An Approach to Resolve NP-Hard Problems of Firewalls

Ahmed Khoumsi[1(✉)], Mohamed Erradi[2], Meryeme Ayache[2], and Wadie Krombi[2]

[1] Department of Electrical and Computer Engineering, University of Sherbrooke, Sherbrooke, Canada
Ahmed.Khoumsi@USherbrooke.ca
[2] ENSIAS, Mohammed V University, Rabat, Morocco

Abstract. Firewalls are a common solution to protect information systems from intrusions. In this paper, we apply an automata-based methodology to resolve several NP-Hard problems which have been shown in the literature to be fundamental for the study of firewall security policies. We also compute space and time complexities of our resolution methods.

1 Introduction

An essential component of a firewall is its security policy that consists of a table of filtering rules specifying which packets are accepted and which ones are discarded from the network [1]. Designing and analyzing a firewall are not easy tasks when we have thousands of filtering rules as is usually the case. To perform such tasks properly, one requires to solve thousands of instances of known fundamental NP-Hard problems identified in [2]. Recognizing the importance of these problems, their solutions can significantly enhance the ability to design and analyze firewalls. Henceforth, the terms *policy* and *rule* denote "firewall security policy" and "filtering rule", respectively.

In this work, we apply the automata-based methodology of [3] to resolve the 13 NP-Hard problems of [2]. The basic principle of the approach is to describe policies as automata and then to develop analysis methods applicable to automata. We also evaluate time and space complexities of the 13 resolutions.

The paper is organized as follows. Section 2 presents related work on analyzing policies. Section 3 contains preliminaries on policies. Section 4 introduces the methodology of [3]. In Sects. 5 and 6, we resolve the 13 NP-Hard problems of [2] by using the methodology of [3]. Section 7 evaluates the space and time complexities of the 13 resolutions. We conclude in Sect. 8.

2 Related Work

Previous work on firewalls, such as [4–6], provide practical analysis algorithms, while [7–11] provide fundamental analysis algorithms with estimations of their time complexities. [2] proves that many firewall analysis problems are NP-Hard.

P.A. Abdulla and C. Delporte-Gallet (Eds.): NETYS 2016, LNCS 9944, pp. 229–243, 2016.
DOI: 10.1007/978-3-319-46140-3_19

[12,13] present techniques to detect anomalies in a policy. An anomaly is defined in [14] as the existence of several rules that match the same packet. A policy is described by a *Policy tree* in [12] and a *Decision tree* in [13].

[15,16] provide solutions to analyze and handle *stateful* firewall anomalies.

[11] proposes a method to detect discrepancies between implementations of a policy. The policy is modeled by a *Firewall Decision Diagram* (FDD) [17] which maps each packet to the decision taken by the firewall for such a packet.

[18] introduces *Fireman*, which is a toolkit that permits to detect errors such as violation of a policy and inconsistency in a policy. Fireman is implemented using *Binary Decision Diagrams* (BDD) [19].

[20] generates test sequences to validate the conformance of a policy, where the system's behavior is specified by an extended finite state machine [21] and the policy is specified with the model OrBAC [22].

[23] verifies equivalence between two policies by extracting and comparing equivalent policies whose filtering rules are disjoint.

[24] presents a visualization tool to analyze firewall configurations, where the policy is modeled in a specific hierarchical way.

In each of the above works, a specific formalism is used to solve a specific problem. A policy is modeled: by a policy tree to study anomalies, by a FDD to study discrepancies, by a BDD to study policy violation and inconsistency, etc. This observation motivated the work of [3,25], where automata are used to study several aspects of policies. The main contribution of the present article is the resolution of the 13 NP-Hard problems of [2] by using the methodology of [3]. Space and time complexities of the 13 resolutions are provided.

3 Preliminaries

The behavior of a firewall is controlled by its policy which consists of a list of rules defining the actions to take each time a packet tries to cross the firewall. The packets are specified by an n-tuple of headers that are taken into account by the policy. A rule is in the form: *if some conditions are satisfied, then a given action must be taken to authorize or refuse the access.* Therefore, a rule can be specified as (*Condition*, *Action*), where:

- *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^i defines the set of values that are authorized to H^i. *Condition* is satisfied for a packet P, if for every $i = 0 \cdots m-1$ the value of H^i of P belongs to F^i. We say that P matches a rule R (or R matches P) when the condition of R is satisfied for P. Otherwise, P does not match R (or R does not match P).
- *Action* is Accept or Deny, to authorize or forbid a packet to go through the firewall, respectively.

The rules are denoted $\mathrm{R}_1, \mathrm{R}_2, \cdots$, and their actions are denoted $a_1, a_2, \cdots$ respectively. The rules are 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.

An *accept-rule* (resp. *deny-rule*) is a rule whose action is Accept (resp. Deny). An *all-rule* is a rule whose condition is TRUE, i.e. it matches all packets. We also combine the definitions to obtain *all-accept-rule* and *all-deny-rule*. A policy is said *complete* if every packet matches at least one of its rules.

Table 1 contains an example of policy. The condition of each rule R_i is defined by four fields: IPsrc, IPdst, Port and Protocol, and its action is in the last column. 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 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: 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, and P is transmitted by a protocol belonging to Protocol.

Table 1. Example of rules

Rule	IPsrc	IPdst	Port	Protocol	Action
R_1	*Any*	212.217.65.201	80	TCP	Accept
R_2	192.168.10.0/24	81.10.10.0/24	*Any*	*Any*	Deny
R_3	194.204.201.0/28	212.217.65.202	21	*Any*	Accept
R_4	192.168.10.0/24	*Any*	*Any*	*Any*	Accept

4 Synthesis Procedure

The basis of the methodology of [3] is a procedure that synthesizes an automaton from a policy. The input of the procedure is a policy $\mathcal{F}$ specified by n rules $R_1, \cdots, R_n$ ordered in decreasing priority. The result is an automaton $\Gamma_{\mathcal{F}}$ implementing $\mathcal{F}$.

The synthesis procedure is presented in detail in [3]. In this section, we illustrate it by the example of policy $\mathcal{F}$ of Table 1, for which the synthesis procedure generates the automaton $\Gamma_{\mathcal{F}}$ of Fig. 1. The states are organized by levels, where the states of level j are reached after j transitions from the initial state (represented with a small incoming arrow). A transition is said of level j if it links a state of level j to a state of level $j+1$. Transitions of level j are labeled by sets of values of the field F^j. There are two types of final states (represented in bold):

– A *match* state is associated to the action Accept or Deny (noted A or D in the figure). There may be one or several match states in a synthesized automaton. The automaton of Fig. 1 has 5 match states.

– A *no-match* state is indicated by a star $*$. There may be at most one no-match state in a synthesized automaton. The automaton of Fig. 1 has 1 no-match state.

In Fig. 1 and subsequent figures, some transitions are labeled in the form *Any* or $not(X)$, where X is one or more sets of values. A label *Any* in a transition of level j denotes the whole domain of values of the field F^j. A label $not(X)$ in a transition of level j denotes the complementary of X in the domain of F^j.

The fundamental characteristics of $\Gamma_{\mathcal{F}}$ is that it implements $\mathcal{F}$ as stated by the following theorem taken from [3]:

Theorem 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 sets $\sigma_0, \cdots, \sigma_{m-1}$ that contain $H^0, \cdots, H^{m-1}$, respectively. Let r be the (final) reached state of $\Gamma_{\mathcal{F}}$:*

– *r is a match state iff*[1] *P matches at least one rule of $\mathcal{F}$.*
– *If r is a match state, then the action (*Accept *or* Deny*) associated to r is the action of the most priority rule matching P.*

Let us illustrate the fact that $\Gamma_{\mathcal{F}}$ of Fig. 1 implements $\mathcal{F}$ of Table 1. Consider a packet P which arrives at the firewall and assume that its four headers H^0 to H^3 are $(192.168.10.12), (212.217.65.201), (25), (\text{TCP})$, respectively. We start in the initial state $\langle 0 \rangle$. The transition labeled 192.168.10.0/24 (comprising H^0) is executed and leads to state $\langle 1 \rangle$. Then, the transition labeled 212.217.65.201 (comprising H^1) is executed and leads to state $\langle 4 \rangle$. Then, the transition labeled $not(80)$ (comprising H^2) is executed and leads to state $\langle 10 \rangle$. Finally, the transition labeled *Any* (comprising H^3) is executed and leads to the second match state. Since the reached match state is associated to Accept, the packet is accepted.

Consider now a packet whose four headers H^0 to H^3 are $(194.204.201.20)$, $(212.217.65.201), (25), (\text{TCP})$, respectively. We start in the initial state $\langle 0 \rangle$. The transition labeled 194.204.201.0/28 (comprising H^0) is executed and leads to state $\langle 2 \rangle$. Then, the transition labeled 212.217.65.201 (comprising H^1) is executed and leads to state $\langle 8 \rangle$. Then, the transition labeled $not(80)$ (comprising H^2) is executed and leads to the no-match state $\langle * \rangle$. Therefore, no rule of the policy matches such a packet.

5 Resolution of FC, FA and SP

Let us demonstrate the applicability of our synthesis procedure for the resolution of 5 of the 13 problems of [2]: **FC**, **FA-d**, **FA-a**, **SP-d** and **SP-a**.

[1] iff means: if and only if.

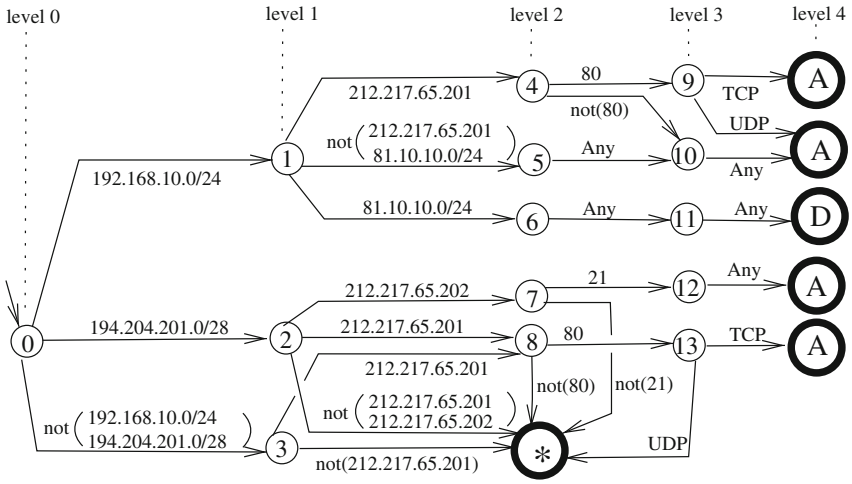

Fig. 1. Automaton synthesized from the policy of Table 1.

5.1 Resolution of Firewall Completeness (FC) Problem

Firewall Completeness (**FC**) problem is to design an algorithm that takes as input a policy $\mathcal{F}$ and determines whether every packet arriving at the firewall matches at least one of the filtering rules of $\mathcal{F}$. From Theorem 1, we obtain:

Proposition 1 (FC). *A policy $\mathcal{F}$ is complete iff its automaton $\Gamma_{\mathcal{F}}$ has no no-match state.*

Therefore, **FC** problem of $\mathcal{F}$ is solved by constructing the automaton $\Gamma_{\mathcal{F}}$ and verifying if it contains the no-match state. For example, the policy $\mathcal{F}$ of Table 1 is incomplete, because $\Gamma_{\mathcal{F}}$ of Fig. 1 contains the no-match state.

5.2 Resolution of Firewall Adequacy Problems: FA-d, FA-a

There are two Firewall Adequacy (**FA**) problems:

FA-d: to design an algorithm that takes as input a policy $\mathcal{F}$ and determines whether there exists at least one packet which is denied by $\mathcal{F}$.

FA-a: to design an algorithm that takes as input a policy $\mathcal{F}$ and determines whether there exists at least one packet which is accepted by $\mathcal{F}$.

From Theorem 1, we obtain:

Proposition 2 (FA-d). *A policy $\mathcal{F}$ denies one or more packets iff its automaton $\Gamma_{\mathcal{F}}$ has one or more match states associated to the action* Deny.

Proposition 3 (FA-a). *A policy $\mathcal{F}$ accepts one or more packets iff its automaton $\Gamma_{\mathcal{F}}$ has one or more match states associated to the action* Accept.

Therefore, **FA-d** (resp. **FA-a**) problem of $\mathcal{F}$ is solved by constructing the automaton $\Gamma_{\mathcal{F}}$ and verifying if it contains match state(s) associated to the action Deny (resp. Accept). For example, the policy $\mathcal{F}$ of Table 1 denies and accepts packets, because $\Gamma_{\mathcal{F}}$ of Fig. 1 contains 1 match state with action Deny and 4 match states with action Accept.

5.3 Resolution of Slice Probing Problems: SP-d, SP-a

We first define the following two types of policies:

Discard slice: it is a policy consisting of zero or more accept rules followed by a last all-deny-rule.

Accept slice: it is a policy consisting of zero or more deny rules followed by a last all-accept-rule.

There are two Slice Probing (**SP**) problems:

SP-d: to design an algorithm that takes as input a discard slice $\mathcal{F}$ and determines whether there exists at least one packet which is denied by $\mathcal{F}$.

SP-a: to design an algorithm that takes as input an accept slice $\mathcal{F}$ and determines whether there exists at least one packet which is accepted by $\mathcal{F}$.

We have two ways to solve **SP**: by using **FC** or **FA**.

Solving SP-d and SP-a by using FC: Consider a discard slice $\mathcal{F}$ consisting of n rules $\mathrm{R}_1, \cdots, \mathrm{R}_n$, i.e. $\mathrm{R}_1, \cdots, \mathrm{R}_{n-1}$ are accept rules and R_n is an all-deny-rule. Therefore, $\mathcal{F}$ denies a packet P iff P matches none of the accept-rules of $\mathcal{F}$, and hence matches only the last all-deny-rule of $\mathcal{F}$. Clearly, this situation occurs iff the policy $\mathcal{F}\backslash\mathrm{R}_n$ is incomplete, where $\mathcal{F}\backslash\mathrm{R}_n$ denotes $\mathcal{F}$ from which R_n is removed. In the same way, we obtain that an accept slice $\mathcal{F}$ accepts at least one packet iff the policy $\mathcal{F}\backslash\mathrm{R}_n$ is incomplete. From Proposition 1:

Proposition 4 (SP-d). *A discard slice $\mathcal{F}$ consisting of rules $\mathrm{R}_1, \cdots, \mathrm{R}_n$ denies one or more packets iff the automaton $\Gamma_{\mathcal{F}\backslash\mathrm{R}_n}$ has the no-match state.*

Proposition 5 (SP-a). *An accept slice $\mathcal{F}$ consisting of rules $\mathrm{R}_1, \cdots, \mathrm{R}_n$ accepts one or more packets iff the automaton $\Gamma_{\mathcal{F}\backslash\mathrm{R}_n}$ has the no-match state.*

Therefore, **SP-d** and **SP-a** problems of $\mathcal{F}$ are solved by constructing the automaton $\Gamma_{\mathcal{F}\backslash\mathrm{R}_n}$ and verifying if it contains the no-match state.

Solving SP-d (resp. SP-a) by using FA-d (resp. FA-a): SP-d is a particular case of **FA-d** which considers only discard slices, instead of any policy. Similarly, **SP-a** is a particular case of **FA-a** which considers only accept slices. Therefore, **SP-d** and **SP-a** can be solved by solving **FA-d** and **FA-a**, respectively. Hence, from Propositions 2 and 3, we obtain:

Proposition 6 (SP-d). *A discard slice $\mathcal{F}$ denies one or more packets iff the automaton $\Gamma_{\mathcal{F}}$ has one or more match states associated to the action* Deny.

Proposition 7 (SP-a). *An accept slice $\mathcal{F}$ accepts one or more packets iff the automaton $\Gamma_{\mathcal{F}}$ has one or more match states associated to the action* Accept.

Example of Accept Slice: Due to the symmetry between **SP-d** and **SP-a**, we will illustrate only the resolution of **SP-a** by the example of the accept slice of Table 2. The symbol # means "same as the field of the preceding rule".

Table 2. Example of accept slice.

Rule	IPsrc	IPdst	Port	Protocol	Action
R_1	192.168.10.0/24	81.10.10.0/24	*Any*	*Any*	Deny
R_2	194.204.201.0/28	212.217.65.202	*not*(21)	*Any*	
R_3	#	212.217.65.201	*not*(80)	*Any*	
R_4	#	#	80	UDP	
R_5	#	*not*(212.217.65.201, 212.217.65.202)	*Any*	*Any*	
R_6	*not*(192.168.10.0/24, 194.204.201.0/28)	212.217.65.201	*not*(80)	*Any*	
R_7	#	#	80	UDP	
R_8	#	*not*(212.217.65.201)	*Any*	*Any*	
R_9	*Any*	*Any*	*Any*	*Any*	Accept

Illustration of SP-a Resolution by Using FC: Figure 2 represents the automaton $\Gamma_{\mathcal{F}\setminus R_9}$ synthesized from the accept slice of Table 2 without R_9. From Proposition 5 and the fact that $\Gamma_{\mathcal{F}\setminus R_9}$ contains the no-match state, we deduce that this accept slice accepts packets.

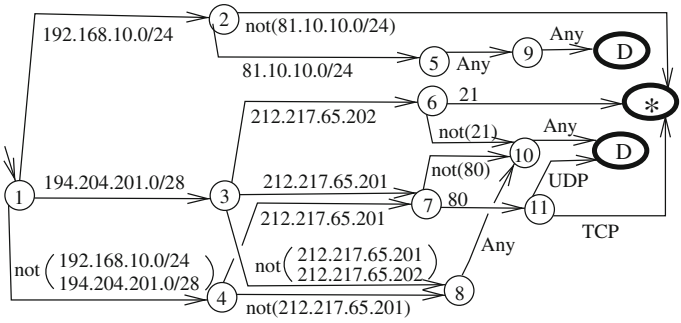

Fig. 2. Automaton $\Gamma_{\mathcal{F}\setminus R_9}$ of the accept slice of Table 2 without R_9.

Illustration of SP-a resolution by using FA-a: Fig. 3 represents the automaton $\Gamma_{\mathcal{F}}$ synthesized from the accept slice $\mathcal{F}$ of Table 2. From Proposition 7 and the fact that $\Gamma_{\mathcal{F}}$ has 4 match states associated to the action Accept, we deduce that this accept slice accepts packets.

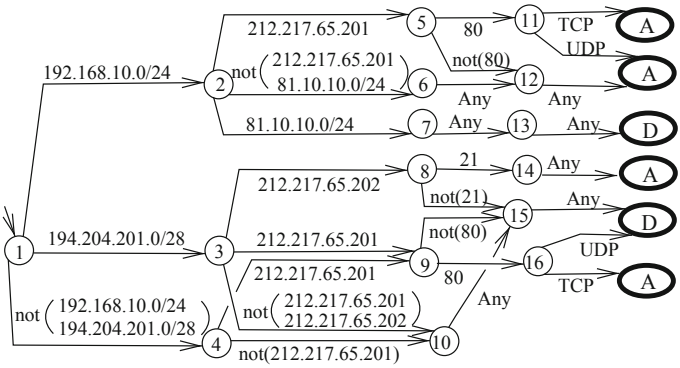

Fig. 3. Automaton $\Gamma_{\mathcal{F}}$ obtained from the accept slice $\mathcal{F}$ of Table 2.

6 Resolution of FI, FV, FE and FR

Let us demonstrate the applicability of our synthesis procedure for the resolution of 8 problems of [2]: **FI-d**, **FI-a**, **FV-d**, **FV-a**, **FE-d**, **FE-a**, **FR-d** and **FR-a**.

6.1 Resolution of Firewall Implication Problems: FI-d, FI-a

There are two Firewall Implication (**FI**) problems:

FI-d: to design an algorithm that takes as input two policies $\mathcal{F}_1$ and $\mathcal{F}_2$ and determines whether $\mathcal{F}_2$ denies all the packets denied by $\mathcal{F}_1$.

FI-a: to design an algorithm that takes as input two policies $\mathcal{F}_1$ and $\mathcal{F}_2$ and determines whether $\mathcal{F}_2$ accepts all the packets accepted by $\mathcal{F}_1$.

We solve **FI-d** and **FI-a** by the 3-step procedure below.

Step 1: We apply the synthesis procedure of Sect. 4 to generate the automata $\Gamma_{\mathcal{F}_1}$ and $\Gamma_{\mathcal{F}_2}$ from $\mathcal{F}_1$ and $\mathcal{F}_2$.

Step 2: $\Gamma_{\mathcal{F}_1}$ and $\Gamma_{\mathcal{F}_1}$ are combined into a single automaton denoted $\Omega_{\mathcal{F}_1,\mathcal{F}_2}$, by applying to them the product operator (this operator is also used in the synthesis procedure of Sect. 4, as shown in [3]). Each state of $\Omega_{\mathcal{F}_1,\mathcal{F}_2}$ is defined in the form $\langle \phi_1, \phi_2 \rangle$, where each ϕ_i is a state of $\Gamma_{\mathcal{F}_i}$. Intuitively, for every packet P, a state $\langle \phi_1, \phi_2 \rangle$ of $\Omega_{\mathcal{F}_1,\mathcal{F}_2}$ is reached, iff the states ϕ_1 and ϕ_2 are reached in $\Gamma_{\mathcal{F}_1}$ and $\Gamma_{\mathcal{F}_1}$, respectively. A state $\langle \phi_1, \phi_2 \rangle$ of $\Omega_{\mathcal{F}_1,\mathcal{F}_2}$ is said final if ϕ_1 and ϕ_2 are final states in Γ_1 and Γ_2, respectively. Hence, a final state of $\Omega_{\mathcal{F}_1,\mathcal{F}_2}$ is in one of the following forms, where q_i is a match state of $\mathcal{F}_i$ and E_i is the no-match state of $\mathcal{F}_i$: $\langle q_1, q_2 \rangle$ associated to two actions a_1 and a_2, $\langle q_1, E_2 \rangle$ associated to a single action a_1, $\langle E_1, q_2 \rangle$ associated to a single action a_2, and $\langle E_1, E_2 \rangle$ associated to no action.

Step 3: From Theorem 1, when a final state r of $\Omega_{\mathcal{F}_1,\mathcal{F}_2}$ is reached for a packet P, the actions a_1 and a_2 associated to r, if any, are dictated by $\mathcal{F}_1$ and $\mathcal{F}_2$, respectively. We obtain:

Proposition 8 (FI-d). *$\mathcal{F}_2$ denies all the packets denied by $\mathcal{F}_1$ iff, for every final state r of $\Omega_{\mathcal{F}_1,\mathcal{F}_2}$ associated to actions (a_1, a_2): $a_1 =$ Deny implies $a_2 =$ Deny.*

Proposition 9 (FI-a). *$\mathcal{F}_2$ accepts all the packets accepted by $\mathcal{F}_1$ iff, for every final state r of $\Omega_{\mathcal{F}_1,\mathcal{F}_2}$ associated to (a_1, a_2): $a_1 =$ Accept implies $a_2 =$ Accept.*

Therefore, **FI-d** (resp. **FI-a**) problem of $(\mathcal{F}_1, \mathcal{F}_2)$ is solved by constructing the automaton $\Omega_{\mathcal{F}_1,\mathcal{F}_2}$ and verifying if all its final states satisfy the condition of Proposition 8 (resp. Proposition 9).

Due to the symmetry between **FI-d** and **FI-a**, we illustrate only the resolution of **FI-d**. We consider the previous policies $\mathcal{F}_1$ of Table 1 and $\mathcal{F}_2$ of Table 2. The automata $\Gamma_{\mathcal{F}_1}$ and $\Gamma_{\mathcal{F}_2}$ have been previously given in Figs. 1 and 3, respectively. The product automaton $\Omega_{\mathcal{F}_1,\mathcal{F}_2}$ of $\Gamma_{\mathcal{F}_1}$ and $\Gamma_{\mathcal{F}_2}$, is represented in Fig. 4. The notation X-Y associated to the final states means that the actions dictated by $\mathcal{F}_1$ and $\mathcal{F}_2$ are X and Y, respectively. * means the absence of action. For example, *-D means that a_2 is Deny and there is no a_1. Since we have no state with D-A or D-*, we deduce from Proposition 8 that the accept slice of Table 2 denies every packet which is denied by the policy of Table 1.

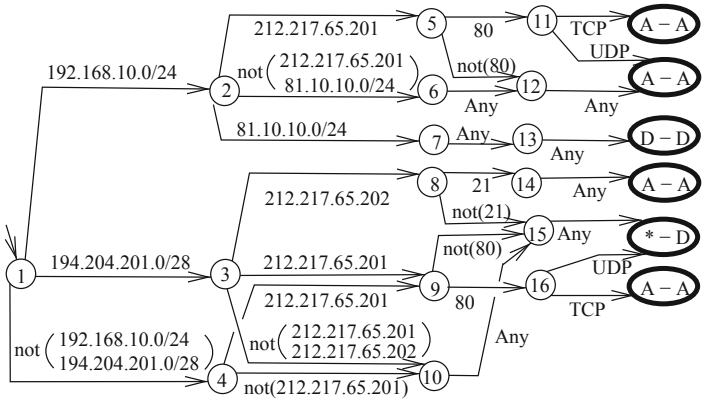

Fig. 4. Product $\Omega_{\mathcal{F}_1,\mathcal{F}_2}$ of $\Gamma_{\mathcal{F}_1}$ and $\Gamma_{\mathcal{F}_2}$ of Figs. 1 and 3.

6.2 Resolution of Firewall Verification Problems: FV-d, FV-a

We first define the following two particular properties:

Discard property: it has exactly the same form and semantics as a filtering rule with the action Deny.

Accept property: it has exactly the same form and semantics as a filtering rule with the action Accept.

There are two Firewall Verification problems (**FV**):

FV-d: to design an algorithm that takes as input a policy $\mathcal{F}$ and a discard property $\mathcal{P}$, and determines whether $\mathcal{F}$ denies all the packets denied by $\mathcal{P}$.

FV-a: to design an algorithm that takes as input a policy $\mathcal{F}$ and an accept property $\mathcal{P}$, and determines whether $\mathcal{F}$ accepts all the packets accepted by $\mathcal{P}$.

FV-d and **FV-a** are particular cases of **FI-d** and **FI-a**, respectively, because a discard property and an accept property are particular policies consisting of a single rule. We can therefore solve **FV-d** and **FV-a** by using exactly the same 3-step method used for solving **FI-d** and **FI-a**. We obtain:

Proposition 10. *$\mathcal{F}$ denies all the packets denied by a discard property $\mathcal{P}$ iff, for every final state r of $\Omega_{\mathcal{P},\mathcal{F}}$ associated to (a_1, a_2): $a_1 =$* Deny *implies $a_2 =$* Deny

Proposition 11. *$\mathcal{F}$ accepts all the packets accepted by an accept property $\mathcal{P}$ iff, for every final state r of $\Omega_{\mathcal{P},\mathcal{F}}$ associated to (a_1, a_2): $a_1 =$* Accept *implies $a_2 =$* Accept

Therefore, **FV-d** (resp. **FV-a**) problem of $(\mathcal{P}, \mathcal{F})$ is solved by constructing the automaton $\Omega_{\mathcal{P},\mathcal{F}}$ and verifying if all its final states satisfy the condition of Proposition 10 (resp. Proposition 11).

Due to the symmetry between **FV-d** and **FV-a**, we illustrate only the resolution of **FV-a**. We consider the policy $\mathcal{F}$ of Table 1 (Sect. 3) and the accept property $\mathcal{P}$ of Table 3. In Step 1, we construct automata $\Gamma_{\mathcal{F}}$ and $\Gamma_{\mathcal{P}}$. $\Gamma_{\mathcal{F}}$ has been seen in Fig. 1 and $\Gamma_{\mathcal{P}}$ is represented in Fig. 5. In Step 2, we construct the product $\Omega_{\mathcal{P},\mathcal{F}}$ of $\Gamma_{\mathcal{P}}$ and $\Gamma_{\mathcal{F}}$, which is represented in Fig. 6. Since $\Omega_{\mathcal{P},\mathcal{F}}$ has no state with A-D or A-*, we deduce from Proposition 11 that $\mathcal{F}$ of Table 1 accepts every packet which is accepted by $\mathcal{P}$ of Table 3.

Table 3. Example of accept property

Rule	IPsrc	IPdst	Port	Protocol	Action
R_1	192.168.10.0/24	212.217.65.201	*Any*	*Any*	Accept

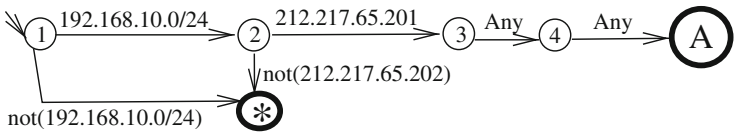

Fig. 5. Automaton $\Gamma_{\mathcal{P}}$ obtained from the accept property $\mathcal{P}$ of Table 3.

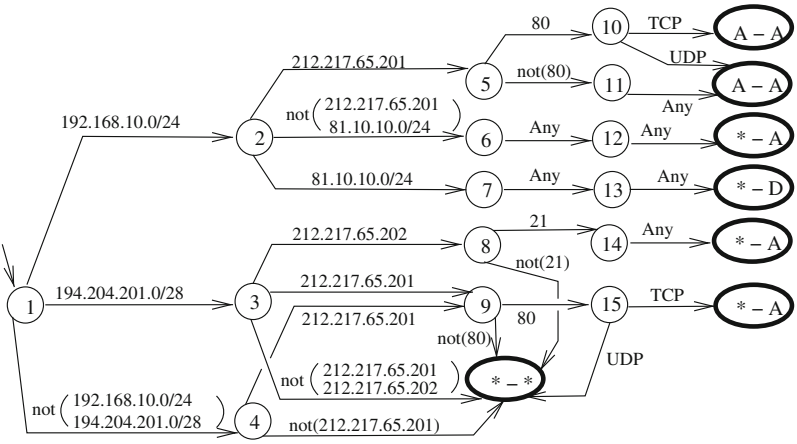

Fig. 6. Product $\Omega_{\mathcal{P},\mathcal{F}}$ of $\Gamma_{\mathcal{P}}$ and $\Gamma_{\mathcal{F}}$ of Figs. 5 and 3.

6.3 Resolution of Firewall Equivalence Problems: FE-d, FE-a

There are two Firewall Equivalence (**FE**) problems:

FE-d: to design an algorithm that takes as input two policies $\mathcal{F}_1$ and $\mathcal{F}_2$ and determines whether $\mathcal{F}_1$ and $\mathcal{F}_2$ deny the same set of packets.

FE-a: to design an algorithm that takes as input two policies $\mathcal{F}_1$ and $\mathcal{F}_2$ and determines whether $\mathcal{F}_1$ and $\mathcal{F}_2$ accept the same set of packets.

FE-d and **FE-a** can be obviously solved by solving **FI-d** and **FI-a**. Indeed, $\mathcal{F}_1$ and $\mathcal{F}_2$ deny the same set of packets is equivalent to: $\mathcal{F}_1$ denies at least all the packets denied by $\mathcal{F}_2$ AND $\mathcal{F}_2$ denies at least all the packets denied by $\mathcal{F}_1$. Similarly, $\mathcal{F}_1$ and $\mathcal{F}_2$ accept the same set of packets is equivalent to: $\mathcal{F}_1$ accepts at least all the packets accepted by $\mathcal{F}_2$ AND $\mathcal{F}_2$ accepts at least all the packets accepted by $\mathcal{F}_1$. We obtain:

Proposition 12. *$\mathcal{F}_1$ and $\mathcal{F}_2$ deny the same set of packets iff, for every final state r of $\Omega_{\mathcal{F}_1,\mathcal{F}_2}$ associated to (a_1, a_2): $a_1 =$ Deny iff $a_2 =$ Deny.*

Proposition 13. *$\mathcal{F}_1$ and $\mathcal{F}_2$ accept the same set of packets iff, for every final state r of $\Omega_{\mathcal{F}_1,\mathcal{F}_2}$ associated to (a_1, a_2): $a_1 =$ Accept iff $a_2 =$ Accept.*

Therefore, **FE-d** (resp. **FE-a**) problem of $(\mathcal{F}_1, \mathcal{F}_2)$ is solved by constructing the automaton $\Omega_{\mathcal{F}_1,\mathcal{F}_2}$ and verifying if all its final states satisfy the condition of Proposition 12 (resp. Proposition 13).

Due to the symmetry between **FE-d** and **FE-a**, we illustrate only the resolution of **FE-a**. We use the same example used to illustrate the resolution of **FI-d**. We consider therefore the previous policies $\mathcal{F}_1$ of Table 1 and $\mathcal{F}_2$ of Table 2. The automata $\Gamma_{\mathcal{F}_1}$, $\Gamma_{\mathcal{F}_2}$ and $\Omega_{\mathcal{F}_1,\mathcal{F}_2}$ have been represented in Figs. 1, 3 and 4, respectively. Since every final state in Fig. 4 has either two actions Accept or no action Accept, we deduce from Proposition 13 that the accept slice of Table 2 and the policy of Table 1 accept the same set of packets.

6.4 Resolution of Firewall Redundancy Problems: FR-d, FR-a

Let $\mathcal{F}\backslash\mathcal{R}$ denote a policy $\mathcal{F}$ from which a filtering rule $\mathcal{R}$ is removed. There are two Firewall Redundancy (**FR**) problems:

FR-d: to design an algorithm that takes as input a policy $\mathcal{F}$ and one of its discard rules $\mathcal{R}$, and determines whether $\mathcal{F}$ and $\mathcal{F}\backslash\mathcal{R}$ deny the same set of packets.

FR-a: to design an algorithm that takes as input a policy $\mathcal{F}$ and one of its accept rules $\mathcal{R}$, and determines whether $\mathcal{F}$ and $\mathcal{F}\backslash\mathcal{R}$ accept the same set of packets.

FR-d and **FR-a** are obviously particular cases of **FE-d** and **FE-a**, respectively. We can therefore solve **FR-d** and **FR-a** as we have solved **FE-d** and **FE-a**. We obtain: ($\mathcal{R}$ is one of the rules of a policy $\mathcal{F}$)

Proposition 14. *$\mathcal{F}$ and $\mathcal{F}\backslash\mathcal{R}$ deny the same set of packets iff, for every final state r of $\Omega_{\mathcal{F},\mathcal{F}\backslash\mathcal{R}}$ associated to actions (a_1, a_2): $a_1 =$ Deny iff $a_2 =$ Deny.*

Proposition 15. *$\mathcal{F}$ and $\mathcal{F}\backslash\mathcal{R}$ accept the same set of packets iff, for every final state r of $\Omega_{\mathcal{F},\mathcal{F}\backslash\mathcal{R}}$ associated to actions (a_1, a_2): $a_1 =$ Accept iff $a_2 =$ Accept.*

Therefore, **FR-d** (resp. **FR-a**) problem of $(\mathcal{F}, \mathcal{R})$ is solved by constructing the automaton $\Omega_{\mathcal{F},\mathcal{F}\backslash\mathcal{R}}$ and verifying if all its final states satisfy the condition of Proposition 14 (resp. Proposition 15).

Due to the symmetry between **FR-d** and **FR-a**, we illustrate only the resolution of **FR-d**. We consider the policy $\mathcal{F}$ of Table 4 which is obtained by adding the rule R_5 to the policy of Table 1. Let us verify that $\mathcal{F}$ and $\mathcal{F}\backslash R_5$ deny the same packets. The automaton $\Gamma_{\mathcal{F}\backslash R_5}$ has been seen in Fig. 1. The automaton $\Gamma_{\mathcal{F}}$ is identical to $\Gamma_{\mathcal{F}\backslash R_5}$, because R_5 is "shadowed" by R_4 and hence never takes effect. The product $\Omega_{\mathcal{F},\mathcal{F}\backslash\mathcal{R}}$ is represented in Fig. 7. Since every final state in Fig. 7 has either two actions Deny or no action Deny, we deduce from Proposition 14 that $\mathcal{F}$ of Table 4 and $\mathcal{F}\backslash R_5$ deny the same set of packets.

Table 4. Policy to illustrate **FR-d** resolution

Rule	IPsrc	IPdst	Port	Protocol	Action
R_1	*Any*	212.217.65.201	80	TCP	Accept
R_2	192.168.10.0/24	81.10.10.0/24	*Any*	*Any*	Deny
R_3	194.204.201.0/28	212.217.65.202	21	*Any*	Accept
R_4	192.168.10.0/24	*Any*	*Any*	*Any*	Accept
R_5	192.168.10.0/24	*Any*	80	UDP	Deny

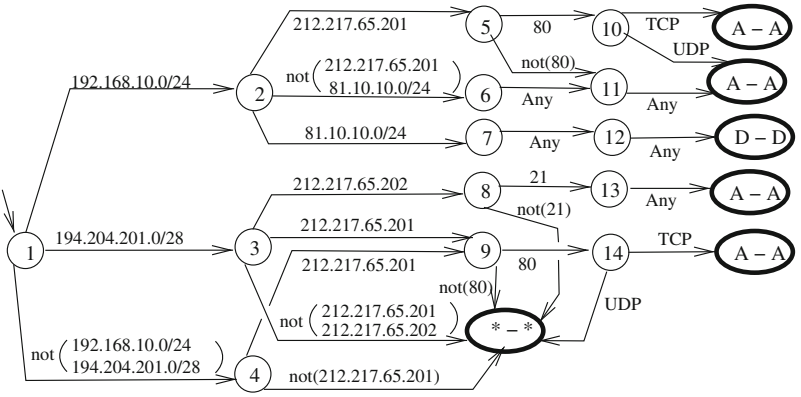

Fig. 7. Product $\Omega_{\mathcal{F},\mathcal{F}\setminus\mathrm{R}_5}$ obtained for the policy of Table 4.

7 Evaluation of Space and Time Complexities

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 define:

$d_i =$ 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. $D = d_0 + \cdots + d_{m-1}$
$\mu =$ number of great fields.
$\delta =$ sum of the number of bits to code the small fields.

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. Since IPsrc, IPdst and Port are great fields and Protocol is a small field, we obtain $\mu = 3$ (number of great fields) and $\delta = d_3 = 1$ (1 bit is used to code the unique small field protocol).

We have the following result:

Theorem 2. *The space and time complexities for solving each of the 13 problems are in* $\mathrm{O}(n^{\mu+1} \times 2^{\delta})$*, which is bounded by both* $\mathrm{O}(n^{m+1})$ *and* $\mathrm{O}(n \times 2^{D})$.

For space limit, we do not present the proof of Theorem 2.

By using results in [26], the authors of [2] prove that the 13 problems studied in this article are NP-Hard. In our context, their result is that the time complexity is in $O(n \times 2^D)$. On the other hand, the authors of [7–11] solve some of the 13 problems with algorithms whose time complexity is in $O(n^{m+1})$. Our contribution here is that the two expressions $O(n^{m+1})$ and $O(n \times 2^D)$ are upper bounds 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.

8 Conclusion

We have applied the automata-based methodology of [3] to resolve the 13 NP-hard problems of firewalls of [2]. We have also evaluated space and time complexities of the 13 resolutions.

As near future work, we plan to apply our synthesis procedure for the design of efficient security policies that adapt dynamically to the filtered traffic. We also plan to adapt our approach in other areas, such as policies in intelligent health-care (e-health).

References

1. Information Technology Security Evaluation Criteria (ITSEC), v1.2. Office for Official Publications of the European Communities, Luxembourg, June 1991
2. Elmallah, E., Gouda, M.G.: Hardness of firewall analysis. In: International Conference on NETworked sYStems (NETYS), Marrakesh, Morocco, May 2014
3. Khoumsi, A., Krombi, W., Erradi, M.: A formal approach to verify completeness and detect anomalies in firewall security policies. In: Cuppens, F., Garcia-Alfaro, J., Zincir Heywood, N., Fong, P.W.L. (eds.) FPS 2014. LNCS, vol. 8930, pp. 221–236. Springer, Heidelberg (2015)
4. Hoffman, D., Yoo, K.: Blowtorch: a framework for firewall test automation. In: 20th IEEE/ACM International Conference on Automated Software Engineering (ASE), Long Beach, California, USA, pp. 96–103, November 2005
5. Kamara, S., Fahmy, S., Schultz, E., Kerschbaum, F., Frantzen, M.: Analysis of vulnerabilities in internet firewalls. Comput. Secur. **22**(3), 214–232 (2003)
6. Wool, A.: A quantitative study of firewall configuration errors. Computer **37**(6), 62–67 (2004)
7. Acharya, H.B., Gouda, M.G.: Firewall verification and redundancy checking are equivalent. In: 30th IEEE International Conference on Computer Communication (INFOCOM), Shanghai, China, pp. 2123–2128, April 2011
8. Liu, A.X., Gouda, M.G.: Complete redundancy removal for packet classifiers in TCAMs. IEEE Trans. Parallel Distrib. Syst. **21**(4), 424–437 (2010)
9. Acharya, H.B., Gouda, M.G.: Projection, division: linear space verification of firewalls. In: 30th International Conference on Distributed Computing Systems (ICDCS), Genova, Italy, pp. 736–743, June 2010
10. Al-Shaer, E., Marrero, W., El-Atawy, A., Elbadawi, K.: Network configuration in a box: towards end-to-end verification of networks reachability and security. In: 17th IEEE International Conference on Network Protocols (ICNP), Princeton, NJ, USA, pp. 736–743, October 2009

11. Liu, A.X., Gouda, M.G.: Diverse firewall design. IEEE Trans. Parallel Distrib. Syst. **19**(9), 1237–1251 (2008)
12. Al-Shaer, E., Hamed, H.: Modeling and management of firewall policies. IEEE Trans. Netw. Serv. Manag. **1**(1), 2–10 (2004)
13. Karoui, K., Ben Ftima, F., Ben Ghezala, H.: Formal specification, verification, correction of security policies based on the decision tree approach. Int. J. Data Netw. Secur. **3**(3), 92–111 (2013)
14. Madhuri, M., Rajesh, K.: Systematic detection and resolution of firewall policy anomalies. Int. J. Res. Comput. Commun. Technol. (IJRCCT) **2**(12), 1387–1392 (2013)
15. Garcia-Alfaro, J., Cuppens, F., Cuppens-Boulahia, N., Martinez Perez, S., Cabot, J.: Management of stateful firewall misconfiguration. Comput. Secur. **39**, 64–85 (2013)
16. 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)
17. Liu, A.X., Gouda, M.G.: Structured firewall design. Comput. Netw.: Int. J. Comput. Telecommun. Netw. **51**(4), 1106–1120 (2007)
18. 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, CA, USA, May 2006
19. Bryant, R.E.: Graph-based algorithms for Boolean function manipulation. IEEE Trans. Comput. **35**(8), 677–691 (1986)
20. 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
21. Lee, D., Yannakakis, M.: Principles and methods of testing finite state machines - a survey. Proc. IEEE **84**, 1090–1126 (1996)
22. 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
23. Lu, L., Safavi-Naini, R., Horton, J., Susilo, W.: Comparing and debugging firewall rule tables. IET Inf. Secur. **1**(4), 143–151 (2007)
24. Mansmann, F., Göbel, T., Cheswick, W.: Visual analysis of complex firewall configurations. In: 9th International Symposium on Visualization for Cyber Security (VizSec), Seattle, WA, USA, pp. 1–8, October 2012
25. Krombi, W., Erradi, M., Khoumsi, A.: Automata-based approach to design and analyze security policies. In: Internernational Conference on Privacy, Security and Trust (PST), Toronto, Canada (2014)
26. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. AW.H. Freeman, San Francisco (1979)