Distributed Firewall Anomaly Detection Through LTL Model Checking

Sylvain Hallé and Éric Lunaud Ngoupé
Université du Québec à Chicoutimi, Canada
Email: shalle@acm.org, eric-lunaud.ngoupe@uqac.ca

Roger Villemaire and Omar Cherkaoui
Université du Québec à Montréal, Canada
Email: {villemaire.roger,cherkaoui.omar}@uqam.ca

***Abstract*—An anomaly in a firewall is a relationship between two of its rules that may hint at a possible misconfiguration of its filter. While checking anomalies within a single firewall is well understood, identifying anomalies across multiple firewalls in a network is a much harder problem that has only been studied for restricted cases. In particular, we show that the correct identification of anomalies must take into account the routing function performed in each node of the network. We introduce a formal model of firewalls and routing tables that generalizes past work on the topic; in particular, we show how the detection of anomalies in this model reduces to the model checking of particular instances of Linear Temporal Logic formulæ. An implementation of an anomaly detector that leverages existing model checkers reveals that distributed anomalies can be identified at a reasonable cost.**

Keywords*—*firewall; rules; Linear Temporal Logic; model checking

I. Introduction

The management of network devices is an intrinsically complex and error-prone task. While each piece of equipment may be analyzed and deemed correct when considered in isolation, the combined interaction of multiple such elements may give rise to unforeseen and often undesirable side effects. Network firewalls are no exception in this respect. Their overall behaviour results from the subtle interaction of tens of filtering rules, and it has long been recognized that an improper configuration of these rules may compromise the correct enforcement of an organization's security policy.

It has hence become desirable to develop tools and techniques to analyze firewall rule bases and identify so-called *anomalies*: relationships between rules that hint at a potential misconfiguration. In Section II, we illustrate on a simple example a set of widely-accepted anomalies; more importantly, we argue that, when considering multiple firewalls dispersed throughout an arbitrary *network*, the correct discovery of anomalies must take into account the routing tables present in every node. However, in Section III, we shall see that research on anomalies in the past has mostly concentrated on single firewalls, and that distributed anomalies have only been studied for restricted cases.

In this paper, we describe a reduction of the problem of checking distributed firewall anomalies to the problem of model checking in Linear Temporal Logic (LTL). Our formal model, described in Section IV, generalizes previous work on the subject in three different respects:

1) The detection of anomalies takes into account the interplay between firewall rules and each node's routing table. An anomaly between two rules is not reported if no packet can visit those rules in the correct order, given the network's routing pattern.
2) The network considered can have an arbitrary topology, and is not restricted to firewalls arranged in a regular tree structure.
3) The anomalies studied in past literature merely become particular instances of LTL formulæ. As such, any other anomaly expressible as an LTL formula can be readily verified by our framework.

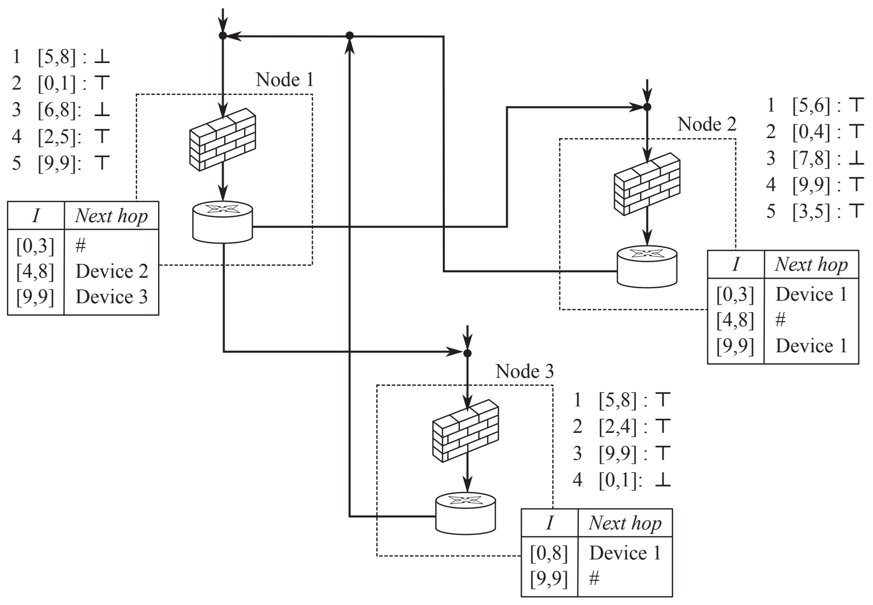
In Section V, we report on experiments made on the implementation of an anomaly detector based on that model, and leveraging existing software called model checkers. We show that, despite its increased generality, our detector accurately spots anomalies in complex networks made of thousands of firewall rules on the order of minutes.

II. Firewall Rule Analysis

In order to secure a corporate network or some subnetwork within it, network traffic is usually filtered according to such criteria as origin, destination, protocol and service. Typically *firewalls* are equipped with *filters* specifying which packets should be forwarded and which should be discarded.

Figure 1 shows a simple setup for firewalls distributed across a network. The network is composed of a number of nodes, each made of an ingress firewall component filtering traffic from outside the node's subnet, and of a routing component responsible for patching the traffic to the next hop based on a routing table. A *filter* is a sequence of *rules* that are tried in order, up to the first matching one. A rule consists of a *condition*, which is a region of the packet's space (in our example, we only show the destination address field), and of a *decision* —usually accept ($\top$) or deny ($\bot$). Each packet hence goes through the rules in sequence up to the first matching condition, whose decision determines whether the packet is forwarded or discarded. In the following, we shall use the notation *x.y* to designate filter rule *y* of node *x*.

The packet is then processed by the routing component, which sends it to the next hop according to its destination address. A special symbol, labelled #, does not stand for any actual node. When # occurs as the destination of a routing rule, it indicates that the packets leave the network. This symbolizes either that the node is directly connected to the destination, or that the next hop is a node outside the network under



I	*Next hop*
[0,3]	#
[4,8]	Device 2
[9,9]	Device 3

I	*Next hop*
[0,3]	Device 1
[4,8]	#
[9,9]	Device 1

I	*Next hop*
[0,8]	Device 1
[9,9]	#

Figure 1. A network of firewalls and routers. The firewall filter and routing table for each node is shown.

consideration. While in our setup filtering is done before routing, the opposite order can easily be taken into account by reversing the order of those two steps throughout. Since the state of routing tables may vary with time through the use of dynamic routing protocols such as RIP and OSPF, we shall assume in the following that the tables model an over-approximation of the possible paths in the network.

A. Anomalies

Configuring a filter is a well-known error-prone task. Network management researchers have introduced filter properties, called *anomalies*, which either reveal or hint to a possible misconfiguration. In particular, Al-Shaer's work [1], [2] considered the following cases, involving a pair of rules.

1) Shadowing: A rule r_1 is *shadowed* if there is a rule r_2, preceding r_1 in the filter, that already blocks all packets covered by r_2. This anomaly can occur either within a single firewall, or across firewalls: for instance in Figure 1, rule 1.2 is shadowed by rule 3.4. This entails that a packet with destination address 0 will be rejected if it enters through Node 3, while Node 1 would have accepted it.

2) Spuriousness: Spuriousness is the opposite of shadowing: this anomaly occurs when some rule in a firewall allows traffic that a later rule will reject. For example rule 3.1 and rule 1.1 present a spuriousness anomaly: a packet with destination address 5 will be rejected right away if it enters the network through Node 1, while the same packet entering at Node 3 will be accepted, and routed to Node 1 where it will then be rejected.

3) Redundancy: A rule is redundant if it is shadowed by a rule with the same decision. If both rules lie within the same firewall, the redundant rule can be removed without changing the packets that are accepted. This is the case for rule 1.3 which is shadowed by rule 1.1. In the case of distributed firewalls, however, discovering a redundancy anomaly does not entail that the rule may be removed immediately; one has to inspect further and check whether the firewall receives its traffic exclusively and directly from the upstream firewall containing the duplicate. For example, if Node 3 received traffic solely from Node 1, then rule 3.3 3 would merely repeat a decision already taken by rule 5 in Node 1, and could be removed.

4) Correlation: Finally, correlation happens when a later rule matches some packet already matched by r while having a different decision.

As was stressed by [1]–[3], anomalies do not entail that the firewalls present errors; they only *hint* at potential misconfigurations, on the grounds that the administrator's intent may be ambiguous.

B. The Impact of Routing

In the case of firewalls distributed across a network, it does not suffice to compare the rules in a pairwise fashion as the previous definitions seem to imply. Obviously, one must take into account the topology of the network to determine whether two rules r, r' can be processed in that order. In our example, it does not make sense to conclude that rule 1 in Node 3 shadows Rule 3 in Node 2, as no traffic ever goes through Node 3's firewall followed immediately by Node 2. Rather, that traffic must flow through Node 1 in between —and all traffic destined to Node 2's rule 3 will be discarded by Node 1's rule 1 beforehand. Hence the sequence in which firewalls can be visited matters in the analysis.

Not only is this sequence relevant, but the actual routing function performed in each node is also important to discover rule anomalies. For example, it would seem natural to declare that rule 2.1 and rule 1.1 present a spuriousness anomaly. However, one shall remark that the packets matching rule 1 in Node 2 are never routed to Node 1: Node 2 is directly connected to those destinations. Simply comparing the rules, without taking into account the actual routing information, makes us falsely conclude to the presence of an anomaly.

Similarly, one might conclude that Rule 1.1 shadows Rule 3.1; this, however, is not true, as the only packets that may hop from Node 1 to Node 3 are those with destination address 9; therefore neither of the rules involved in the anomaly apply to that traffic, and we are this time witnessing "false shadowing". Finally, rule 1.4 seems to make rule 2.5 redundant; however only packets in the interval $[4,8]$ flow from Node 1 to Node 2; the projection of rule 1.4 over this range no longer covers the entire interval of rule 2.5, and hence redundancy was falsely concluded.

These examples show that an appropriate framework for the detection of anomalies in firewall rules should model the nodes' routing tables, and take this information into account to automatically discard candidate pairs of rules that do not correspond to a possible path in the network. As a matter of fact, one has to intersect the first rule's interval with the interval of every routing rule fired along the path to the second rule, in order to correctly conclude that an anomaly is indeed present.

III. Related Work

The detection of anomalies in firewall rules has been the subject of a number of related works in the past. We shall first dismiss works such as [4], for which errors amount to the presence (or absence) of specific rules inside firewalls; examples of such errors include "Allowing TCP on port 23" or "Outbound POP3". In contrast, the firewall anomalies we are concerned with correlate fields and decisions of two different entries in a firewall rule base.

The remaining works can be divided in two categories, whether they attempt to discover anomalies within a single firewall, or consider the distributed case.

A. Intra-Firewall Anomalies

The Firewall Policy Adviser [3] is one of the earliest tools for firewall analysis. It uses Binary Decision Diagrams (BDDs) to represent rules, as is the case for FIREMAN [5]. This approach has been evaluated experimentally, detecting anomalies in a 800-rule filter in less than 3 seconds. Alternatively, a tree structure that represents a spatial decomposition of regions into non-overlapping axis-parallel regions is used in [6] in a prototype tool. Special decision tree data structures have also been introduced in [7], [8] to process sequences of regions, but with neither theoretical nor experimental evaluation.

While these tools are efficient at detecting a predefined set of firewall anomalies, their algorithms are hard-coded and built-in. To detect different patterns in rule bases, one therefore has to modify the existing tools, and implement the algorithms that detect the desired patterns. In contrast, the Margrave tool [9] allows a user to write queries in a first-order language; however, this language is closer to a scripting language, and ultimately amounts to the user programming the desired detection mechanism.

It was suggested in [10] that anomalies rather be expressed as formulæ in a more general language called Visibility Logic. Alas, VL is a relatively new concept, and verification of VL formulæ currently works only for single firewalls.

B. Inter-Firewall Anomalies

In contrast with intra-firewall anomalies, very few works tackled the problem of detecting inter-firewall configuration anomalies. Since the presence of multiple firewalls brings up the question of paths in the network, a first class of related works used model checking techniques; [11] models the path of a packet into a network to check, for example, that two given endpoints are reachable, or to discover quality of service violations. A similar approach is followed by [12], which uses a Boolean satisfiability solver to check two properties on multi-firewall rule base: 1) reachability (for each rule, there exists a packet that fires that rule) and 2) acyclicity (no packet returns to a firewall that already processed it).

Both these frameworks model the path of *individual* packets into a network. This approach, however, is not appropriate for the discovery of the rule anomalies considered in the present paper. For example, discovering that some packet is accepted in some firewall A and gets re-accepted by the next firewall B does not reveal a redundancy anomaly. Rather, redundancy requires that all packets accepted by the *same* rule in B are also accepted by the *same* rule in A. The discovery of firewall anomalies requires reasoning on intervals, not packets.

This is the approach followed by Al-Shaer's work [1], [2], which presents an extension of the Firewall Policy Adviser for the distributed case. The algorithm computes whether a single pair (r_1, r_2) of rules exhibits any of the aforementioned anomalies. Checking an entire firewall rule base involves repeating the process for every possible pair of rules along every possible path in the network. The same principle is used in another firewall analysis tool called Prometheus [13]. We shall see in Section V how this approach presents severe scalability issues; they were mitigated by the fact that the case study considered a small number of firewalls arranged in a directed tree, thereby considerably limiting the number of possible paths in the network. Moreover, in neither case, the routing information is used to compute the presence of anomalies, thereby leading to the false positive issue described earlier.

Finally, recent work by Rupali [14] discusses inter-firewall anomalies but only provides experimental results for intra-firewall analysis.

IV. A Formal Model of Distributed Firewalls

To address the issues mentioned above, we shall define in this section a new formal model for the detection of firewall anomalies. The model can be seen as a hybrid of the two main approaches discussed above: the system manipulates intervals of values, but reasons over them using model checking techniques. In line with our example in Figure 1, the presentation is built by taking into consideration a single packet field (the packet's destination address). Other fields can later be added to the model in a straightforward way.

A. Firewall and Routing Tables

Let $A = \{0, 1, \ldots, k\}$ be a range of contiguous numbers standing for values of some packet field, where $\max A = k$ denotes the upper bound of the range. Let $I = \{(x,y) \in A^2 : x \leq y\}$ be the set of valid intervals over A. The set of firewall rules is defined as $R_F = I \times \{\top, \bot\}$; that is, each rule is a tuple (i,x), where $i \in I$ is an interval and x is either "accept" ($\top$) or "reject" ($\bot$). The set of firewalls $F = (R_F)^*$ is the set of all possible sequences of rules taken from R_F. In the following, instance of firewalls will be denoted $\overline{f}$, while firewall rules will be denoted as f.

It shall be noted that this representation of intervals is more general than, e.g. [2], which assumes hierarchical intervals, where only some rightmost string of bits can vary (such as in 10.10.10.*); consequently, if two intervals are not disjoint, one is necessarily included within the other —intervals can never partially overlap. In contrast, our model supports *arbitrary* intervals. Hence, it can be applied without modification to any packet field, including those that are not hierarchical in nature, such as port ranges.

A similar treatment can be made for routing tables. Let D be a set of device (or node) names. Since we assume that the single packet field stands for its destination address, the set of routing rules can hence be defined as the set $R_R = I \times D$. That is, given a destination range, a routing rule gives the next hop for a packet whose field lies in that range. The set of all routing tables is then $R = (R_R)^*$. In the case where a routing table contains two rules $(I,d),(I',d') \in R_R$ such that $I \cap I' \neq \emptyset$ and $d \neq d'$, the conflict is interpreted as a non-deterministic choice: packets whose destination lies in $I \cap I'$ may be forwarded either to d or to d'.

A network node is made of an ingress firewall and a routing table. The set of network nodes is therefore $N_N = F \times R$. Finally, a network N is a function from node names to network nodes; formally, $N : D \to N_N$; hence the expression $N(d)$ returns the node $(\overline{f}, \overline{r}) \in N_N$ consisting of the firewall and routing table for the node named d. We will abuse notation and write $N(d) = \emptyset$ to denote the fact that N contains no device named d. Similarly, we shall write $n \in N$ to denote that there exists a device name $d \in D$ such that $n = N(d)$. We assume the network to be *consistent*: for each tuple $(d,(\overline{f},\overline{r})) \in N$ and for each routing rule $(i,x) \in \overline{r}$, either $N(x) \neq \emptyset$ or $x = \#$ —that is, every routing rule leads to a next hop that exists in the network or has a packet leave altogether.

For a list $\overline{r} = r_0, r_1, \ldots$ of routing (resp. firewall) rules, we denote $\overline{r}[k]$ as the k-th routing (resp. firewall) rule of $\overline{r}$ (i.e. r_k). The length of $\overline{r}$ will be written $|\overline{r}|$. We will also assume that each firewall rule in the network is given a unique global number, and let K be the set of such numbers. K will also be overridden as a function $R_F \to K$; given some rule r in some firewall of some network node, we denote by $K(r)$ the unique number associated to that rule.

B. From a Network to a Kripke Structure

From a network N formalized as above, we then proceed to build a special kind of finite-state machine M that will be used for verification purposes. Intuitively, each state in M stands for one of the firewall rules in the nodes of N; the states will be connected in such a way that a path in M represents a possible sequence in which the firewall rules may be processed.

Somewhere along a path, the system takes a "snapshot" of the current rule, and memorizes that snapshot for the remainder of the trace. That snapshot contains the rule's interval bounds and decision. The choice of whether to take a snapshot is non-deterministic; this entails that, for every possible sequence of rules $r_1, r_2, \ldots, r_m$ in N and every rule r_i in that sequence, there exists a trace in M that follows the same rules in the same sequence and that takes its snapshot at rule r_i.

Using this mechanism, an anomaly in the firewall rules will hence express a property of the states along some path of M, where the snapshot of a past rule and the fields of the current rule are in a specific relationship. To represent that information, we define the structure of M in terms of the following 8 state variables:

- $\chi \in K$ is the unique number associated to each rule;
- $\rho_L, \rho_R \in A$ are respectively the left and right bounds of the rule's interval;
- $\rho_D \in \{\top, \bot\}$ is the rule's decision;
- $\iota_L, \iota_R \in A$ and $\iota_D \in \{\top, \bot\}$ are called *freeze* variables; they carry the same meaning as their ρ counterparts, and are used to memorize (or "freeze") the values of some past rule for later comparison;
- $\iota_F \in \{\top, \bot\}$ is an auxiliary variable that indicates whether the other ι have already frozen some rule or are still uninitialized.

Formally, we build a *Kripke structure* $M = \langle S, S_0, \delta, L \rangle$, where S is a set of states, S_0 a set of initial states, and $\delta \subseteq S^2$ a transition relation. Each state is defined uniquely by the values taken by each state variable; the function L, called a state labelling, associates to each state s and each state variable a value in their respective domains. Hence the expression $L(s_3, \rho_D)$ designates the value in A of state variable ρ_D in state s_3.

To completely characterize M, one needs to define S_0 and the transition relation δ.

1) Initial States: We first assume that traffic may enter the network from any network node. Therefore, the first state of a trace in M should correspond to an encoding of the topmost firewall rule of any node, hence deciding on the possible values for χ, ρ_L, ρ_R and ρ_D. In the first state of the trace, no snapshot has yet been taken, hence $\iota_F = \bot$ in all initial states of the structure. Since the snapshot interval has not been constrained in any way, the whole range of values are possible, and therefore $\iota_L = 0$ and $\iota_R = \max A$. This can be formalized as follows:

Definition 1 (Initial states)**.** *Let $a, a' \in A$, $x \in \{\top, \bot\}$, $k \in K$ be arbitrary values and $s \in S$ be the state of M uniquely defined as: $L(s,\chi) = k$, $L(s,\rho_L) = a$, $L(s,\rho_R) = a'$, $L(s,\rho_D) = x$, $L(s,\iota_L) = 0$, $L(s,\iota_R) = \max A$, $L(s,\iota_D) = \bot$, $L(s,\iota_F) = \bot$. Then s is an initial state of M if and only if there exists a network node $n = (\overline{f}, \overline{r}) \in N$ such that $\overline{r}[0] = ((a,a'),x)$ and $K(\overline{r}[0]) = k$.*

To take into account the fact that some nodes may not be entry points to the network, it suffices to impose that n be designated as an ingress node using some additional marking, which we omit from the presentation.

2) Transition Relation: The transition relation describes how the system may move from one state to another. We need to distinguish between two different cases. The first case occurs when the current state s encodes any firewall rule of some node, except the last. Since firewall rules are handled sequentially, the next state encodes the next rule in that node's firewall, which determines the next values for χ, ρ_L, ρ_R and ρ_D. If a snapshot has already been taken (i.e. $\iota_F = \top$ in the current state s), the next state retains the values of that snapshot, thereby fixing ι_L, ι_R and ι_D. On the contrary, if no snapshot has been taken, the system can non-deterministically move to a next state with still no snapshot, or freeze the values of the current rule into its variables ι_L, ι_R and ι_D. This can be formalized as a first part of the transition relation:

Definition 2 (Non-final rules). *Let $\delta_1 \subseteq S^2$ be some transition relation. Let $s, s' \in S$ be two states of M, $n = (\overline{f}, \overline{r})$ be some network node of N. Let $f = ((a_L, a_R), x)$, $f' = ((a'_L, a'_R), x')$ be two firewall rules such that $f = \overline{f}[i]$, $f' = \overline{f}[i+1]$ for some $i \in [0, |\overline{f}| - 2]$. Moreover suppose that $L(s, \chi) = K(f)$, $L(s, \rho_L) = a_L$, $L(s, \rho_R) = a_R$ and $L(s, \rho_D) = x$. The transition (s, s') is part of δ_1 if and only if $L(s', \chi) = K(f')$, $L(s', \rho_L) = a'_L$, $L(s', \rho_R) = a'_R$, and either of the following holds:*

1) *$L(s, \iota_F) = \top$ and $L(s', \iota_F) = \top$, $L(s', \iota_L) = L(s, \iota_L)$, $L(s', \iota_R) = L(s, \iota_R)$, $L(s', \iota_D) = L(s, \iota_D)$; or*
2) *$L(s, \iota_F) = \bot$ and either $L(s', \iota_F) = \bot$, $L(s', \iota_L) = L(s, \iota_L)$, $L(s', \iota_R) = L(s, \iota_R)$, $L(s', \iota_D) = L(s, \iota_D)$, or $L(s', \iota_F) = \top$, $L(s', \iota_L) = \max(L(s, \iota_L), a'_L)$, $L(s', \iota_L) = \min(L(s, \iota_R), a'_R)$, $L(s', \iota_D) = x'$*

It shall be noted in case 2 of the above definition that the freezing of some rule's interval into the state variables is actually its *intersection* with whatever interval is already encoded in the freeze variables (hence the use of max and min). This is necessary, for conditions on the packet's interval may have already started to add up if the system has gone through a routing table at least once. An example was given in Figure 1; a packet entering in Device 2 and moving to Device 1 must lie in the interval $[0, 3]$; therefore, taking a snapshot of Device 1's rule 4 according to that path should result in the interval $[2, 3]$ (rather than $[2, 5]$) being frozen for later comparison.

The second case to be considered is when the current rule is the last of the node's firewall. As with the previous firewall states, a snapshot of the current rule's interval may be taken if no interval has yet been frozen. Then, to represent the routing of packets across the network, the next possible rule must be taken according to the node's routing table *and* constraints gathered on the snapshot interval (whether frozen or not). Any routing rule whose interval intersects with that of the snapshot interval qualifies as a possible next hop; the system should then transition to the topmost firewall rule of this next hop. Finally, the hypotheses on the snapshot interval must be updated by intersecting it with the selected routing rule's interval, as was explained above.

This intuition can be formalized as a second part of the transition relation:

Definition 3 (Final rules). *Let $\delta_2 \subseteq S^2$ be some transition relation. Let $s, s' \in S$ be two states of M, $n = (\overline{f}, \overline{r})$ and $n' = (\overline{f}', \overline{r}')$ be two network nodes of N, such that $N(x) = n'$ for some node name $x \in D$. Let $f = ((a_L, a_R), d)$ be a firewall rule such that $f = \overline{f}[i]$, for $i = |\overline{f}| - 1$. Suppose that $L(s, \chi) = K(f)$, $L(s, \rho_L) = a_L$, $L(s, \rho_R) = a_R$ and $L(s, \rho_D) = d$. Let $f' = ((a'_L, a'_R), d')$ be a firewall rule such that $f' = \overline{f}'[0]$. The transition (s, s') is part of δ_2 if and only if:*

- *There exists a routing rule $r = ((b_L, b_R), x) \in \overline{r}$ such that $[b_1, b_2] \cap [L(s, \iota_L), L(s, \iota_R)] \neq \emptyset$, and*
- *$L(s', \chi) = K(f')$, $L(s', \rho_L) = a'_L$, $L(s', \rho_R) = a'_R$, and*
- *Either of the following holds:*
  1) *$L(s, \iota_F) = L(s', \iota_F)$, $L(s', \iota_L) = \max(L(s, \iota_L), b_L)$, $L(s', \iota_R) = \max(L(s, \iota_R), b_R)$, $L(s', \iota_D) = L(s, \iota_D)$, or*
  2) *$L(s, \iota_F) = \bot$, $L(s', \iota_F) = \top$, $L(s', \iota_D) = d$, $L(s', \iota_L) = \max(L(s, \iota_L), a_L, b_L)$, $L(s', \iota_R) = \max(L(s, \iota_R), a_R, b_R)$*

Finally, we represent the fact that a packet leaves the network by adding transitions to a sink state, whenever the destination of the routing rule is # or as soon as $\iota_L > \iota_R$ in the current state (indicating that no packet can ever reach that point due to conflicting conditions). We omit the formal definition of this last condition. The total transition relation for M is then given as $\delta_1 \cup \delta_2$.

C. Firewall Anomalies as LTL formulæ

With the Kripke structure M defined above, the verification of distributed firewall anomalies amounts to the verification of specific conditions on the values of state variables along every possible path of the system. Rather than devise a dedicated algorithm to search for those anomalies in M, we shall express each anomaly as an expression in some forme of logical language called Linear Temporal Logic (LTL) [15].

Formally, a trace $\overline{s}$ of M is a sequence of states $s_0, s_1, \ldots$ such that for every $i \geq 0$, $(s_i, s_{i+1}) \in \delta$. Informally, $\overline{s}$ is the list of states visited by starting at some $s_0 \in S$ and only following transitions defined by δ.

The basic building blocks of LTL formulæ, called *ground terms*, are inequalities over the values of state variables with respect to some current state. For example, the expression $\rho_L \leq \iota_L$ indicates that, in some current state $s \in S$, $L(s, \rho_L) \leq L(s, \iota_L)$.

On top of these propositional variables, LTL allows *Boolean connectives* $\vee$ (or), $\wedge$ (and), $\neg$ (not), bearing their usual meaning and *temporal operators* to express constraints on the sequence of states. The temporal operator $\mathbf{G}$ means "globally"; the formula $\mathbf{G}\,\varphi$ means that formula φ is true in every state of the trace, starting from the current state. The operator $\mathbf{F}$ means "eventually"; the formula $\mathbf{F}\,\varphi$ is true if φ holds for some future state of the trace. The operator $\mathbf{X}$ means "next"; it is true whenever φ holds in the next state of the trace. Finally, the $\mathbf{U}$ operator means "until"; the formula $\varphi\,\mathbf{U}\,\psi$ is true if φ holds for all states until some state satisfies ψ.

The fact that a formula φ holds for some trace $\overline{s}$, noted $\overline{s} \models \varphi$, can be computed recursively as defined in Table I. A Kripke structure M is said to be a model of φ (noted $M \models \varphi$) when φ holds for every trace starting at one of M's initial states.

Using this logic, it is possible to convert the firewall anomalies described in Section II into LTL formulæ.

$\bar{s} \models v_1 \star v_2$	$\equiv$	$L(s,v_1) \star L(s,v_2)$
$\bar{s} \models \neg\varphi$	$\equiv$	$\bar{s} \not\models \varphi$
$\bar{s} \models \varphi \wedge \psi$	$\equiv$	$\bar{s} \models \varphi$ and $\bar{s} \models \psi$
$\bar{s} \models \varphi \vee \psi$	$\equiv$	$\bar{s} \models \varphi$ or $\bar{s} \models \psi$
$\bar{s} \models \varphi \rightarrow \vee\psi$	$\equiv$	$\bar{s} \not\models \varphi$ or $\bar{s} \models \psi$
$\bar{s} \models \mathbf{X}\,\varphi$	$\equiv$	$\bar{s}^1 \models \varphi$
$\bar{s} \models \mathbf{G}\,\varphi$	$\equiv$	$m_0 \models \varphi$ and $\bar{s}^1 \models \mathbf{G}\,\varphi$
$\bar{s} \models \mathbf{F}\,\varphi$	$\equiv$	$m_0 \models \varphi$ or $\bar{s}^1 \models \mathbf{F}\,\varphi$
$\bar{s} \models \varphi\,\mathbf{U}\,\psi$	$\equiv$	$m_0 \models \psi$ or both
		$m_0 \models \varphi$ and $\bar{s}^1 \models \varphi\,\mathbf{U}\,\psi$

Table I. THE SEMANTICS OF LTL OPERATORS. THE SYMBOLS v_1 AND v_2 REPRESENT ARBITRARY STATE VARIABLES OR CONSTANTS; $\star$ IS ANY OPERATOR IN $\{\leq, \geq, =\}$ AND φ AND ψ ARE ARBITRARY LTL FORMULÆ.

1) Shadowing and Spuriousness: We recall that a rule r is shadowed when its decision is accept and there exists a previous rule blocking some traffic and whose interval completely covers that of r. In the Kripke structure M, this will be the case when an interval has been frozen (i.e. $\iota_F = \top$), the bounds of the frozen interval cover that of the current rule (i.e. both $\iota_L \leq \rho_L$ and $\iota_R \geq \rho_R$), when the frozen decision is $\bot$ and that of the current rule is $\top$. The *absence* of such an anomaly can be stated by the fact that, globally along any trace, those four conditions can never be fulfilled simultaneously by the same state:

$$\mathbf{G}\,\neg(\iota_F = \top \wedge \iota_L \leq \rho_L \wedge \iota_R \geq \rho_R \wedge \iota_D = \bot \wedge \rho_D = \top) \qquad (1)$$

Spuriousness is expressed like shadowing, with decisions reversed:

$$\mathbf{G}\,\neg(\iota_F = \top \wedge \iota_L \leq \rho_L \wedge \iota_R \geq \rho_R \wedge \iota_D = \top \wedge \rho_D \neq \bot) \qquad (2)$$

2) Redundancy: Redundancy can be expressed as a variant of the previous formulæ where the decision of both rules involved must match:

$$\mathbf{G}\,\neg(\iota_F = \top \wedge \iota_L \leq \rho_L \wedge \iota_R \geq \rho_R \wedge \iota_D = \rho_D) \qquad (3)$$

3) Correlation: A rule r is correlated when there exists a subsequent rule, having a different decision from r, and whose interval overlaps that of r. In the Kripke structure M, this will be the case when an interval has been frozen (i.e. $\iota_F = \top$) and when the frozen decision is different to that of the current rule (i.e. $\iota_D \neq \rho_D$). Moreover the bounds of the frozen interval must overlap that of the current rule; this occurs when at least one of $\iota_L \leq \rho_L$ and $\iota_R \leq \rho_R$ is true, and at least one of $\iota_L \geq \rho_L$ and $\iota_R \geq \rho_R$ is true. Again, the *absence* of such an anomaly can be stated by the fact that, globally along any trace, those four conditions can never be fulfilled simultaneously:

$$\begin{aligned}\mathbf{G}\,\neg(\iota_F = \top \wedge (\iota_L \leq \rho_L \vee \iota_R \leq \rho_R)&\\ \wedge\, (\iota_L \geq \rho_L \vee \iota_R \geq \rho_R) \wedge \iota_D \neq \rho_D)& \qquad (4)\end{aligned}$$

A formal proof of the correctness of this construction is omitted due to lack of space.

V. IMPLEMENTATION

The results in the previous section provide a straightforward workflow for the detection of anomalies in a network of firewalls. For a given network N, it suffices to build the corresponding Kripke structure M, and then to decide whether $M \models \varphi$, with φ any of the LTL formulæ (1)–(4). If φ does not hold in M (that is, $M \not\models \varphi$), then there exists a *counter-example* trace[1], starting from some initial state and leading into a state that satisfies the undesirable conditions stated by φ —thereby showing why it is false.

Our presentation, however, has not yet described how $M \models \varphi$ can be computed. A frontal approach to this problem would amount to enumerate all traces $\bar{s}$ in M and compute $\bar{s} \models \varphi$ separately for every such trace. This would follow the strategy used in [1]. It is easy to see that this approach hardly scales for networks of even small sizes: for a network of n nodes, there exist up to $\sum_{k=0}^{n} k!$ non-cyclic paths to check, hence yielding a *super-factorial* complexity.

The interest of our construction rather lies in the fact that, when properties on paths are expressed as Linear Temporal Logic formulæ, there exists a much simpler, PSPACE-complete algorithm that can compute $M \models \varphi$ without the need for enumerating every trace. Moreover, existing and heavily-optimized software called *model checkers* can be leveraged to take care of the actual verification. Therefore, no dedicated algorithm needs to be designed from this point on.

A. Experimental Setup

To assess the feasibility of the approach, we implemented a distributed firewall anomaly detector as a Java application made of approximately 1,100 lines of code.[2] Its schematics are described in Figure 2. The anomaly detector takes as input a set of text files describing the firewall rules and routing table of each node in the network. These files are then processed by an Encoder, which is responsible for creating a Kripke structure corresponding to that network by applying the construction described in Section IV.

The Kripke structure is then piped to the standard input of an external model checker. For the present paper, this external software is NuSMV 2.5.4 [16], an open source, state-of-the-art model checker for LTL. Each anomaly to be checked on the network is expressed as an LTL formula and appended to the NuSMV model. As we have seen, these formulæ are independent from the actual network to verify: the same expressions are added upon every invocation of NuSMV.

The model checker is then run as a background command line process, and its output is collected by the verifier. If the given LTL formula is true, then no anomaly has been found and the verifier ends with that verdict. If the formula is false, NuSMV provides, without any further work, a (finite) counter-example trace for that formula; actual values of every state variable are given for every state of the trace. A Decoder takes care of reading that counter-example, discover transitions corresponding to firewall rules, snaphots, and routing rules, and translate that counter-example back into an explanation for the reported anomaly. Figure 3 shows an explanation produced by the tool on the network of Figure 1.

[1]This is not the case for some other temporal logics, such as CTL and CTL*.

[2]http://github.com/sylvainhalle/FirewallCheck

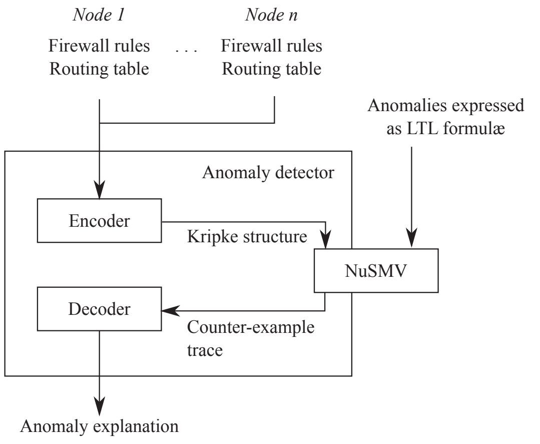


Figure 2. Schematics of the distributed firewall anomaly detector.

```
- Starting at Node 1
- Considering firewall rule 2 on Node 1: [1-5] accept
- Going to Node 2 through routing rule 2:
  Node 1 -> [2-6] -> Node 2
- The considered interval becomes restricted to [2-5]
- Going to Node 3 through routing rule 1:
  Node 2 -> [2-6] -> Node 3
- Shadowing anomaly with firewall rule 2 on
  Node 3: [2-5] accept vs. [3-4] reject
```

Figure 3. Explanation produced by the distributed firewall anomaly detector.

B. Results and Discussion

We then measured the performance of the anomaly detector by analyzing a variety of synthesized network configurations. In all the experiments described, we programatically generated text files containing the firewall and routing table of every node in a network. The tool then followed the workflow shown in Figure 2. The loading of text files, generation of the Kripke structure and interpretation of the counter-example trace take in the order of milliseconds and are hence negligible in the total running time. The processing times are therefore measured as the elapsed time between the cold-start of NuSMV until the termination of its system process. All times have been computed on an AMD Athlon II X4 running at 3.0 GHz under Ubuntu 12.04.

We first repeated the experiments described in [2] and generated sets of random firewalls arranged in a four-tier, directed tree structure. Each upstream firewall in this structure is linked to exactly two or three downstream firewalls, thus yielding a hierarchy of the form 2-2-2 or 3-3-3. The traffic flows in a single direction: no packet is ever routed back to an upstream firewall. It shall be noted that we use this experimental setup so that our results can be compared to past work; no claim is made regarding the realistic nature of this topology. The running times are shown in Figure 4, for nodes of increasing firewall sizes.

The linear slope of the log-log plot, both for the 2-2-2 and the 3-3-3 case indicates that, for a fixed network topology, the detection of anomalies is polynomial in the total number of firewall rules. In the case of the 2-2-2 hierarchy (15 network nodes), it takes between roughly 1 and 10 seconds to analyze the firewall rule base; this time varies approximately between 2 and 20 minutes for the 3-3-3 hierarchy (40 nodes).

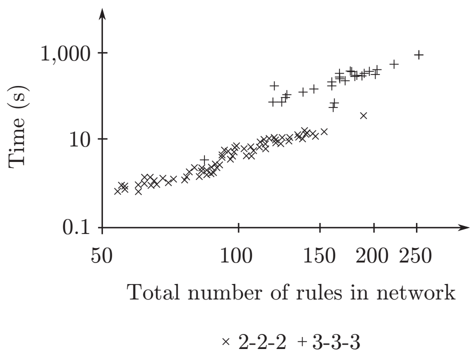


Figure 4. Verification time for two tree configurations, with an increasing number of firewall rules in the network (log-log scale).

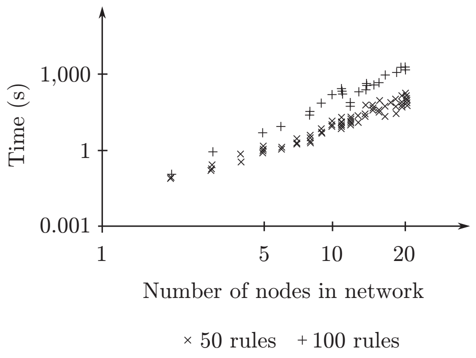


Figure 5. Verification time for randomly generated topologies, with a fixed number of firewall rules and an increasing number of nodes rules in the network (log-log scale).

We also repeated the experiment, this time fixing the number of firewall rules, and varying the number of network nodes. We generated *random* routing tables for each of the nodes, therefore resulting in arbitrary topologies instead of the fixed hierarchy of the first experiment. The running times are shown in Figure 5. The plot shows verification times varying between 1 second and 16 minutes, depending on the number of nodes and the fixed number of rules used for the experiment.

The results obtained can be used as the basis for the comparison of existing methods. We shall first discard experimental results reported for *intra*-firewall anomalies or for anomalies along a single path in the network [3], [5], [10], [14], which amount to a much simpler problem.

For firewalls arranged as a 3-3-3 hierarchy, [2] reports an analysis time of 180 seconds, and 20 seconds with additional optimizations. These figures however, have been reportedly computed on a 400 MHz Pentium III computer. A conservative approximation based on the runtime ratio of each processor in the SPEC benchmarks,[3] yields a speed-corrected running time of 12 seconds for the basic version, and 1.3 second for the optimized version.

We have seen, however, that those faster running times are achieved at the expense of precision and generality. First, the content of routing tables is not modelled by [2] —thereby leading to potential false positives as described in Section II.

Moreover, the optimized technique only applies when the considered network is a directed tree; for any other network, only the generic technique, which performs a separate analysis on every path, is applicable. For example, the 2-2-2 hierarchy contains 15 nodes; assuming an *arbitrary* topology of those same 15 nodes, the generic method should potentially verify more than $15! \approx 10^{12}$ paths. At this rate, the tool should be more than one hundred million times faster on every path to achieve the same running time as the model checking approach.

Another basis for comparison is [12], as part of the routing information is taken into account in the model of the network. For a random network of 3,000 rules, checking cyclicity is reported to take 7 seconds, and reachability 450 seconds. It shall be noted, however, that the properties checked are simpler than firewall anomalies, and only model the path of single packets. We have discussed earlier how this approach is not appropriate for the detection of anomalies, which require the propagation of rules' intervals and their comparison to other rules later on.

VI. Conclusion

Early experimental results on the use of model checking for the detection of patterns in firewall rules indicate that the approach can spot firewall anomalies in reasonable time for networks of realistic sizes. However, two factors unrelated to performance make the approach even more appealing. First, existing anomalies studied in the literature are specific formulæ in Linear Temporal Logic; as such, any other anomaly expressible as an LTL formula is readily handled by our framework. Moreover, our approach takes into account routing tables and possible paths between firewalls, and can analyze arbitrary network topologies without the need for explicit enumeration of every possible trace.

Future improvements include the optimization of the representation of intervals in the model, and the use of visibility logic, rather than Linear Temporal Logic, to express the rules. The freeze variables of the Kripke structure are actually used to simulate the modal operators available in Visibility Logic. Designing a model checking algorithm where VL constraints would be checked natively might lead to better performances than the LTL design proposed here.

References

[1] E. S. Al-Shaer and H. H. Hamed, "Discovery of policy anomalies in distributed firewalls," in *INFOCOM*, 2004.

[2] E. Al-Shaer, H. Hamed, R. Boutaba, and M. Hasan, "Conflict classification and analysis of distributed firewall policies," *IEEE Journal on Selected Areas in Communications*, vol. 23, no. 10, pp. 2069–2084, 2005.

[3] E. S. Al-Shaer and H. H. Hamed, "Firewall policy advisor for anomaly discovery and rule editing," in *IM*, ser. IFIP Conference Proceedings, G. S. Goldszmidt and J. Schönwälder, Eds., vol. 246. Kluwer, 2003, pp. 17–30.

[4] A. Wool, "Firewall configuration errors revisited," Tel Aviv University, Tech. Rep. arXiv:0911.1240v1, 2009.

[5] L. Yuan, J. Mai, Z. Su, H. Chen, C.-N. Chuah, and P. Mohapatra, "FIREMAN: A toolkit for firewall modeling and analysis," in *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2006, pp. 199–213.

[6] Y. Yin, R. Bhuvaneswaran, Y. Katayama, and N. Takahashi, "Analysis methods of firewall policies by using spatial relationships between filters," in *ICSCN*, 2007, pp. 348–354.

[7] A. Liu and M. Gouda, "Complete redundancy detection in firewalls," in *Proceedings of the 19th Annual IFIP WG 11.3 Working Conference on Data and Applications Security*, ser. Lecture Notes in Computer Science, vol. 3654. Springer, 2005.

[8] M. G. Gouda and A. X. Liu, "Structured firewall design," *Computer Networks*, vol. 51, no. 4, pp. 1106–1120, 2007.

[9] T. Nelson, C. Barratt, D. J. Dougherty, K. Fisler, and S. Krishnamurthi, "The Margrave tool for firewall analysis," in *LISA*, R. van Drunen, Ed. USENIX Association, 2010, pp. 1–18.

[10] B. Khorchani, S. Hallé, and R. Villemaire, "Firewall anomaly detection with a model checker for visibility logic," in *NOMS*. IEEE, 2012, pp. 466–469.

[11] A. El-Atawy and T. Samak, "End-to-end verification of QoS policies," in *NOMS*. IEEE, 2012, pp. 426–434.

[12] A. Jeffrey and T. Samak, "Model checking firewall policy configurations," in *POLICY*. IEEE Computer Society, 2009, pp. 60–67.

[13] R. M. Oliveira, S. Lee, and H. S. Kim, "Automatic detection of firewall misconfigurations using firewall and network routing policies," in *IEEE DSN Workshop on Proactive Failure Avoidance, Recovery, and Maintenance (PFARM)*, 2009.

[14] R. Chaure, "An implementation of anomaly detection mechanism for centralized and distributed firewalls," *International Journal of Computer Applications*, vol. 7, no. 4, pp. 5–8, September 2010, published By Foundation of Computer Science.

[15] A. Pnueli, "The temporal logic of programs," in *FOCS*. IEEE, 1977, pp. 46–57.

[16] A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella, "NuSMV 2: An opensource tool for symbolic model checking," in *CAV*, ser. Lecture Notes in Computer Science, E. Brinksma and K. G. Larsen, Eds., vol. 2404. Springer, 2002, pp. 359–364.

[17] *2012 IEEE Network Operations and Management Symposium, Maui, HI, USA, April 16-20, 2012*. IEEE, 2012.

[3]http://www.spec.org/. The computed ratio is 13.9 for a single core (our tool does not take advantage of multiple cores and runs on a single processor) and has been rounded to 15.