A Formal Approach to Verify Completeness and Detect Anomalies in Firewall Security Policies

Ahmed Khoumsi[1(✉)], Wadie Krombi[2], and Mohammed Erradi[2]

[1] Department of Electrical and Computer Engineering, University of Sherbrooke, Sherbrooke, Canada
ahmed.khoumsi@usherbrooke.ca
[2] ENSIAS, Mohammed V University, Rabat, Morocco
krombi@yahoo.com, erradi@ensias.ma

Abstract. Security policies are a relevant solution to protect information systems from undue accesses. In this paper, we develop a formal and rigorous automata-based approach to design and analyze security policies. The interest of our approach is that it can be used as a common basis for analyzing several aspects of security policies, instead of using a distinct approach and formalism for studying each aspect. We first develop a procedure that synthesizes automatically an automaton which implements a given security policy. Then, we apply this synthesis procedure to verify completeness of security policies and detect several types of anomalies in security policies. We also study space and time complexities of the developed procedures.

1 Introduction

In this paper, we develop a formal automata-based approach to study firewall security policies described by filtering rules. Our fundamental contribution is the development of a procedure that synthesizes an automaton which implements a security policy given as a table of filtering rules. Then, we apply our automata-based approach for verifying completeness of security policies and detecting several types of anomalies in security policies. We also detect and correct an error in the commonly used definition of redundancy anomaly. Finally, we give a precise and accurate evaluation of space and time complexities of our developed procedures.

Our results are presented as 14 propositions which have all been formally proved. Due to space limitation, we do not present the proofs.

The rest of this paper is organized as follows. Section 2 presents related work on modeling and analyzing firewall security policies. In Sect. 3, we present preliminaries on firewall security policies. Section 4 contains a procedure that synthesizes an automaton which implements a security policy given as a table of filtering rules. In Sects. 5, 6, 7, and 8, we apply the synthesis procedure to verify completeness and detect various types of anomalies in a security policy. Section 9 evaluates space and time complexities of the procedures we have developed throughout Sects. 4, 5, 6, 7, and 8. We conclude in Sect. 10.

F. Cuppens et al. (Eds.): FPS 2014, LNCS 8930, pp. 221–236, 2015.
DOI: 10.1007/978-3-319-17040-4_14

2 Related Work

Many studies on security policies focus on firewalls. In [1,2], the authors present techniques and algorithms to detect anomalies in the filtering policy of a firewall. Firewall policy anomaly is defined in [3] as the existence of two or more filtering rules that match the same packet. The security policy is described by a tree called *Policy tree* in [1] and *Decision tree* in [2], where each node represents a field of a filtering rule and each link corresponds to a possible value of a field. A path from the root to a leaf describes a filtering rule of the policy. A leaf specifies the action (Accept, Deny) to be taken when the corresponding filtering rule is satisfied. The authors of [4] also study anomaly detections, but we find their categorization of anomalies less rigorous than in [1]. While in [1,2] the study of anomaly detection is limited to stateless firewalls, in [5,6] the authors provide solutions to analyze and handle stateful firewall anomalies.

In [7], the authors propose a method to detect functional discrepancies between various implementations of a security policy of a firewall. The data structure used to model the security policy is a tree called *Firewall Decision Diagram* (FDD) [8] which maps each packet to the decision to be taken by the firewall for such a packet. Each non-terminal node in a FDD specifies a test performed on a field of the packet, and each branch descending from this node corresponds to possible values of this field.

In [9], the authors introduce *Fireman*, which is a toolkit for modeling and analyzing firewalls. Fireman detects errors such as violation of a policy and inconsistency in a policy of a firewall. Fireman is implemented using *Binary Decision Diagrams* (BDD) [10] to represent predicates and perform all the set of the available operations.

In [11], the authors propose a framework to generate automatically test sequences to validate the conformance of a security policy. In such a framework, the behavior of the system is specified by an extended finite state machine [12] and the security policy is specified with the model OrBAC [13].

In [14], the authors present a visualization tool to analyze firewall configurations. To use this visualization, the firewall policy is modeled using a specific hierarchical way. The first level after the root in that model consists of the names of the different rules. The second level contains the action to execute, the third one the protocol, followed by the source and destination address. The hierarchical representation of the firewall is formed from rules data by grouping identical rule elements in a recursive manner.

In [15], the authors develop a method to verify equivalence between two security policies by extracting and comparing equivalent policies whose filtering rules are disjoint.

In each of the above works, the security policy is modeled by a data structure designed to solve a specific problem. To study a given security policy, we need to model it: by a policy tree to study anomalies, by a FDD to study discrepancies, by a BDD to study policy violation and inconsistency in a policy of a firewall, etc. This observation motivated the recent work of [16] which develops an automata-based approach to design and analyze several aspects of firewall security policies,

instead of using a distinct approach and formalism for studying each aspect. The present article adds the following contributions:

- We improve our synthesis procedure, in particular by defining formally the product of automata and labeling the transitions by intervals instead of values, which implies an enormous reduction of the number of transitions.
- We develop detection procedures, not only for shadowing anomaly, but also for generalization, correlation, and redundancy anomalies.
- We detect and correct an error in the definition commonly used for redundancy anomaly.
- We compute space and time complexities of our developed procedures.

3 Preliminaries on Firewall Security Policies

As defined in [17], firewalls are devices or programs that control the flow of network traffic between networks or hosts that employ differing security postures. The main objective of a firewall is to separate logically networks that have distinct security requirements. This is done by using rules on which communications are permitted between networks. The behavior of a firewall is controlled by its security policy which is specified by an ordered list of filtering rules defining the decisions and actions to be taken each time a packet arrives at the firewall. The packets arriving at the firewall are specified by an n-tuple of headers that are taken into account by the security policy. Examples of headers: source IP address, destination IP address, port number, protocol.

A firewall security rule (also called filtering rule) is expressed in the form: *if some conditions are satisfied, then a given action must be taken to authorize or refuse the access.* A rule can be represented as a couple (*Condition*, *Action*):

- *Condition* is a set of filtering fields $F^0, \cdots, F^{m-1}$ corresponding to respective headers $H^0, \cdots, H^{m-1}$ of a packet arriving at the firewall. Each F^j defines the set of values that are authorized to H^j. *Condition* is satisfied for a given packet P, if for every $j = 0, \cdots m-1$ the value of the header H^j of P belongs to the field F^j. We say that P matches a rule R (we may also say: R matches P) when the condition of R is satisfied for P. Otherwise, we say that P does not match R (or R does not match P).
- The field *Action* is *Accept* or *Deny*, to authorize or forbid a packet to go through the firewall, respectively.

Table 1 shows a structure of table that describes a security policy by a list of filtering rules $\mathrm{R}_1, \cdots, \mathrm{R}_n$. Each line of the table corresponds to a rule R_i $(1 \leq i \leq n)$ which is defined by: (1) its condition consisting of m filtering fields $F_i^0, \cdots, F_i^{m-1}$; and (2) its action a_i. The rules are given in decreasing priority order, that is, when a packet P arrives at the firewall, matching of P and R_1 is verified: if P matches R_1, then action a_1 is executed; if P does not match R_1, then matching of P and R_2 is verified. And so on, the process is repeated until a rule R_i matching P is found or all the rules are examined. If none of the rules

R_1 to R_n matches P, then usually a default rule is used for P. Such a default rule has its condition equal to True, that is, it matches any packet.

A security policy of a firewall is said *complete* if for any packet P arriving at the firewall, there exists at least one of its filtering rules which matches P. Note that a security policy is complete if we integrate at the end of its list of rules a default rule with condition True.

Table 1. Structure of a table of filtering rules

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 contains an example of security policy with four rules R_1 to R_4, where each line of the table corresponds to a rule. The condition of each rule R_i is defined by four fields named IPsrc, IPdst, Port and Protocol, and its action is the last field. The term *Any* in the column of a field F^j means any value in the domain of F^j. The term a.b.c.0/x is a usual notation that denotes an interval of IP addresses obtained from the 32-bit address a.b.c.0 by keeping constant the first x bits and varying the other bits. A packet P arriving at the firewall matches a rule R_i if the following four points are satisfied:

- P comes from an address belonging to IPsrc,
- P is destined to an address belonging to IPdst,
- P is transmitted through a port belonging to Port,
- P is transmitted by a protocol belonging to Protocol.

This security policy is complete because R_4 has the condition True (i.e., it matches every packet). Consider for example a packet P specified by the headers (88.120.10.15), (65.22.23.11), (25), (TCP), that is, P comes from 88.120.10.15, is destined to 65.22.23.11, and is transmitted through the port 25 by the protocol TCP. Only R_4 matches P which is accepted because the action of R_4 is *Accept*.

The example of Table 2 has been constructed specifically to illustrate the detections of all the anomalies studied in Sects. 7 and 8.

Table 2. Example of security rules

Rule	IPsrc	IPdst	Port	Protocol	Action
R_1	*Any*	81.10.10.0/24	*Any*	*Any*	*Accept*
R_2	192.168.10.0/24	81.10.10.0/24	21, 80	TCP	*Deny*
R_3	192.168.10.0/24	81.10.10.0/24	21, 85	*Any*	*Accept*
R_4	*Any*	*Any*	*Any*	*Any*	*Accept*

4 Synthesis Procedure

In this section, we develop a procedure that synthesizes an automaton which implements a security policy. The input of the procedure is a firewall security policy $\mathcal{F}$ specified by an array of n filtering rules $\mathrm{R}_1, \cdots, \mathrm{R}_n$ ordered in decreasing priority, where each R_i is defined by a condition consisting of m fields and by an action a_i. The result is an automaton $\Gamma_{\mathcal{F}}$ implementing $\mathcal{F}$ which is generated in three steps which are presented below and illustrated by the example of Table 2.

Due to space limitation, we omit theoretical details on automata. An automaton $\mathcal{A}$ consists of states related by labeled transitions, and will be represented by a graph whose nodes and arcs are the states and the transitions of $\mathcal{A}$, respectively. An arc from node q to node r labeled by the event σ represents the transition $q \xrightarrow{\sigma} r$. There is one initial state (with a small incoming arrow) and one or more final states (in bold).

We will use the following notation of intervals of integers: $[a, b] = \{x \mid a \leq x \leq b\}$, $[a, b[= \{x \mid a \leq x < b\}$, $]a, b] = \{x \mid a < x \leq b\}$ and $]a, b[= \{x \mid a < x < b\}$. For the sake of clarity, in Figs. 1 and 2 we will use the shorthands of Table 3.

Table 3. Shorthands in Figs. 1 and 2

Figure 1	$\alpha_1 = 192.168.10.0$	$\beta_1 = 81.10.10.0$
	$\alpha_2 = 192.168.10.255$	$\beta_2 = 81.10.10.255$

Figure 2	$I_1 = [0, \alpha_1[$	$I_2 = [\alpha_1, \alpha_2]$	$I_3 =]\alpha_2, 2^{32}[$	
	$J_1 = [0, \beta_1[$	$J_2 = [\beta_1, \beta_2]$	$J_3 =]\beta_2, 2^{32}[$	
	$K_1 = [0, 21[$	$K_2 =]21, 80[$	$K_3 =]80, 85[$	$K_4 =]85, 2^{16}[$

4.1 Step 1: Automaton for Each Filtering Rule

In Step 1, for each rule R_i, we synthesize an automaton $\mathcal{A}_i$ that has $m + 1$ consecutive states $q_i^0, q_i^1, \cdots, q_i^m$, where q_i^0 is the initial state and q_i^m is the final state. Every pair of consecutive states q_i^j and q_i^{j+1} $(0 \leq j < m)$ are linked by as many as transitions as the number of intervals defining the values of the field F_i^j. The transitions are labeled by the respective interval names. Then, each automaton $\mathcal{A}_i$ is completed as follows:

- A new final state E_i is added.
- In every non final state q_i^j of $\mathcal{A}_i$, we add the transitions labeled by the intervals which are complementary to the intervals labeling the transitions leading from q_i^j to q_i^{j+1}. All the added transitions lead to state E_i.

As we will explain it in Proposition 1, the intuition of E_i is that it is reached when a packet does not match rule R_i.

Step 1 is illustrated in Fig. 1 which represents the automata $(\mathcal{A}_i)_{i=1\cdots4}$ obtained from the rules of Table 2. We assume that IP addresses are in 32 bits, port numbers are in 16 bits, and only two protocols are possible: TCP and UDP. Note that some intervals consist of a single value, e.g. 21, 80, 85, TCP and UDP.

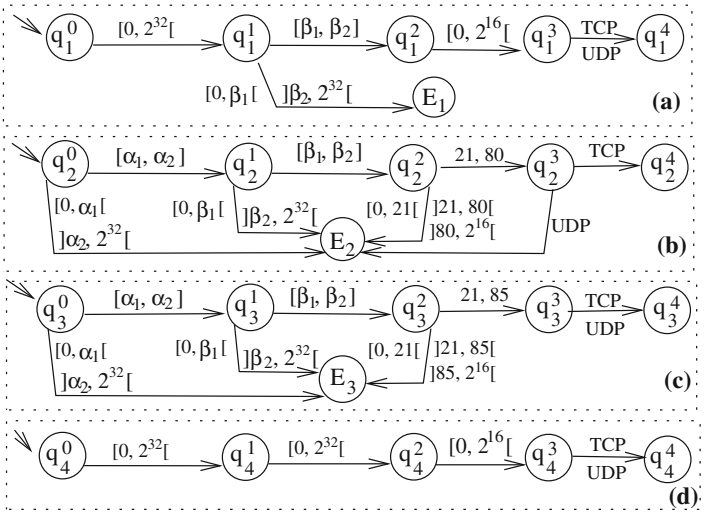

Fig. 1. Automata $(\mathcal{A}_i)_{1\cdots4}$ obtained from Table 2.

4.2 Step 2: Uniform Intervals

We say that a state is of level i if it is reached after i transitions from the initial state. Hence, each automaton $\mathcal{A}_i$ has $m+1$ states, from level 0 to level m. The automata $(\mathcal{A}_i)_{i=1\cdots n}$ constructed in Step 1 do not use the same intervals in the same level. For example, at level 0 (i.e. from q_i^0) of Fig. 1, we have the following intervals: $[0, 2^{32}[$ in $\mathcal{A}_1$ and $\mathcal{A}_4$; and $[\alpha_1, \alpha_2]$, $[0, \alpha_1[$ and $]\alpha_2, 2^{32}[$ in $\mathcal{A}_2$ and $\mathcal{A}_3$.

To be able to combine the n automata in Step 3, the objective of Step 2 is to rewrite the n automata so that at each level, they all use the same intervals. For that purpose, for each level j, the domain of the field F^j is partitioned into a set of disjoint intervals, so that every interval that labels a transition of level j of $\mathcal{A}_i$ is an union of intervals of the partition. We proceed as follows:

- For every automaton $\mathcal{A}_i$ and level j, we consider the set of the intervals labeling the transitions from state q_i^j. We construct the list L_i^j consisting of the values of the extremities of the considered intervals. The values in L_i^j are sorted in increasing order.
  Consider for example the level 0 of Fig. 1: in $\mathcal{A}_1$ and $\mathcal{A}_4$ we obtain the list $L_1^0 = L_4^0 = \langle 0, 2^{32}\rangle$ from the interval $[0, 2^{32}[$; in $\mathcal{A}_2$ and $\mathcal{A}_3$ we obtain the list $L_2^0 = L_3^0 = \langle 0, \alpha_1, \alpha_2, 2^{32}\rangle$ from the intervals $[0, \alpha_1[$, $[\alpha_1, \alpha_2]$, $]\alpha_2, 2^{32}[$.
- For every level j, the lists $(L_i^j)_{i=1\cdots n}$ are merged into a single increasing sorted list L^j. Then, we construct a set of disjoint intervals whose extremities are consecutive elements of L^j (we may have to define some single-value intervals). The obtained set of intervals is a partition of the domain of the field F^j, because the set of outgoing transitions in each state q_i^j of an automaton $\mathcal{A}_i$ constitute the whole domain of the field F^j.

For example, if we merge the above $(L_i^0)_{i=1\cdots4}$, we obtain the sorted list: $L^0 = \langle 0, \alpha_1, \alpha_2, 2^{32} \rangle$, from which we construct the three disjoint intervals I_1, I_2 and I_3, which constitute a partition of the domain $[0, 2^{32}[$ of IPsrc.

– For every automaton $\mathcal{A}_i$ and level j, each transition labeled by an interval I is replaced by several transitions labeled by the intervals of the partition that constitute I.
Consider for example the level 0 of $\mathcal{A}_1$ in Fig. 1. The transition labeled $[0, 2^{32}[$ is replaced by 3 transitions labeled I_1, I_2 and I_3, respectively.

Step 2 is illustrated in Fig. 2 which represents the automata $(\mathcal{A}_i^*)_{i=1\cdots4}$ obtained from the automata $(\mathcal{A}_i)_{i=1\cdots4}$ of Fig. 1. The constructed intervals are: I_1, I_2, I_3 at level 0; J_1, J_2, J_3 at level 1; $K_1, K_2, K_3, K_4, 21, 80, 85$ at level 2; and TCP, UDP at level 3.

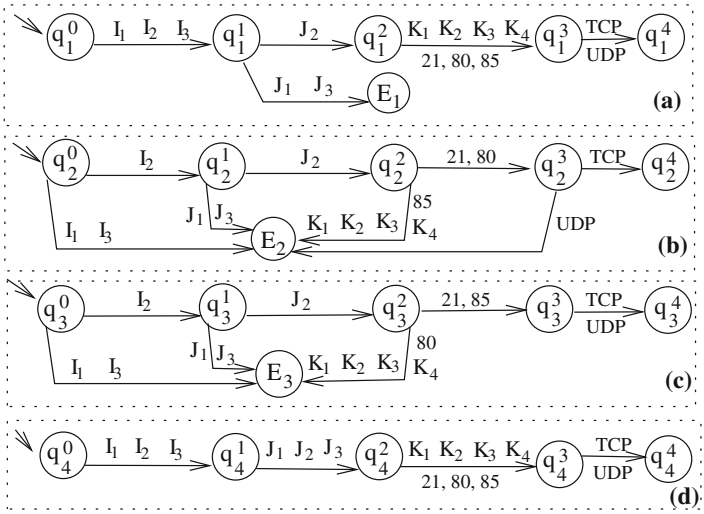

Fig. 2. Automata $(\mathcal{A}_i^*)_{1\cdots4}$ obtained from automata $(\mathcal{A}_i)_{1\cdots4}$ of Fig. 1.

4.3 Step 3: Product and Association of Actions

Consider two automata $\mathcal{A}_1^*$ and $\mathcal{A}_2^*$ constructed in Step 2. The product $\mathcal{A}_1^* \otimes \mathcal{A}_2^*$ is defined inductively as follows:

– Its initial state $\langle q_1^0, q_2^0 \rangle$ is a combination of the initial states q_1^0 and q_2^0 of $\mathcal{A}_1^*$ and $\mathcal{A}_2^*$.
– For every state $\langle r_1, r_2 \rangle$ of $\mathcal{A}_1^* \otimes \mathcal{A}_2^*$ and label σ:
  1. If $\mathcal{A}_1^*$ and $\mathcal{A}_2^*$ have $r_1 \xrightarrow{\sigma} s_1$ and $r_2 \xrightarrow{\sigma} s_2$ respectively, then $\mathcal{A}_1^* \otimes \mathcal{A}_2^*$ has the state $\langle s_1, s_2 \rangle$ and the transition $\langle r_1, r_2 \rangle \xrightarrow{\sigma} \langle s_1, s_2 \rangle$.
  2. If $\mathcal{A}_1^*$ has $r_1 \xrightarrow{\sigma} s_1$ and $r_2 = E_2$, then $\mathcal{A}_1^* \otimes \mathcal{A}_2^*$ has the state $\langle s_1, E_2 \rangle$ and the transition $\langle r_1, E_2 \rangle \xrightarrow{\sigma} \langle s_1, E_2 \rangle$.

3. If $\mathcal{A}_2^*$ has $r_2 \xrightarrow{\sigma} s_2$ and $r_1 = E_1$, then $\mathcal{A}_1^* \otimes \mathcal{A}_2^*$ has the state $\langle E_1, s_2\rangle$ and the transition $\langle E_1, r_2\rangle \xrightarrow{\sigma} \langle E_1, s_2\rangle$.

– $\mathcal{A}_1^* \otimes \mathcal{A}_2^*$ has no other state and transition.

The product of all automata $\mathcal{A}_1^*, \cdots, \mathcal{A}_n^*$ is an iteration of products of two automata. Let $\Gamma_{\mathcal{F}}$ denote the product of the n automata. Each state of $\Gamma_{\mathcal{F}}$ is defined in the form $(r_1, \cdots, r_n)$, where each r_i may be q_i^j or E_i, for $i = 1 \cdots n$ and $j = 0 \cdots m$. We define two types of final states:

– A *match* state $(r_1, \cdots, r_n)$ is such that $r_i = q_i^m$ for at least one i;
– The (unique) *no-match* state is $(E_1, \cdots, E_n)$.

To each match state $u = (r_1, \cdots, r_n)$ of $\Gamma_{\mathcal{F}}$, we associate the action a_i of the rule R_i that has the smallest index i such that $r_i \neq E_i$.

The fundamental characteristics of $\Gamma_{\mathcal{F}}$ is that it implements $\mathcal{F}$ as stated by the following proposition:

Proposition 1. *Consider a packet P arriving at the firewall, and let $H^0, \cdots, H^{m-1}$ be its headers. From the initial state of $\Gamma_{\mathcal{F}}$, we execute the m consecutive transitions labeled by the intervals $\sigma_0, \cdots, \sigma_{m-1}$ that contain $H^0, \cdots, H^{m-1}$, respectively. Let $r = \langle r_1, \cdots, r_n\rangle$ be the (final) reached state of $\Gamma_{\mathcal{F}}$. r is a match state if and only if P matches at least one rule of $\mathcal{F}$. More precisely, for every $i = 1, \cdots, n$:*

– *$r_i = q_i^m$ or $r_i = E_i$,*
– *if $r_i = q_i^m$, P matches the rule R_i,*
– *if $r_i = E_i$, P does not match R_i.*

And if r is a match state, then the action (Accept or Deny) associated to r is the action dictated by the most priority rule matching P.

Figure 3 represents the automaton $\Gamma_{\mathcal{F}}$ obtained in Step 3 from $(\mathcal{A}_i^*)_{i=1\cdots4}$ of Fig. 2. In Figs. 1 and 2, we have represented explicitly each interval labeling a transition, because that was useful to understand the operations of Step 2. In Fig. 3, we use a simpler and more symbolic notation based on the expressions *Any* and $not(X)$ where X is one or more intervals. A transition labeled *Any* from a state q_i^j is a symbolic representation of the transitions labeled by all the intervals defined at level j in Step 2. For example, the transition *Any* from $\langle q_1^2, E_2, E_3, q_4^2\rangle$ represents the 7 transitions labeled $K_1, K_2, K_3, K_4, 21, 80, 85$. $not(X)$ corresponds to *Any* without the transitions labeled by the intervals in X. For example, the transition labeled not(192.168.10.0/24) (i.e. $not(I_2)$) represents the 2 transitions labeled I_1 and I_3, and the transition labeled $not(21, 80, 85)$ represents the 4 transitions labeled K_1, K_2, K_3 and K_4. The 5 match states of $\Gamma_{\mathcal{F}}$ are in bold, there is no no-match state. Let us illustrate the fact that $\Gamma_{\mathcal{F}}$ of Fig. 3 implements $\mathcal{F}$ of Table 2. Consider a packet P which arrives at the firewall and assume that its four headers H^0 to H^3 are (192.168.10.12), (81.10.10.20), (25), (TCP), respectively. We start in the initial state $\langle q_1^0, q_2^0, q_3^0, q_4^0\rangle$. The transition labeled 192.168.10.0/24 (comprising H^0) is executed and leads to state $\langle q_1^1, q_2^1, q_3^1, q_4^1\rangle$. Then, the transition labeled 81.10.10.0/24 (comprising H^1) is

executed and leads to state $\langle q_1^2, q_2^2, q_3^2, q_4^2\rangle$. Then, the transition labeled $]21, 80[$ (comprising H^2) is executed and leads to state $\langle q_1^3, E_2, E_3, q_4^3\rangle$. Finally, the transition labeled TCP (comprising H^3) is executed and leads to the match state $\langle q_1^4, E_2, E_3, q_4^4\rangle$. Since the reached match state is associated to *Accept*, the packet is accepted.

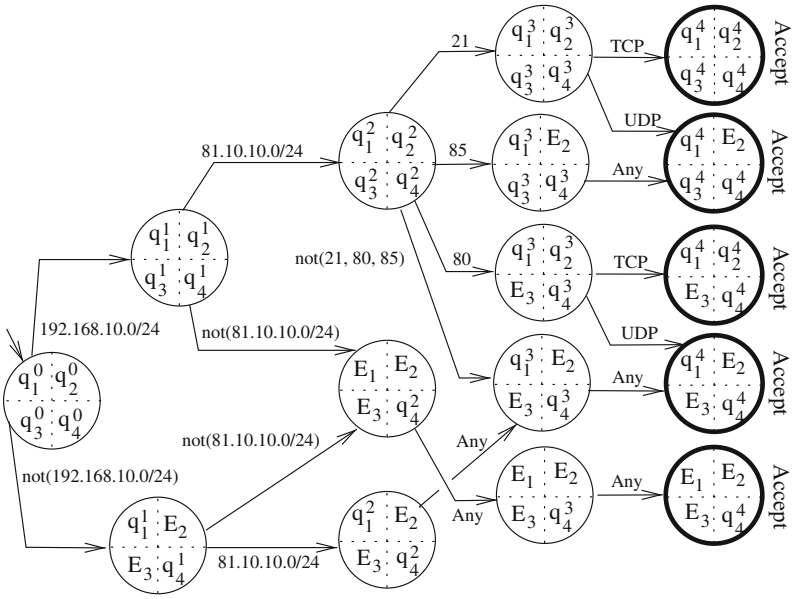

Fig. 3. Automaton $\Gamma_{\mathcal{F}}$ synthesized from automata $(\mathcal{A}_i^*)_{1\cdots4}$ of Fig. 2.

In the following four Sects. 5, 6, 7, and 8, we show how our 3-step synthesis procedure can be applied for verifying completeness of security policies and detecting several types of anomalies in security policies.

5 Verifying Completeness

We consider a security policy $\mathcal{F}$ defined by a table of n filtering rules R_1 to R_n and its corresponding automaton $\Gamma_{\mathcal{F}}$. A firewall security policy is said *complete* if every packet reaching the firewall matches at least one of the rules R_1 to R_n. From Proposition 1, we obtain:

Proposition 2. *A security policy $\mathcal{F}$ is complete if and only if its automaton $\Gamma_{\mathcal{F}}$ has no no-match state.*

For example, the security policy of Table 2 is complete because all final states of $\Gamma_{\mathcal{F}}$ in Fig. 3 are match states. If we remove rule R_4, we obtain an incomplete security policy because the corresponding automaton contains the no-match state, which is reached for any packet whose header IPdst does not belong to 81.10.10.0/24.

6 Anomaly Categorization, General Anomaly Detection

6.1 Categories of Anomalies

Recall that an anomaly in a firewall security policy is defined as the existence of two or more filtering rules matching the same packet. From this basic definition, and as noted in [18], several related work has categorized different types of firewall policy anomalies [1,9,19]. We propose to classify anomalies in two categories:

Anomaly with Conflict: It is an anomaly where the filtering rules matching the same packet are associated to *different* actions. Typical anomalies of this category are shadowing, generalization, and correlation anomalies, which we study in Sects. 7.1, 7.2, and 7.3.

Anomaly Without Conflict: It is an anomaly where the filtering rules matching the same packet are associated to the *same* action. Typical anomaly of this category is redundancy anomaly, which we study in Sect. 8.

6.2 Detecting General Anomalies

By *general anomaly* we mean any anomaly without considering its category. From Proposition 1, we obtain:

Proposition 3. *A security policy $\mathcal{F}$ contains a general anomaly if and only if the automaton $\Gamma_{\mathcal{F}}$ has at least one match state that contains two or more q_i^m.*

Intuitively, the presence of several q_i^m in a match state means the existence of packets that are accepted by several filtering rules. For example, the security policy of Table 2 contains several general anomalies, because the automaton of Fig. 3 has four match states with two or more q_i^m.

The nonexistence of general anomaly implies obviously the nonexistence of any category of anomalies. On the other hand, the existence of general anomaly is a symptom (but not a guarantee) of the existence of specific anomalies. In the following two Sects. 7 and 8, we show how to detect anomalies with conflict and without conflict, respectively.

7 Detecting Anomalies with Conflict

7.1 Detecting Shadowing Anomaly

A rule R_j is shadowed by a preceding rule R_i (i.e. $i < j$) if, on the one hand R_i matches all the packets that match R_j, and on the other hand actions of R_i and R_j are distinct [19]. In this case, all the packets that R_j intends to deny (resp. accept) are accepted (resp. denied) by R_i; thus, R_j never takes effect. It is important to discover shadowed rules and alert the administrator to correct this error by reordering or removing these rules [1]. Based on this definition and Proposition 1, we obtain:

Proposition 4. *In a security policy $\mathcal{F}$, rule R_j is shadowed by rule R_i if and only if the following three conditions hold: (1) $i < j$; (2) $a_i \neq a_j$; and (3) for every match state $\langle r_1, \cdots, r_n \rangle$ of $\Gamma_{\mathcal{F}}$, if $r_j = q_j^m$ then $r_i = q_i^m$.*

For example, in Table 2, R_2 is shadowed by R_1 because: $1 < 2$, $a_1 = Accept \neq a_2 = Deny$, and $\Gamma_{\mathcal{F}}$ of Fig. 3 has no match state with q_2^4 and without q_1^4. Consequently, R_2 never takes effect and hence its action *Deny* is never taken. Assume that the administrator decides to remove this anomaly by switching the orders of rules R_1 and R_2. After this switch, it is not necessary to construct a new automaton $\Gamma_{\mathcal{F}}$ from the beginning. Since R_1 becomes R_2 and vice versa, we have just to switch between r_1 and r_2 in all states to give priority to R_2 over R_1, and then to re-assign actions to match states. As a result, in Fig. 3 we will have the action *Deny* associated to the first and third match states from the top, and the action *Accept* associated to the three other match states.

7.2 Detecting Generalization Anomaly

A rule R_j generalizes a preceding rule R_i (i.e. $i < j$) if, on the one hand R_j matches more packets than R_i, and on the other hand actions of R_i and R_j are distinct [19]. Based on this definition and Proposition 1, we obtain:

Proposition 5. *In a security policy $\mathcal{F}$, rule R_j generalizes rule R_i if and only if the following four conditions hold: (1) $i < j$; (2) $a_i \neq a_j$; (3) for every match state $\langle r_1, \cdots, r_n \rangle$ of $\Gamma_{\mathcal{F}}$, if $r_i = q_i^m$ then $r_j = q_j^m$; and (4) there exists a match state $\langle r_1, \cdots, r_n \rangle$ of $\Gamma_{\mathcal{F}}$ such that $r_i = E_i$ and $r_j = q_j^m$.*

For example, in Table 2, R_4 generalizes R_2 because: $2 < 4$, $a_2 = Deny \neq a_4 = Accept$, and $\Gamma_{\mathcal{F}}$ of Fig. 3 has no match state with q_2^4 and without q_4^4 and has 3 match states with E_2 and q_4^4.

We have seen in Sect. 7.1 that R_2 is shadowed by R_1 and that the shadowing anomaly can be removed by switching the orders of R_1 and R_2. It can be easily checked that after this reordering, the shadowing anomaly is transformed into a generalization anomaly: R_1 (which becomes R_2) generalizes R_2 (which becomes R_1).

7.3 Detecting Correlation Anomaly

Rules R_i and R_j correlate if their actions are distinct and:

- there exist packets that match R_i and not R_j,
- there exist packets that match R_j and not R_i,
- there exist packets that match both R_i and R_j [19].

Based on this definition and Proposition 1, we obtain:

Proposition 6. *In a security policy $\mathcal{F}$, rules R_i and R_j correlate if and only if the following five conditions hold: (1) $i \neq j$; (2) $a_i \neq a_j$; (3) there exists a match state $\langle r_1, \cdots, r_n \rangle$ of $\Gamma_{\mathcal{F}}$ such that $r_i = E_i$ and $r_j = q_j^m$; (4) there exists a match state $\langle r_1, \cdots, r_n \rangle$ of $\Gamma_{\mathcal{F}}$ such that $r_i = q_i^m$ and $r_j = E_j$; and (5) there exists a match state $\langle r_1, \cdots, r_n \rangle$ of $\Gamma_{\mathcal{F}}$ such that $r_i = q_i^m$ and $r_j = q_j^m$.*

For example, in Table 2, rules R_2 and R_3 correlate because: $2 \neq 3$, $a_2 = Deny \neq a_3 = Accept$, and the automaton $\Gamma_{\mathcal{F}}$ of Fig. 3 has the three match states $\langle q_1^4, q_2^4, q_3^4, q_4^4\rangle$, $\langle q_1^4, E_2, q_3^4, q_4^4\rangle$, $\langle q_1^4, q_2^4, E_3, q_4^4\rangle$.

8 Redefining and Detecting Redundancy Anomaly

In the present section, we study redundancy anomaly which has been defined in [1] as follows: A rule R_i is redundant to a rule R_j if: (1) R_j is associated to the same action as R_i and matches the same or more packets than R_i, and (2) the security policy will not be affected if R_i is removed.

We have detected that the formal definition of redundancy anomaly given by [1] is incompatible with this informal definition. Indeed, there are situations where a rule is diagnosed by [1] as redundant to another rule, while its removal affects the security policy.

We will distinguish two cases of redundancy, LP-redundancy and MP-redundancy, which are compatible with the informal definition. We will show how to detect each of them. Then, we will show why the formal definition of [1] is problematic.

8.1 Definition and Detection of LP-Redundancy Anomaly

Redundancy of the Least Priority (LP) of the Two Rules: R_j is LP-redundant to R_i if the following three conditions hold: (1) $i < j$; (2) $a_i = a_j$; and (3) R_i matches all the packets matching R_j.

The intuition of LP-redundancy is that if R_j has less priority than R_i and matches less packets than R_i, then R_j is obviously useless and can be removed.

Detecting LP-Redundancy Anomaly: Based on the definition of LP-redundancy and Proposition 1, we obtain:

Proposition 7. *In a security policy $\mathcal{F}$, rule R_j is LP-redundant to R_i if the following three conditions hold: (1) $i < j$; (2) $a_i = a_j$; and (3) for every match state $\langle r_1, \cdots, r_n\rangle$ of $\Gamma_{\mathcal{F}}$, if $r_j = q_j^m$ then $r_i = q_i^m$;*

For example, in Table 2, R_3 is LP-redundant with R_1 because: $1 < 3$, $a_1 = a_3 = Accept$, and in the automaton $\Gamma_{\mathcal{F}}$ of Fig. 3, the two match states that contain q_3^4 contain also q_1^4.

8.2 Definition and Detection of MP-Redundancy Anomaly

Redundancy of the Most Priority (MP) of the Two Rules: R_i is MP-redundant to R_j if the following four conditions hold: (1) $i < j$; (2) $a_i = a_j$; (3) R_j matches more packets than R_i; (4) there exists no rule R_k satisfying the following three sub-conditions: (4.a) $i < k < j$, (4.b) $a_k \neq a_i$, and (4.c) there exist packets matching both R_i and R_k.

The intuition of MP-redundancy is that if R_i has more priority than R_j but matches less packets than R_j, then R_i is useless if its removal does not give the chance to another rule R_k to take an effect different from the effect of R_i.

The following proposition expresses MP-redundancy by using the three anomalies with conflict.

Proposition 8. *R_i is MP-redundant to R_j if: $i < j$; $a_i = a_j$; R_j matches more packets than R_i; and there exists no rule R_k between R_i and R_j satisfying one of the following three conditions: R_k is shadowed by R_i, R_k generalizes R_i, or R_k and R_i correlate.*

Detecting MP-Redundancy Anomaly: Based on the definition of MP-redundancy and Proposition 1, we obtain:

Proposition 9. *In a security policy $\mathcal{F}$, rule R_i is MP-redundant to R_j if the following five conditions hold: (1) $i < j$; (2) $a_i = a_j$; (3) for every match state $\langle r_1, \cdots, r_n \rangle$ of $\Gamma_{\mathcal{F}}$: if $r_i = q_i^m$, then $r_j = q_j^m$; (4) there exists a match state $\langle r_1, \cdots, r_n \rangle$ of $\Gamma_{\mathcal{F}}$ such that: $r_i = E_i$ and $r_j = q_j^m$; and (5) for every k such that $i < k < j$ and $a_k \neq a_i$, there exists no match state $\langle r_1, \cdots, r_n \rangle$ of $\Gamma_{\mathcal{F}}$ such that: $r_i = q_i^m$ and $r_k = q_k^m$.*

For example, in Table 2, R_3 is MP-redundant with R_4 because: $3 < 4$, $a_3 = a_4 =$ *Accept*, and in the automaton $\Gamma_{\mathcal{F}}$ of Fig. 3: the two match states that contain q_3^4 contain also q_4^4, there exist three match states containing E_3 and q_4^4, and there exists no k between 3 and 4.

8.3 Difference with [1]

The difference between our definition of MP-redundancy and the definition of [1] is in the condition related to R_k. In our definition, the condition is that for every rule R_k between R_i and R_j that has a different action than R_i, the set of packets matching R_i must be *disjoint* with the set of packets matching R_k. In the definition of [1], the condition is weaker because its permits that the set of packets matching R_k be included in or equal to the set of packets matching R_i. In other words, we obtain the definition of [1] by removing the line "R_k is shadowed by R_i" in Proposition 8. Consequently, there are situations where a rule is diagnosed by [1] as redundant to another rule while in reality its removal affects the security policy.

To illustrate the problematic definition of MP-redundancy of [1], let us consider R_1 and R_4 of Table 2. With our definition, R_1 is not MP-redundant to R_4, because R_1 accepts all the packets accepted by R_2, while R_2 is between R_1 and R_4 and has a different action than R_1. But with the definition of [1], R_1 is MP-redundant to R_4, because the definition of [1] accepts that between R_1 and R_4 there exists a rule (R_2) which is shadowed by R_1. This is problematic because in reality removing R_1 affects the security policy.

9 Space and Time Complexities

Recall that n and m are the number of rules and fields, respectively. We call *great field* a field whose domain contains more than n values, and *small field* a field whose domain contains at most n values. Consider for example the four fields IPsrc, IPdst, Port and Protocol and assume $n = 1000$. IPsrc, IPdst and Port are great fields, because their domains contain 2^{32}, 2^{32} and 2^{16} values, respectively, hence more than 1000 values. Protocol is a small field, because its domain contains much less than 1000 values (the number of considered protocols is negligible to 1000). In addition to n and m, we will use the following parameters:

d_i = number of bits necessary to code the values of field F^i, for $i = 0, \cdots, m-1$, hence 2^{d_i} is the number of possible values of F^i;
D = sum of the number of bits to code all the fields, i.e. $D = d_0 + \cdots + d_{m-1}$;
μ = number of great fields, hence $\mu \leq m$;
δ = sum of the number of bits to code the small fields, hence $\delta \leq D$.

In our computation of complexities, we assume that $d_i \geq 1$ (i.e. several possible values for each field), $n > D$ (hence $n > m$) and $2^n > n^m$ which is realistic when we have hundreds or thousands of filtering rules.

For example, for the $m = 4$ fields IPsrc, IPdst, Port and Protocol, we have used $d_0 = d_1 = 32$ (each IPsrc and IPdst is coded in 32 bits), $d_2 = 16$ (Port is coded in 16 bits), $d_3 = 1$ (Protocol is coded in 1 bit since we consider only TCP and UDP), and $D = 32 + 32 + 16 + 1 = 81$. For $81 < n < 2^{16}$, the above assumptions (all $d_i \geq 1$, $n > 81$ and $2^n > n^4$) are obviously satisfied. IPsrc, IPdst and Port are great fields (theirs domains have more than n values), and Protocol is a small field (its domain has less than n values). We obtain $\mu = 3$ (number of great fields) and $\delta = d_3 = 1$ (1 bit is used to code the unique small field Protocol).

Proposition 10. *The space and time complexities of the synthesis procedure are in* $O(n^{\mu+1} \times 2^{\delta})$*, which is bounded by both* $O(n^{m+1})$ *and* $O(n \times 2^D)$.

Proposition 11. *The space and time complexities for verifying completeness of a security policy are in* $O(n^{\mu+1} \times 2^{\delta})$*, which is bounded by both* $O(n^{m+1})$ *and* $O(n \times 2^D)$.

Proposition 12. *The space and time complexities for detecting general anomalies in a security policy are in* $O(n^{\mu+1} \times 2^{\delta})$*, which is bounded by both* $O(n^{m+1})$ *and* $O(n \times 2^D)$.

Proposition 13. *The space and time complexities for detecting anomalies with conflict (shadowing, generalization, correlation) and LP-redundancy anomalies are in* $O(n^{\mu+2} \times 2^{\delta})$*, which is bounded by both* $O(n^{m+2})$ *and* $O(n^2 \times 2^D)$.

Proposition 14. *The space complexity for detecting MP-redundancy anomalies is in* $O(n^{\mu+2} \times 2^{\delta})$*, which is bounded by both* $O(n^{m+2})$ *and* $O(n^2 \times 2^D)$*. The time complexity for detecting MP-redundancy anomalies is in* $O(n^{\mu+3} \times 2^{\delta})$*, which is bounded by both* $O(n^{m+3})$ *and* $O(n^3 \times 2^D)$.

By using results in [20], the authors of [21] prove that several fundamental problems encountered in analysis and design of firewalls are NP-hard. In our context, their result is that the time complexity is at least in $O(n \times 2^D)$. Our contribution here is that the expression $O(n \times 2^D)$ is an upper bound of our more precise expression $O(n^{\mu+1} \times 2^{\delta})$ which shows explicitly the influence of the size of fields (through μ and δ) on the complexity.

10 Conclusion and Future Work

We have first proposed a procedure that synthesizes an automaton which implements a security policy. This synthesis procedure can be a common basis to design and analyze several aspects of firewall security policies, instead of using a distinct approach and formalism for studying each aspect. We have demonstrated this statement by applying our synthesis procedure for verifying completeness of security policies and detecting several types of anomalies in security policies. Another contribution is that we have identified and corrected an error in the commonly used definition of redundancy anomaly. Last but not least, we have evaluated precisely space and time complexities of our developed procedures.

As near future work, we intend to use our automata-based approach to design security policies that can adapt dynamically to the filtered traffic. The approach we will adopt is that from a given security policy, we extract a new equivalent policy whose filtering rules can be put in any order without influencing the semantics of the policy. When the filtered traffic varies, the filtering rules will be reordered dynamically in order to keep a low average delay of determining the rule that is applied for an arriving packet.

References

1. Al-Shaer, E., Hamed, H.: Modeling and management of firewall policies. IEEE Trans. Netw. Serv. Manage. **1**(1), 2–10 (2004)
2. Karoui, K., Ben Ftima, F., Ben Ghezala, H.: Formal specification, verification and correction of security policies based on the decision tree approach. Int. J. Data Netw. Secur. **3**(3), 92–111 (2013)
3. Madhuri, M., Rajesh, K.: Systematic detection and resolution of firewall policy anomalies. Int. J. Res. Comput. Commun. Technol. (IJRCCT) **2**(12), 1387–1392 (2013)
4. Chen, Z., Guo, S., Duan, R.; Research on the anomaly discovering algorithm of the packet filtering rule sets. In 1st International Conference on Pervasive Computing, Signal Processing and Applications (PCSPA), Harbin, China, pp. 362–366, September 2010
5. Garcia-Alfaro, J., Cuppens, F., Cuppens-Boulahia, N., Martinez Perez, S., Cabot, J.: Management of stateful firewall misconfiguration. Comput. Secur. **39**, 64–85 (2013)
6. Cuppens, F., Cuppens-Boulahia, N., Garcia-Alfaro, J., Moataz, T., Rimasson, X.: Handling stateful firewall anomalies. In: Gritzalis, D., Furnell, S., Theoharidou, M. (eds.) SEC 2012. IFIP AICT, vol. 376, pp. 174–186. Springer, Heidelberg (2012)

7. Liu, A.X., Gouda, M.G.: Diverse firewall design. IEEE Trans. Parallel Distrib. Syst. **19**(9), 1237–1251 (2008)
8. Liu, A.X., Gouda, M.G.: Structured firewall design. Comput. Netw. Int. J. Comput. Telecommun. Netw. **51**(4), 1106–1120 (2007)
9. Yuan, L., Mai, J., Su, Z., Chen, H., Chuah, C.-N., Mohapatra, P.: FIREMAN: a toolkit for firewall modeling and analysis. In: IEEE Symposium on Security and Privacy (S&P), Berkeley/Oakland, May 2006
10. Bryant, R.E.: Graph-based algorithms for boolean function manipulation. IEEE Trans. Comput. **35**(8), 677–691 (1986)
11. Mallouli, W., Orset, J., Cavalli, A., Cuppens, N., Cuppens, F.: A formal approach for testing security rules. In: 12th ACM Symposium on Access Control Models and Technologies (SACMAT), Sophia Antipolis, France, June 2007
12. Lee, D., Yannakakis, M.: Principles and methods of testing finite state machines - a survey. Proc. IEEE **84**, 1090–1126 (1996)
13. El Kalam, A.A., El Baida, R., Balbiani, P., Benferhat, S., Cuppens, F., Deswarte, Y., Miège, A., Saurel, C., Trouessin, G.: Organization based access control. In: IEEE 4th International Workshop on Policies for Distributed Systems and Networks (POLICY), Lake Come, Italy, June 2003
14. Mansmann, F., Göbel, T., Cheswick, W.: Visual analysis of complex firewall configurations. In: 9th International Symposium on Visualization for Cyber Security (VizSec), Seattle, pp. 1–8, October 2012
15. Lu, L., Safavi-Naini, R., Horton, J., Susilo, W.: Comparing and debugging firewall rule tables. IET Inf. Secur. **1**(4), 143–151 (2007)
16. Krombi, W., Erradi, M., Khoumsi, A.: Automata-based approach to design and analyze security policies. In: International Conference on Privacy, Security and Trust (PST), Toronto, Canada (2014)
17. Scarfone, K., Hauffman, P.: Guidelines on Firewalls and Firewall Policy, Recommendations of the National Institute of Standards and Technology (NIST). Special Publication 800–41, Revision 1, 2–1, September 2009
18. Madhavi, S., Raghu, G.: Segment generation approach for firewall policy anomaly resolution. Int. J. Comput. Sci. Inf. Technol. (IJCSIT) **5**(1), 6–11 (2014)
19. Hu, H., Ahn, G., Kulkarni, K.: Detecting and resolving firewall policy anomalies. IEEE Trans. Dependable Secure Comput. **9**(3), 318–331 (2012)
20. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. A.W.H. Freeman, San Francisco (1979)
21. Elmallah, E., Gouda, M.G.: Hardness of firewall analysis. In: International Conference on NETworked sYStems (NETYS), Marrakesh, Morocco, May 2014