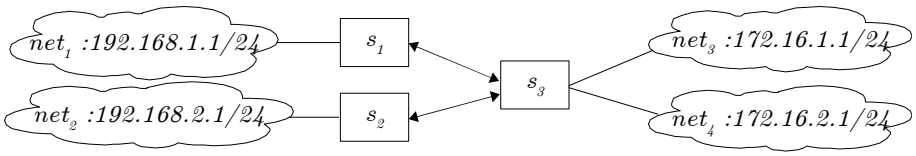


Figure 1: Example of network topology

Definition 10 (Network Topology). A network topology τ is given by:

- a finite set $\mathcal{N}et_\tau$ of subnetwork names ;
- a total map $\mathcal{D}om_\tau : \mathcal{N}et_\tau \rightarrow \mathcal{T}_\mathcal{X}^{\mathsf{IP}}$ which associates any subnetwork name $net \in \mathcal{N}et_\tau$ with a linear term of $\mathcal{T}_\mathcal{X}^{\mathsf{IP}}$ called domain of a ;
- a finite set $\mathcal{SH}_\tau$ of security hosts names ;

[5]Note that without loss of generality, we can consider that any subnetwork has only one gateway. If one wants to describe a topology in which a subnetwork is connected to several security hosts, it suffices to add an intermediate security host.

- a total map $\mathcal{GW}_\tau : \mathcal{N}et_\tau \to \mathcal{SH}_\tau$ which associates any subnetwork name $net \in \mathcal{N}et_\tau$ with a security host called gateway of net such that for any security host s and any distinct $net_1, net_2 \in \mathcal{GW}_\tau^{-1}(s)$, $\mathcal{D}om_\tau(net_1)$ and $\mathcal{D}om_\tau(net_2)$ are not unifiable (that is to say, address ranges of subnetworks connected to the same security host must be disjoint) ;
- a relation $\mathcal{CON}_\tau$ over $\mathcal{SH}_\tau \times \mathcal{SH}_\tau$ describing the connection between the security hosts.

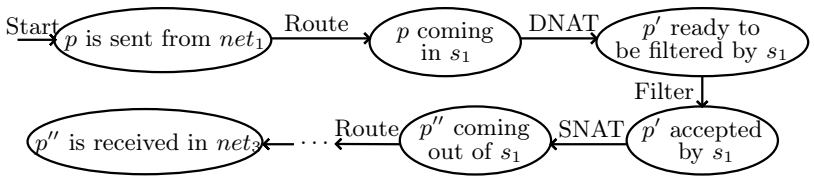
Example 11. The topology depicted in Figure 1 is formally defined as follows:
- $\mathcal{N}et_\tau = \{net_1, net_2, net_3, net_4\}$, $\mathcal{SH}_\tau = \{s_1, s_2, s_3\}$,
- $\mathcal{D}om_\tau(net_1) = \mathbf{ip}(192.168.1.1/24[x])$, $\mathcal{GW}_\tau(net_1) = s_1$,
- $\mathcal{D}om_\tau(net_2) = \mathbf{ip}(192.168.1.2/24[x])$, $\mathcal{GW}_\tau(net_2) = s_2$,
- $\mathcal{D}om_\tau(net_3) = \mathbf{ip}(172.16.1.1/24[x])$, $\mathcal{GW}_\tau(net_3) = s_3$,
- $\mathcal{D}om_\tau(net_4) = \mathbf{ip}(172.16.2.1/24[x])$, $\mathcal{GW}_\tau(net_4) = s_3$,
- and $\mathcal{CON}_\tau = \{(s_1, s_3), (s_3, s_1), (s_2, s_3), (s_3, s_2)\}$.

To entirely define a network security policy, one must associate to any security host a firewall together with routing rules.

Definition 12 (Network security policy). A network security policy $\wp$ over a network topology τ is given by a set of firewalls $\mathcal{F}_\wp = \{\mathfrak{f}_1, \ldots, \mathfrak{f}_n\}$, a total mapping $\mathcal{R}oute_\wp : \mathcal{SH}_\tau \to \mathcal{T}_\mathcal{X}^{IP} \to \mathcal{SH}_\tau$ and total mapping $\mathcal{A}ssign_\wp : \mathcal{SH}_\tau \to \mathcal{F}_\wp$.

$\mathcal{A}ssign_\wp$ indicates the firewall which must be placed into a given security host whereas $\mathcal{R}oute_\wp$ associates to any security host a routing map which indicates the next security host to which a given packet must be forwarded depending on its destination.

The questions we are interested in are the same as for single firewalls. Which packets reach their recipient ? Which are dropped ? However, the analysis become more complicated. Indeed, contrary to the case of a single firewall, there are numerous intermediate steps between the sending and the receipt of a packet. In other terms, a packet can be in different states: sent, received, about to be filtered by a given security host, ... The following picture represents an example of successive states in which a packet p can be:



To analyze traffic under a network security policy, our approach consists in labeling packets to indicate the state in which they are and seeing their state evolution

as a rewriting process. To represent states, we introduce the following symbols:

$$\begin{array}{ll}
\textbf{sent@net} & : \mathsf{SrcAddress} \times \mathsf{DestAddress} \rightarrow \mathsf{State} \\
\textbf{received@net} & : \mathsf{SrcAddress} \times \mathsf{DestAddress} \rightarrow \mathsf{State} \\
\textbf{pre@s} & : \mathsf{SrcAddress} \times \mathsf{DestAddress} \rightarrow \mathsf{State} \\
\textbf{post@s} & : \mathsf{SrcAddress} \times \mathsf{DestAddress} \rightarrow \mathsf{State} \\
\textbf{filter@s} & : \mathsf{SrcAddress} \times \mathsf{DestAddress} \rightarrow \mathsf{State} \\
\textbf{route@s} & : \mathsf{SrcAddress} \times \mathsf{DestAddress} \rightarrow \mathsf{State}
\end{array}$$

for any $net \in \mathcal{N}et_\tau$ and $s \in \mathcal{SH}_\tau$.

As for single firewalls, we can associate to any network security policies $\wp$ a function which takes as input a packet together with the subnetwork from which it is sent and returns drop (if the packet is dropped during its transmission) or the packet under which it is delivered together with the recipient subnetwork. This function is denoted by $[\![\wp]\!]$ and is called the *semantics* of $\wp$. The Figure 2 describes how to build for any network security policy $\wp$ a rewrite system $\mathrm{Flow}_\wp$ computing the successive states in which a given packet is during its traversal of the network.

To make clear the role of each rule scheme, the figure is divided into five categories corresponding to the functionality that the rules simulate. The rule scheme (B) expresses that two hosts which are in a same subnetwork can communicate each other without going through any security host. (R_1) expresses that a packet which is sent from a given subnetwork *net* toward another subnetwork must pass through the gateway of *net*. (R_2) describes the packet forwarding from a security host to the one selected by the routing rules. (R_3) indicates that if no routing rule applies and if the packet destination belong to a subnetwork connected to the current security host, then the packet must be delivered to its recipient. $(DNAT_1)$ and $(SNAT_1)$ describes the address translation process when the packet matches a NAT rule whereas $(DNAT_2)$ and $(SNAT_2)$ expresses that a packet that does not match any NAT rule must be go to the next step. Finally, (F_{drop}) (resp. (F_{accept})) describes that a packet is dropped or forwarded according to the filtering rules of the current security host. Indeed, $\mathrm{Filter}_\mathsf{f}^{-1}(\textbf{drop})$ (resp. $\mathrm{Filter}_\mathsf{f}^{-1}(\textbf{accept})$) is the automaton which recognizes the (regular) set of packets which rewrites to **drop** (resp. **accept**) *w.r.t* the rewrite system $\mathrm{Filter}_\mathsf{f}$.

To conform with the classical idea that a security policy evaluates access requests to decisions, we use, for any network topology τ, the following vocabulary:

- network access requests refer to ground terms of the form:
  $$\textbf{sent@net}(\textbf{from}(t_1, t_2), \textbf{dest}(t_3, t_4))$$
  for some $net \in \mathcal{N}et_\tau$ and ground terms t_1, t_2, t_3 and t_4 such that $t_1 = \sigma(t)$ for $t = \mathcal{D}om_\tau(net)$ and some ground substitution σ and
- decisions refer to either **drop** or ground terms of head **received@net** for some $net \in \mathcal{N}et_\tau$.

We denote by $\mathcal{R}equest_\tau$ the set of network access requests and by $\mathcal{D}ecision_\tau$ the set of decisions (over τ). Thereby, $[\![\wp]\!]$ refers to the relation associating any term of $\mathcal{R}equest_\tau$ with its normal forms *w.r.t.* $\mathrm{Flow}_\wp$ (when they exist).

4.2 Policy properties

In this section, we discuss some crucial policy properties (completeness and consistency) and see how the rewrite-based encoding we proposed allows us to reason about these properties.

Broadcast

(B) $\quad \mathbf{sent@net}(x,y) \twoheadrightarrow \mathbf{received@net}(x,y) \parallel y \in A$
for any $net \in \mathcal{N}et_\tau$ *and* $A = rec(t)$
where $t = \mathbf{dest}(t_{ip}, z)$ *and* $t_{ip} = \mathcal{D}om_\tau(net)$ *and* $z \in \mathcal{X}$

Route

(R_1) $\quad \mathbf{sent@net}(x,y) \twoheadrightarrow \mathbf{pre@s}(x,y) \parallel y \in A$
for any $net \in \mathcal{N}et_\tau$, $s = \mathcal{GW}_\tau(net)$ *and* $A = \overline{rec(t)}$
where $t = \mathbf{dest}(t_{ip}, z)$ *and* $t_{ip} = \mathcal{D}om_\tau(net)$ *and* $z \in \mathcal{X}$,

(R_2) $\quad \mathbf{route@s}(x,y) \twoheadrightarrow \mathbf{pre@s'}(x,y) \parallel y \in A$
for any $(s,s') \in \mathcal{CON}_\tau$, $t \in \mathcal{D}om(\mathcal{R}oute_\zeta(s))$ *such that*
$\mathcal{R}oute_\zeta(s,t) = s'$ *and* $A = rec(\mathbf{dest}(t,z))$ *and* $z \in \mathcal{X}$

(R_3) $\quad \mathbf{route@s}(x,y) \twoheadrightarrow \mathbf{received@net}(x,y) \parallel y \in A$
for any $s \in \mathcal{SH}_\tau$, $net \in \mathcal{N}et$, *such that* $\mathcal{GW}(net) = s$ *and*
$A = rec(t) \setminus rec(\mathcal{D}om(\mathcal{R}oute_\zeta(s)))$
where $t = \mathbf{dest}(t_{ip}, z)$ *and* $t_{ip} = \mathcal{D}om_\tau(net)$ *and* $z \in \mathcal{X}$,

Destination NAT

$(DNAT_1)$ $\quad \mathbf{pre@s}(x,y) \twoheadrightarrow \mathbf{filter@s}(x, destAdd) \parallel y \in A$
for any $s \in \mathcal{SH}_\tau$, $\mathfrak{f} = \mathcal{A}ssign_\zeta(s)$, $r \in \mathrm{Pre}_\mathfrak{f}$,
$A = rec(r/\mathrm{Pre}_\mathfrak{f})$, *and destAdd right-hand side of* r

$(DNAT_2)$ $\quad \mathbf{pre@s}(x,y) \twoheadrightarrow \mathbf{filter@s}(x,y) \parallel y \in A$
for any $s \in \mathcal{SH}_\tau$, $\mathfrak{f} = \mathcal{A}ssign_\zeta(s)$ *and* $A = \overline{\bigcup_{r \in \mathrm{Pre}_\mathfrak{f}} rec(r)}$

Filter

(F_{drop}) $\quad \mathbf{filter@s}(x,y) \twoheadrightarrow \mathbf{drop} \parallel \mathbf{packet}(x,y) \in A$
for any $s \in \mathcal{SH}_\tau$, $\mathfrak{f} = \mathcal{A}ssign_\zeta(s)$ *and* $A = \mathrm{Filter}_\mathfrak{f}^{-1}(\mathbf{drop})$

(F_{accept}) $\quad \mathbf{filter@s}(x,y) \twoheadrightarrow \mathbf{post@s}(x,y) \parallel \mathbf{packet}(x,y) \in A$
for any $s \in \mathcal{SH}_\tau$, $\mathfrak{f} = \mathcal{A}ssign_\zeta(s)$ *and* $A = \mathrm{Filter}_\mathfrak{f}^{-1}(\mathbf{accept})$

Source NAT

$(SNAT_1)$ $\quad \mathbf{post@s}(x,y) \twoheadrightarrow \mathbf{route@s}(srcAdd, y) \parallel y \in A$
for any $s \in \mathcal{SH}_\tau$, $\mathfrak{f} = \mathcal{A}ssign_\zeta(s)$, $r \in \mathrm{Post}_\mathfrak{f}$ *and*
$A = rec(r/\mathrm{Post}_\mathfrak{f})$, *and srcAdd right-hand side of* r

$(SNAT_2)$ $\quad \mathbf{post@s}(x,y) \twoheadrightarrow \mathbf{route@s}(x,y) \parallel y \in A$
for any $s \in \mathcal{SH}_\tau$, $\mathfrak{f} = \mathcal{A}ssign_\zeta(s)$ *and* $A = \overline{\bigcup_{r \in \mathrm{Post}_\mathfrak{f}} rec(r)}$

Figure 2: Rewrite system $\mathrm{Flow}_\wp$ computing the traffic under a network security policy $\wp$

As said in the previous section, completeness is the ability to take a decision for every network access request. In the case of a single firewall, it was sufficient to check that the set of filtering rules covers all possible packets. However, the case of a network security policy is more complicated. Indeed, incompleteness can arise from an incomplete firewall or from the fact that a packet is never dropped nor reaches its destination (in the latter case, we say that the packet is lost).

Definition 13 (Completeness). A network security policy $\wp$ over a network topology τ is *complete* if for any $t_{request} \in \mathcal{R}equest_\tau$, there exists $t_{decision} \in \mathcal{D}ecision_\tau$ such that $[\![\wp]\!](t_{request}) = t_{decision}$.

Notice that a network security policy can be complete even if the firewalls it is made of are not complete and conversely, it can be incomplete even if all the firewalls it contains are complete.

Proposition 14. The completeness of a network security policy $\wp$ is decidable.

More precisely, we have the following property:

Proposition 15. Given a network security policy $\wp$ over a network topology τ, the sets $(\twoheadrightarrow^*_{\mathrm{Flow}_\wp})^{-1}(\mathbf{drop})$ and $(\twoheadrightarrow^*_{\mathrm{Flow}_\wp})^{-1}(\mathcal{D}ecision_\tau \setminus \mathbf{drop})$ are effectively regular.

Proof. The proofs of Propositions 14 and 15 are based on the fact that $\mathrm{Flow}_\wp$ is a constrained growing rewrite system. Jacquemard proposes in [Jac96] a method for computing a tree automaton which recognizes the set $(\twoheadrightarrow^*_{\mathrm{R}})^{-1}(L)$ for any regular tree language L and growing rewrite system R. We can easily show that his result can be extended to constrained growing rewrite systems (see appendix 5). Since $\{\mathbf{drop}\}$ and $\mathcal{D}ecision_\tau \setminus \{\mathbf{drop}\}$ are regular sets, we can build the two tree automata which recognize $(\twoheadrightarrow^*_{\mathrm{Flow}_\wp})^{-1}(\mathbf{drop})$ and $(\twoheadrightarrow^*_{\mathrm{Flow}_\wp})^{-1}(\mathcal{D}ecision_\tau \setminus \mathbf{drop})$. Completeness of a security policy can be verified by checking that the union of these two automata covers $\mathcal{R}equest_\tau$. □

Another crucial security property is consistency. In the context of network security policies, consistency strongly relies on routing rules. Let us recall that routing is the mechanism by which paths are selected in a network to forward packets from their sender to their recipient. Generally, their objective is to forward packets by finding the best path towards its destination, that is the shortest, the most reliable or that balancing best the network load (to avoid congestion for example). In this perspective, routing does not have to alter the delivery of a packet to its recipient. Moreover, most of the time, routing is not static and is dynamically calculated to take into account the situation of the traffic. We understand why it can be dangerous from a security point of view that according to the path it follows in the network, a packet can be or not delivered to its recipient. In other terms, whatever the selected path, any packet could always have the same destiny. Routing thus has to remain a strategy for achieving a goal and does not have to alter the goal. We call consistency the property ensuring that a security policy is not affected by routing.

Definition 16 (Consistency). A network security policy $\wp$ over a network topology τ is *consistent* iff its semantics does not depend on $\mathcal{R}oute_{\wp}$.

Given a network security policy $\wp$, we define $\overline{\wp}$ as the policy such that:
- $\mathcal{F}_{\overline{\wp}} = \mathcal{F}_{\wp}$
- $\mathcal{A}ssign_{\overline{\wp}} = \mathcal{A}ssign_{\wp}$ and
- $\mathcal{R}oute_{\overline{\wp}}(s, x) = s'$ for any $(s, s') \in \mathcal{CON}_{\tau}$ (where $x \in \mathcal{X}$)

Roughly speaking, $\overline{\wp}$ is $\wp$ in which security hosts route any packet to all security host to which it is connected. Consistency of a policy can be expressed as a property over the rewrite system $\text{Flow}_{\overline{\wp}}$ as follows:

Proposition 17. A network security policy $\wp$ over a network topology τ is consistent iff $\twoheadrightarrow^*_{\text{Flow}_{\overline{\wp}}}$ is confluent.

A good advantage of using rewrite systems is the large number of existing tools such as *e.g.* A3PAT [CPU+10], ACP (Automated Confluence Prover) [AYT09], CafeOBJ [DF98], ELAN [VDBMR02], Maude [C+07] or TOM [BBK+07] to execute and reason about them.

To benefit the possibilities opened by these tools, one must represent $\text{Flow}_{\overline{\wp}}$ as an unconstrained rewrite system. For that, we use the fact that automata used in $\text{Flow}_{\overline{\wp}}$ are all based on prefix-automata. Since for any prefix-based automaton A, we can compute a finite set of linear terms $\{t_1, \ldots, t_n\}$ such that $\cup_i rec(t_i) = \mathcal{L}(A)$ and (t_i, t_j) not unifiable for any $i \neq j$, we can transform any rule of $\text{Flow}_{\overline{\wp}}$ to an equivalent finite set of rules which do not overlap. We obtain a finite linear rewrite system containing only trivial overlaps (between rules corresponding to the routing).

5 Conclusion

We have proposed in this paper an original way to describe firewalls using rewrite systems. We have shown that this approach allows us to perform several kinds of analyses, namely single firewall analyses as well as firewall composition, which are generally treated in different way by different frameworks. On one hand, we have shown that rewrite-based specification of firewalls is suitable to determine usual semantics properties (such as completeness or comparison), to detect misconfigurations and that it provides a way to perform query analyses. With regard to most of the existing works, we can handle the network address translation functionality and we perform all analyses in a unique framework. On the other hand, we have shown how to automatically build from rewrite-based firewall specifications together with the description of a topology and routing rules a rewrite system describing the traffic. The obtained rewrite system provides an executable specification of the traffic under the security policy and has in addition good properties which allow us to address some problems such as completeness, reachability and consistency. Compared with previous works, we take into account routing information and topology. Moreover, we do not make traffic analysis for a given path, but for all possible paths induced by routing information. Furthermore, the modeling hypotheses assumed in this paper allow to apply the obtained results in real cases. We currently work on the problem

of minimalizing firewalls, *i.e.* of building from a firewall another equivalent one with a minimum of filtering and translation rules. We are also attached to automatically analyze conformance of network security polices deployed as a firewall composition with a higher level specification of the policy, *e.g.* expressed as a RBAC or OrBAC policy. Finally, we start the development of a tool implementing the present work.

References

[ABR08] T. Abbes, A. Bouhoula, and M. Rusinowitch. An inference system for detecting firewall filtering rules anomalies. In *ACM Symp. on Applied Computing*, pages 2122–2128. ACM, 2008.

[ASHBH05] E Al-Shaer, H. Hamed, R. Boutaba, and M. Hasan. Conflict classification and analysis of distributed firewall policies. In *IEEE Journal on Selected Areas in Communications*, volume 23, pages 2069 – 2084, 2005.

[AYT09] T. Aoto, J. Yoshida, and Y. Toyama. Proving confluence of term rewriting systems automatically. In *Rewriting Techniques and Applications*, pages 93–102. Springer, 2009.

[BB07] A. Benelbahri and A. Bouhoula. Tuple based approach for anomalies detection within firewall filtering rules. In *IEEE Symp. on Computers and Communications*, pages 63–70. IEEE C.S., 2007.

[BBK+07] E. Balland, P. Brauner, R. Kopetz, P.-E. Moreau, and A. Reilles. Tom: Piggybacking rewriting on java. In *Rewriting Techniques and Applications*, LNCS, pages 36–47. Springer, 2007.

[BN98] F. Baader and T. Nipkow. *Term rewriting and all that.* C.U.Press, 1998.

[C+07] Manuel Clavel et al., editors. *All About Maude - A High-Performance Logical Framework*, volume 4350 of *LNCS*. Springer, 2007.

[CCBGA05] F. Cuppens, N. Cuppens-Boulahia, and J. Garcia-Alfaro. Detection and removal of firewall misconfiguration. In *Intl Conf. on Communication, Network and Information Security*. ACTA Press, 2005.

[CCBGA06] F. Cuppens, N. Cuppens-Boulahia, and J. Garcia Alfaro. Detection of network security component misconfiguration by rewriting and correlation. In *Joint Conf. on Security in network ARchitectures and Security of Information Systems*, 2006.

[CDG+08] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on: `http://www.grappa.univ-lille3.fr/tata`, 2008.

[CF02] B. Conoboy and E. Fictner. Ip filter based firewalls howto. Available on: `http://www.obfuscation.org/ipf/ipf-howto.pdf`, 2002.

[CPU+10] É. Contejean, A. Paskevich, X. Urbain, P. Courtieu, O. Pons, and J. Forest. A3PAT, an approach for certified automated termination proofs. In *ACM SIGPLAN Work. on Partial evaluation and program manipulation*, pages 63–72. ACM, 2010.

[DF98] R. Diaconescu and K. Futatsugi. *CafeOBJ Report: The Language, Proof Techniques, and Methodologies for Object-Oriented Algebraic Specification*. World Scientific Pub Co Inc, 1998.

[EZ01] P. Eronen and J. Zitting. An expert system for analyzing firewall rules. In *Nordic Work. on Secure IT Systems*, pages 100–107, 2001.

[FL06] V. Fuller and T. Li. *Classless Inter-Domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan*. The Internet Society, 2006. RFC 4632.

[GL04] M.G. Gouda and A.X. Liu. Firewall design: Consistency, completeness, and compactness. In *IEEE Intl Conf. on Distributed Computing Systems*. IEEE C.S., 2004.

[HAS06] H. Hamed and E. Al-Shaer. Taxonomy of conflicts in network security policies. *IEEE Communications Magazine*, 44(3):134–141, 2006.

[Haz00] S. Hazelhurst. Algorithms for analysing firewall and router access lists. Technical Report TR-WITS-CS-1999-5, University of the Witwatersrand, South Africa, 2000.

[Jac96] F. Jacquemard. Decidable approximations of term rewriting systems. In *Rewriting Techniques and Applications*, pages 362–376. Springer, 1996.

[LG09] Alex X. Liu and Mohamed G. Gouda. Firewall policy queries. *IEEE Transactions on Parallel and Distributed Systems*, 20(6):766–777, 2009.

[LGMN05] A.X. Liu, M.G. Gouda, H.H. Ma, and A.H.H. Ngu. Firewall queries. *Principles of Distributed Systems*, pages 197–212, 2005.

[Liu08] A.X. Liu. Formal verification of firewall policies. In *IEEE Intl Conf. on Communications*, pages 1494 – 1498. IEEE C.S., 2008.

[LTM08] A.X. Liu, E. Torng, and C.R. Meiners. Firewall compressor: An algorithm for minimizing firewall policies. In *Conf. on Computer Communications (INFOCOM)*, pages 176–180. IEEE C.S., 2008.

[Rus02] R. Russell. Linux 2.4 packet filtering howto. Available on: `http://www.netfilter.org/documentation`, 2002.

[SdO08] A. Santana de Oliveira. *Réécriture et modularité pour les politiques de sécurité*. PhD thesis, Université Henri Poincaré, Nancy, 2008.

[VDBMR02] M.G.J. Van Den Brand, P.E. Moreau, and C. Ringeissen. The ELAN Environment: an Rewriting Logic Environment based on ASF+ SDF Technology. In *Work. on Language Descriptions, Tools and Applications*, 2002.

About the representation of packets and subnetworks

Let us recall that a subnetwork is a logically visible subdivision of an IP network. Subnetworks are characterized by the partition of IP addresses into to parts: a "network prefix" and a "host number". More precisely, defining a subnetwork consists in giving a number $n < max$ of bits (where max is 32 for IPv4 and 128 for IPv6) together with a sequence of n bits (characterizing the network prefix). The remaining $n - max$ bits identify the host within the subnetwork. For example, the subnetwork (CIDR notation [FL06]) 192.168.5.64/26 corresponds to the range of IP $[192.168.5.64, 192.168.5.127]$ and is represented by the term $t_S = 11000000\,10101000\,00000101\,01(x)$ (brackets are omitted for readability). A term represents a host of this subnetwork iff there is a ground substitution σ such that $\sigma(t_S) = t$. Moreover, if we denote by $E[n]$ the subset of terms of E of length n, for any boolean operation $\oplus$, $E[n] \oplus F[n] = (E \oplus F)[n]$. Since automaton recognizing E is smaller than the automaton recognizing $E[n]$ when E is a regular set characterized by a finite set of prefixes of length $\leq n$, it is relevant to restrict only *a posteriori* the length of terms of sort Binary.

Furthermore, this representation allows to easily translate any automaton recognizing an IP range to a finite set of subnetworks whose union exactly covers the considered range. In particular, it allow to translate rewrite-based specification of firewalls into a usual (*e.g.* iptable) specifications.

To increase the compacity of automata recognizing IP ranges, we represent, during computations, binary numbers with their "inverse" (in the word theory sense), for example, $1(0(0(\#)))$ is represented by $0(0(1(\#)))$. Thus, the automaton A recognizing ground instances of $1(0(0(x)))$ is replaced by $\tilde{A}$ which recognizes those of $X(0(0(1(\#))))$ where X is a second order variable (called context variable). It is easy to see that boolean operations are stables for the inverse operation $\tilde{}$. This allows to have a direct correspondance between word automata over $\{0, 1\}$ and tree automata over $\mathbf{0}, \mathbf{1}$ and $\#$.

Note that in many works, subnetworks are represented by integer ranges. We believe that such a representation has the following major drawback : one range of IP does not necessarily correspond to one subnetwork and most ranges can not be exactly covered by a finite set of subnetworks. For example, the range $[192.168.1.3; 192.168.1.4]$ does not correspond to any subnetwork. The only one subnetwork of size 2 containing 192.168.1.3 is 192.168.1.3/31 = 192.168.1.2/31 which corresponds to the range $[192.168.1.2; 192.168.1.3]$ and the smallest subnetwork covering $[192.168.1.3; 192.168.1.4]$ is 192.168.1.0/29 corresponding to the range $[192.168.1.0; 192.168.1.7]$. Thus, a specification of a firewall with n rules can correspond to a real firewall with $m >> n$ rules. This leads to the fact that most of algorithms, such as the minimization algorithm [LTM08] can not be applied to real situations.

Extension of Jacquemard's result to CGRS

Without restriction, we can consider that any CGRS is a set of rules of one of the following forms

$$x \twoheadrightarrow r[x] \quad \| \quad x \in A \tag{1}$$
$$f(x_1, \ldots, x_n) \twoheadrightarrow r[x_1, \ldots, x_n] \quad \| \quad x_1 \in A_1, \ldots, x_n \in A_n \tag{2}$$

knowing that any unconstrained variable x can be seen as a variable constrained by $x \in A$ where A is the automaton recognized all ground terms. Thus, we consider that any variable is constrained.

Let be L a regular language recognized by A_L and R a CGRS. The automaton recognizing $(\twoheadrightarrow_{\mathrm{R}}^*)^{-1}(L)$ is built as follows:

$$\mathcal{R}each_0 = (Q, Q_0, \Delta_0) := \biguplus_{(l \twoheadrightarrow r \| C) \in \mathrm{R}} \left(\biguplus_{(x \in A) \in C} A \right) \uplus A_L$$

where the disjoint sum ($\uplus$) of two automata over the same signature is the automaton whose set of states, set of final states and set of rules are the union of corresponding sets of the two automata, provided that they are all disjoint. Then, we transform Δ_k ($k \geq 0$) into Δ_{k+1} by applying the following rules:

$$(i) \ \frac{(f(x_1, \ldots, x_n) \twoheadrightarrow g(r_1, \ldots, r_m) \| \bigwedge_i x_i \in A_i) \in \mathrm{R} \ \ ; \ \ g(q_1, \ldots, q_m) \twoheadrightarrow q \in \Delta_k}{f(q'_1, \ldots, q'_n) \twoheadrightarrow q \in \Delta_{k+1}}$$

with the conditions:

1. for all $1 \leq i \leq n$, q'_i is a final state of A_i
2. for all $1 \leq j \leq m$, there exists a substitution $\theta : \mathcal{X} \twoheadrightarrow Q$ such that $\theta(r_j) \xrightarrow{*}_{\Delta_k} q_j$ and for each x_i occuring in $g(r_1, \ldots, r_m)$, we have $\theta(x_i) = q'_i$.

$$(ii) \ \frac{(f(x_1, \ldots, x_n) \twoheadrightarrow x \| \bigwedge_i x_i \in A_i) \in \mathrm{R} \ \ ; \ \ q \in Q}{f(q'_1, \ldots, q'_n) \twoheadrightarrow q \in \Delta_{k+1}}$$

with the conditions:

1. $x = x_i$ for some $1 \leq i \leq n$
2. for all $1 \leq i \leq n$, if $x_i = x$ then $q'_i = q$, otherwise q'_i is a final state of A_i.

The desired automaton is $\mathcal{R}each = (Q, Q_L, \Delta)$ where Q_L is the set of final states of A_L.