Real Time Verification of Firewalls with Dynamic Rulebase Update

Amjad Gawanmeh
Department of Electrical and Computer Engineering
Khalifa University of Science, Technology and Research,
Abu Dhabi, UAE
Email: amjad.gawanmeh@kustar.ac.ae

Sofiène Tahar
Department of Electrical and Computer Engineering
Concordia University
Montreal, Quebec, Canada
Email: tahar@ece.concordia.ca

***Abstract*—Firewalls provide the required security for private communication networks since they protect them from undesired traffic and unauthorized access. They are required to implement several security policies that are specified at a high level of abstraction. The verification of firewalls and the security policies they implement is a challenging problem because of the critical role of their dynamic operation. In this work, we introduce a novel method for verifying the correct implementation of security policies in firewalls. The method is used to show that, during the firewall runtime, security policies are implemented in the firewall rulebase with no conflicts. The method is tested on synthetic firewalls of practical size. The evaluation of this method shows its ability to verify real time security policy implementation in firewalls during their runtime.**

***Index Terms*—Firewall Security, Formal Methods, Policy Verification**

I. Introduction and Motivation

Firewalls [1] are the most widely adopted technology for the protection of networks in the theme of enterprise security. Firewalls provide security through rules that implement policies. Hence, their quality is affected by the quality of these policies. A security policy contains high level rules about the security and the privacy of the network. These rules are implemented with firewalls by stating the desirable actions with regard to a certain network traffic. However, when policies are implemented as a set of rules, conflicts may arise between various rules because many policies may have a wide range of addresses, ports, or protocols and only two possible actions. This issue occurs because firewall rules are defined based on higher level security policies, and are updated frequently either by adding new rules or changing the order of rules. Therefore, it is essential to detect conflicting rules in firewalls when new rules are implemented.

Formal methods [2] use rigorous mathematical reasoning to verify design specifications and implementations of a variety of hardware, security, software, and several other types of systems [3]. This work presents a novel method for the verification of correct security policies implementation during firewalls runtime. In [4], [5] we presented a formal model for firewall rulebase, a domain restriction method, and an algorithm that can identify conflicting rules in firewalls with an embedding in Event-B method [6] at the network address level of abstraction. The Event-B framework was used because it provides the necessary constructs to directly implement domain restriction operations. However, there were several limitations up to that point. For instance, the embedding in Event-B did not allow automatic rulebase verification since we had to define all rules manually into the Event-B model. This also prevented the verification of large rulebases. Moreover, the Event-B model could not be integrated into the firewall models themselves, hence making it inappropriate for runtime verification. Despite all of these limitations, the Event-B embedding illustrated the applicability and feasibility of the method.

This paper extends our previous work with a formal model and the conflict detecting algorithm to handle firewall rules based on ranges of network addresses, protocol types and port numbers. In addition, this work introduces theorems about the completeness, soundness and termination of the method are defined and proven. The implementation of the proposed algorithm is tested on a large firewall rulebase. Finally, the algorithm is used for the real time verification of firewalls that are dynamically updated with new security policies, and then tested on a rulebase of more than 40000 rules.

The performance of the verification algorithm is tested on various rulebases of sizes that range between 2000 rules and 44000 rules. The algorithm identified several conflicting rules and terminated in less than 30 seconds for the largest rulebase. This time is needed in order to establish the rulebase abstract domains, which is practically done at the initialization step. In addition, the runtime verification can be conducted in less than 0.03 seconds where the algorithm could verify that a rulebase is conflict free when a new rule that implements a security policy is added to the rulebase. This time needs to be as short as possible in order to identify possible conflicts when a new policy is to be implemented in the firewall during its real time operation.

The rest of the paper is organized as follows: Section II provides a review on formal verification of firewalls. Section III presents the formal model. Section IV describes the conflict detection algorithm in the firewall rulebase. Section V elaborates on termination, soundness, and completeness proofs of the algorithm. Section VI presents the runtime verification of security policies with related experimental results. Finally, Section VII concludes the paper.

II. Related Work

This section presents the state of the art on the verification of firewalls. A thorough review on related work on firewalls modeling and verification could be found in [4]. We address here the most pertinent ones. For instance, Abbes *et al.* [7] presented a method for detecting conflicts between packet filters that are defined within one single firewall. The work in [8] provided a method for certifying that a firewall configuration is sound and complete with respect to a given high level security policy. Brucker *et al.* [9] used higher-order logic (HOL) to model security policies and firewalls. Matoušek *et al.* [10] provided a method to verify that security constraints are preserved within a network that runs dynamic routing protocols. The work in [11] presented a deterministic algorithm based on a projection pass, a division pass, and a probe procedure that checks if a firewall satisfies a given property. Kotenko and Polubelova [12] used a model checking approach for detection and resolution of filtering anomalies in high security policies.

Liu [13] addressed the problem of conflicts between rules defined by a decision path of the firewall decision diagram. Jeffery and Samak [14] used Boolean satisfiability solvers for the analysis of firewall policy with regards to reachability and cyclicity properties. More recently, [15] presented a model checking based technique and a tool for verifying access-control policies, where a set of policies are translated into the NuSMV model checker language and then formally verified.

Souayeh and Bouhoula [16] and Alsaleh [17] proposed methods for verifying that a firewall configuration respects the security policy it implements. Khorchani *et al.* [18] used a modal logic, called Visibility Logic, to define arbitrary patterns between rules inside a firewall and verify any formula expressed in visibility logic using model checking. The work of Windmuller [19] handled the problem of defining the required rule set based on an existing, informal security concept and in validating the resulting rulebase. Finally, Kotenko and Polubelova [20] used model checking techniques for the verification of firewall security policy.

In the aformentioned works, the focus was mainly on two issues, the first was verifying that a firewall's rulebase is anomality free, while the second was verifying that the rulebase correctly implements security policies. However, a firewall rulebase is dynamic, where new rules can be added, or existing rules can be edited. Therefore, verifying that this change creates no anomalities and, at the same time, maintains the correct implementation of security policies is a problem that has not been addressed.

Al-Shaer *et al.* [21], [17], introduced a method to identify anomalies that could exist in a single and multi-firewall environment along with a set of techniques to discover policy anomalies in centralized and distributed firewalls. The algorithm is only tested on a small number of rules, and its efficiency is not convenient and cannot be applied during firewalls operation. On the other hand, identifying anomalies in rules can be misleading because certain security policies can be implemented only by introducing anomalies. For instance, in order to block traffic from a specific address on certain port and allow all other traffic from that same address into any other port, two rules with anomalies must be used. Yet, the inspection sequence matters in this case.

Since conflicts may arise when implementing a new policy, the updated rulebase needs to be verified. This particular problem has not been addressed so far. The work we present in this paper, compared to the above described state of the art, proposes a method that can be used for runtime verification of correct security policy implementation, and hence, it can validate firewall rulebases with regard to their security policy during their runtime. Therefore, it can be used to identify any conflicts that may occur when implementing a new policy or editing an existing one during the runtime of firewalls.

III. Modeling Firewall Rules

When packets arrive at a firewall, they are inspected based on their contents of source and destination addresses. The firewall will either accept or deny an incoming and outgoing packet based on the first rule that matches the address of the packet, the firewall is assumed to be stateless where no history information is kept. Network addresses are formally defined as $\mathcal{N}_a$, where every address, a source or a destination, is composed of three parts: IP address (ip), protocol type (p), and port number (t), hence, $s = (ip, p, t)$ and $d = (ip, p, t)$ are source and destination network addresses, respectively, where, $s \in \mathcal{N}_a$ and $d \in \mathcal{N}_a$. The address ip can represent a single address, a network address, or an address range. We assume that a source and destination network address pair in a firewall rule cannot be empty. We use $\mathcal{N}$ to represent the set of network address pairs such that $(s, d) \in \mathcal{N}$, where s is for the source address and d is for the destination address. We use $\mathcal{A}$ to denote the set of possible firewall actions: $\mathcal{A} = \{accept,\ deny\}$. Firewall rules are defined as a mapping relation from between elements in $\mathcal{N}$ and an action element in $\mathcal{A}$, formally, $r : n \mapsto a$, where $n \in \mathcal{N}$, $a \in \mathcal{A}$, and $\mapsto$ is a mapping relation from addresses to actions that maps one element in $\mathcal{N}$ to an element in $\mathcal{A}$, which indicates that if the address n matches the address of the packet, then action a is taken for that packet. We use $\sigma(r)$ and $\delta(r)$ to denote a function that extracts the set of source and destination addresses that appear in rule r, respectively. We also use $\alpha(r)$ to denote the action that appears in rule r. The set of all possible firewall rules is defined as $\Re : \mathcal{N} \times \mathcal{A}$. We use $\mathcal{N}_s$ and $\mathcal{N}_d$ to denote the set of source and destination addresses, respectively, where each is a subset of $\mathcal{N}_a$.

We formally define a firewall rulebase as a finite set of rules: $\mathcal{R}$: $\{r_1, r_2, \ldots r_n\}$, such that $\mathcal{R} \subset \Re$. In firewalls, $\mathcal{R}$ is inspected in an arbitrary order. On the other hand, a firewall that implements certain security policies, $\mathcal{F}$, is an ordered sequence of rules in the form: $\mathcal{F} = r_1, r_2, \ldots, r_n$ such that $r_1 \in \mathcal{R} \wedge r_2 \in \mathcal{R} \wedge \ldots r_n \in \mathcal{R}$. We use $\mathcal{R}(\mathcal{F})$ to denote $\mathcal{R}$ for a given firewall rulebase $\mathcal{F}$.

Equality between addresses is modeled using a predicate function that represents network address comparison such that

$\mathcal{N}_a \times \mathcal{N}_a \rightarrow \mathcal{B}$, where, $\mathcal{B} = \{True, False\}$, and we use the operator $==$ to denote this predicate function. This definition covers all possible cases of single address, network address, or a range of addresses. $ip_i == ip_j \ iff \ \exists x, y \cdot \ x \in ip_i \wedge y \in ip_j \wedge x == y$, where $x == y$ is true if and only if both single addresses are identical.

The conflict set, $\mathcal{S}$, is a set of all conflicting rules in the firewall and is formally defined as:
$\mathcal{S} = \{(r_i, r_j) | r_i \in \mathcal{R} \wedge \in r_j \mathcal{R} \wedge r_i == (s_i, d_i) \mapsto a_i \wedge r_j == (s_j, d_j) \mapsto a_j \wedge s_i == s_i \wedge d_i == d_j \wedge a_i \neq a_j\}$.

A given firewall correctly implements a security policy if there are no conflicts in its rules with regard to the rules that implement the new policy. When the address of the packet that is being inspected matches two different rules, then the resulting firewall action will depend on the sequence of these rules. If rules give different actions, the action of the first rule inspected will be applied.

We model the correct implementation of a security policy using a set of rules as a property, ϕ, over the set of firewall rules, $\mathcal{F}$, hence, if a given firewall rulebase correctly implements a security policy, then we state it as: $\phi \models \mathcal{F}$.

IV. Algorithm for Detecting Conflicts in Firewall Rules

The methodology we use in order to verify the consistency of the rulebase is based on the formal model for firewall rules that defines domain and codomain restriction operators [5]. Figure 1 illustrates the methodology to calculate the conflict set, $\mathcal{S}$, for the firewall rulebase, denoted as $\mathcal{R}(\mathcal{F})$, by applying domain restriction on every rule r_i in $\mathcal{R}(\mathcal{F})$. First, domain rules sets, R_s and R_d are calculated for source and destination address of rule r_i using domain restriction. Next, co-domain restriction is applied on the latter sets, R_s and R_d, in order to calculate co-domain rule sets, denoted as R_{sx} and R_{dx}. Finally, we check conflicts for source and destination addresses depending on whether the rule contains both source and delineation address or only one of them, and update the conflict set. This operation is repeated for all rules. At the end of the algorithm, $\mathcal{S}$ shall contain the set of all conflicting rules.

The rulebase is consistent and conflict free if and only if the conflict set calculated by the second part is empty. Then, source and destination addresses are extracted for every rule in $\mathcal{R}(\mathcal{F})$, r_i, and domain restriction sets, called R_{sx} and R_{dx}, are calculated for r_i and $\mathcal{R}(\mathcal{F})$. This is achieved by obtaining the source address n_s, the destination address, n_d, and the action of the rule $a_r = \alpha(r_i)$, then the domain restriction operator $\lhd$ is applied on $\mathcal{R}(\mathcal{F})$ and both n_s and n_d separately in order to obtain two subsets of $\mathcal{R}(\mathcal{F})$ named R_s and R_d. Then co-domain restriction operator, $\rhd$, is applied on both R_s and R_d and either A_a or A_d based on the action in the rule, a_r, in order to obtain R_{sx} from R_s and a_r, and R_{dx} from R_d and a_r.

Finally, $\mathcal{S}$ is updated based on source and destination addresses existence of the rule r_i, where three cases are distinguished: in the first, where r_i contains both, r_i is mapped into every common rule in R_{sx} and R_{dx}, to create pairs of rules in the form (r_i, r_j) and add them to $\mathcal{S}$. The operator $\odot$ denotes the mapping between the rule r_i and the set of rules conflicting with it, i.e., $R_{sx} \cap R_{dx}$. In the second case in which r_i contains only a source address, we add $r_i \odot R_{sx}$ into $\mathcal{S}$, and finally, in the third case in which r_i contains only a destination address, we add $r_i \odot R_{dx}$ into $\mathcal{S}$.

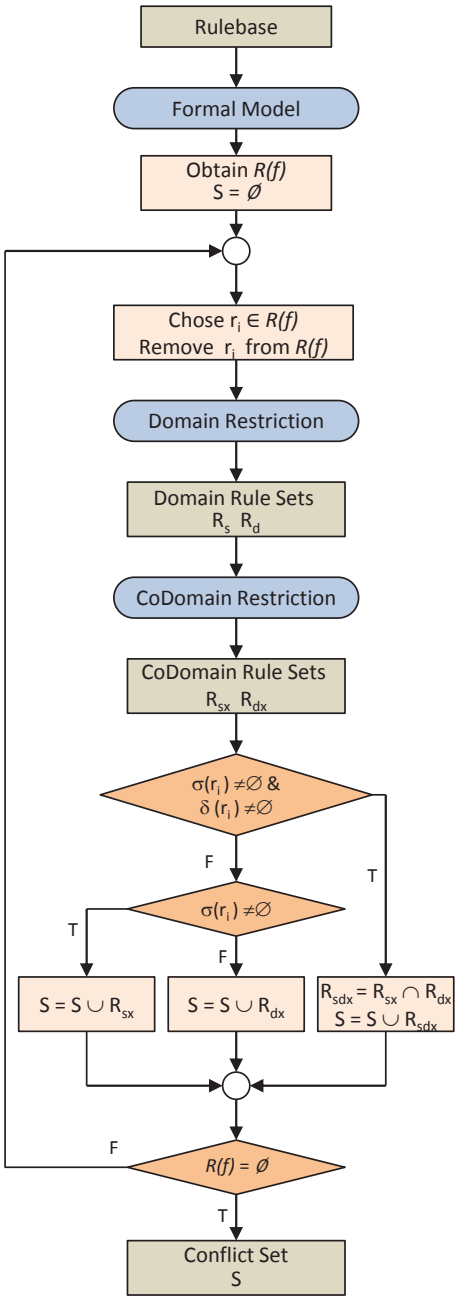

Fig. 1. Detecting Conflicts Methodology

The algorithm has n^2 complexity, since domain restriction operators have a complexity of n, where n represents the number of rules. Hence it can be efficiently used for runtime verification of firewalls. In fact, the above algorithm is used for firewalls verification at runtime, where a new rule (r_i) defines

the security policy to be implemented in the firewall. The algorithm then generates conflicts with the firewall rulebase, $\mathcal{R}(\mathcal{F})$, if they exist. On the other hand, the absence of conflicts proves that the policy can be safely implemented in the firewall. In addition, it is required to prove the termination of the algorithm in order to avoid deadlock situations. Therefore, in the next section soundless and completeness of the algorithm shall be discussed.

V. Soundness and Completeness of the Method

A firewall configuration is conflict-free if its calculated conflict set is empty, therefore, we need to generate the conflict set for the given set of rules in order to verify the property. Let $\mathcal{R}$ be a firewall rulebase, ϕ be a conflict-free property, and $\mathcal{S}$ be the conflict set for $\mathcal{R}$. Then $\phi \models \mathcal{R}$ states that property ϕ is correct for $\mathcal{R}$, and $\mathcal{S}$ can be calculated from $\mathcal{R}$ using the above algorithm. Based on this notation, theorems about the correctness of the algorithm are defined and proven.

Theorem V.1. *Soundness.*

Let $\mathcal{R}$ be a firewall rulebase, ϕ be a consistency property, and $\mathcal{S}$ be the conflict set for $\mathcal{R}$, then, if the conflict set for $\mathcal{F}$ is not empty, then, there exist conflicts in $\mathcal{R}$. Formally, $\mathcal{S} \neq \emptyset \Rightarrow \phi \nvDash \mathcal{R}$.

Given: $\mathcal{S} \neq \emptyset$, Goal: $\phi \nvDash \mathcal{R}$

Proof:

$\mathcal{S} \neq \emptyset \Rightarrow \quad \exists r_i, r_j \cdot r_i \in \mathcal{R} \quad \wedge \quad r_j \in \mathcal{R} \quad \wedge \quad (r_i, r_j) \in \mathcal{S}$

three cases are identified:

(1) $\sigma(r_i) = \emptyset \Rightarrow r_j \in R_{dx} \ \wedge \ \delta(r_i) \neq \emptyset$,

(2) $\delta(r_i) = \emptyset \Rightarrow r_j \in R_{sx} \ \wedge \ \sigma(r_i) \neq \emptyset$, and

(3) $\sigma(r_i) \neq \emptyset \ \wedge \ \delta(r_i) \neq \emptyset \Rightarrow r_j \in R_{sx} \bigcap R_{dx}$

For case (1): from the co-domain restriction operator ($\triangleright$), it can be deduced that

if $\alpha(r_i) = accept$, then $r_j \in R_{dx} \Rightarrow R_j \in R_d \ \wedge \ \alpha(r_j) = deny$, and

if $\alpha(r_i) = deny$, then $r_j \in R_{dx} \Rightarrow R_j \in R_d \ \wedge \ \alpha(r_j) = accept$.

For both cases, it can be deduced from domain restriction that $R_j \in R_d \Rightarrow \delta(r_j) = \delta(r_i) \ \wedge \ \alpha(r_i) \neq \alpha(r_j)$, therefore, $\mathcal{R}$ is not conflict-free, and, ϕ does not hold, hence $\phi \nvDash \mathcal{F}$.

For case (2):

if $\alpha(r_i) = accept$, then $r_j \in R_{sx} \Rightarrow R_j \in R_s \ \wedge \ \alpha(r_j) = deny$, and $ifaction(r_i) = deny$, then $r_j \in R_{sx} \Rightarrow R_j \in R_d \ \wedge \ action(r_j) = accept$.

For both cases, $R_j \in R_s \Rightarrow \sigma(r_j) = \sigma(r_i) \ \wedge \ \alpha(r_i) \neq action(r_j)$, therefore, $\mathcal{R}$ is not conflict-free, and ϕ does not hold, hence $\phi \nvDash \mathcal{F}$.

For case (3):

if $\alpha(r_i) = accept$, then $r_j \in R_{sx} \bigcap R_{dx} \Rightarrow R_j \in R_s \ \wedge \ R_j \in R_d \ \wedge \ \alpha(r_j) = deny$,

if $\alpha(r_i) = deny$, then $r_j \in R_{sx} \bigcap R_{dx} \Rightarrow R_j \in R_s \ \wedge \ R_j \in R_d \ \wedge \ \alpha(r_j) = accept$.

For both cases, $R_j \in R_s \ \wedge \ R_j \in R_d \Rightarrow (\sigma(r_j) == \sigma(r_i)) \ \wedge \ (\delta(r_j) == \delta(r_i)) \ \wedge \ (\alpha(r_i) \neq \alpha(r_j))$,

therefore, $\mathcal{R}$ is not conflict-free, and ϕ does not hold, hence $\phi \nvDash \mathcal{R}$.

■

Corollary V.1. *Absence of Conflicts.*

Assuming the same conditions as Theorem V.1, if the there is no conflict in $\mathcal{R}$, then the conflict set of $\mathcal{R}$ is empty. Formally, $\phi \models \mathcal{R} \Rightarrow \mathcal{S} = \emptyset$.

This corollary is deduced from Theorem V.1 above and illustrates the ability of the algorithm to prove that the firewall rulebase is conflict-free.

Theorem V.2. *Completeness.*

Let $\mathcal{F}$ be a firewall configuration, ϕ be a consistency property, and $\mathcal{S}$ be the conflict set for $\mathcal{R}$. If the conflict set is empty, then there exist no conflicts in $\mathcal{R}$. Formally, $\mathcal{S} = \emptyset \Rightarrow \phi \models \mathcal{R}$.

Given: $\mathcal{S} = \emptyset$, Goal: $\phi \models \mathcal{R}$

Proof:

$\mathcal{S} = \emptyset \Rightarrow \quad \forall r_i \cdot r_i \in \mathcal{R}$, then

(1) $\sigma(r_i) = \emptyset \Rightarrow R_{dx} = \emptyset$, or

(2) $\delta(r_i) = \emptyset \Rightarrow R_{sx} = \emptyset$, or

(3) $\sigma(r_i) \neq \emptyset \ \wedge \ \delta(r_i) \neq \emptyset \Rightarrow R_{sx} \bigcap R_{dx} = \emptyset$

For case (1):

if $\alpha(r_i) = accept$, then it can be deduced from the co-domain restriction operator, ($\triangleright$), that

$R_{dx} = \emptyset \Rightarrow \nexists r_j \cdot r_j \in \mathcal{R} \ \wedge \ \alpha(r_j) = deny \ \wedge \ r_j \in R_d$.

Similarly, if $\alpha(r_i) = deny$, then

$R_{dx} = \emptyset \Rightarrow \nexists r_j \cdot r_j \in \mathcal{R} \ \wedge \ \alpha(r_j) = accept \ \wedge \ r_j \in R_d$.

For both cases, it can be deduced from the above and the domain restriction operator definition, $\triangleleft$, that $\sigma(r_i) neq \sigma(r_j)$, or the co-domain restriction operator $\triangleright$ that $\alpha(r_i) == \alpha(r_j)$. Therefore, $\mathcal{R}$ is conflict-free, and, ϕ does hold, hence $\phi \models \mathcal{R}$.

For case (2):

if $\alpha(r_i) = accept$, then

$R_{sx} = \emptyset \Rightarrow \nexists r_j \cdot r_j \in \mathcal{R} \ \wedge \ \alpha(r_j) = deny \ \wedge \ r_j \in R_s$.

Similarly, if $\alpha(r_i) = deny$, then

$R_{sx} = \emptyset \Rightarrow \nexists r_j \cdot r_j \in \mathcal{R} \ \wedge \ \alpha(r_j) = accept \ \wedge \ r_j \in R_s$.

For both cases, $\sigma(r_i) \neq \sigma(r_j)$, or $\alpha(r_i) == \alpha(r_j)$, therefore, $\mathcal{R}$ is conflict-free, and ϕ does hold, hence $\phi \models \mathcal{R}$.

For case (3):

if $\alpha(r_i) = accept$, then

$R_{sx} \bigcap R_{dx} = \emptyset \Rightarrow \nexists r_j \cdot r_j \in \mathcal{R} \ \wedge \ \alpha(r_j) = deny \ \wedge \ r_j \in R_s \ \wedge \ r_j \in R_d$.

Similarly, if $\alpha(r_i) = deny$, then

$R_{sx} = \emptyset \Rightarrow \nexists r_j \cdot r_j \in \mathcal{R} \wedge \alpha(r_j) = accept \wedge r_j \in R_s \wedge r_j \in R_d$.

For both cases, $\sigma(r_i) \neq \sigma(r_j)$, or $\delta(r_i) neq \delta(r_j)$, or $\alpha(r_i) = \alpha(r_j)$, therefore, $\mathcal{R}$ is conflict-free, and ϕ does hold, hence $\phi \models \mathcal{R}$.

■

Corollary V.2. *Detecting Conflicts.*

Assuming the same conditions as Theorem V.2, if there are conflicts in $\mathcal{R}$, then the conflict set of $\mathcal{R}$ is not empty. Formally, $\phi \nvDash \mathcal{R} \Rightarrow \mathcal{S} \neq \emptyset$.

This corollary is deduced from Theorem V.2 above, and states that when there are conflicts in the rulebase, then the conflict set is not empty. This illustrates the ability of the algorithm to detect conflicts.

Theorem V.3. *Termination.*

The Algorithm terminates in polynomial time.

We construct the proof for termination showing in every iteration in the algorithm, there is at least one single dependence solved, that the space is finite and it is decreasing in every iteration.

Proof:

Starting with a set of rules, $\mathcal{R}(\mathcal{F})$ contains a finite number of elements.

Every iteration in the algorithm executes $\mathcal{R}(\mathcal{F}) = \mathcal{R}(\mathcal{F}) - \{r_i\}$, where $r_i \in \mathcal{R}(\mathcal{F})$.

This implies that the space is decreasing, and eventually, $\mathcal{R}(\mathcal{F})$ becomes $\emptyset$, which is the precondition for algorithm termination. ■

In the next section, we consider an application for the proposed method, where the algorithm is implemented and tested on a synthetic firewall.

VI. Experimental Results and Discussion

Figure 2 illustrates an application of the proposed methodology. When a new security policy is to be implemented in a firewall, the domain restriction method is used in order to confirm that the new policy creates no conflicts in the rulebase during runtime. On the other hand, the method can identify the conflicting rules if they exist. Hence, this method can be used in the iterative process of firewall policy implementation, verification, and maintenance. In order to illustrate the efficiency of this method, a C++ implementation is developed for the method presented in this paper. For experimental purpose, the implementation is run on a machine with an Intel Core 2 Duo 3 GHz processor and 4GB of RAM.

Due to security concerns, firewalls and their rulebase are considered confidential, therefore, we run the algorithm on synthetic firewalls of huge sizes in order to evaluate their performance. Rules that have different fields are generated: source IP address, destination IP address, network address, IP address range, source port number, destination port number, and common protocol types with their default port numbers. In order to vary rules to mimic a practical firewall rulebase, we considered different types of rules that may or may not contain any of the above fields. For these experiments, we run the algorithm on rulebases of 2000 rules, and incremented the number by 2000 up to 44000 rules, which is larger than what a typical firewall can handle [22]. For every case, the algorithm was run a specific number of times where the conflict set was constructed and execution time was measured.

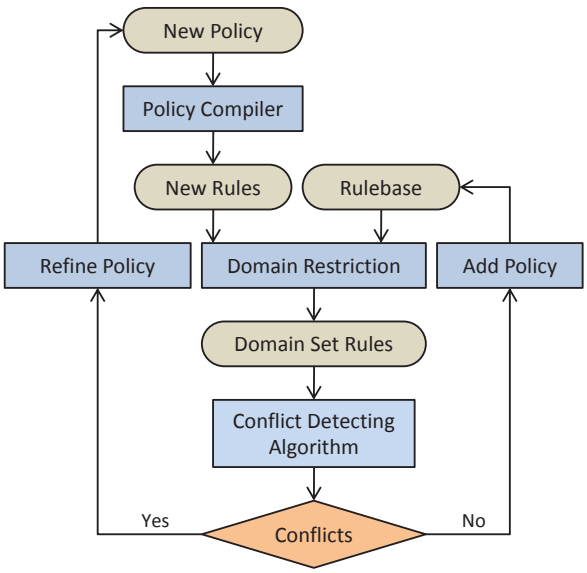

Fig. 2. Real-time Verification of New Security Policies in Firewalls

Table I displays the average execution time for generating the conflict set and the number of conflicting rules in the generated rulebase for the implementation. Table I shows that the algorithm can identify conflicts when they exist in a reasonable time. The largest rulebase consisting of 44000 rules could be run in the implementation in less than 30 seconds.

Number of Rules	Execution Time (second)	Number of Conflicts
4000	0.29	0
8000	0.92	0
12000	1.85	1
16000	3.29	1
20000	5.14	1
24000	7.34	2
28000	10.50	3
32000	13.17	3
36000	17.01	5
40000	20.60	6
44000	24.80	8

TABLE I
EXPERIMENTAL RESULTS: EXECUTION TIME VS TOTAL NUMBER OF RULES

In addition, we tested the algorithm by defining a number of rules that implement a certain security policy and generating a firewall rulebase of 40000 rule. We considered a security policy of a various number of rules and run the method in order to obtain the time needed to process it. Table II shows the verification time for the security policy vs number of rules in the policy. We observe that the method could terminate in less than 0.03 seconds for a security policy of 20 rules tested on a rulebase of 40000 rules. This proves that the algorithm we present in this work is practical and can be used in real time verification of firewalls, and hence, verify the consistency of the rulebase right after any dynamic modification on the rulebase. Therefore, it can be integrated into firewalls without affecting their efficiency, while at the same time preventing undesired security holes.

VII. Conclusion

The verification of firewall rulebase is essential, in particular during their real time operation due to their dynamic characteristics. In this paper, we presented a method for runtime verification of security policies implemented in firewalls taking into consideration their dynamic operations. In this paper, we extended our previous work in [4], [5] by considering the

Number of Rules	Verification Time (milisecond)
2	2.18
4	4.34
6	6.63
8	8.67
10	10.84
12	12.98
14	15.08
16	17.27
18	19.40
20	21.09

TABLE II
EXPERIMENTAL RESULTS: EXECUTION TIME VS NUMBER OF RULES IN THE POLICY

firewall rulebase at a lower level of abstraction where rules can have port types and port numbers. We also provided an implementation for the algorithm and used it to test a generated firewall rulebase that ranges between 2000 rules and 44000 rules. The algorithm is shown to be efficient in detecting all conflicting rules and their performance shows that they can be used in real time verification of firewalls. In addition, the algorithm was used to conduct runtime verification of firewalls considering the dynamic sequence of the rulebases, where it was able to detect any conflicts for newly added rules in less than 0.03 seconds. Finally, we presented and proved three theorems related to termination, soundness and completeness of the proposed algorithm.

Compared to the state of the art work, the performance of the algorithm shows that it can be used for the runtime verification of firewalls while they are dynamically configured or modified, an issue that has not been addressed properly in the state of the art. On the other hand, the proposed method can verify the consistency of new security policies when implemented in a firewall. As future work, we plan to investigate this issue further and utilize the efficiency of the conflict method in order to provide a real-time consistency check for dynamic firewalls during runtime operation.

REFERENCES

[1] A. X. Liu, *Firewall Design and Analysis*, World Scientific Publishing Inc., 2011.

[2] J.R. Abrial, "Faultless Systems: Yes We Can!," *IEEE Computer Journal*, vol. 42, no. 9, pp. 30–36, 2009.

[3] P. Boca and J.P. Bowen J. Siddiqi, *Formal Methods: State of the Art and New Directions*, Springer-Verlag London Limited, 2010.

[4] A. Gawanmeh and S. Tahar, "Domain Restriction based Formal Model for Firewall Configurations," *International Journal for Information Security Research*, vol. 2, no. 1–2, pp. 294–302, 2012.

[5] A. Gawanmeh and S. Tahar, "Novel Algorithm for Detecting Conflicts in Firewall Rules," in *IEEE Canadian Conference on Electrical and Computer Engineering*. 2012, pp. 1–4, IEEE Press.

[6] J.R. Abrial, *Modelling in Event-B: System and Software Engineering*, Cambbridge University Press, 2009.

[7] T. Abbes, A. Bouhoula, and M. Rusinowitch, "An Inference System for Detecting Firewall Filtering Rules Anomalies," in *ACM Symposium on Applied Computing*. 2008, pp. 2122–2128, ACM Press.

[8] N. Ben Youssef, A. Bouhoula, and F. Jacquemard, "Automatic Verification of Conformance of Firewall Configurations to Security Policies," in *IEEE Symposium on Computers and Communications*. 2009, pp. 526–531, IEEE Press.

[9] A. Brucker, L. Brügger, and B. Wolff, "Model-Based Firewall Conformance Testing," in *Testing of Software and Communicating Systems*. 2008, vol. 5047 of *LNCS*, pp. 103–118, Springer-Verlag.

[10] P. Matoušek, J. Ráb, O. Ryšavý, and M. Švéda, "A Formal Model for Network-Wide Security Analysis," in *IEEE International Conference on Engineering of Computer Based Systems*. 2008, pp. 171–181, IEEE Press.

[11] H. Acharya and M. Gouda, "Projection and Division: Linear-Space Verification of Firewalls," in *IEEE International Conference on Distributed Computing Systems*. 2010, pp. 736–743, IEEE Press.

[12] I. Kotenko and O. Polubelova, "Verification of Security Policy Filtering Rules by Model Checking," in *IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems*. 2011, pp. 706–710, IEEE Press.

[13] A.X. Liu, "Formal Verification of Firewall Policies," in *IEEE International Conference on Communications*. 2008, pp. 1494–1498, IEEE Press.

[14] A. Jeffrey and T. Samak, "Model Checking Firewall Policy Configurations," in *IEEE Symposium on Policies for Distributed Systems and Networks*. 2009, pp. 60–67, IEEE Press.

[15] Karthick Jayaraman, Mahesh Tripunitara, Vijay Ganesh, Martin Rinard, and Steve Chapin, "MOHAWK: Abstraction-Refinement and Bound-Estimation for Verifying Access Control Policies," *ACM Transactions on Information and System Security*, vol. 15, no. 4, pp. 18, 2013.

[16] N. Souayeh and A. Bouhoula, "A Fully Automatic Approach for Fixing Firewall Misconfigurations," in *IEEE International Conference on Computer and Information Technology*. 2011, pp. 461–466, IEEE Press.

[17] E. Al-Shaer and M.N. Alsaleh, "ConfigChecker: A Tool for Comprehensive Security Configuration Analytics," in *IEEE Symposium on Configuration Analytics and Automation*. 2011, pp. 1–2, IEEE Press.

[18] B. Khorchani, S. Halle, and R. Villemaire, "Firewall Anomaly Detection with a Model Checker for Visibility Logic," in *IEEE Network Operations and Management Symposium*. 2012, pp. 466–469, IEEE Press.

[19] S. Windmuller, "Offline Validation of Firewalls," in *IEEE Software Engineering Workshop*. 2011, pp. 36–41, IEEE Press.

[20] I. Kotenko and O. Polubelova, "Verification of Security Policy Filtering Rules by Model Checking," in *IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems*. 2011, pp. 706–710, IEEE Press.

[21] E. Al-Shaer and H. Hamed, "Discovery of Policy Anomalies in Distributed Firewalls," in *Annual Joint Conference of the IEEE Computer and Communications Societies*. 2004, vol. 4, pp. 2605–2616, IEEE Press.

[22] A. Wool, "Trends in Firewall Configuration Errors: Measuring the Holes in Swiss Cheese," *IEEE Journal of Internet Computing*, vol. 14, no. 4, pp. 58–65, 2010.