Journal of King Saud University – Computer and Information Sciences (2018) 30, 51–66

King Saud University

Journal of King Saud University – Computer and Information Sciences

www.ksu.edu.sa
www.sciencedirect.com

# A formal basis for the design and analysis of firewall security policies

Ahmed Khoumsi [a,*], Mohammed Erradi [b], Wadie Krombi [b]

[a] Department of Electrical & Computer Engineering, University of Sherbrooke, Canada
[b] ENSIAS, Mohammed V University, Rabat, Morocco

Received 29 May 2016; revised 16 November 2016; accepted 16 November 2016
Available online 30 November 2016

**KEYWORDS**

Firewall security policy;
Automata-based policy;
Completeness verification;
Anomaly detection;
Discrepancy detection;
Mixable policy;
Space and time complexities

**Abstract** A firewall is the core of a well defined network security policy. This paper presents an automata-based method to study firewall security policies. We first propose a procedure that synthesizes an automaton that describes a security policy given as a table of rules. The synthesis procedure is then used to develop procedures to detect: incompleteness, anomalies and discrepancies in security policies. A method is developed to represent the automaton by a policy qualified as mixable and that has practical utilities, such as ease to determine the whitelist and the blacklist of the policy. The developed procedures have been deeply evaluated in terms of time and space complexities. Then, a real case study has been investigated. The obtained results confirm that the developed procedures have reasonable complexities and that their actual execution times are of the order of seconds. Finally, proofs of all results are provided.


## 1. Introduction

The world today is fully connected through the internet, while its security remains a big challenge for the research and industrial communities. Network attacks gain a tremendous attention and constitute daily threats and preoccupations of network managers. Firewalls as crucial network security elements have been widely used as the frontier defense against these attacks.

A firewall today is considered as the core element of any well defined network security policy. A firewall security policy consists of filtering rules that are used to filter incoming and outgoing traffic (packets) from the secured network. A badly designed firewall security policy may lead to the acceptance of malicious packets or the rejection of acceptable packets. Therefore, the correct design and analysis of firewall security policies is an important issue that has been addressed by many researchers, such as Acharya and Gouda (2010, 2011), Al-Shaer and Hamed (2004), Al-Shaer et al. (2009), Bryant (1986), Cuppens et al. (2012), Garcia-Alfaro et al. (2008, 2013), Hoffman and Yoo (2005), Kalam et al. (2003), Kamara et al. (2003), Karoui et al. (2013), Liu et al. (2007, 2008, 2010), Lee and Yannakakis (1996), Lu et al. (2007), Mallouli et al. (2007), Madhuri and Rajesh (2013), Mansmann et al. (2012), Pozo et al. (2012), Wool (2004), and Yuan et al. (2006). Henceforth, the terms *policy* and *rule*

* Corresponding author.
E-mail address: Ahmed.Khoumsi@USherbrooke.ca (A. Khoumsi).
Peer review under responsibility of King Saud University.

ELSEVIER Production and hosting by Elsevier

http://dx.doi.org/10.1016/j.jksuci.2016.11.008

denote "firewall security policy" and "filtering rule", respectively.

In existing works, each proposed formalism addresses a specific firewall design or analysis aspect to resolve a specific problem. This has motivated the present work where an automata-based methodology is developed to address different problems using a *single* formalism. This methodology is based on the construction of an automaton that describes a policy initially specified by a table of rules. This automaton construction is used to detect incompleteness, anomalies and discrepancies in policies. A method is also proposed to represent the automaton by a table of rules called *mixable policy* that has practical utilities. The proposed procedures have been deeply evaluated in terms of time and space complexities; note that our complexity evaluation is more precise than in the literature. The obtained results are presented as formally proved propositions. Also, we discuss the results of a real-life policy use case taken from Chen et al. (2012).

The rest of the paper is structured as follows. Section 2 presents related work. Preliminaries on policies are given in Section 3. In Section 4, we propose a procedure that constructs an automaton which describes a policy initially specified by a table of rules. Sections 5–10 show how the procedure of Section 4 is applied for a rigorous study of policies. In Section 5, we present a method that determines if a policy is complete. Section 6 defines two categories of anomalies, qualified as conflicting and nonconflicting. In Section 7, we propose methods to detect three types of conflicting anomalies: shadowing, generalization and correlation anomalies. Section 8 proposes methods to detect two types of nonconflicting anomalies: LP-redundancy and MP-redundancy. In Section 9, we show how an automaton describing a policy can be represented in a practical form called mixable policy. Section 10 presents a method that detects and resolves discrepancies between several designs of the same policy. In Section 11, we evaluate the performances of the procedures developed throughout Sections 4 to 10 in terms of space and time complexities. Section 12 discusses the application of our methodology in a real case study. We conclude in Section 13 by recalling our contributions and presenting ideas of future studies. Finally, formal proofs of our results are presented in Appendix A.

## 2. Related work and contributions

Previous work on firewalls, such as Hoffman and Yoo (2005), Wool (2004), Kamara et al. (2003) provide practical analysis algorithms, for example to test, analyze configuration and detect vulnerability in policies. Acharya and Gouda (2010, 2011), Liu and Gouda (2008, 2010), Al-Shaer et al. (2009) are more fundamental and provide analysis algorithms with estimations of time complexities. Elmallah and Gouda (2014) show that the analyzes of several problems of firewalls are NP-hard.

Madhuri and Rajesh (2013) define an anomaly in a policy by the existence of at least one packet that matches several rules of the policy. Al-Shaer and Hamed (2004), Karoui et al. (2013) present techniques to detect anomalies in a policy, where a policy is specified by a *Policy tree* in Al-Shaer and Hamed (2004) and a *Decision tree* in Karoui et al. (2013).

Garcia-Alfaro et al. (2013), Cuppens et al. (2012) propose methods to study *stateful* anomalies.

Liu and Gouda (2008) show how to detect discrepancies between several designs of the same policy, where the policy is modeled by a *Firewall Decision Diagram* (FDD) defined in Liu and Gouda (2007).

Yuan et al. (2006) introduce a toolkit *Fireman* which detects several types of errors, for example a violation or inconsistency of a policy. Fireman is implemented by *Binary Decision Diagrams* (BDD) (Bryant, 1986).

Mallouli et al. (2007) propose a framework to generate test sequences to check the conformance of a policy to a specification. The system behavior is described by an extended automaton (Lee and Yannakakis (1996)) and the policy that we wish to apply to this system is described by organization-based access control (OrBAC) (Kalam et al., 2003).

Lu et al. (2007) propose a method to verify if two policies are equivalent.

Mansmann et al. (2012) present a tool to visualize and analyze firewall configurations, where the policy is modeled in a hierarchical way.

Pozo et al. (2012) propose CONFIDDENT, a model-driven design, development and maintenance framework for firewalls.

Garcia-Alfaro et al. (2008) propose mechanisms to detect anomalies in configuration rules of security policies.

In each of the above works, a specific problem is solved using a given formalism: anomalies, discrepancies and violation/inconsistencies are studied using a policy tree (Al-Shaer and Hamed, 2004), a FDD (Liu and Gouda 2007) and a BDD (Bryant, 1986), respectively. This observation motivated the works of Krombi et al. (2014), Khoumsi et al. (2014), where the same model of automata is used to solve various problems of policies. The present paper improves the latter two references and completes them with the following new contributions:

1. We show how our approach can be more interesting than FDD, especially in terms of efficiency for deleting, adding, modifying and switching rules in a policy (see Section 4.5).
2. We propose a method to resolve discrepancies between several implementations of a policy (see Section 10).
3. We show how to construct mixable policies and clarify their utility, in particular to determine the whitelist and the blacklist of a policy (see Section 9).
4. We evaluate space and time complexities to execute the automaton of a policy, and show that such automaton execution is more efficient than executing the table of rules of the policy (see Prop. 18).
5. We formally prove all the results given throughout Sections 4 to 11 (including those already in Krombi et al. (2014), Khoumsi et al. (2014)) (see Appendix A).
6. We illustrate the application of our approach to a real-life policy (see Section 12).
7. We explain much more clearly the reason why our complexity evaluation is more precise than in the literature (Sections 11.2, 11.3 and 11.4).

To have a self-contained paper, we present also the main contributions of Krombi et al. (2014), Khoumsi et al. (2014). Indeed, since this paper is the first one that formally proves the results of these two references, their contributions are presented.

Security enforcement is another relevant issue which can be briefly defined as preventing automatically an untrusted system

from violating a security policy. Sui and Mejri (2013) propose a method to enforce a security policy on an untrusted program. Shen et al. (2013) study security enforcement in the context of cloud storage. Security enforcement is one of our near future work (more details will be given in the list of future work, Section 13).

## 3. Preliminaries

A policy of a firewall is a set of rules specifying actions (e.g. accept, deny) to apply to packets arriving at the firewall. A rule $R$ can therefore be defined by a pair (*Condition*, *Action*) which means: if *Condition* is satisfied for a packet $P$ arriving at the firewall, then *Action* (authorize or refuse the access) must be applied to $P$. Here are more precisions about *Condition* and *Action* of a rule $R$:

- *Condition* is specified by several (say $m$) filtering fields $F^0, \ldots, F^{m-1}$, where each $F^j$ is a set of values. Every packet $P$ that reaches the firewall has several headers $H^0, \ldots, H^{m-1}$, where each $H^j$ is a value. *Condition* is satisfied for $P$ (which is also termed as: $P$ matches $R$), if for every $j = 0, \ldots m-1$: $H^j$ belongs to $F^j$.
- *Action* may be Accept or Deny, which consists in authorizing or preventing $P$ from crossing the firewall.

Table 1 represents the structure of a policy description: each rule $R_i$ is specified by several fields $F_i^0, \ldots, F_i^{m-1}$ representing the condition of $R_i$, and by the action $a_i$. The rules are ordered in decreasing priority, i.e. when a packet $P$ arrives at the firewall, we first verify if $P$ matches $R_1$: if yes, $a_1$ is executed; otherwise (i.e. $P$ does not match $R_1$), we verify if $P$ matches $R_2$. And so on, until we find a rule matched by $P$ or all the rules are verified.

An example of policy is given in Table 2, where the condition of each rule is defined by the fields IPsrc, IPdst, Port and Protocol. The symbol $*$ in the column of $F^j$ means any value of the domain of $F^j$. The expression a.b.c.0/x represents the set of 32-bit values obtained by giving all the possible values to the $32 - x$ last bits in the 32-bit value a.b.c.0. A packet $P$ arriving at the firewall matches a rule $R_i$ if it comes from and is destined to addresses belonging to IPsrc and IPdst respectively, and is transmitted through a port belonging to Port by a protocol belonging to Protocol.

For example, a packet with the headers (190.170.15.10), (82.15.15.11), (25), (UDP), is a packet that comes from 190.170.15.10, is destined to 82.15.15.11, and is transmitted through the port 25 by the protocol UDP. This packet is accepted by the firewall, because Accept is the action of $R_4$, the unique rule matched by this packet.

## 4. Automaton construction

In this section, we present the automaton construction procedure. Given a table of several (say $n$) rules $R_1, \ldots, R_n$ specifying a policy $\mathcal{F}$, the procedure constructs an automaton describing $\mathcal{F}$.

An automaton $\mathcal{A}$ can be represented as a graph of states (represented as nodes) connected by labeled transitions. We consider automata with a single initial state and one or more final states. An automaton accepts all the sequences of transitions starting in its initial state and reaching a final state; the set of these sequences is called the language of the automaton.

The intervals of integers are specified by the usual notation $[a,b]$, $[a,b)$, $(a,b]$ and $(a,b)$. Table 3 defines some notation used in Figs. 1 and 2.

### 4.1. First Step: specify each rule by an automaton

Each rule $R_i$ is described by an automaton $\mathcal{A}_i$ with $m+1$ states $q_i^0, q_i^1, \ldots, q_i^m$, where $q_i^0$ and $q_i^m$ are the initial and final states, respectively. Every $q_i^j$ ($j \neq m$) is connected to $q_i^{j+1}$ by a transition $T_i^j$ labeled by the name of an interval of values of $F_i^j$. $\mathcal{A}_i$

**Table 3** Notations used in Figs. 1 and 2.

| | | | |
|---|---|---|---|
| $\alpha_1 = 190.170.15.0$ | $\beta_1 = 80.15.15.0$ | | |
| $\alpha_2 = 190.170.15.255$ | $\beta_2 = 80.15.15.255$ | | |
| $I_1 = [0, \alpha_1)$ | $I_2 = [\alpha_1, \alpha_2]$ | $I_3 = (\alpha_2, 2^{32})$ | |
| $J_1 = [0, \beta_1)$ | $J_2 = [\beta_1, \beta_2]$ | $J_3 = (\beta_2, 2^{32})$ | |
| $K_1 = [0, 25)$ | $K_2 = (25, 81)$ | $K_3 = (81, 83)$ | $K_4 = (83, 2^{16})$ |

**Table 1** Structure of a policy description.

| Rule | $F^0$ | ... | $F^j$ | ... | $F^{m-1}$ | Action |
|---|---|---|---|---|---|---|
| $R_1$ | $F_1^0$ | ... | $F_1^j$ | ... | $F_1^{m-1}$ | $a_1$ |
| ... | ... | ... | ... | ... | ... | ... |
| $R_n$ | $F_n^0$ | ... | $F_n^j$ | ... | $F_n^{m-1}$ | $a_n$ |

**Table 2** Example of policy.

| Rule | IPsrc | IPdst | Port | Protocol | Action |
|---|---|---|---|---|---|
| $R_1$ | * | 80.15.15.0/24 | * | * | Accept |
| $R_2$ | 190.170.15.0/24 | 80.15.15.0/24 | 25, 81 | TCP | Deny |
| $R_3$ | 190.170.15.0/24 | 80.15.15.0/24 | 25, 83 | * | Accept |
| $R_4$ | * | * | * | * | Accept |

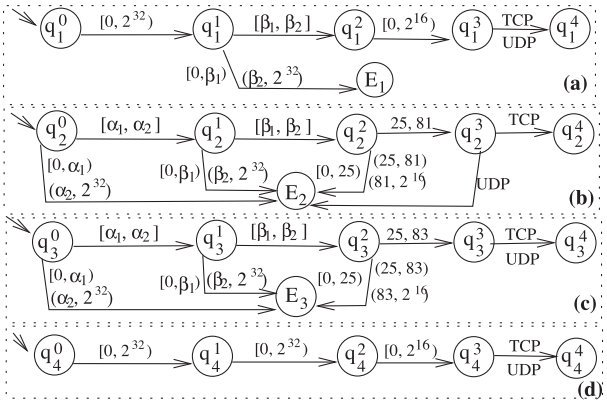


**Figure 1** Step 1: automata obtained from the policy of Table 2.

contains also a state $E_i$ such that each state $q_i^j$ $(j \neq m)$ is connected to $E_i$ by the transitions labeled by the intervals which are complementary to the interval labeling $T_i^j$. Intuitively, the state $q_i^m$ (resp. $E_i$) is reached when a packet matches (resp. does not match) $R_i$.

For example, from the four rules of the policy of Table 2, we obtain respectively the four automata of Fig. 1. IP addresses occupy 32 bits, port numbers occupy 16 bits, and two protocols are considered: UDP and TCP. Singleton intervals are specified by their unique value, like 25, 81, 83.

### 4.2. Second Step: standardize the intervals of the automata

Let the level of a state $q$ be the number of transitions to be executed from the initial state to reach $q$, and the level of a transition be the level of its origin state. The various automata obtained in Step 1 do not use the same alphabet. Consider for example the level 0 transitions of the automata of Fig. 1: $\mathcal{A}_1$ and $\mathcal{A}_4$ use $[0, 2^{32})$, while $\mathcal{A}_2$ and $\mathcal{A}_3$ use $[\alpha_1, \alpha_2]$, $[0, \alpha_1)$ and $(\alpha_2, 2^{32})$.

The objective of Step 2 is to rewrite the transitions of the automata so that they have the same alphabet (this rewriting is useful for Step 3). This is realized by partitioning the domain of each field $F^j$ into a set of disjoint intervals, such that every interval labeling a level $j$ transition of any $\mathcal{A}_i$ is an union of intervals of the partition. For each level $j$, we proceed as follows:

- We construct a sorted list (in increasing order) $L^j$ containing the endpoints of the intervals labeling the level $j$ transitions.For example, for level 0 of the automata of Fig. 1, the obtained sorted list is $L^0 = \langle 0, \alpha_1, \alpha_2, 2^{32} \rangle$.
- We define the intervals whose endpoints are consecutive elements of the sorted list. These intervals form a partition of the domain of the field $F^j$.

For example, from the above sorted list $L^0$, we define the intervals $I_1, I_2$ and $I_3$, which form a partition of $[0, 2^{32})$. (See Table 3 for the notation $I_1, I_2$ and $I_3$.)

- We consider each transition labeled by an interval $I$ and replace it by the transitions labeled by the intervals of the partition that form $I$.For example, the transition labeled $[0, 2^{32})$ is replaced by 3 transitions labeled $I_1, I_2$ and $I_3$, respectively.

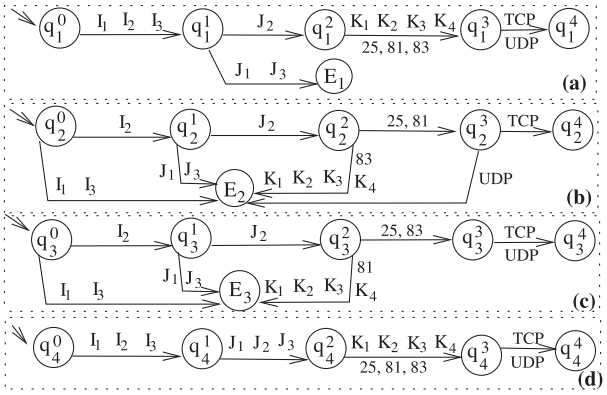


**Figure 2** Step 2: automata obtained by standardizing the intervals of the automata of Fig. 1.

For example, we obtain the automata of Fig. 2 from the automata of Fig. 1.

### 4.3. Third Step: combine the automata

The objective of Step 3 is to construct an automaton by applying a product operator that combines the automata obtained in Step 2. This operator requires that the automata to be combined have the same alphabet, which justifies Step 2. Let us first show inductively how two automata $\mathcal{A}_1^*$ and $\mathcal{A}_2^*$ obtained in Step 2 are combined into an automaton noted $\mathcal{A}_1^* \otimes \mathcal{A}_2^*$:

- Construct the initial state $\langle q_1^0, q_2^0 \rangle$ by combining the initial states $q_1^0$ and $q_2^0$ of $\mathcal{A}_1^*$ and $\mathcal{A}_2^*$.
- For every constructed state $\langle r_1, r_2 \rangle$ of $\mathcal{A}_1^* \otimes \mathcal{A}_2^*$ and every label $\sigma$:
  - –If $\mathcal{A}_1^*$ and $\mathcal{A}_2^*$ have the transitions $r_1 \xrightarrow{\sigma} s_1$ and $r_2 \xrightarrow{\sigma} s_2$ respectively, then construct the state $\langle s_1, s_2 \rangle$ and the transition $\langle r_1, r_2 \rangle \xrightarrow{\sigma} \langle s_1, s_2 \rangle$.
  - –If $\mathcal{A}_1^*$ has the transition $r_1 \xrightarrow{\sigma} s_1$ and $r_2 = E_2$, then construct the state $\langle s_1, E_2 \rangle$ and the transition $\langle r_1, E_2 \rangle \xrightarrow{\sigma} \langle s_1, E_2 \rangle$.
  - –If $\mathcal{A}_2^*$ has the transition $r_2 \xrightarrow{\sigma} s_2$ and $r_1 = E_1$, then construct the state $\langle E_1, s_2 \rangle$ and the transition $\langle E_1, r_2 \rangle \xrightarrow{\sigma} \langle E_1, s_2 \rangle$.

A simple example of product of two automata is given in Fig. 3.

The product of more than two automata is simply obtained by iterating the product of two automata.

We denote by $\Gamma_{\mathcal{F}}$ the product of $\mathcal{A}_1^*, \ldots, \mathcal{A}_n^*$. A state of $\Gamma_{\mathcal{F}}$ is defined by $(r_1, \ldots, r_n)$, where $r_i$ may be $q_i^j$ or $E_i$. Each final state (i.e. without outgoing transition) of $\Gamma_{\mathcal{F}}$ is of one of the following types:

- A *match* state is a state such that $r_i = q_i^m$ for one or several $i$. Let $R_i$ be the rule that has the smallest index $i$ such that $r_i \neq E_i$; the action of $R_i$ is associated to such a match state.
- The *no-match* state is the state $(E_1, E_2, \ldots, E_n)$. No action is associated to it.

For example, if we apply Step 3 to the four automata of Fig. 2, we obtain the automaton $\Gamma_{\mathcal{F}}$ of Fig. 4. In the automata

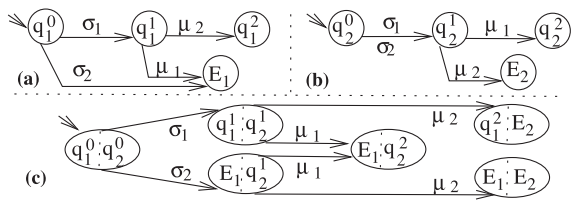

**Figure 3** Product of two simple automata: (c) = product of (a) and (b).

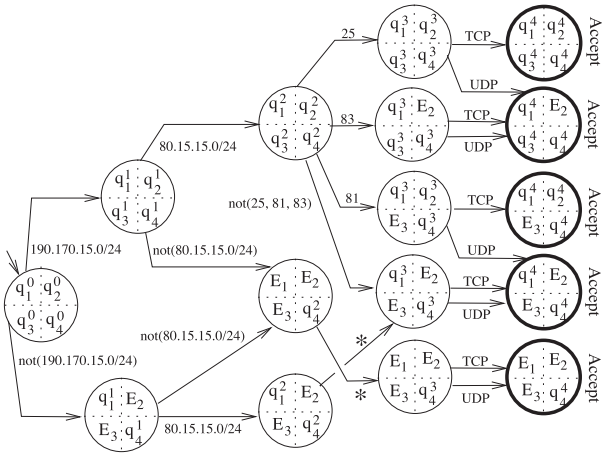


**Figure 4** Step 3: automaton $\Gamma\mathcal{F}$ obtained by combining the automata of Fig. 2.

of Figs. 1 and 2, the interval labeling each transition was explicitly specified to help the reader understand Step 2. In Fig. 4, a more compact notation is used with the symbols $*$ and not$(X)$. The label $*$ in a transition of level $j$ corresponds to the domain of the field $F^j$. For example, the label $*$ in two transitions of level 2 corresponds to the interval $[0, 2^{16})$ (domain of $F^2$), which is in fact the union $K_1 \cup K_2 \cup K_3 \cup K_4 \cup \{25\} \cup \{81\} \cup \{83\}$. If $X$ is a set of values, then not$(X)$ denotes the set $*$ from which we remove $X$. For example, the label not$(25, 81, 83)$ in a transition of level 2 corresponds to $[0, 2^{16})$ without the values $25, 81, 83$, which gives $K_1 \cup K_2 \cup K_3 \cup K_4$.

### 4.4. Automaton executing a policy

The following proposition states that the automaton $\Gamma_{\mathcal{F}}$ executes $\mathcal{F}$, in the sense that when a packet $P$ reaches a firewall whose policy is $\mathcal{F}$, we can determine if $P$ is accepted or rejected by $\mathcal{F}$ by executing $\Gamma_{\mathcal{F}}$ for $P$:

**Proposition 1.** *Consider a firewall of policy $\mathcal{F}$ which is reached by a packet $P$ whose headers are $H^0, \ldots, H^{m-1}$. We start in the initial state of $\Gamma_{\mathcal{F}}$ and execute the sequence of $m$ transitions labeled respectively $\ell_0, \ldots, \ell_{m-1}$ such that each $\ell_j$ contains $H^j$. Let $r = \langle r_1, \ldots, r_n \rangle$ denote the reached state. For each $r_i$, we have $r_i = q_i^m$ iff $P$ matches $\mathrm{R}_i$. If $r$ is a match state (which occurs if $P$ matches one or more rules), the action of the most priority rule matched by $P$ is exactly the action associated to $r$ in $\Gamma_{\mathcal{F}}$.*

Let us for example show how the automaton $\Gamma_{\mathcal{F}}$ of Fig. 4 can be used to execute the policy $\mathcal{F}$ of Table 2. Consider that the firewall is reached by a packet $P$ whose headers $H^0$ to $H^3$ are (190.170.15.12), (80.15.15.20), (23) and (TCP), respectively. We start in $\langle q_1^0, q_2^0, q_3^0, q_4^0 \rangle$. We execute the transition labeled 190.170.15.0/24 (containing $H^0$) and reach $\langle q_1^1, q_2^1, q_3^1, q_4^1 \rangle$. Then, we execute the transition labeled 80.15.15.0/24 (containing $H^1$) and reach $\langle q_1^2, q_2^2, q_3^2, q_4^2 \rangle$. Then, we execute the transition labeled not$(25, 81, 83)$ (containing $H^2$) and reach $\langle q_1^3, E_2, E_3, q_4^3 \rangle$. Finally, we execute the transition labeled TCP (containing $H^3$) and reach $\langle q_1^4, E_2, E_3, q_4^4 \rangle$ associated to Accept, hence $P$ is accepted.

### 4.5. Efficient construction of the automaton

An interesting related work presented by Liu and Gouda (2007, 2008) models and analyzes policies. Our approach has the advantage to store the policy rule information in the states of the automaton. Moreover, with our approach, *deleting*, *adding*, *modifying* and *switching* rules in a policy does not necessitate to reconstruct the automaton from the beginning.

Let us first consider rule *deletion*, i.e. we have to obtain the automaton $\mathcal{A}_{\mathcal{F}\setminus R}$ from $\mathcal{A}_{\mathcal{F}}$. We will illustrate our procedure by the example of Fig. 3, where (a) and (b) are the automata of two rules $R_1$ and $R_2$ and (c) is the automaton of the policy $\mathcal{F}$ consisting of $R_1$ and $R_2$. So, if we remove $R_2$ from $\mathcal{F}$, our procedure should generate (a) from (c). In a first step, we erase the state labels of the automaton of $R$ from the states of the automaton of $\mathcal{F}$. In our example, we erase the state labels of (b) (i.e. $q_2^0, q_2^1, q_2^2, E_2$) from (c) and obtain (1) of Fig. 5. In a second step, we remove the transitions whose origin and destination states are labeled identically. In our example, we remove the outgoing transitions of state $E_1$ in (1) of Fig. 5 and obtain (2) of Fig. 5. In a third step, we minimize the automaton. In our example, we combine the two states $E_1$ in (2) of Fig. 5 and obtain (a) of Fig. 3.

*Adding* $R$ to $\mathcal{F}$ is simple: we construct the automaton of $R$ (as shown in Section 4.1) and then make the product of the automata of $\mathcal{F}$ and $R$ (as shown in Section 4.3).
*Replacing* a rule $R_1$ by a rule $R_2$ is achieved by combining the procedures of deleting $R_1$ and adding $R_2$.
*Switching* two rules $\mathrm{R}_i$ and $\mathrm{R}_j$ consists in switching between $r_i$ and $r_j$ in each state $\langle r_1, r_2, \ldots \rangle$ and then reassigning actions to match states.

Therefore, an advantage of our approach is that deleting, adding, modifying and switching rules do not necessitate to reconstruct the automaton from the beginning.

## 5. Detecting incompleteness

A policy $\mathcal{F}$ is said complete if every packet that could reach the firewall matches one or more rules of $\mathcal{F}$. Otherwise, the policy

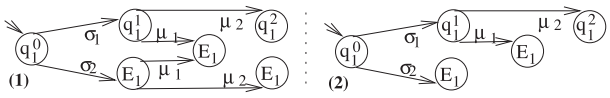


**Figure 5** Illustration of rule deletion.

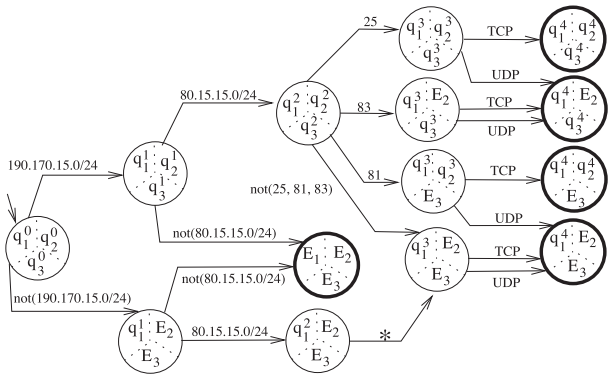


**Figure 6** Automaton $\Gamma_{\mathcal{F}\setminus R_4}$ obtained from the policy $\mathcal{F}$ of Table 2 without $R_4$.

is said incomplete. In other words, a policy is incomplete if it does not define actions to take for some packets. It is therefore desirable that a designed policy is complete, and hence it is relevant to determine if a policy is complete or not. The following proposition allows to determine if a policy is incomplete, it is deduced from Prop. 1:

**Proposition 2.** *A policy $\mathcal{F}$ is incomplete iff its automaton $\Gamma_{\mathcal{F}}$ contains the no-match state.*

For example, the policy of Table 2 is complete because the no-match state is absent from its automaton of Fig. 4. Consider the policy consisting of only the rules $R_1$ to $R_3$ (i.e. without $R_4$), whose automaton $\Gamma_{\mathcal{F}\setminus R_4}$ is represented in Fig. 6. This policy is incomplete because the no-match state is present in its automaton. Concretely, none of the rules $R_1$ to $R_3$ is matched by the packets that are not destined to an address in 80.15.15.0/24.

## 6. Anomaly categorization

By anomaly, we mean the possibility that the same packet matches several rules. As noted in Madhavi and Raghu (2014), several studies have been done to classify anomalies, such as Al-Shaer and Hamed (2004), Yuan et al. (2006), Hu et al. (2012). We have identified two categories of anomalies:

- A *conflicting anomaly* occurs when the same packet matches several rules that have different actions. Conflicting anomalies are studied in Section 7.
- A *nonconflicting anomaly* occurs when the same packet matches several rules that have the same action. In Section 8 we study redundancy anomalies, which are a relevant type of nonconflicting anomalies.

## 7. Detection of conflicting anomalies

We consider three anomalies defined in Al-Shaer and Hamed (2004) and propose methods to detect them.

### 7.1. Detection of shadowing anomalies

Consider two rules $R_i$ and $R_j$ that have different actions (i.e. $a_i \neq a_j$) and such that $R_i$ precedes $R_j$ (i.e. $i < j$). $R_j$ is shadowed by $R_i$ if $R_i$ is matched by all the packets matching $R_j$. Therefore, $R_j$ is never applied. From this definition and Prop. 1, we deduce the following proposition:

**Proposition 3.** *Consider a policy $\mathcal{F}$ and two of its rules $R_i$ and $R_j$. $R_j$ is shadowed by $R_i$ iff: $i < j$; $a_i \neq a_j$; and $\forall$ match state $\langle r_1, \ldots, r_n\rangle$ of $\Gamma_{\mathcal{F}}$, $r_j = q_j^m \Rightarrow r_i = q_i^m$.*

Consider for example the policy of Table 2 and its automaton $\Gamma_{\mathcal{F}}$ of Fig. 4. $R_2$ is shadowed by $R_1$ because: $1 < 2, a_1 = \text{Accept} \neq a_2 = \text{Deny}$, and $\Gamma_{\mathcal{F}}$ has no match state with $q_2^4$ and without $q_1^4$. Therefore, $R_2$ is never applied.

### 7.2. Detection of generalization anomalies

Consider two rules $R_i$ and $R_j$ that have different actions and such that $R_i$ precedes $R_j$. $R_j$ generalizes $R_i$ if $R_j$ matches more packets than $R_i$. From this definition and Prop. 1, we deduce the following proposition:

**Proposition 4.** *Consider a policy $\mathcal{F}$ and two of its rules $R_i$ and $R_j$. $R_j$ generalizes $R_i$ iff: $i < j$; $a_i \neq a_j$; $\forall$ match state $\langle r_1, \ldots, r_n\rangle$ of $\Gamma_{\mathcal{F}}$, $r_i = q_i^m \Rightarrow r_j = q_j^m$; and $\exists$ match state $\langle r_1, \ldots, r_n\rangle$ of $\Gamma_{\mathcal{F}}$ s.t. $r_i = E_i$ and $r_j = q_j^m$.*

Consider for example the policy of Table 2 and its automaton $\Gamma_{\mathcal{F}}$ of Fig. 4. $R_4$ generalizes $R_2$ because: $2 < 4, a_2 = \text{Deny} \neq a_4 = \text{Accept}$, and $\Gamma_{\mathcal{F}}$ has no match state with $q_2^4$ and without $q_4^4$ and has several match states with both $q_4^4$ and $E_2$.

### 7.3. Detection of correlation anomalies

Consider two rules $R_i$ and $R_j$ that have different actions. $R_i$ and $R_j$ correlate if: (1) some packets match $R_i$ but not $R_j$, (2) some packets match $R_j$ but not $R_i$, and (3) some packets match both $R_i$ and $R_j$. From this definition and Prop. 1, we deduce the following proposition:

**Proposition 5.** *Consider a policy $\mathcal{F}$ and two of its rules $R_i$ and $R_j$. $R_i$ and $R_j$ correlate iff: $i \neq j$; $a_i \neq a_j$; $\exists$ match state $\langle r_1, \ldots, r_n\rangle$ of $\Gamma_{\mathcal{F}}$ s.t. $r_i = E_i$ and $r_j = q_j^m$; $\exists$ match state $\langle r_1, \ldots, r_n\rangle$ of $\Gamma_{\mathcal{F}}$ s.t. $r_i = q_i^m$ and $r_j = E_j$; and $\exists$ match state $\langle r_1, \ldots, r_n\rangle$ of $\Gamma_{\mathcal{F}}$ s.t. $r_i = q_i^m$ and $r_j = q_j^m$.*

Consider for example the policy of Table 2 and its automaton $\Gamma_{\mathcal{F}}$ of Fig. 4. $R_2$ and $R_3$ correlate because: $2 \neq 3, a_2 = \text{Deny} \neq a_3 = \text{Accept}$, and $\Gamma_{\mathcal{F}}$ has the following 3 match states which are respectively of the three categories specified in Prop. 5: $\langle q_1^4, E_2, q_3^4, q_4^4\rangle$, $\langle q_1^4, q_2^4, E_3, q_4^4\rangle$, $\langle q_1^4, q_2^4, q_3^4, q_4^4\rangle$.

## 8. Detection of redundancy anomalies

In this section, we study a type of nonconflicting anomalies, referred to as redundancy anomalies as defined in Al-Shaer and Hamed (2004).

### 8.1. Formal definitions of redundancy anomalies

We have determined two types of redundancy between two rules $R_i$ and $R_j$: LP-redundancy and MP-redundancy.

#### 8.1.1. LP-redundancy

It is used to define redundancy of the least priority (LP) rule among two rules. $R_j$ is said LP-redundant to $R_i$ if:

$i < j$; $a_i = a_j$; and $\forall$ packet $P$: $P$ matches $R_j \Rightarrow P$ matches $R_i$.

Intuitively, among two rules with the same action, if the most priority rule matches more packets than the other rule, then removing the latter does not affect the policy.

#### 8.1.2. MP-redundancy

It is used to define redundancy of the most priority (MP) rule among two rules. $R_i$ is said MP-redundant to $R_j$ if:

$i < j$; $a_i = a_j$; $\forall$ packet $P$: if $P$ matches $R_i$, then $P$ matches also $R_j$; $\exists$ packet $P$ s.t. $P$ matches $R_j$ and not $R_i$; and $\forall$ $R_k$ between $R_i$ and $R_j$ *s.t.* $a_k \neq a_i$: $\nexists$ packet matching both $R_i$ and $R_k$.

Intuitively, among two rules with the same action, if the least priority rule matches more packets than the other rule, then removing the latter does not affect the policy if such removal does not allow the application of another rule with a different action.

MP-redundancy can be expressed using the three conflicting anomalies (Section 7) as follows:

**Proposition 6.** *$R_i$ is MP-redundant to $R_j$ iff: $i < j$; $a_i = a_j$; $\forall$ packet $P$: $P$ matches $R_i \Rightarrow P$ matches $R_j$; $\exists$ packet that matches $R_j$ and not $R_i$; and $\nexists$ $R_k$ between $R_i$ and $R_j$ s.t.: $R_k$ is shadowed by $R_i$, or $R_k$ generalizes $R_i$, or $R_k$ and $R_i$ correlate.*

Our definition is *local* in the sense that we define redundancy of a rule with respect to (w.r.t) another rule. We can use the global definition "$R_i$ is redundant in a firewall $\mathcal{F}$" from the local definition, as follows: $R_i$ is said redundant in a firewall $\mathcal{F}$, if $\mathcal{F}$ contains a $R_j$ *s.t.* $R_i$ is redundant to $R_j$. Acharya and Gouda (2011) use the global definition of redundancy without using the local definition.

Note that our definition of MP-redundancy is different from the definition of redundancy given in Al-Shaer and Hamed (2004). The latter is erroneous because there are situations where a rule is diagnosed by Al-Shaer and Hamed (2004) as redundant to another rule, while in reality its removal affects the policy.

### 8.2. Detecting LP-redundancy anomaly

Based on the definition of LP-redundancy and Prop. 1:

**Proposition 7.** *Consider a policy $\mathcal{F}$ and its automaton $\Gamma_{\mathcal{F}}$. A rule $R_j$ is LP-redundant to a rule $R_i$ iff: $i < j$; $a_i = a_j$; and $\forall$ match state $\langle r_1, \ldots, r_n \rangle$ of $\Gamma_{\mathcal{F}}$: $r_j = q_j^m \Rightarrow r_i = q_i^m$.*

Consider for example the policy $\mathcal{F}$ of Table 2 and its automaton $\Gamma_{\mathcal{F}}$ of Fig. 4. $R_3$ is LP-redundant to $R_1$ because: $1 < 3, a_1 = a_3 =$ Accept, and there is no match state of $\mathcal{F}$ containing $q_3^4$ and not $q_1^4$.

### 8.3. Detecting MP-redundancy anomalies

Based on the definition of MP-redundancy and Prop. 1:

**Proposition 8.** *Consider a policy $\mathcal{F}$ and its automaton $\Gamma_{\mathcal{F}}$. A rule $R_i$ is MP-redundant to $R_j$ iff: $i < j$; $a_i = a_j$; $\forall$ match state $\langle r_1, \ldots, r_n \rangle$ of $\Gamma_{\mathcal{F}}$: $r_i = q_i^m \Rightarrow r_j = q_j^m$; $\exists$ match state $\langle r_1, \ldots, r_n \rangle$ of $\Gamma_{\mathcal{F}}$ s.t. $r_i = E_i$ and $r_j = q_j^m$; and $\forall$ $k$ s.t. $i < k < j$ and $a_k \neq a_i$, $\nexists$ match state $\langle r_1, \ldots, r_n \rangle$ of $\Gamma_{\mathcal{F}}$ s.t. $r_i = q_i^m$ and $r_k = q_k^m$.*

Consider for example the policy $\mathcal{F}$ of Table 2 and its automaton $\Gamma_{\mathcal{F}}$ of Fig. 4. $R_3$ is MP-redundant with $R_4$ because: $3 < 4, a_3 = a_4 =$ Accept, $\nexists$ match state containing $q_3^4$ and not $q_4^4$, $\exists$ match states containing both $E_3$ and $q_4^4$, and $\nexists$ rule $R_k$ between $R_3$ and $R_4$.

## 9. Mixable policies

The objective is to construct a specific representation of the automaton $\Gamma_{\mathcal{F}}$ describing a policy $\mathcal{F}$. Such a representation is denoted $\mathcal{F}_m$ and called mixable policy. We will explain in Section 9.2 why $\mathcal{F}_m$ is useful.

### 9.1. Extraction of a mixable policy

Let the *semantics* of a policy be the set of pairs (packet, action) specifying the action generated for each packet that could arrive at the firewall. Two policies are said *equivalent* if they have the same semantics. Let a *path* of the automaton $\Gamma_{\mathcal{F}}$ be a sequence $[q^0, q^1, \ldots]$ of connected states of $\Gamma_{\mathcal{F}}$, that leads from the initial state $q^0$ to a final state of $\Gamma_{\mathcal{F}}$. For a policy $\mathcal{F}$ and its automaton $\Gamma_{\mathcal{F}}$, we extract a mixable policy $\mathcal{F}_m$ from $\Gamma_{\mathcal{F}}$ as follows: for every path $[q^0, q^1, \ldots, q^m]$ whose final state $q^m$ is a match state, we extract the rule (*Condition*, *Action*) such that:

- each field $F^j$ of *Condition* is the set of values labeling the transitions linking $q^j$ to $q^{j+1}$, for $j = 0, \ldots, m-1$;
- *Action* is the action associated to $q^m$.

The obtained mixable policy $\mathcal{F}_m$ is equivalent to $\mathcal{F}$, because it is extracted from $\Gamma_{\mathcal{F}}$ that executes $\mathcal{F}$.

For example, for the policy $\mathcal{F}$ of Table 4, and assuming that Protocol can be TCP or UDP, we obtain the automaton of Fig. 7. Its 15 paths are: [1, 2, 5, 11, 17], [1, 2, 5, 11, 18], [1, 2, 5, 12, 18], [1, 2, 6, 12, 18], [1, 2, 7, 13, 19], [1, 3, 8, 14, 20], [1, 3, 8, 15, 21], [1, 3, 9, 15, 21], [1,3,9,16,21], [1,3,9,16,22], [1,3,10,15,21], [1,4,9,15,21], [1, 4, 9, 16, 21], [1, 4, 9, 16, 22], [1, 4, 10, 15, 21].

Since $\Gamma_{\mathcal{F}}$ has 15 paths, the extracted mixable policy $\mathcal{F}_m$ has 15 rules; it is represented in Table 5. The symbol # means "same label as in the above box".

### 9.2. Utility of mixable policies

The automaton $\Gamma_{\mathcal{F}}$ and its corresponding mixable policy $\mathcal{F}_m$ are closely related: each rule of $\mathcal{F}_m$ corresponds to a path of $\Gamma_{\mathcal{F}}$ leading from the initial state to a final state, and conversely. We obtain the following propositions:

**Table 4** Example of policy.

| Rule | IPsrc | IPdst | Port | Protocol | Action |
|---|---|---|---|---|---|
| $R_1$ | * | 210.215.68.203 | 81 | TCP | Accept |
| $R_2$ | 190.170.15.0/24 | 80.15.15.0/24 | * | * | Deny |
| $R_3$ | 194.204.201.0/28 | 210.215.68.204 | 25 | * | Accept |
| $R_4$ | 190.170.15.0/24 | * | * | * | Accept |
| $R_5$ | * | * | * | * | Deny |

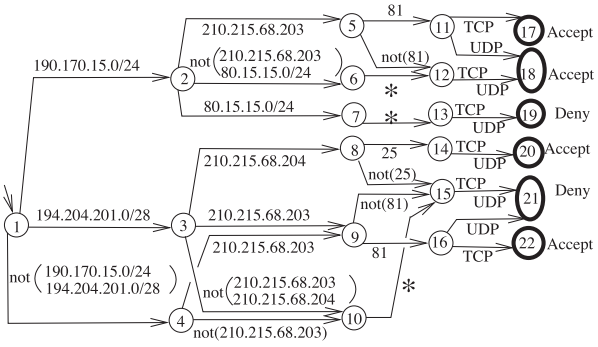


**Figure 7** Automaton $\Gamma\mathcal{F}$ of the policy $\mathcal{F}$ of Table 4.

**Proposition 9.** *A mixable policy consists of disjoint rules, i.e. every packet matches at most one rule.*

**Proposition 10.** *The semantics of a mixable policy is not influenced by the order of its rules.*

Props. 9 and 10 imply that each rule of $\mathcal{F}_m$ can be analyzed independently of the other rules. Indeed, when we consider a rule of $\mathcal{F}_m$, we know that it cannot be in competition with another rule (due to Prop. 9) and we do not have to take into account its position in the table of rules (due to Prop. 10). Let us explain how these properties imply that a mixable policy is an adequate representation for answering to the following question: *In a policy, what are the set of accepted packets and the set of denied packets?*

Indeed, since the rules of $\mathcal{F}_m$ are disjoint, the condition of each rule is a relevant description of a subset of packets. The conditions of the Accept-rules (i.e. rules with action Accept) of $\mathcal{F}_m$ constitute a partition of the whole set of packets accepted by $\mathcal{F}$. In other words, the set of Accept-rules of $\mathcal{F}_m$ constitutes a "whitelist" of $\mathcal{F}$. Similarly, the set of Deny-rules of $\mathcal{F}_m$ constitutes a "blacklist" of $\mathcal{F}$. Hence, the complexity of determining the whitelist and the blacklist of a policy $\mathcal{F}$ is in the order of the number of rules of $\mathcal{F}_m$.

With the original table of $\mathcal{F}$, the whitelist and blacklist can be determined in general only by verifying the reaction of the policy for each possible packet. The number of possibilities to be verified is bounded by the product of the domain sizes of the $m$ fields. For example, consider the 4 fields IPsrc, IPdst, Port and Protocol, and assume that their domains are $[0, 2^{32})$, $[0, 2^{32})$, $[0, 2^{16})$ and $[0, 2^{8})$, respectively. Hence, we may have to consider $2^{32} \times 2^{32} \times 2^{16} \times 2^{8} = 2^{88}$ possibilities. Since the number of possibilities of packets to consider in a policy $\mathcal{F}$ is in general much higher than the number of rules of $\mathcal{F}_m$, we deduce that $\mathcal{F}_m$ is more adequate than the original table of $\mathcal{F}$ to determine the withelist and blacklist of $\mathcal{F}$.

However, the price to be paid for the above advantage of $\mathcal{F}_m$ is that in general $\mathcal{F}_m$ has much more rules than the original table of $\mathcal{F}$. For this reason, $\mathcal{F}_m$ is not intended to be an implementation of $\mathcal{F}$. Mixable policies are only intended to be consulted by designers or managers. An efficient implementation of $\mathcal{F}$ is to execute its automaton (as it will be stated in Prop. 18).

**Table 5** Mixable policy $\mathcal{F}_m$ of the policy $\mathcal{F}$ of Table 4.

| Rule | IPsrc | IPdst | Port | Protocol | Action |
|---|---|---|---|---|---|
| $R_1$ | 190.170.15.0/24 | 210.215.68.203 | 81 | TCP | Accept |
| $R_2$ | # | # | 81 | UDP | Accept |
| $R_3$ | # | # | not(81) | * | Accept |
| $R_4$ | # | not(210.215.68.203, 80.15.15.0/24) | * | * | Accept |
| $R_5$ | # | 80.15.15.0/24 | * | * | Deny |
| $R_6$ | 194.204.201.0/28 | 210.215.68.204 | 25 | * | Accept |
| $R_7$ | # | # | not(25) | * | Deny |
| $R_8$ | # | 210.215.68.203 | not(81) | * | Deny |
| $R_9$ | # | # | 81 | TCP | Accept |
| $R_{10}$ | # | # | 81 | UDP | Deny |
| $R_{11}$ | # | not(210.215.68.203, 210.215.68.204) | * | * | Deny |
| $R_{12}$ | not(190.170.15.0/24, 194.204.201.0/28) | 210.215.68.203 | not(81) | * | Deny |
| $R_{13}$ | # | # | 81 | TCP | Accept |
| $R_{14}$ | # | # | 81 | UDP | Deny |
| $R_{15}$ | # | not(210.215.68.203) | * | * | Deny |

**Table 6** First table of rules.

| Rule | Interface | IPsrc | IPdst | Port | Protocol | Action |
|---|---|---|---|---|---|---|
| $R_1$ | 0 | * | 190.170.0.1 | 25 | TCP | Accept |
| $R_2$ | 0 | 224.168.0.0/16 | * | * | * | Deny |
| $R_3$ | * | * | * | * | * | Accept |

**Table 7** Second table of rules.

| Rule | Interface | IPsrc | IPdst | Port | Protocol | Action |
|---|---|---|---|---|---|---|
| $R_1$ | 0 | 224.168.0.0/16 | * | * | * | Deny |
| $R_2$ | 0 | * | 190.170.0.1 | 25 | TCP | Accept |
| $R_3$ | 0 | * | 190.170.0.1 | * | * | Deny |
| $R_4$ | * | * | * | * | * | Accept |

## 10. Detecting and resolving discrepancies

To ensure a rigorous design of a policy, the authors of Liu and Gouda (2008) propose that the same informal specification of the policy is given to several teams: each team designs a table of rules based on the specification, and then a comparative study of the various tables is made to detect possible discrepancies between them.

Let us propose a procedure to detect discrepancies between tables of rules. We assume that the tables of rules are complete, because incompleteness is undesirable and must be detected and removed before going further (see Section 5). The application of our procedure will be illustrated with the pair of Tables 6 and 7 taken from Liu and Gouda (2008). Note that these tables have an additional field *Interface* (taking value 0 or 1). We will use the notation of Table 8 in the automata of Figs. 8 and 9.

### 10.1. Discrepancy detection

#### 10.1.1. Step 1: computing the automaton of each table of rules

Consider that several teams have designed several tables of rules $T_1, \ldots, T_p$ for the same policy. We apply the procedure of Section 4 to each table of rules to construct automata $\Gamma_1, \ldots, \Gamma_p$. However, when constructing standard intervals (Section 4.2) of each $\Gamma_i$, we consider the intervals used in all the tables so that all the automata $(\Gamma_i)_{i=1\ldots p}$ will have the same alphabet.

For example, for Tables 6 and 7, the resulting automata $\Gamma_1$ and $\Gamma_2$ are represented in Fig. 8. Match states are represented by ellipses, where *a* or *d* denotes the action Accept or Deny associated to a match state.

**Table 8** Notations used in Figs. 8 and 9.

| Notation | Meaning | Notation | Meaning |
|---|---|---|---|
| $\alpha$ | 224.168.0.0 | *a* | Accept |
| $\beta$ | 224.168.255.255 | *d* | Deny |
| $\gamma$ | 190.170.0.1 | | |

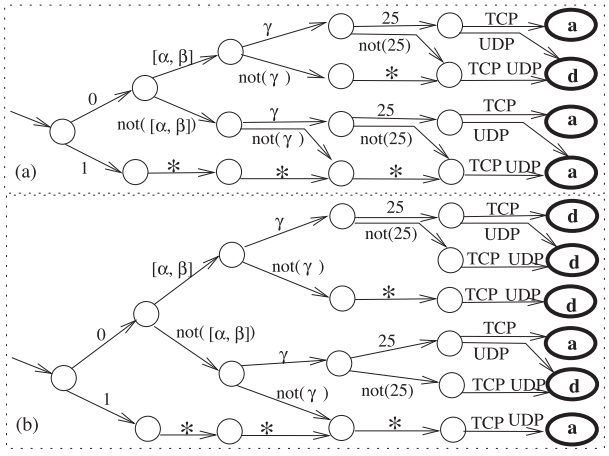


**Figure 8** Step 1 of discrepancy detection: automata resulting from Tables 6 and 7.

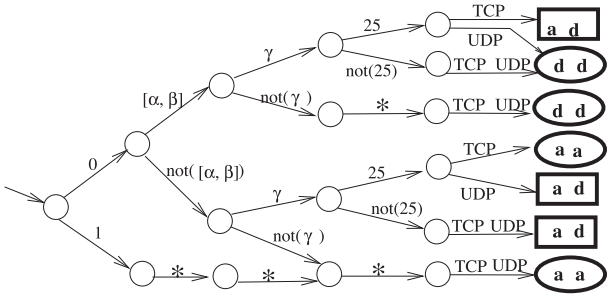


**Figure 9** Step 2 of discrepancy detection: product automaton $\Gamma$ obtained from automata $(\Gamma i)_{i=1,2}$ of Fig. 8.

#### 10.1.2. Step 2: computing a product automaton

We compute the product $\Gamma$ of the automata $(\Gamma_i)_{i=1\ldots p}$ obtained in Step 1, using the product operator defined in Section 4.3. Each state of $\Gamma$ is defined by $\langle \phi_1, \ldots, \phi_p \rangle$, where $\phi_i$ is a state of $\Gamma_i$. $\langle \phi_1, \ldots, \phi_p \rangle$ is said a match state if each $\phi_i$ is a match state in $\Gamma_i$.

For example, the automaton $\Gamma$ of Fig. 9 is the product of the automata $\Gamma_1$ and $\Gamma_2$ of Fig. 8.

*10.1.3. Step 3: detecting discrepancies*

Discrepancy detection is based on the following proposition:

**Proposition 11.** *Each match state of* $\Gamma$ *associated to distinct actions represents a discrepancy.*

For example, three discrepancies are detected in the automaton $\Gamma$ of Fig. 9. Rectangles represent the match states with discrepancies, while ellipses represent the match states without discrepancy.

*10.2. Discrepancy resolution*

For each detected discrepancy, the various teams of designers discuss and agree on a single action. For the example of Fig. 9, the result of resolution is that one action is selected for each rectangular state.

## 11. Space and time complexities

*11.1. Computation of complexities*

We define two categories of fields: a *great field* takes values in a domain of size $> n$, and a *small field* takes values in a domain of size $\leqslant n$. If for example we take $n = 1000$, then IPsrc, IPdst and Port are great fields (since their domain sizes $2^{32}$ and $2^{16}$ are greater than 1000), and Protocol is a small field (in practice, much less than 1000 protocols are considered). Our calculations of complexities are based on the following parameters:

$n$ = number of rules of a table; when we study several ($p$) tables (to detect discrepancies), $n$ is the sum of the sizes of the $p$ tables (hence $n > p$);
$m$ = number of fields $(F^j)_{j=0,\ldots,m-1}$ in a rule;
$\mu$ = number of great fields;
$f_j$ = domain size of $F^j$;
$\Phi$ = product of all domain sizes $f_j$, i.e. $\Phi = f_0 \times \ldots \times f_{m-1}$;
$\varphi$ = product of the domain sizes $f_j$ of the small fields;
$\Delta$ = sum of the domain sizes $f_j$ of the small fields.

To present our results in a suitable form for comparison with the literature, we also define:

$d_j = \log_2(f_j)$, i.e. we need $\lceil d_j \rceil$ bits to code all the values of $F^j$ ($\lceil d_j \rceil$ is the smallest integer $\geqslant d_j$);
$D = \log_2(\Phi) = d_0 + \cdots + d_{m-1}$;
$\delta = \log_2(\varphi)$ sum of the $d_j$ of the small fields.

We assume $d_j \geqslant 1$ (i.e. $f_j \geqslant 2$, each field takes a value among *several* values), $n > D$ and $2^n > n^m$ (realistic when $n$ is sufficiently greater than $m$). These assumptions and the above definitions imply: $\mu \leqslant m \leqslant D$, $\delta \leqslant D$ (i.e. $\varphi \leqslant \Phi$), and $\Delta \leqslant (m - \mu) \times n$.

For example, with 4 fields IPsrc, IPdst, Port and Protocol, we have $m = 4$. Assume that $d_0 = d_1 = 32$ (32 bits to code IPsrc and IPdst), $d_2 = 16$ (16 bits to code Port), $d_3 = 8$ (8 bits to code Protocol). Hence, $D = 32 + 32 + 16 + 8 = 88$, $f_0 = f_1 = 2^{32}, f_2 = 2^{16}, f_3 = 2^8$ and $\Phi = 2^{88}$. For $88 < n < 2^{16}$, the assumptions "$d_i \geqslant 1, n > D$ and $2^n > n^m$" are satisfied, the great fields are IPsrc, IPdst and Port, and the unique small field is Protocol. Therefore, $\mu = 3$ (number of great fields), $\delta = d_3 = 8$ (number of bits to code the small field), and $\Delta = \varphi = f_3 = 2^8$ (domain size of the small field).

**Proposition 12.** *The procedure to construct the automaton has space and time complexities in* $O(n^{\mu+1} \times 2^{\delta})$*; the latter is upper-bounded by* $O(n^{m+1})$ *and* $O(n \times 2^D)$.

**Proposition 13.** *Incompleteness detection has space and time complexities in* $O(n^{\mu+1} \times 2^{\delta})$*; the latter is upper-bounded by* $O(n^{m+1})$ *and* $O(n \times 2^D)$.

**Proposition 14.** *Detecting conflicting and LP-redundancy anomalies has space and time complexities in* $O(n^{\mu+2} \times 2^{\delta})$*; the latter is upper-bounded by* $O(n^{m+2})$ *and* $O(n^2 \times 2^D)$.

**Proposition 15.** *Detecting MP-redundancy anomalies has a space complexity in* $O(n^{\mu+2} \times 2^{\delta})$*; the latter is upper-bounded by* $O(n^{m+2})$ *and* $O(n^2 \times 2^D)$*. Detecting MP-redundancy anomalies has a time complexity in* $O(n^{\mu+3} \times 2^{\delta})$*; the latter is upper-bounded by* $O(n^{m+3})$ *and* $O(n^3 \times 2^D)$.

**Proposition 16.** *Extracting a mixable policy has space and time complexities in* $O(n^{\mu+1} \times 2^{\delta})$*; the latter is upper-bounded by* $O(n^{m+1})$ *and* $O(n \times 2^D)$.

Consider $p$ policies $\mathcal{F}_1, \ldots, \mathcal{F}_p$, let $n_1, n_2, \ldots, n_p$ be their respective numbers of rules, and $n = n_1 + \cdots + n_p$.

**Proposition 17.** *Detecting and resolving discrepancies between* $\mathcal{F}_1, \ldots, \mathcal{F}_p$ *has space and time complexities in* $O(n^{\mu+1} \times 2^{\delta})$*; the latter is upper-bounded by* $O(n^{m+1})$ *and* $O(n \times 2^D)$.

For a policy $\mathcal{F}$ and a packet $P$, we can determine the action Accept or Deny dictated by $\mathcal{F}$ for $P$, by executing the table of $\mathcal{F}$ (by consulting its rules, as explained in Section 3) or the automaton $\Gamma_{\mathcal{F}}$ (as shown in Prop. 1). If we execute the table of $\mathcal{F}$, we may have to verify the $m$ fields of the $n$ rules; the time complexity is therefore in $O(m \times n)$. The next Prop. 18 implies that executing $\Gamma_{\mathcal{F}}$ is less costly than executing the table of $\mathcal{F}$.

**Proposition 18.** *Executing the automaton* $\Gamma_{\mathcal{F}}$ *for a packet P has a space complexity in* $O(1)$*, while its time complexity is in* $O(\mu \times n + \Delta)$*; the latter is upper-bounded by* $O(m \times n)$.

*11.2. Comparison with complexities in related work*

In Elmallah and Gouda (2014), it is proved that several analysis and design problems of firewalls are NP-hard. With our terminology, their result is that the time complexity of several problems is in $O(n \times 2^D)$. On the other hand, the authors of Acharya and Gouda (2010, 2011), Liu and Gouda (2008, 2010), Al-Shaer et al. (2009) solve analysis problems with algorithms whose estimated time complexity is in $O(n^{m+1})$. Our contribution is that the estimations $O(n^{m+1})$ and $O(n \times 2^D)$ are upper bounds of our estimation $O(n^{\mu+1} \times 2^{\delta})$.

Note that we are not pretending that our approach is less complex than in the literature. What we want to show is that

our estimation of complexity is *more precise* than in the literature. This better precision has been possible because we have separated the fields in two categories: great fields and small fields. When all fields are great, our expression $n^{\mu+1} \times 2^{\delta}$ is equal to $n^{m+1}$ (i.e. expression of Acharya and Gouda (2010, 2011), Liu and Gouda (2008, 2010), Al-Shaer et al. (2009)). When all fields are small, our expression $n^{\mu+1} \times 2^{\delta}$ is equal to $n \times 2^{D}$ (i.e. expression of Elmallah and Gouda (2014)).

### *11.3. Intuitive justification of the precision of our complexity evaluation*

In this section, we justify intuitively why our complexity evaluation is more precise than in the literature. After Steps 1 and 2 of the automaton construction procedure (Sections 4.1 and 4.2), the domain of each field becomes defined by a partition of subsets, where we can reason on each subset as if it were a single value. The partition of each field $F^j$ is constructed as follows. For each field $F^j$ and each rule, we have an interval of values of $F^j$ defined by a pair of values (minimal and maximal values). We regroup the pairs of values for all the rules and then order them in increasing order into a list $L^j$. The important fact is that we have realized that the size of $L^j$ has the following two upper-bounds:

- The first upper-bound is the order of the number of rules: $O(n)$. This upper-bound is due to the fact that we have 2 values per rule, and hence at most $2n$ values for all the rules.
- The second upper-bound is the order of the domain size of $F^j : O(2^{d_j}) = O(f_j)$. This upper-bound is due to the fact that the number of subsets cannot exceed the number of possible values of $F^j$.

These two upper-bounds of the size of $L^j$ are also upper-bounds of the number of subsets that constitute the partition of $F^j$.

If we use only the upper-bound $O(n)$, we obtain complexities in the form $O(n^{m+1})$, i.e. the expression of Acharya and Gouda (2010, 2011), Liu and Gouda (2008, 2010), Al-Shaer et al. (2009).

If we use only the upper-bound $O(2^{d_j})$, we obtain complexities in the form $O(n \times 2^{D})$, i.e. the expression of Elmallah and Gouda (2014).

### *11.4. Field reduction*

Let us define the notion of field reduction to clarify the fact that the worst case complexities of Section 11.1 are generally much higher than the actual complexities obtained with concrete examples.

As we have seen in Section 11.3, after Steps 1 and 2 of the automaton construction procedure, the number of subsets that constitute the partition of each field $F^j$ is upper-bounded by both $O(n)$ and $O(2^{d_j})$, i.e. by $O(\Psi_j)$ where $\Psi_j = \min(2^{d_j}, n)$ i.e. the smallest of $2^{d_j}$ and $n$. In other words, Steps 1 and 2 of the synthesis procedure imply a conceptual reduction of the domain size of each $F^j$: from $O(2^{d_j})$ to $O(\Psi_j)$. This is in fact the worst case reduction, which is in general much smaller than the reductions actually obtained with concrete examples, because in general the obtained partitions have much less than $\Psi_j$ subsets.

This important difference between the worst case reduction and an actual reduction implies that the worst case complexities of Section 11.1 are generally much higher than the actual complexities obtained with concrete examples. This fact will be illustrated in Section 12.

## 12. Case study

We have applied our method to several examples. Due to space limitation, we have opted to present the results obtained for one example, namely the real-life policy of Chen et al. (2012) which is anonymized due to the privacy and security concern.

The policy of Chen et al. (2012) consists of 87 rules[1] ($R_1$ to $R_{87}$) expressed in 5 fields, whose domain sizes are reduced as follows by Steps 1 and 2 of the synthesis procedure:

*Source IP address*: its domain $[0, 2^{32})$ is partitioned in 19 subsets, hence its domain size is actually reduced from $2^{32}$ to 19.
*Destination IP address*: its domain $[0, 2^{32})$ is partitioned in 18 subsets, hence its domain size is actually reduced from $2^{32}$ to 18.
*Source port*: its domain $[0, 2^{16})$ is partitioned in 2 subsets, hence its domain size is actually reduced from $2^{16}$ to 2.
*Destination port*: its domain $[0, 2^{16})$ is partitioned in 35 subsets, hence its domain size is actually reduced from $2^{16}$ to 35.
*Protocol*: its domain $[0, 2^{8})$ is partitioned in 6 subsets, hence its domain size is actually reduced from $2^{8}$ to 6.

In this example, the worst case reduction is to reduce the domain size of each field to the number of rules, i.e. 87, because all the fields are great ($\mu = 5$ and $\delta = 0$). After the actual reduction, all fields become small fields whose domain sizes are 19, 18, 2, 35, 6, i.e. $\mu = 0$ and $\delta = \log_2(19 \times 18 \times 2 \times 35 \times 6) \approx 17.13$. Hence, $O(n^{\mu+1} \times 2^{\delta})$ is equal to $O(87^6) \approx 433 \times 10^9$ with the worst case reduction, and becomes $O(87 \times 2^{17.13}) \approx 12 \times 10^6$ with the actual reduction. As for the expression $O(\mu \times n + \Delta)$ of the time complexity to apply the policy by executing its automaton, it is equal to $O(5 \times 87) = O(435)$ with the worst case reduction, and becomes $O(19 + 18 + 2 + 35 + 6) = O(80)$ with the actual reduction.

The constructed automaton has 14,543 states, among which there are 296 match states. Since the automaton contains also the no-match state, the policy is incomplete. The following anomalies are detected:

- $R_{21}$ generalizes $R_2$;
- $R_{54}$ generalizes $R_{48}$;
- $R_{55}$ generalizes $R_{49}$;
- $R_{56}$ generalizes $R_{50}$;
- $R_{87}$ generalizes: $R_7, R_9$, $R_{27}$–$R_{38}$, $R_{45}$–$R_{47}, R_{51}, R_{76}, R_{79}$–$R_{84}, R_{86}$;
- $R_{12}$ correlates with: $R_2, R_{39}$-$R_{41}$;
- $R_6$ correlates with: $R_7, R_9$, $R_{27}$–$R_{38}$, $R_{45}, R_{46}, R_{51}$;
- $R_{85}$ correlates with: $R_7, R_9$, $R_{27}$–$R_{38}$, $R_{45}, R_{46}, R_{51}$;

[1] Due to space limitation, the table of 87 rules is not presented here; it can be consulted in Chen et al. (2012).

- $R_{85}$ is MP-redundant to $R_{87}$.

The generated mixable policy contains 101,780 rules, among which there are 2317 Accept-rules and 99,463 Deny-rules. The great number of rules can make this mixable policy unattractive, but the relatively reduced number of Accept-rules could motivate the designer to examine the whitelist.

Despite the large sizes of the automaton and the mixable policy, their constructions last less than 10 s, while the other operations (completeness verification, anomaly detections, execution of the automaton) last less than 1 s. The implementation of our method runs on a VMware ESXi5.1 server using a 4vCPU in a Linux environment.

## 13. Conclusion

### 13.1. Recapitulation of our contributions

A procedure is proposed to construct an automaton which describes a (firewall security) policy given as a table of rules. The procedure is then applied as a common basis to develop analysis and design procedures to:

- apply a policy by executing its automaton;
- detect incompleteness in a policy;
- detect conflicting and nonconflicting anomalies in a policy;
- extract a mixable policy;
- detect and resolve discrepancies between table of rules.

We evaluate space and time complexities of every developed procedure more precisely than in the literature. Our methodology is illustrated by applying it to a real-life policy. All the obtained results are formally proved.

### 13.2. Future work

We plan to study the following subjects:

- To study *stateful* firewalls, i.e. policies that behave depending on the filtered traffic history.
- To design adaptive policies that change dynamically with the filtered traffic.
- To study how to resolve conflicting anomalies in security policies whose rules have the same priority. In the present paper, conflicting anomalies are automatically resolved by selecting the action (Accept or Deny) of the most priority rule among the conflicting rules.
- To study how our formalism can be used for security enforcement on untrusted programs. The objective is to correct automatically a program $P$ so that it satisfies a security policy $\Phi$. Our basic approach would be to model $P$ by an automaton $\mathcal{A}_P$, and then to combine $\mathcal{A}_P$ in a certain way with an automaton $\mathcal{A}_\Phi$ modeling $\Phi$. The result would be an automaton modeling a program which can make *all and only* the executions of $P$ that satisfy $\Phi$. Our idea is inspired from Sui and Mejri (2013), with the difference that Sui and Mejri (2013) model $P$ and $\Phi$ by a specific process algebra instead of automata.
- We are presently working on a project on security policies in a medical cloud environment. The idea is to provide a cloud infrastructure through which doctors in different regions collaborate to perform an effective diagnosis to a patient in emergency situation. In such a heterogeneous distributed environment, a particular problem that may arise is the situation where security policies of different sites/providers are contradictory, which we call inter-policies anomaly. So, in addition to the (intra-policy) anomalies studied in Sections 7 and 8, the objective is now to study how to detect and resolve inter-policies anomalies.In this project, we will also study security enforcement in the context of cloud storage. We will be inspired especially by Shen et al. (2013).
- Consider a component $\mathcal{C}$ that communicates with other components without crossing a firewall, while we want the firewall policy to be respected. We are looking at how to develop a method that extracts a local policy applicable to $\mathcal{C}$ from the firewall policy.

## Acknowledgements

We would like to thank Prof. Yahya Benkaouz for his valuable participation in the preparation of the final version of the paper.

## Appendix A. Proofs of Propositions

### A.1. Proof of Proposition 1

1. In each $\mathcal{A}_i^*$ computed in Step 1, when we execute the $m$ consecutive transitions whose labels contain $H^0, \ldots, H^{m-1}$, respectively: we reach state $q_i^m$ if $P$ matches $R_i$, and we reach $E_i$ if $P$ does not match $R_i$.
2. The final state reached in $\Gamma_{\mathcal{F}}$ for a packet $P$ is $\langle r_1, \ldots, r_n \rangle$ *iff* the final state reached in each $\mathcal{A}_i^*$ is $r_i$, for $i = 1 \ldots n$.
3. From items 1 and 2, we deduce: $r_i = q_i^m$ or $r_i = E_i$, and $P$ matches the rule $R_i$ *iff* $r_i = q_i^m$.
4. To each match state $\langle r_1, \ldots, r_n \rangle$ of $\Gamma_{\mathcal{F}}$, we associate the action $a_i$ with the smallest index $i$ *s.t.* $r_i = q_i^m$.
5. From items 3 and 4, we deduce that if $r$ is a match state, it is associated with the action of the most priority rule that is matched by $P$.

### A.2. Proof of Proposition 2

Prop. 1 implies that the final state of $\Gamma_{\mathcal{F}}$ reached for a packet $P$ is the no-match state $\langle E_1, E_2, \ldots, E_n \rangle$ *iff* $P$ matches none of the rules $R_1$ to $R_n$. Therefore, $\Gamma_{\mathcal{F}}$ has not the no-match state *iff* every packet matches one or more rules.

### A.3. Proof of Proposition 3

By definition, $R_j$ is shadowed by $R_i$ *iff*: $i < j$; $a_i \neq a_j$; and for every packet $P$, if $P$ matches $R_j$ then $P$ matches $R_i$. From Prop. 1, the latter condition is equivalent to: for every match state $\langle r_1, \ldots, r_n \rangle$ of $\Gamma_{\mathcal{F}}$, $r_j = q_j^m \Rightarrow r_i = q_i^m$.

### A.4. Proof of Proposition 4

By definition, $R_j$ generalizes $R_i$ *iff*: (1) $i < j$; (2) $a_i \neq a_j$; and (3) $R_j$ matches more packets than $R_i$. Condition 3 is equivalent to:

(3.1) for every packet $P$, if $P$ matches $R_i$ then $P$ matches $R_j$; and (3.2) there exists a packet that matches $R_j$ and not $R_i$. From Prop. 1, condition 3.1 is equivalent to: for every match state $\langle r_1, \ldots, r_n \rangle$ of $\Gamma_{\mathcal{F}}$, $r_i = q_i^m \Rightarrow r_j = q_j^m$; and condition 3.2 is equivalent to: there exists a match state $\langle r_1, \ldots, r_n \rangle$ of $\Gamma_{\mathcal{F}}$ *s.t.* $r_i = E_i$ and $r_j = q_j^m$.

### *A.5. Proof of Proposition 5*

By definition, $R_i$ and $R_j$ correlate *iff*: (1) $i \neq j$; (2) $a_i \neq a_j$; (3) there exist packets that match $R_j$ and not $R_i$; (4) there exist packets that match $R_i$ and not $R_j$; and (5) there exist packets that match both $R_i$ and $R_j$. From Prop. 1, condition 3 is equivalent to: there exists a match state $\langle r_1, \ldots, r_n \rangle$ of $\Gamma_{\mathcal{F}}$ *s.t.* $r_i = E_i$ and $r_j = q_j^m$; condition 4 is equivalent to: there exists a match state $\langle r_1, \ldots, r_n \rangle$ of $\Gamma_{\mathcal{F}}$ *s.t.* $r_i = q_i^m$ and $r_j = E_j$; and condition 5 is equivalent to: there exists a match state $\langle r_1, \ldots, r_n \rangle$ of $\Gamma_{\mathcal{F}}$ *s.t.* $r_i = q_i^m$ and $r_j = q_j^m$.

### *A.6. Proof of Proposition 6*

The three anomalies with conflict can be expressed as follows, where $\mathcal{P}_x$ denotes the set of packets matching $R_x$:

- $R_k$ is shadowed by $R_i$ *iff*: $i < k, a_i \neq a_k$, and $\mathcal{P}_k \subseteq \mathcal{P}_i$.
- $R_k$ generalizes $R_i$ *iff*: $i < k, a_i \neq a_k$, and $\mathcal{P}_i \subset \mathcal{P}_k$.
- $R_k$ and $R_i$ correlate *iff*: $i \neq k, a_i \neq a_k$, $\mathcal{P}_i \cap \mathcal{P}_k \neq \varnothing$, $\mathcal{P}_i \not\subseteq \mathcal{P}_k$ and $\mathcal{P}_k \not\subseteq \mathcal{P}_i$.

Using the above expressions, let us first prove that for a rule $R_k$, the following two expressions E1 and E2 are equivalent:

*E1:*($i < k$) and ($a_k \neq a_i$) and (there exist packets matching both $R_i$ and $R_k$).
*E2:*($R_k$ is shadowed by $R_i$) or ($R_k$ generalizes $R_i$) or ($R_k$ and $R_i$ correlate).

1. The existence of packets matching both $R_i$ and $R_k$ is equivalent to $\mathcal{P}_i \cap \mathcal{P}_k \neq \varnothing$.
2. $\mathcal{P}_i \cap \mathcal{P}_k \neq \varnothing$ is equivalent to: ($\mathcal{P}_k \subseteq \mathcal{P}_i$) or ($\mathcal{P}_i \subset \mathcal{P}_k$) or ($\mathcal{P}_i \cap \mathcal{P}_k \neq \varnothing$ and $\mathcal{P}_i \not\subseteq \mathcal{P}_k$ and $\mathcal{P}_k \not\subseteq \mathcal{P}_i$).
3. From items 1 and 2, we deduce that E1 is equivalent to the disjunction: ($i < k, a_i \neq a_k$, and $\mathcal{P}_k \subseteq \mathcal{P}_i$) or ($i < k, a_i \neq a_k$, and $\mathcal{P}_i \subset \mathcal{P}_k$) or ($i < k, a_i \neq a_k$, $\mathcal{P}_i \cap \mathcal{P}_k \neq \varnothing$, $\mathcal{P}_k \not\subseteq \mathcal{P}_i$, and $\mathcal{P}_k \not\subseteq \mathcal{P}_i$).
4. The three components of the disjunction of item 3 are equivalent to "$R_k$ is shadowed by $R_i$", "$R_k$ generalizes $R_i$ and "$R_k$ and $R_i$ correlate", respectively.
5. From items 3 and 4, we deduce that E1 and E2 are equivalent.

Using the equivalence between E1 and E2, the definition of "$R_i$ is MP-redundant to $R_j$" can be expressed as Prop. 6.

### *A.7. Proof of Proposition 7*

By definition, $R_j$ is LP-redundant to $R_i$ if: (1) $i < j$, (2) $a_i = a_j$, and (3) $R_i$ matches all the packets matching $R_j$.

From Prop. 1, condition 3 is equivalent to: for every match state $\langle r_1, \ldots, r_n \rangle$ of $\Gamma_{\mathcal{F}}$: if $r_j = q_j^m$, then $r_i = q_i^m$.

### *A.8. Proof of Proposition 8*

By definition, $R_i$ is MP-redundant to $R_j$ if: (1) $i < j$; (2) $a_i = a_j$; (3) if a packet $P$ matches $R_i$, then $P$ matches $R_j$; (4) there exists a packet that matches $R_j$ and does not match $R_i$; and (5) there exists no rule $R_k$ *s.t.* $i < k < j$, $a_k \neq a_i$, and there exist packets matching both $R_i$ and $R_k$.

From Prop. 1, condition 3 is equivalent to: for every final state $\langle r_1, \ldots, r_n \rangle$ of $\Gamma_{\mathcal{F}}$: $r_i = q_i^m \Rightarrow r_j = q_j^m$; condition 4 is equivalent to: there exists a match state $\langle r_1, \ldots, r_n \rangle$ of $\Gamma_{\mathcal{F}}$ *s.t.* $r_i = E_i$ and $r_j = q_j^m$; and condition 5 is equivalent to: for every $k$ *s.t.* $i < k < j$ and $a_k \neq a_i$, there exists no match state $\langle r_1, \ldots, r_n \rangle$ of $\Gamma_{\mathcal{F}}$ *s.t.* $r_i = q_i^m$ and $r_k = q_k^m$.

### *A.9. Proof of Proposition 9*

By construction, the automaton $\Gamma_{\mathcal{F}}$ is deterministic, hence to each packet corresponds exactly one path of $\Gamma_{\mathcal{F}}$. To each path of $\Gamma_{\mathcal{F}}$ leading to a match state corresponds exactly one rule of $\mathcal{F}_{\mathrm{m}}$, and vice versa. No rule of $\mathcal{F}_{\mathrm{m}}$ corresponds to any path of $\Gamma_{\mathcal{F}}$ leading to a no-match state. In conclusion, each packet matches at most one rule of $\mathcal{F}_{\mathrm{m}}$.

### *A.10. Proof of Proposition 10*

The order of the rules influences the semantics of a policy only if there exist packets matching several rules. From Prop. 9, there exists no packet matching several rules of a mixable policy. Therefore, the semantics of a mixable policy is not influenced by the order of its rules.

### *A.11. Proof of Proposition 11*

For any packet $P$, the match state reached in $\Gamma$ is $\langle \phi_1, \ldots, \phi_p \rangle$ *iff* the match state reached in each $\Gamma_i$ is $\phi_i$, for $i = 1 \ldots p$. A match state $\langle \phi_1, \ldots, \phi_p \rangle$ of $\Gamma$ is associated to actions $(a_1, \ldots, a_p)$ *iff* each state $\phi_i$ of $\Gamma_i$ is associated to action $a_i$, for $i = 1, \ldots, p$. From Prop. 1, the action dictated by $T_i$ for a packet $P$ is the action associated to the match state reached in $\Gamma_i$, for $i = 1, \ldots, p$. From these three facts, reaching a state of $\Gamma$ associated to distinct actions corresponds to a situation where distinct actions are dictated to a packet.

### *A.12. Preliminary result for the computation of complexities*

Let $\Psi_j = \min(f_j, n) = \min(2^{d_j}, n)$, i.e. $\Psi_j$ is the smallest of $f_j$ and $n$.

By definition of a great field $F^j$, we have $n < 2^{d_j}$ and hence $\Psi_j = n$. Therefore, the product of the $\Psi_j$'s of the $\mu$ great fields is equal to $n^\mu$ and smaller than $2^{D-\delta}$.

By definition of a small field $F^j$, we have $2^{d_j} \leqslant n$ and hence $\Psi_j = 2^{d_j}$. Therefore, the product of the $\Psi_i$'s of the $m - \mu$ small fields is $= 2^\delta$ and $\leqslant n^{m-\mu}$.

Therefore, $\Psi_0 \times \cdots \times \Psi_{m-1} = n^\mu \times 2^\delta$, which is smaller than both $n^\mu \times n^{m-\mu} = n^m$ and $2^{D-\delta} \times 2^\delta = 2^D$.

### A.13. Proof of Prop. 12

*Complexity of Step 1*

Space and time complexities to construct one state or one transition of $\mathcal{A}_i$ are in O(1). Each $\mathcal{A}_i$ contains $m+2$ states and a limited number (i.e. O(1)) of transitions from each state. Hence, space and time complexities to construct each $\mathcal{A}_i$ are in O($m$). Hence, space and time complexities to construct the $n$ $(\mathcal{A}_i)_{1\ldots n}$ are in O($m \times n$).

*Complexity of Step 2*

For every level $j$, the sorted list $L^j$ is constructed as follows:

- For every automaton $\mathcal{A}_i$: we consider the intervals labeling the transitions from the state $q_i^j$. We construct a sorted list (in increasing order) $L_i^j$ which contains the endpoints of those intervals.
- The lists $(L_i^j)_{i=1\ldots n}$ are combined to obtain a sorted list $L^j$.

Space and time complexities to construct each $L_i^j$ are in O(1), because the cardinality of $L_i^j$ is in O(1). Hence, space and time complexities to construct all $L_i^j$ ($i = 1\ldots n$, $j = 0\ldots m-1$) are in O($n \times m$).

$L^j$ is constructed by merging $(L_i^j)_{i=1\ldots n}$. If we use the merge sort algorithm, space and time complexities are in O($n$) and O($n \times \log(n)$), respectively. Hence, space and time complexities to construct all $L^j$ ($j = 0\ldots m-1$) are in O($m \times n$) and O($m \times n \times \log(n)$), respectively.

The number of transitions from each state $q_i^j$ of each automaton $\mathcal{A}_i^*$ is bounded by both O($2^{d_j}$) and O($n$) i.e. by O($\Psi_j$). The bound O($2^{d_j}$) is due to the fact that $2^{d_j}$ is the number of possible values of $F^j$, which is $\geqslant$ the number of transitions from $q_i^j$. The bound O($n$) is due to the fact that it is the order of the cardinality of $L^j$, which is $\geqslant$ the number of transitions from $q_i^j$. Hence, space and time complexities to construct all the transitions of each automaton $\mathcal{A}_i^*$ are in O($\Psi_0 + \cdots + \Psi_{m-1}$). Space and time complexities to construct all the states of each $\mathcal{A}_i^*$ are in O($m$). Therefore, space and time complexities to construct all the $\mathcal{A}_i^*$ are in O($n(\Psi_0 + \cdots + \Psi_{m-1})$).

*Complexity of Step 3*

Let us consider the construction of $\Gamma_{\mathcal{F}}$ in Step 3 level by level. At each level $j$, we evaluate space and time complexities to construct the states of level $j$ (i.e. the states that are reached after $j$ transitions from the initial state) and the transitions from level $j-1$ to level $j$.

Space and time complexities to construct a state $r = \langle r_1, \ldots, r_n \rangle$ of $\Gamma_{\mathcal{F}}$ are in O($n$), because we construct and store the $n$ components of $r$. Space and time complexities to construct a transition $t$ between two constructed states of levels $j$ and $j+1$ are in O(1), because we store the label of $t$.

*Level* **0**: The initial state $r^0$ is the unique state of level 0. Space and time complexities of its construction are in O($n$).

*Level* **1**: Using the same reasoning as in the proof of Step 2, the number of transitions from $r^0$ is in O($\Psi_0$). Hence, the number of states at level 1 is in O($\Psi_0$). Therefore: space and time complexities to construct the transitions from level 0 to level 1 are in O($\Psi_0$), and space and time complexities to construct the states at level 1 are in O($n \times \Psi_0$).

*Level* **2**: Using the same reasoning as in the proof of Step 2, the number of transitions from each state of level 1 is in O($\Psi_1$). Since the number of states of level 1 is in O($\Psi_0$), we obtain that: the number of states at level 2 and the number of transitions from level 1 to level 2 are in O($\Psi_0 \times \Psi_1$). Therefore: space and time complexities to construct the states at level 2 are in O($n \times \Psi_0 \times \Psi_1$), and space and time complexities to construct the transitions from level 1 to level 2 are in O($\Psi_0 \times \Psi_1$). And so on …until level $m$.

*Level* $m$: Using the same reasoning as in the proof of Step 2, we obtain that: space and time complexities to construct the states at level $m$ are in O($n \times \Psi_0 \times \cdots \times \Psi_{m-1}$); and space and time complexities to construct the transitions from level $m-1$ to level $m$ are in O($\Psi_0 \times \cdots \times \Psi_{m-1}$).

At each level $j$, the number of states is also bounded by $2^n$, because each state is defined by $n$ 2-value components $r_i$ ($r_i = q_i^j$ or $r_i = E_i$, for $i = 1\ldots n$). But this bound has no influence because it is greater than the other identified bounds, due to the assumptions $n > D$ and $2^n > n^m$.

*All levels*: By adding the complexities of all levels, we obtain that space and time complexities to construct $\Gamma_{\mathcal{F}}$ are in O($n \times \Psi_0$) + O($n \times \Psi_0 \times \Psi_1$) + $\cdots$ + O($n \times \Psi_0 \times \cdots \times \Psi_{m-1}$). From $d_i \geqslant 1$ and $n > D$, we obtain $\Psi_i \geqslant 2$, from which we deduce that $\Psi_0 + (\Psi_0 \times \Psi_1) + \cdots + (\Psi_0 \times \cdots \times \Psi_{m-1}) \leqslant 2\times(\Psi_0 \times \cdots \times \Psi_{m-1})$. Hence, space and time complexities to construct $\Gamma_{\mathcal{F}}$ are in O($n \times \Psi_0 \times \cdots \times \Psi_{m-1}$).

*Associating actions*: The space complexity to associate an action to a match state of $\Gamma_{\mathcal{F}}$ is in O(1), because we only need to store the action associated to the state. Since there are at most O($\Psi_0 \times \cdots \times \Psi_{m-1}$) match states in $\Gamma_{\mathcal{F}}$, the space complexity to associate actions to all these states is in O($\Psi_0 \times \cdots \times \Psi_{m-1}$). The time complexity to associate an action to a match state of $\Gamma_{\mathcal{F}}$ is in O($n$), because we may need to consult the $n$ components of the state. Since there are at most O($\Psi_0 \times \cdots \times \Psi_{m-1}$) match states in $\Gamma_{\mathcal{F}}$, the time complexity to associate actions to all these states is in O($n \times \Psi_0 \times \cdots \times \Psi_{m-1}$).

*Total complexity*

Since Steps 1 and 2 are less complex than Step 3, we obtain that space and time complexities to construct $\Gamma_{\mathcal{F}}$ are in O($n \times \Psi_0 \times \cdots \times \Psi_{m-1}$), which is equal to O($n^{\mu+1} \times 2^{\delta}$) and smaller than O($n^{m+1}$) and O($n \times 2^D$).

### A.14. Proof of Proposition 13

We consider that in the construction of the automaton $\Gamma_{\mathcal{F}}$, the no-match state (if any) is placed at the top of the list of states of $\Gamma_{\mathcal{F}}$.

Verifying completeness from $\Gamma_{\mathcal{F}}$ can be achieved by checking if the top state is associated to an action. Therefore, space and time complexities to verify completeness from $\Gamma_{\mathcal{F}}$ are in O(1).

Since we need to construct $\Gamma_{\mathcal{F}}$, space and time complexities to verify completeness from the original table of $\mathcal{F}$ are the same as those given by Prop. 12.

### A.15. Proof of Propositions 14and 15

#### A.15.1. Space complexity of Props. 14 and 15

For each pair $(i,j)$, we store the identifiers of the match states of $\Gamma_{\mathcal{F}}$ where anomalies between rules $R_i$ and $R_j$ are detected. Since the number of match states is in $O(\Psi_0 \times \cdots \times \Psi_{m-1})$, the space to store all the identifiers for one pair $(i,j)$ is in $O(\Psi_0 \times \cdots \times \Psi_{m-1})$. Since the number of pairs $(i,j)$ is in $O(n^2)$, the total space complexity to detect anomalies from $\Gamma_{\mathcal{F}}$ is in $O(n^2 \times \Psi_0 \times \cdots \times \Psi_{m-1})$, which is equal to $O(n^{\mu+2} \times 2^{\delta})$ and smaller than $O(n^{m+2})$ and $O(n^2 \times 2^D)$. This is the result because it is greater than the space complexity to construct $\Gamma_{\mathcal{F}}$.

#### A.15.2. Time complexity of Prop. 14

For each pair $(i,j)$, the time complexity to check if a match state of $\Gamma_{\mathcal{F}}$ has an anomaly (shadowing, generalization, correlation or LP-redundancy) between rules $R_i$ and $R_j$ is in $O(1)$, because we only need to check the components $i$ and $j$ of the state. Since the number of match states is in $O(\Psi_0 \times \cdots \times \Psi_{m-1})$ and the number of pairs $(i,j)$ is in $O(n^2)$, the total time complexity is in $O(n^2 \times \Psi_0 \times \cdots \times \Psi_{m-1})$, which is equal to $O(n^{\mu+2} \times 2^{\delta})$ and smaller than $O(n^{m+2})$ and $O(n^2 \times 2^D)$. This is the result because it is greater than the time complexity to construct $\Gamma_{\mathcal{F}}$.

#### A.15.3. Time complexity of Prop. 15

For each pair $(i,j)$, the time complexity to check if a match state of $\Gamma_{\mathcal{F}}$ contains an MP-redundancy anomaly between rules $R_i$ and $R_j$ is in $O(n)$, because we may need to check the components $i,\ldots,j$ of the state. Since the number of match states is in $O(\Psi_0 \times \cdots \times \Psi_{m-1})$ and the number of pairs $(i,j)$ is in $O(n^2)$, the total time complexity is in $O(n^3 \times \Psi_0 \times \cdots \times \Psi_{m-1})$, which is equal to $O(n^{\mu+3} \times 2^{\delta})$ and smaller than $O(n^{m+3})$ and $O(n^3 \times 2^D)$. This is the result because it is greater than the time complexity to construct $\Gamma_{\mathcal{F}}$.

### A.16. Proof of Proposition 16

In $\Gamma_{\mathcal{F}}$, the number of paths is in the order of the number of final states, i.e. $O(\Psi_0 \times \cdots \times \Psi_{m-1})$. The maximum space and time complexities are obtained with this maximum number of paths, which is reached when there is a single value in each field of each path. Hence, in this extreme case, to extract the rule corresponding to each path, we need to store all the fields $F^j$ of the path, for $j = 0,\ldots,m-1$, where each field is of size 1. Therefore, space and time complexities to extract one rule from $\Gamma_{\mathcal{F}}$ are in $O(m)$. Consequently, space and time complexities to extract all the rules of $\mathcal{F}_{\mathrm{m}}$ are in $O(m \times \Psi_0 \times \cdots \times \Psi_{m-1})$, which is equal to $O(m \times n^{\mu} \times 2^{\delta})$, and hence less costly than the construction of $\Gamma_{\mathcal{F}}$, due to the assumption $n > D$ (which implies $n > m$).

Since we need to construct $\Gamma_{\mathcal{F}}$ which has been proved to be more costly than the above complexities, space and time complexities to extract $\mathcal{F}_{\mathrm{m}}$ from $\mathcal{F}$ are the same as those given by Prop. 12.

### A.17. Proof of Proposition 17

#### A.17.1. Detecting Discrepancies

Let $n_1,\ldots,n_p$ be the numbers of rules of the $p$ tables, respectively, and $n = n_1 + \cdots + n_p$. Computing $\Gamma$ is as costly as computing the automaton of a single policy $\mathcal{F}$ with $n$ rules. Hence, space and time complexities to compute $\Gamma$ are in $O(n \times \Psi_0 \times \cdots \times \Psi_{m-1})$, which is equal to $O(n^{\mu+1} \times 2^{\delta})$ and smaller than $O(n^{m+1})$ and $O(n \times 2^D)$.

$O(1)$ is the space complexity to verify the equality of the actions of each match state $r$ of $\Gamma$, because we store an identifier of the state if it contains a discrepancy. For all the match states of $\Gamma$, we obtain $O(\Psi_0 \times \cdots \times \Psi_{m-1})$ which is the order of the number of match states of $\Gamma$.

$O(p)$ is the time complexity to verify the equality of the actions of each match state $r$ of $\Gamma$, because we check at most the $p$ actions of $r$. For all the match states of $\Gamma$, we obtain $O(p \times \Psi_0 \times \cdots \times \Psi_{m-1})$, because the number of match states of $\Gamma$ is in $O(\Psi_0 \times \cdots \times \Psi_{m-1})$.

Since $n > p$, the total space and time complexities to detect discrepancies are in $O(n \times \Psi_0 \times \cdots \times \Psi_{m-1})$, which is equal to $O(n^{\mu+1} \times 2^{\delta})$ and smaller than $O(n^{m+1})$ and $O(n \times 2^D)$.

#### A.17.2. Resolving discrepancies

The teams of designers agree on which action Accept or Deny is selected for each detected discrepancy. We assume that the time complexity of an agreement for a discrepancy is in $O(1)$. Since the maximum number of discrepancies is in the order of the number of match states of $\Gamma$, i.e. $O(\Psi_0 \times \cdots \times \Psi_{m-1})$, the time complexity of all agreements is in $O(\Psi_0 \times \cdots \times \Psi_{m-1})$. Since agreement is a human process, its space complexity is in $O(0)$.

From the assumption $n > D$ (hence $n > m$), we can easily check that the above time and space complexities to agree on one action for each detected discrepancy are smaller than the time and space complexities to detect discrepancies. Hence, time and space complexities to detect and resolve discrepancies are those to detect discrepancies.

### A.18. Proof of Proposition 18

The automaton $\Gamma_{\mathcal{F}}$ is executed in at most $m$ steps by going through a path of length $m$ from the initial state to a match state. In each step $j = 1 \ldots m$, we are in a state of level $j-1$ in which we have to verify the outgoing transitions and select one of them. We have seen in Section A.13 that the number of outgoing transitions from each state of level $j$ is in $\Psi_j$. Hence, the time complexity to go through the $m$ levels is in $O(\Psi_0 + \cdots + \Psi_{m-1})$. For each great field $F^j$, we have $\Psi_j = n$. By summing the $\Psi_j$ of the $\mu$ great fields, we obtain $\mu \times n$. For each small field $F^j$, we have $\Psi_j = 2^{d_j}$ which is the domain size of $F^j$. By summing the $\Psi_j$ of the small fields, we obtain $\Delta$. Hence, the total time complexity to execute $\Gamma_{\mathcal{F}}$ is in $O(\mu \times n + \Delta)$.

From $\Delta \leqslant (m-\mu) \times n$, we obtain that $O(\mu \times n + \Delta)$ is bounded by $O(m \times n)$.

The space complexity to execute $\Gamma_{\mathcal{F}}$ is in $O(1)$, because we only need to store the current state and the current verified transition.

## References


Acharya, H.B., Gouda, M.G., 2010. Projection and division: linear space verification of firewalls. In: 30th Int. Conf. on Distributed Computing Systems (ICDCS). Genova, Italy, pp. 736–743.

Acharya, H.B., Gouda, M.G., 2011. Firewall verification and redundancy checking are equivalent. In: 30th IEEE Int. Conf. on Computer Communication (INFOCOM). Shanghai, China, pp. 2123–2128.

Al-Shaer, E., Hamed, H., 2004. Modeling and management of firewall policies. IEEE Trans. Netw. Service Manage. 1 (1), 2–10.

Al-Shaer, E., Marrero, W., El-Atawy, A., Elbadawi, K., 2009. Network configuration in a box: Towards end-to-end verification of networks reachability and security. In: 17th IEEE Int. Conf. on Network Protocols (ICNP). Princeton, NJ, USA, pp. 736–743.

Bryant, R.E., 1986. Graph-based algorithms for boolean function manipulation. IEEE Trans. Comput. 35 (8).

Chen, F., Liu, A., Hwang, J., Xie, T., 2012. First step towards automatic correction of firewall policy faults. ACM Trans. Autonomous Adaptive Syst. 7 (2).

Cuppens, F., Cuppens-Boulahia, N., Garcia-Alfaro, J., Moataz, T., Rimasson, X., 2012. Handling stateful firewall anomalies. In: 27th IFIP International Information Security and Privacy Conference (SEC). Heraklion, Crete, Greece, pp. 174–186.

Elmallah, E., Gouda, M.G., 2014. Hardness of firewall analysis. In: Intern. Conf. on NETworked sYStems (NETYS). Marrakesh, Morocco.

Garcia-Alfaro, J., Cuppens, F., Cuppens-Boulahia, N., 2008. Complete analysis of configuration rules to guarantee reliable network security policies. Int. J. Inf. Security 7 (2), 103–122.

Garcia-Alfaro, J., Cuppens, F., Cuppens-Boulahia, N., Perez, S.M., Cabot, J., 2013. Management of stateful firewall misconfiguration. Comput. Security 39, 64–85.

Hoffman, D., Yoo, K., 2005. Blowtorch: a framework for firewall test automation. In: 20th IEEE/ACM Int. Conf. on Automated Software Engineering (ASE). Long Beach, California, USA, pp. 96–103.

Hu, H., Ahn, G., Kulkarni, K., 2012. Detecting and resolving firewall policy anomalies. IEEE Trans. Dependable Secure Comput. 9 (3).

Kalam, A.A.E., Baida, R.E., Balbiani, P., Benferhat, S., Cuppens, F., Deswarte, Y., Miège, A., Saurel, C., Trouessin, G., 2003. Organization based access control. In: IEEE 4th International Workshop on Policies for Distributed Systems and Networks (POLICY). Lake Come, Italy.

Kamara, S., Fahmy, S., Schultz, E., Kerschbaum, F., Frantzen, M., 2003. Analysis of vulnerabilities in internet firewalls. Comput. Security 22 (3), 214–232.

Karoui, K., Ftima, F.B., Ghezala, H.B., 2013. Formal specification, verification and correction of security policies based on the decision tree approach. Int J Data Netw Security 3 (3), 92–111.

Khoumsi, A., Krombi, W., Erradi, M., 2014. A formal approach to verify completeness and detect anomalies in firewall security policies. In: 7th Intern. Symposium on Foundations & Practice of Security (FPS). Montreal, Canada.

Krombi, W., Erradi, M., Khoumsi, A., 2014. Automata-based approach to design and analyze security policies. In: Intern. Conf. on Privacy, Security and Trust (PST). Toronto, Canada.

Lee, D., Yannakakis, M., 1996. Principles and methods of testing finite state machines – a survey. Proc. IEEE 84, 1090–1126.

Liu, A.X., Gouda, M.G., 2007. Structured firewall design. Comput. Netw. Int. J. Comput. Telecommun. Netw. 51 (4), 1106–1120.

Liu, A.X., Gouda, M.G., 2008. Diverse firewall design. IEEE Trans. Parallel Distributed Syst. 19 (9), 1237–1251.

Liu, A.X., Gouda, M.G., 2010. Complete redundancy removal for packet classifiers in TCAMs. IEEE Trans. Parallel Distribut. Syst. 21 (4), 424–437.

Lu, L., Safavi-Naini, R., Horton, J., Susilo, W., 2007. Comparing and debugging firewall rule tables. IET Inf. Security 1 (4), 143–151.

Madhavi, S., Raghu, G., 2014. Segment generation approach for firewall policy anomaly resolution. Int. J. Comput. Sci. Inf. Technol. (IJCSIT) 5 (1), 6–11.

Madhuri, M., Rajesh, K., 2013. Systematic detection and resolution of firewall policy anomalies. Int. J. Res. Comput. Commun. Technol. (IJRCCT) 2 (12), 1387–1392.

Mallouli, W., Orset, J., Cavalli, A., Cuppens, N., Cuppens, F., 2007. A formal approach for testing security rules. In: 12th ACM Symposium on Access control models and technologies (SACMAT). Sophia Antipolis, France.

Mansmann, F., Göbel, T., Cheswick, W., 2012. Visual analysis of complex firewall configurations. In: 9th International Symposium on Visualization for Cyber Security (VizSec). Seattle, WA, USA, pp. 1–8.

Pozo, S., Gasca, R., Reina-Quintero, A., Varela-Vaca, A., 2012. CONFIDDENT: a model-driven consistent and non-redundant layer-3 firewall ACL design, development and maintenance framework. J Syst Softw 85 (2), 425–457.

Shen, Q., Yang, Y., Wu, Z., Wang, D., Long, M., 2013. Securing data services: a security architecture design for private storage cloud based on HDFS. Int. J. Grid Utility Comput. 4 (4), 242–254.

Sui, G., Mejri, M., 2013. FASER (Formal and Automatic Security Enforcement by Rewriting) by BPA algebra with test. Int. J. Grid Utility Comput. 4 (2/3), 204–211.

Wool, A., 2004. A quantitative study of firewall configuration errors. Computer 37 (6), 62–67.

Yuan, L., Mai, J., Su, Z., Chen, H., Chuah, C.-N., Mohapatra, P., 2006. FIREMAN: A toolkit for firewall modeling and analysis. In: IEEE Symposium on Security and Privacy (S&P). Berkeley/Oakland, CA, USA.