DOMAIN RESTRICTION BASED FORMAL MODEL FOR FIREWALL CONFIGURATIONS

Amjad Gawanmeh

Khalifa University
Sharjah Campus, PO.Box 573 Sharjah, UAE
amjad.gawanmeh@kustar.ac.ae

Sofiène Tahar

Concordia University
Montreal, Quebec, H3G 1M8 Canada
tahar@ece.concordia.ca

ABSTRACT

Firewalls are widely adopted for protecting private networks by filtering out undesired network traffic in and out of secured networks. Therefore, they play an important role in the security of communication systems. The verification of firewalls is a great challenge because of the dynamic characteristics of their operation, their configuration is highly error prone, and finally, they are considered the first defense to secure networks against attacks and unauthorized access. In this paper, we present a formal model for firewalls rulebase using domain restriction method, and based on this model, a novel algorithm for detecting and identifying conflicts in firewalls rulebase. The algorithm is based on calculating the conflict set of firewall configurations using the domain restriction. The domain restriction method is implemented using Event-B formal techniques, where we model firewall configuration rules, and then use invariant checking to verify the consistency of firewall configurations.

1. INTRODUCTION

Firewalls [1] are part of network security that were designed to enable secure connections between private and outside networks. With the growing complexity of computer networks, security has become a crucial issue. Firewalls are part of network security that were designed to enable secure connections between private networks and outside networks, the growing complexity of networks made them indispensable to control information flow within a network. Therefore, they are widely adopted for protecting private networks by filtering out undesired network traffic in and out of the secured network. Therefore, firewalls are the front defense for secure networks against attacks.

Protection provided by a firewall directly depends on the quality of its configuration and the consistency of its rulebase. Firewall configurations and maintenance of their rulebase is highly error prone, therefore solutions are needed in order to verify their correctness. A firewall policy error either creates security holes that will allow malicious traffic to sneak into a private network or block legitimate traffic and disrupt normal operation, which in turn could lead to undesired consequences. Therefore the central role of firewalls in the security of networks make their verification a critical task.

Testing and verification of firewalls is a great challenge because of the dynamic characteristics of their operation, their configuration is highly error prone, and finally, they are considered the first defense to secure networks against attacks and unauthorized access. In addition, firewalls can be used extensively before it turns out that they are vulnerable to attacks, even though they receive intensive analysis, and are thought to be correct. Most firewall operations depend on an existing sequence of rules, which is intentionally made dynamic in order to eliminate certain denial of service (DoS) attacks. Therefore, it is essential to detect conflicting rules in firewalls configurations, and at the same time be able to decide if they conform to the security requirement of the firewall.

Formal methods [2] are based on using mathematical reasoning to verify that design specifications comprehend certain design requirements. Formal methods have been successfully used for the precise analysis of a variety of hardware and software systems [3]. In this paper we present a formal model for analyzing firewall configurations in order to show that they are correctly implemented. Our method for verification of firewall configuration rules is based on *domain restriction* method. First, a formal model for firewall configuration rules is defined, then, domain restriction operations are defined on this model. Finally, an algorithm is established using these operations to formally verify the consistency of the configuration rules. For illustration purposes, we use the Event-B [4] formal method in order to apply our approach on a firewall example, where we model firewall

rules in Event-B, then define an appropriate invariant to check consistency in firewall rules.

This paper extends the previous work in [5] by proposing a formal model for firewalls rulebase and an algorithm for the verification of firewalls rulebase that can efficiently detect conflicts in firewall rules. In [5], we presented the domain restriction method implemented in Event-B [4], while in this work, the algorithm is based on formally calculating the *conflict set* for a given firewall rulebase. In addition, the algorithm can verify rulebase consistency, and identify conflicting rules, if they exist.

The rest of the paper is organized as follows. Section 2 discusses related work. In Section 3, we present a formal model for firewall configurations and rules. Section 4 presents the domain restriction method and an algorithm for the verification of firewall configurations. In Section 5, we present an implementation of the domain restriction method in Event-B, and illustrate it on a firewall example. Finally, Section 6 concludes the paper with open issues and future work directions.

2. RELATED WORK

In this section we discuss related work on using formal methods for the verification of firewalls and their configurations. Abbes *et al.* [6] proposed a method to detect overlaps between packet filters within one firewall, they classify rules based on the conditions of each filtering rule to separate non-overlapping rules. Ben Youssef *et al.* [7] proposed a method for checking whether a firewall reacts correctly with respect to a security policy given in an high level declarative language. The method is implemented in satisfiability solver modulo theories (SMT). These works are limited to the problem of conflict avoidance, and do not consider verifying whether a firewall reacts correctly with respect to a given security policy.

In another approach, Brucker *et al.* [8] presented a case study to model firewalls and their policies in higher-order logic (HOL) throughout a set of derived theories for simplifying policies. Matoušek *et al.* [9] introduced a formal method approach for the verification of security constraints on networks with dynamic routing protocols in use. Cuppens *et al.* [10] proposed an automatic process generating firewall rules from an abstract specification of a security policy. However, the validity of the translations was not proved and the obtained results are not proved to be conflict free.

Yuan *et al.* [11] introduced FIREMAN, a static analysis toolkit for firewall modeling and analysis that treats firewall configurations as specialized programs and applies static analysis techniques to check problems of configurations, such as policy violations, inconsistencies, and inefficiencies in firewalls. The tool performs symbolic model checking of the firewall configurations and is implemented using binary decision diagrams (BDDs), this approach is based on propositional logic, which has limitations in terms of langauge expressiveness and model size. In another work, Matoušek *et al.* [9] introduced a formal method approach for the verification of security constraints on networks with dynamic routing protocols in use. We believe a formal model that captures the details of firewall rulebases is necessary prior to the definition of an effecting algorithm in this particular case.

Acharya and Gouda [12] proposed a verification algorithm that takes a firewall F and a property R, and determines whether or not F satisfies R. Kotenko and Polubelova [13] used Promela for detecting anomalies in the specification of a security policy of computer networks with model checking. The method is implemented in the SPIN model checker. Liu [14] verifies in his work whether a firewall policy satisfies a given property. The method is based on showing that a property about rules does not conflict with any rule defined by a decision path of the firewall decision diagram. Jeffery and Samak [15] used SAT (Satisfiability) solvers for the model analysis of reachability and cyclicity properties of interest in firewall policies. The model for network configurations is based on a single firewall model and is shown to be efficient compared to BDD based approaches. The use of model checking has the problem of state space explosion, specially for a large number of firewall rules. These methods require a high level of abstraction in order to avoid the state space explosion problem, in addition, manual encoding of firewall policy rules in languages such as Promela is tedious and time consuming.

Most of above approaches only check for conflicts between rules which is obtained by inspecting certain fields in the policy, rather than checking them in the rulebase, in addition, they consider only firewall policies at a high level of abstraction. They also ignore the dynamic update of these rules, and do not consider the sequence at which these rules are inspected. Therefore the verification of firewalls rulebase, considering their dynamic sequence, is essential and have not been yet explored thoroughly. This paper will address this issue by presenting a precise formal model and an algorithm that can automatically detect conflicts in firewalls rulebase. The algorithm is based on the simplicity of calculating the conflict set, and therefore, will be efficient once implemented. In addition, most formally related work verifies firewall rulebase with regard to the security policy, while in this work, we are concerned about the consistency of the rulebase. We believe first-order theorem proving will be more efficient in this particular case.

3. A FORMAL MODEL FOR FIREWALLS RULEBASE

In this section we present our formal model for firewall configurations, where we formally define components and relations in firewalls, then our verification methods based on this model. Firewalls [1] are network elements that control packets in a secured network based on a set of rules. These rules define the actions performed by the firewall based on certain configured filtering conditions. Firewall rules filter traffic based on protocol type, port used, or source and destination IP addresses. Firewall actions are either to accept, or deny. The first allows packets to pass through, while the second blocks them. Rules are examined in sequence, the packet is accepted or denied by a specific rule if it matches the required network addressing fields of this rule. Otherwise, the following rule is examined until a matching rules is found. In case no rule is found, a default policy action can be performed.

In order to provide a formal and precise model for the above description, we will use first-order logic that allows reasoning about firewall operations and primitives, while at the same time, they can be implemented directly in supporting verification methods such as Event-B.

We assume a finite domain containing the possible network addresses pairs in a firewall rulebase $\langle s, d\rangle$, i.e., source and destination, where either s or d can be empty. Let $\mathcal{N}$ be the set of possible network address pairs for packets incoming to and outgoing from a network such that $\langle s, d\rangle \in \mathcal{N}$. We define two sets based on $\mathcal{N}$, the first, $\mathcal{N}_s$, is for source addresses, and the second, $\mathcal{N}_d$, is for destination addresses. $\mathcal{N}$ is an abstract set that will be refined in order to represent actual network addresses, it can be refined further to represent protocol type or port numbers in IP network addressing. Let $\mathcal{A}$ be the abstract set of all possible actions that a firewall can perform, this set can be defined as follows: $\mathcal{A} = \{accept,\ deny\}$. We define every firewall rule to be a mapping relation from an address pair in $\mathcal{N}$ into an action 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 $\mathcal{A}$. n may contain either source or destination or both. We use $source(r)$ and $dest(r)$ to denote the address that appears in rule r. We define $\mathbb{R}$ to be the set firewall rules such that $\mathcal{R} = \mathcal{N} \times \mathcal{A}$, therefore, $r \in \mathbb{R}$. We use $source(r)$ and $dest(r)$ to denote the address that appears in rule r.

A firewall rulebase, denoted as $\mathcal{R}$ is a finite set of rules: $\{r_1, r_2, \ldots r_n\}$ that are inspected in a specific order by the firewall, such that $\mathcal{R} \subset \mathbb{R}$. A firewall is configured so that $\mathcal{R}$ is inspected in an arbitrary order. A firewall configuration, $\mathcal{F}$, 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 configuration $\mathcal{F}$.

For a given firewall rulebase with a set of rules, a packet with a source address, a destination address, or both, is checked by inspecting the rules in sequence. The basic principle of firewalls operation states that the order in which rules are inspected should not affect the result. $\mathcal{F}'$ is an arbitrary firewall configuration for $\mathcal{F}$ if the following conditions are satisfied:

- Every rule in $\mathcal{F}$ is also in $\mathcal{F}'$ and every rule in $\mathcal{F}'$ is also in $\mathcal{F}$: $\forall r_i \cdot r_i \in \mathcal{R}(\mathcal{F}) \Rightarrow r_i \in \mathcal{R}(\mathcal{F}') \wedge r_i \in \mathcal{R}(\mathcal{F}') \Rightarrow r_i \in \mathcal{R}(\mathcal{F})$,
- There are at least two rules that are inspected in different order: $\mathcal{F} \neq \mathcal{F}'$

  $\exists i, j \cdot r_i \in \mathcal{F} \wedge r_j \in \mathcal{F}' \wedge i \neq j \wedge (r_i = r_j)$

A conflict set, $\mathcal{CS}$, is a set of all rules that satisfy the following conditions:

- $\forall r \cdot r \in \mathcal{CS} \Rightarrow r \in \mathcal{R}(\mathcal{F})$, and consequently $r \in \mathcal{R}(\mathcal{F}')$,
- $\exists n \cdot n \in \mathcal{N} \wedge n =< source(r), dest(r) >$,
- $\exists r' \cdot r' \in \mathcal{R}(\mathcal{F}) \wedge r' \neq r \wedge n =$

  $< source(r'), dest(r') >$

A given firewall configuration is considered consistent if there are no conflicts in its rules. The inconsistency occurs when two different rules match the packet being inspected, and each rule gives a different action. The consistency property is denoted as ϕ. If a given firewall configuration is consistent then we can write: $\phi \models \mathcal{F}$.

The domain of firewall configuration rules, $\mathcal{D}$, is defined as:
$\mathcal{D}(\mathcal{F}) = \{n | n \in \mathcal{N} \wedge \exists a, r \cdot (a \in \mathcal{A} \wedge r \in \mathcal{R}(\mathcal{F}) \wedge r = n \mapsto a)\}$

Furthermore, two domains can be defined for source and destination addresses, $\mathcal{N}_s$ and $\mathcal{N}_d$, respectively, as:
$\mathcal{D}_s(\mathcal{F}) =$
$\{s | n = \langle s, d\rangle \wedge n \in \mathcal{N} \wedge s \in \mathcal{N}_s \wedge d \in \mathcal{N}_d \wedge \exists a, r \cdot (a \in \mathcal{A} \wedge r \in \mathcal{R}(\mathcal{F}) \wedge r = n \mapsto a)\}$
$\mathcal{D}_d(\mathcal{F}) =$
$\{d | n = \langle s, d\rangle \wedge n \in \mathcal{N} \wedge s \in \mathcal{N}_s \wedge d \in \mathcal{N}_d \wedge \exists a, r \cdot (a \in \mathcal{A} \wedge r \in \mathcal{R}(\mathcal{F}) \wedge r = n \mapsto a)\}$

Similarly, the configuration co-domain, $\mathcal{C}$, can be defined as:
$\mathcal{C}(\mathcal{F}) = \{a | a \in \mathcal{A} \wedge \exists n, r \cdot (n \in \mathcal{N} \wedge r \in \mathcal{R}(\mathcal{F}) \wedge r = n \mapsto a)\}$

4. VERIFICATION OF FIREWALL RULES USING DOMAIN RESTRICTION

In this section we define the domain restriction method based on the above firewall rules mode. We first define domain restriction, then introduce an algorithm to detect conflicts in fierwall rules, and finally, we illustrate the algorithm on a firewall rulebase example.

4.1. Domain Restriction Method

Domain restriction is applied on firewall rulebase in order to obtain a subset of $\mathcal{R}(\mathcal{F})$. The operators $\lhd$ and $\rhd$ are used to represent domain restriction and co-domain restriction over a set of firewall rules, respectively. First, we formally define domain restriction based on a set of network address pairs, then we refine this definition further for source and destination addresses.

Domain restriction is defined using the operator $\lhd$ over a set of network addresses, $\mathcal{N}$, where $\mathcal{N} \in \mathcal{P}(\mathcal{D}(\mathcal{F}))$, and a set of firewall rules $\mathcal{R}(\mathcal{F})$ as follows:

$$\mathcal{N} \lhd \mathcal{R}(\mathcal{F}) = \{n \mapsto a | n \in \mathcal{N} \land a \in \mathcal{A} \land \exists r \cdot (r \in \mathcal{R}(\mathcal{F}) \land r = n \mapsto a)\}$$

Domain restriction of firewall configurations network source and destination addresses, $\mathcal{N}_s$ and $\mathcal{N}_d$, where $\mathcal{N}_s \in \mathcal{P}(\mathcal{D}_s(\mathcal{C}))$ and $\mathcal{N}_d \in \mathcal{P}(\mathcal{D}_d(\mathcal{C}))$, is defined respectively as:

$$\mathcal{N}_s \lhd \mathcal{R}(\mathcal{F}) = \{n \mapsto a | n \in \mathcal{N} \land a \in \mathcal{A} \land \exists r \cdot (r \in \mathcal{R}(\mathcal{F}) \land \exists d \cdot (d \in \mathcal{D}_d \land n = \langle s, d \rangle \land r = n \mapsto a))\}$$

$$\mathcal{N}_d \lhd \mathcal{R}(\mathcal{F}) = \{n \mapsto a | n \in \mathcal{N} \land a \in \mathcal{A} \land \exists r \cdot (r \in \mathcal{R}(\mathcal{F}) \land \exists s \cdot (s \in \mathcal{D}_s \land n = \langle s, d \rangle \land r = n \mapsto a))\}$$

Co-domain restriction is defined for a chosen set of actions $A \in \mathcal{P}(\mathcal{A})$, the operator $\rhd$ is used to represent this operation, which is formally defined as follows:

$$A \rhd \mathcal{R}(\mathcal{F}) = \{n \mapsto a | n \in \mathcal{N} \land a \in \mathcal{A} \land \exists r \cdot (r \in \mathcal{R}(\mathcal{F}) \land r = n \mapsto a))\}$$

4.2. Detecting Conflicts in Firewall Rules

In order to efficiently use domain restriction operations, we introduce an algorithm that generates a conflict set for a given firewall configuration $\mathcal{F}$, then based on this conflict set, this configuration can be verified to be consistent. Since a domain restriction operation is closed under $\mathcal{N}$, $\mathcal{N}_s$ and $\mathcal{N}_d$, the consistency of $\mathcal{F}$ can be defined based on the conflict set that is obtained using these operators. In addition $\mathcal{N} \lhd \mathcal{R}(\mathcal{F})$, $\mathcal{N}_s \lhd \mathcal{R}(\mathcal{F})$, and $\mathcal{N}_d \lhd \mathcal{R}(\mathcal{F})$ obtain the same set, namely, $\mathcal{R}(\mathcal{F})$. Similarly, co-domain restriction is closed under $\mathcal{A}$. This property helps in the definition of the properties of the algorithm we propose here.

We divide our algorithm into two steps, the first is used to model consistency of firewall configurations based on the concept of conflict set and is shown below in Algorithm 1.

Algorithm 1 Detecting Conflicts in Firewall Configurations

Input: Firewall Configuration $\mathcal{F}$
Output: Consistency of $\mathcal{F}$: $\phi \models \mathcal{F}$
Calculate Conflict Set for $\mathcal{F}$: $\mathcal{CS}$ using Algorithm 2
if $\mathcal{CS} = \emptyset$ **then**
  $\phi \vdash \mathcal{F}$
else
  $\phi \nvdash \mathcal{F}$
end if

Algorithm 2 Calculating Conflict Set

Input: Firewall Configuration $\mathcal{F}$
Output: Conflict Set: $\mathcal{CS}$
Initialize:
  $\mathcal{CS} = \emptyset$
  $A_a = \{accept\}$
  $A_d = \{deny\}$
REPEAT
• Chose rule $r_i \in \mathcal{R}(\mathcal{F})$ and obtain network addresses for r_i
    $n_s = source(r_i)$
    $n_d = dest(r_i)$
• Apply domain restriction:
    $R_s = n_s \lhd \mathcal{R}(\mathcal{F})$; Source address
    $R_d = n_d \lhd \mathcal{R}(\mathcal{F})$; Destination address
• Apply co-domain restriction:
    $R_{sa} = A_a \rhd R_s$; *Accept* action
    $R_{da} = A_a \rhd R_d$
    $R_{sd} = A_d \rhd R_s$; *Deny* action
    $R_{dd} = A_d \rhd R_d$
• Check for conflicts for r_i :
if $R_{sa} \neq \emptyset \land R_{sd} \neq \emptyset$ **then**
  $CS = CS \bigcup \{r_i\}$
else
  No conflict for source address
end if
if $R_{da} \neq \emptyset \land R_{dd} \neq \emptyset$ **then**
  $CS = CS \bigcup \{r_i\}$
else
  No conflict for destination address
end if
  $\mathcal{R}(\mathcal{F}) = \mathcal{R}(\mathcal{F}) - \{r_i\}$
UNTIL $\mathcal{R}(\mathcal{F}) = \emptyset$
END

The second step is used to calculate the conflict set, $\mathcal{CS}$, of a given firewall configuration $\mathcal{F}$, or alternatively, a firewall rulebase $\mathcal{R}(\mathcal{F})$. The algorithm works by inspecting all rules in $\mathcal{R}(\mathcal{F})$ and obtaining source and destination addresses for every rule, then two simple sets of actions $A_a = \{accept\}$ and $A_d = \{deny\}$ are defined. Then we calculate the set of rules that occur in the do-

main of this network address for the source, R_s, and another for the destination, R_d, by applying the domain restriction method: $R_s = \mathcal{N}_s \lhd \mathcal{R}$ and $R_d = \mathcal{N}_d \lhd \mathcal{R}$. Next, we calculate two sets of rules using co-domain restriction for A_a and A_d by applying co-domain restriction operators on the calculated sets R_s and R_d as follows:

$$R_{sa} = A_a \rhd R_s$$
$$R_{sd} = A_d \rhd R_s$$
$$R_{da} = A_a \rhd R_d$$
$$R_{dd} = A_d \rhd R_d$$

In the last step, we check for existing conflicts for source and destination and update the conflict set. This operation is repeated for all rules. The details of the algorithm is given below in Algorithm 2.

The algorithm has n^2 complexity, since domain restriction operators have a complexity of n, where n represents the number of rules. It is essential to show that the algorithm, once applied on a finite number of rules will terminate. Termination is constructed by observing the behavior of each step in the algorithm, and showing that one dependence is solved in every iteration, the space is finite, and it is decreasing in every iteration, starting with a set of rules, in every iteration of the algorithm this set will be reduced to $\mathcal{R}(\mathcal{F}) = \mathcal{R}(\mathcal{F}) - \{r_i\}$, and eventually, $\mathcal{R}'(\mathcal{F})$ becomes $\emptyset$, which is the precondition for algorithm termination.

4.3. Firewall Rulebase Example

An example of a firewall rulebase is given below. The firewall checks its rules when a packet arrives to its entry, the corresponding chain of rules decides if the packet must be dropped or allowed to pass.

```
r1 = If source IP address = 10.*.*.*, DENY

r2 = If source IP address = 192.168.*.*,ACCEPT

r3 = If source IP address = 0.0.0.0, DENY

r4 = If source IP address = 10.1.*.* to
     10.3.*.*, DENY

r5 = If source IP address = 60.40.*.*, ACCEPT

r6 = If source IP address = 1.2.3.4, DENY


r7 = If destination IP address = 60.47.3.9 AND
     destination port=80 OR 443, ACCEPT


r8 = If destination IP address = 60.47.3.* AND
     destination port= 21, ACCEPT

DENY ALL default rule
```

First we obtain $\mathcal{F} = r_1, r_2, \ldots, r_8$, we first identify the set of rules to be $\mathcal{R}(\mathcal{F}) = \{r_1, r_2, \ldots, r_8\}$. Applying the algorithm for the first iteration, $r_i = r_1$ means that $n_s = 10.*.*.*$, and $n_d = null$, applying domain restriction, $R_s = \{r_1, r_4\}$, $R_d = \emptyset$, then, applying co-domain restriction, $R_{sa} = \emptyset$, $R_{sd} = \{r_1, r_4\}$, $R_{da} = \emptyset$, and $R_{dd} = \emptyset$, and therefore $\mathcal{CS} = \emptyset$, then $\mathcal{R}(\mathcal{F}) = \{r_2, \ldots, r_8\}$. We repeat the same procedure for all rules, and $\mathcal{CS}$ will remain empty, which indicates that there is no conflict in the above rulebase. Now we consider that an additional rule r_9 is added:

```
r9 = If source IP address = 10.40.*.*, ACCEPT
```

For the new rulebase, $\mathcal{R}(\mathcal{F}) = \{r_1, r_2, \ldots, r_9\}$. Applying Algorithm 2 and starting with $r_i = r_1$, we obtain $n_s = 10.*.*.*$ and $n_d = null$, then $R_s = \{r_1, r_4, r_9\}$, $R_d = \emptyset$, next step, $R_{sa} = \{r_9\}$, $R_{sd} = \{r_1, r_4\}$, $R_{da} = \emptyset$, and $R_{dd} = \emptyset$. Since $R_{sa} \neq \emptyset \wedge R_{sd} \neq \emptyset$ then, $\mathcal{CS} = \{r_1\}$, which indicates that this rule has conflict with another one in the rulebase.

The above algorithm can be applied dynamically while updating the rulebase. In this case the complexity of the algorithm is reduced to linear for a newly added rule. The algorithm can verify consistency of firewall rulebase, and in addition, identify all the rules with conflicts if they occur.

5. CASE STUDY

In this section we present an implementation of the domain restriction method in Event-B, and illustrate it on a firewall example. Many of the algorithm steps are internally executed by Event-B operators.

5.1. Event-B

Event-B [4] is a formal method for modeling guarded operations. The Event-B method provides invariants proofs for state- based systems that are updated by guarded events. Event-B has been shown suitable to perform verification of wide range of systems, but have not been explored for checking properties over firewall configurations. Since Event-B provides a library of operators for set theory operations in first-order logic, we can use it efficiently for the implementation of our method. On the other hand, the Event-B language [16] allows modeling firewall specifications at different levels of abstraction.

In Event-B [4], the guard is a predicate built on state variables while an action is a generalized substitution that defines a state transition. A guard activates an event when it evaluates to true. A descriptive specification describes what the system does by using a set of variables,

constants, properties over constants and invariants which specify properties that the machines state verify.

The correctness of an event-B model is established by proof obligations for the invariants, where each event, including the initialization event, should preserve these invariants. Event-B guards are used to define preconditions that should hold before the event can be executed. The guard and the action of an event defines a relation between variables before the event holds and after. Proof obligations are produced from events in order to state that the invariant condition is preserved. These proof obligations need to be verified in order to proof the correctness of the invariants.

The Rodin tool [17] is a theorem prover that is designed to run automatically and use a large library of mathematical rules, provided with the system, however, interactive guidance from the user is required for certain proof obligations. We use the Rodin platform in order to define and implement two models for the firewall: an abstract model at the network address level and a refined model at the IP address level. In addition, the consistency of firewalls configurations are defined as Event-B invariants, and then are verified for the refined model by discharging all the proof obligations generated by the tool.

5.2. Verification of Firewall Configurations

An example of a firewall is given in Figure 1, where net_a, net_b, net_c, and net_d, along with their architecture, in addition to the IP addresses are only illustrative. The firewall contains filter rules. When a packet arrives to its entry, the corresponding chain of rules decides if the packet must be dropped or must continue its traversal of the rules. The chain is made up of a list of rules. When inspecting packets, chains use the following: the source and destination network, the IP source address, the IP destination address, the protocol, the source port and the destination port.

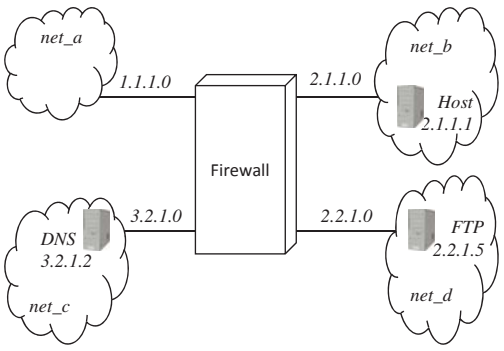

Fig. 1. A Firewall Controlling Traffic in a Network

We consider representing the network at different levels of abstraction. With each refinement of the network, we add more details about the address, and hence, we obtain more concrete firewall rules. In order to illustrate our verification method, we chose a network represented by three zones controlled by a firewall whose initial configuration corresponds to a given set of rules. For the firewall controlled network given in Figure 1, will consider the a security policy, which can be implemented in a firewall configuration that is composed of a set of rules, for instance, consider the following set of rules:

$\mathcal{R}1$: net_a has the right to access net_b.

$\mathcal{R}2$: net_b has the right to access net_c.

$\mathcal{R}3$: net_c has no right to access net_a.

$\mathcal{R}4$: host in net_b has the right to access DNS and FTP servers.

$\mathcal{R}5$: net_c has no right to access an FTP server in net_d.

The above firewall rules can contain any of the following parameters: network, IP address, protocol or and port. At the first level of abstraction, we consider network parameters, where we abstract rules to be at the network level. The abstract rules are contained in the concrete ones. An Event-B model is defined for this level of abstraction along with certain invariants to check the consistency of the firewall configuration.

Abstraction of the above set of rules results in an abstract configuration at the network level. Note that this abstraction may result in contradiction in case certain part of one network is granted access to an outer network, and another part of the same network is denied access to that outer network. Abstraction here will definitely result in contradiction between the abstract rule and one of the concrete rules. To overcome this, we consider both rules result in an undefined action. This issue, however, will be resolved at the refined level when we consider the model at the IP address level where both cases can be defined with the appropriate action.

In the first step we will present an abstract model of the above policy at the network level. We define two types *ACTION* and *NET*, the first type defines behavior of the firewall on filtered packets: *ACCEPT* or *DENY*. In order to have a complete specification, which is necessary for proof obligations discharge, we assume that there is always a default last rule that is either *ACCEPT* or *DENY*.

CONTEXT X0
SETS
ACTION NET
CONSTANTS

ACCEPT DENY
net_a net_b net_c net_d

AXIOMS

axm1 : $partition(ACTION, \{ACCEPT\}, \{DENY\},)$
axm2 : $partition(NET, \{net\_a\}, \{net\_b\}, \{net\_c\}, \{net\_d\})$
axm3 : $DENY \neq ACCEPT$
usu

END

MACHINE F0
SEES X0
VARIABLES
RULE
INVARIANTS
inv7 : $RULE \in NET \times NET \rightarrowtail ACTION$
EVENTS
Initialisation
begin
act2 : $RULE :=$
$\{(net\_a \mapsto net\_b) \mapsto ACCEPT,$
$(net\_b \mapsto net\_c) \mapsto ACCEPT,$
$(net\_c \mapsto net\_a) \mapsto DENY,$
$(net\_b \mapsto net\_d) \mapsto ACCEPT,$
$(net\_c \mapsto net\_d) \mapsto DENY\}$
end
END

In the above Event-B model, the set *RULE* is defined as a total injection function from the set of $NET \times NET$ into the set *ACTION*. The actual rules of the firewall configurations are defined in the initialization event, where the address of every rules is abstracted into the network address. This may result in merging two or more rules into one single rule.

This model is refined by defining the network address NET over the range of possible IP addresses in the axioms below. In order to make the variants below readable and simple, we assume IP addresses are of class A only, however, the invariants can simply be extended to include other classes by adding the constraints below.

The firewall configuration is defined at this level by mapping every pair of addresses (source and destination) into its possible action throughout the term $RULE$ below. Rules are checked for consistency in Rodin by evaluating the invariant $RuleCheck$ below for the rules map that defines the firewall configurations. This invariant generates a number of proof obligations that were discharged using the Rodin proof control.

CONTEXT X1
CONSTANTS
RN NET
AXIOMS
axm1 : $RN \in 0..255$
axm2 : $NET \in RN \times RN \times RN \times RN$

END

MACHINE F1
SEES X1
VARIABLES
RULE
INVARIANTS
inv1 : $RULE \in NET \times NET \rightarrowtail ACTION$
RuleCheck : $\forall src1, src2, dst1, dst2 \cdot src1 \in NET \wedge src2 \in NET \wedge dst1 \in NET \wedge dst2 \in NET \wedge (RULE(src1 \mapsto dst1) = ACCEPT \wedge RULE(src2 \mapsto dst2) = DENY) \Longrightarrow \neg(src1 = src2 \wedge dst1 = dst2) \wedge ((NetAdd(src1) = \top) \vee (NetAdd(src1) = \top \wedge SUBNET(src1) \neq SUBNET(src2))) \wedge ((NetAdd(src2) = F) \vee (NetAdd(src2) = \top \wedge SUBNET(src1) \neq SUBNET(src2))) \wedge (NetAdd(dst1) = \top) \vee (NetAdd(dst1) = \top \wedge SUBNET(dst1) \neq SUBNET(dst2))) \wedge ((NetAdd(dst2) = F) \vee (NetAdd(dst2) = \top \wedge SUBNET(dst1) \neq SUBNET(dst2)))$
EVENTS
Initialisation
begin
act1 : $RULE := \{$
$(1 \mapsto 1 \mapsto 1 \mapsto 0 \mapsto 2 \mapsto 1 \mapsto 1 \mapsto 0) \mapsto ACCEPT,$
$(2 \mapsto 1 \mapsto 1 \mapsto 0 \mapsto 3 \mapsto 2 \mapsto 1 \mapsto 0) \mapsto ACCEPT,$
$(2 \mapsto 2 \mapsto 1 \mapsto 0 \mapsto 1 \mapsto 1 \mapsto 0 \mapsto 0) \mapsto DENY,$
$(2 \mapsto 1 \mapsto 1 \mapsto 1 \mapsto 3 \mapsto 2 \mapsto 1 \mapsto 2) \mapsto ACCEPT,$
$(2 \mapsto 1 \mapsto 1 \mapsto 1 \mapsto 2 \mapsto 2 \mapsto 1 \mapsto 5) \mapsto ACCEPT,$
$(3 \mapsto 2 \mapsto 1 \mapsto 0 \mapsto 2 \mapsto 2 \mapsto 1 \mapsto 5) \mapsto DENY\}$
end
Event *evt1*
any
$w4\ \ w3\ \ w2\ \ w1$
where
grd1 : $w1 = 0 \wedge w4 \in 0..255 \wedge w3 \in 0..255 \wedge w2 \in 0..255$
then
act1 : $NetAdd(w4 \mapsto w3 \mapsto w2 \mapsto w1) := TRUE$
end
Event *evt2*
any
$w4\ \ w3\ \ w2\ \ w1$
where
grd1 : $w1 = 0..255 \wedge w4 \in 0..255 \wedge w3 \in 0..255 \wedge w2 \in 0..255$
then
act1 : $SUBNET(w4 \mapsto w3 \mapsto w2 \mapsto w1) := w4 \mapsto w3 \mapsto w2$
end
END

The next step is to implement the set theory primitives and their domain restriction operators in Event-B. These operators are embedded in the platform, therefore,

we directly use them to implement our model. The consistency of the firewall configurations is defined using the invariant, *inv3*, within the Rodin platform, as shown below. The tool generates proof obligations that were successfully discharged using Event-B proof control.

MACHINE F2
SEES X2
VARIABLES
`n Rs Rd Aa Ad Rsa Rsd Rda Rdd`
INVARIANTS
`inv1` : $Rs \in RULE \land Rd \in RULE \land Rsa \in RULE \land Rsd \in RULE \land Rda \in RULE \land Rdd \in RULE$
`inv2` : $Aa \in ACTION \land Ad \in ACTION$
`inv3` : $Rsa \cap Rsd = \emptyset \ \land \ Rda \cap Rdd = \emptyset$
EVENTS
Initialisation
begin
`act1` : $Aa := \{ACCEPT\}$
`act2` : $Ad := \{DENY\}$
end
Event *evt1*
any
n
where
`grd1` : $n \in NET$
then
`act1` : $Rs := n \lhd RULE$
`act2` : $Rd := n \rhd RULE$
`act3` : $Rsa := Aa \rhd Rs$
`act4` : $Rsd := Ad \rhd Rs$
`act5` : $Rda := Aa \rhd Rd$
`act6` : $Rdd := Ad \rhd Rd$
end
END

The above example shows that the method allows modeling the firewall configurations at different levels of abstraction. We presented a high level model at an abstract network address level. This model is further refined in order to include more details about the addresses in firewall rules, while preserving the correctness of the invariants. The verification of a more refined model will be straightforward, and will require a refinement of this model based on addresses by including protocol types or port numbers. This is going to be covered in a future work. In order to make this method more appealing and applicable on industrial size firewalls, an interface is required in order to map firewall rules into Event-B data structure models. The semantics of this translation can be deduced using our model, and the interface can provide an automatic translation from firewall configuration rules into Event-B syntax. This issue will be addressed in the future work.

6. CONCLUSION

Firewall configurations and the maintenance of their rulebase is highly error prone, therefore, the verification of their correctness is essential. In this paper we present a formal model for firewall configuration rules based on domain restriction. This model is used to define an algorithm to formally verify the consistency of the configuration rules in firewalls. The algorithm is based on a formal model that utilizes the domain restriction method we presented in our previous work [5]. Compared to [5], where the verification method was based on Event-B theorem proving, in this work we present an implementation independent algorithm based on a formal model for firewalls rulebase.

We illustrate our method on a case study by modeling firewall configurations at the network level of abstraction, then, we refine this model by considering the network at the IP address level. Firewall configuration rules are embedded in Event-B, where the consistency of firewall configurations is defined as Event-B invariants, then the Rodin firstorder theorem prover is used to prove the consistency of this configuration by proving each of the proof obligations automatically.

The advantage of our method is the ability to model firewall configurations at different levels of abstraction. A high level model representing firewall rules at an abstract network address level is used first. This model is further refined by using IP network addresses in firewall rules, while preserving the correctness of the invariants, and hence the consistency of firewall configurations. This method can model firewall configuration rules at the network address level of abstraction, which we believe is the major domain, where most conflicts happen to be in firewalls rulebase. However, the method still can be modified to support the detection of conflicts in rules at the protocol and port level of abstraction. This issue will be considered for future work. The algorithm is implementation independent, another method to implement it would be using an existing SAT solver. Alternatively, it can be integrated into an existing theorem prover that supports first-order set theory operations such as the Event-B or HOL theorem prover [18].

As further future work, we will provide a formal proof of the correctness of the method by showing the completeness and soundness of the presented model. In addition we intend to use the same method to prove firewall consistency at more refined levels by allowing rules at the protocol and port number levels.

7. REFERENCES

[1] D. Chapman and E. Zwicky, *Building Internet Firewalls*, Orielly & Associates Inc., 2000.

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

[5] A. Gawanmeh and S. Tahar, "Modeling and Verification of Firewall Configurations Using Domain Restriction Method," in *IEEE International Conference on Internet Technology and Secured Transactions*. pp. 642–647, IEEE Computer Society Press, 2011.

[6] T. Abbes, A. Bouhoula, and M. Rusinowitch, "An Inference System for Detecting Firewall Filtering Rules Aanomalies," in *ACM Symposium on Applied computing*. pp. 2122–2128, ACM press, 2008.

[7] 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*. pp. 526–531, IEEE Computer Society Press, 2009.

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

[9] 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*. pp. 171–181, IEEE Computer Society Press, 2008.

[10] F. Cuppens, N. Cuppens-Boulahia, T. Sans, and A. Miège, "A Formal Approach to Specify and Deploy a Network Security Policy," in *Formal Aspects in Security and Trust*. vol. 173 of *Lecture Notes in Computer Science*, pp. 203–218, Springer-Verlag, 2004.

[11] L. Yuan, H. Chen, J. Mai, C. Chuah, Z. Su, and P. Mohapatra, "FIREMAN: a Toolkit for Firewall Modeling and Analysis," in *IEEE Symposium on Security and Privacy*. pp. 199–213, IEEE Computer Society Press, 2006.

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

[13] 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*. pp. 706–710, IEEE Computer Society Press, 2011.

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

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

[16] C. Metayer, J. Abrial, and L. Voisin, "RODIN Deliverable 3.2: Event-B Language," Tech. Rep. Project IST-511599, School of Computing Science, University of Newcastle, UK, 2005.

[17] *Rodin Platform*, "http://www.event-b.org, 2010," .

[18] *HOL Sourceforge Project. The HOL System Reference*, "http://hol.sourceforge.net, 2011," .