Linear-Time Verification of Firewalls

H. B. Acharya
Department of Computer Science
University of Texas
Austin, TX, USA
Email: acharya@cs.utexas.edu

Mohamed G. Gouda
Department of Computer Science
University of Texas
Austin, TX, USA
Email: gouda@cs.utexas.edu

February 12, 2009

Abstract

A firewall is a packet filter placed at the entrance of a private network. Its function is to examine each packet that is incoming into the private network and decide, based on the specified rules of the firewall, whether to accept the packet and allow it to proceed, or to discard the packet. A property of a firewall is a specified set of packets that is supposed to be accepted or discarded by the firewall. In this paper, we present the first linear time algorithm to verify whether a given firewall satisfies a given property. The time complexity of our algorithm is $O(nd)$, where n is the number of specified rules in the given firewall and d is the number of fields that are checked by the firewall or the property. Our verification algorithm consists of two passes: a deterministic pass followed by a probabilistic pass. In most cases, the algorithm correctly determines whether the given firewall satisfies the given property. But in some rare cases, the algorithm may erroneously determine that the firewall satisfies the property (where in fact it does not). Using a combination of analysis and extensive simulation, we show that the probability of an erroneous determination by the algorithm is of the order of 3×10^{-5}.

1 Introduction

A firewall is a security system that acts as a protective boundary between a private computer network and the "outside world", usually the rest of the Internet. Firewalls help make private computer networks invisible to online attackers and protect them against malicious software - viruses, worms, and Trojan horses.

A firewall is a packet filter that is placed at a point where a private computer network connects to the Internet. The firewall intercepts each packet that is exchanged between the network and the Internet, examines the headers of the packet, and based on a sequence of configuration rules in the firewall, takes action on the packet: it may allow the packet to proceed, drop the packet and notify its sender, or drop it silently. Then, the firewall may or may not log the action. In this paper, we take the simplified view that a firewall simply either accepts or discards each packet.

The decision that a firewall implements when it receives a packet depends on the values in the headers of the packet. A firewall is a sequence of rules; each rule matches certain packets, with specific header values, and specifies the action (accept or discard) to be taken on the packet. In case there are multiple rules matching a packet, the first rule (according to the order of the rule sequence) to match the packet takes precedence. A firewall property is specified by a set of packets and a decision: accept or discard. A firewall is said to satisfy a property iff the action taken by the firewall on each packet specified by the property matches the decision specified by the property.

A firewall is an important line of security; for it to be effective, it needs to satisfy the properties the designers intend the firewall to satisfy. However, a firewall may be required to satisfy very many properties

- $10,000$ or more for an industrial firewall - and a slow check may need the whole network to remain offline until it is completed, resulting in very expensive downtime. If the checks are too slow, an impatient IT department may simply go online with an improperly configured firewall, and become vulnerable to attacks. The process, of checking whether a firewall satisfies the properties of interest, needs to be as convenient and fast as possible.

In this paper, we present a firewall verification algorithm. This algorithm takes a firewall and a property, and determines whether or not the firewall satisfies the property. The time complexity of our algorithm is $O(nd)$, where n is the number of rules in the given firewall and d the number of fields checked by the firewall. In practice, d has a value between 5 and 7, so the time complexity of our algorithm is linear in the number of rules in the firewall.

Prior algorithms for verifying that a given firewall satisfies a given property (or a query), [1] and [2], have a time complexity of $O(n^d)$ [3]. Thus, our firewall verification algorithm, in the current paper, is the first one to have a linear time complexity. As a large industrial firewall may easily have more than $n = 2,000$ rules, it becomes significant that our verification algorithm remains tractable in the worst case for large values of n. (There are other approaches to ensure the correctness of a firewall besides firewall verification. These approaches include firewall design, firewall analysis and firewall testing. We review these approaches in Section 10.)

Our verification algorithm for firewalls takes as input a firewall F, represented as a sequence of rules and a property R, and decides whether F satisfies R. The verification algorithm is based on a novel graphical representation of the given firewall F, called the rule diagram. The algorithm consists of two passes: a deterministic pass, possibly followed by a probabilistic pass. The time complexity of each pass is $O(nd)$. The deterministic pass takes as input the given firewall F and property R, and produces one of three outcomes:

(a) F satisfies R.

(b) F does not satisfy R.

(c) No conclusion can be reached.

We show that, for several special classes of F and R, the deterministic pass produces either outcome (a) or outcome (b). In those cases where the deterministic pass produces outcome (c), the

(c_1) F satisfies R with a high probability.

(c_2) F does not satisfy R.

Thus, if the probabilistic pass produces the outcome (c_1), it is possible that the decision (that F satisfies R) is erroneous. We show that the probability of an erroneous outcome when the probabilistic pass produces the outcome (c_1) is very small, less than .006.

We have implemented our algorithm and applied it extensively to verify whether a randomly-generated firewall satisfies a given property. Our experiments, using $(80,000)$ firewall-property pairs, have showed that our verification algorithm produces outcome (c_1) very rarely, about 0.5% of the time.

Therefore, we conclude that the probability of our verification algorithm producing an erroneous result is less than $.005 \times .006 = 3 \times 10^{-5}$.

2 Firewalls, Packets and Properties

In this section, we formally define the three central concepts in this paper, firewalls, packets, and properties.

A *field* is a variable, whose value is taken from an interval of non-negative integers. Examples of fields are source IP address, destination IP address, transport protocol, source port number, and destination port number. The domain of values of the source IP address field, for example, is the interval $[0, 2^{32} - 1]$.

In this paper, we consider d fields, denoted $f_1, f_2, \ldots, f_d$. The domain of values of any field f_j is denoted by $D(f_j)$. Note that $D(f_j)$ is an interval of non-negative integers.

A *firewall* F is a sequence of n rules, where the i-th rule is of the form:

$$f_1 \in S_{i,1} \wedge f_2 \in S_{i,2} \wedge \ldots \wedge f_d \in S_{i,d} \rightarrow < decision_i >$$

Each $S_{i,j}$ is a non-empty interval of non-negative integers taken from the domain $D(f_j)$, and $< decision_i >$ is either *accept* or *discard*. The last rule in a firewall is of the form

$$f_1 \in S_{n,1} \wedge f_2 \in S_{n,2} \wedge \ldots \wedge f_d \in S_{n,d} \rightarrow < decision_n >$$

Each $S_{n,j}$ is the domain $D(f_j)$ of field f_j.

A *packet* is a d-tuple $(p_1, p_2, \ldots, p_d)$, where each p_j is an element from the domain $D(f_j)$ of field f_j .

A packet $(p_1, p_2, \ldots, p_d)$ is said to *match* a rule

$$f_1 \in S_{i,1} \wedge f_2 \in S_{i,2} \wedge \ldots \wedge f_d \in S_{i,d} \rightarrow < decision_i >$$

in a firewall F iff the predicate $p_1 \in S_{i,1} \wedge p_2 \in S_{i,2} \wedge \ldots \wedge p_d \in S_{i,d}$ holds.

A firewall F is said to *accept*, or *discard* respectively, a packet iff F has a rule that satisfies the following three conditions:

1. The packet matches the rule.
2. The packet does not match any earlier rule in F .
3. The decision of the rule is accept, or discard, respectively.

A property is of the form:

$$f_1 \in T_1 \wedge f_2 \in T_2 \wedge \ldots \wedge f_d \in T_d \rightarrow < decision >$$

Each T_j is a non-empty interval of non-negative integers taken from the domain $D(f_j)$ of field f_j , and $< decision >$ is either *accept* or *discard*.

A packet $(p_1, p_2, \ldots, p_d)$ is said to match a property

$$f_1 \in T_1 \wedge f_2 \in T_2 \wedge \ldots \wedge f_d \in T_d \rightarrow < decision >$$

iff the predicate $p_1 \in T_1 \wedge p_2 \in T_2 \wedge \ldots \wedge p_d \in T_d$ holds.

A firewall F is said to satisfy a property

$$f_1 \in T_1 \wedge f_2 \in T_2 \wedge \ldots \wedge f_d \in T_d \rightarrow < decision >$$

iff one of the following two conditions holds:

1. $< decision >= accept$, and every packet that matches the property is accepted by F .
2. $< decision >= discard$, and every packet that matches the property is discarded by F .

A firewall example F_1 , with three rules, is given below:

$$f_1 \in [4, 10] \wedge f_2 \in [5, 7] \wedge f_3 \in [4, 10] \rightarrow \textit{accept}$$
$$f_1 \in [1, 6] \wedge f_2 \in [5, 7] \wedge f_3 \in [5, 7] \rightarrow \textit{accept}$$
$$f_1 \in [0, 10] \wedge f_2 \in [0, 10] \wedge f_3 \in [0, 10] \rightarrow \textit{discard}$$

Each rule checks the three fields f_1 , f_2 , and f_3 . The domain $D(f_j)$ of each field f_j is the interval $[0, 10]$. A packet for firewall F_1 is the 3-tuple $(1, 7, 5)$, where $f_1 = 1$, $f_2 = 7$, and $f_3 = 5$. This packet does not match the first rule in F_1 , but matches the second rule in F_1, and because the decision of the second rule is *accept*, firewall F_1 is said to accept the packet.

Three properties of firewall F_1, named R_1 through R_3, are as follows:

$$R_1 : f_1 \in [3, 9] \wedge f_2 \in [8, 10] \wedge f_3 \in [6, 9] \rightarrow \textit{discard}$$
$$R_2 : f_1 \in [2, 3] \wedge f_2 \in [1, 9] \wedge f_3 \in [6, 9] \rightarrow \textit{discard}$$
$$R_3 : f_1 \in [1, 9] \wedge f_2 \in [4, 9] \wedge f_3 \in [3, 7] \rightarrow \textit{accept}$$

Note that packet $(1, 7, 5)$ matches R_3 , but does not match R_1 or R_2 .

3 Rule Diagrams of Firewalls

The rule diagram of a firewall F is a directed acyclic connected graph that represents the rules of F and has the structure shown in Figure 1. From this figure, the rule diagram of F satisfies the following conditions:

1. The rule diagram has no directed cycles, and no parallel edges.

2. The rule diagram has one node, the root, with no incoming edges.

3. The rule diagram has n terminal nodes, with no outgoing edges, where n is the number of rules in firewall F. The i-th terminal node is labeled with $decision_i$, the decision of the i-th rule in firewall F .

4. Each non-terminal node in the rule diagram is labeled with a field f_j .

5. Each edge e is labeled with either one interval (possibly empty) of non-negative integers, or two intervals of non-negative integers.

6. The rule diagram has n horizontal chains. The i-th horizontal chain corresponds to the i-th rule in firewall F . For example, the top horizontal chain corresponds to the first rule in firewall F :

   $$f_1 \in S_{1,1} \wedge f_2 \in S_{1,2} \wedge .. \wedge f_d \in S_{1,d} \rightarrow < decision_1 >$$

   Similarly, the bottom horizontal chain corresponds to the last rule in firewall F :

   $$f_1 \in S_{n,1} \wedge f_2 \in S_{n,2} \wedge .. \wedge f_d \in S_{n,d} \rightarrow < decision_n >$$

   If $i < i'$, then we say that the chain that corresponds to rule i in F *precedes* the chain that corresponds to rule i' in F .

7. Each non-terminal node labeled f_j , in a horizontal chain other than the bottom horizontal chain, has two outgoing edges: a horizontal edge labeled $S_{i,j}$, which is specified in the i-th rule in firewall F , and a vertical or inclined edge labeled $C_{i,j}$, which is specified as follows: $C_{i,j} = D(f_j) - S_{i,j}$

   Note that both $D(f_j)$ and $S_{i,j}$ are intervals of non-negative integers. Thus, $C_{i,j}$ is either an interval of non-negative integers, or the union of two intervals of non-negative integers. Each non-terminal node labeled f_j in the bottom horizontal chain has one outgoing edge labeled $S_{n,j}$, which is specified in the n-th rule in firewall F .

Figure 1: Structure of the rule diagram of a firewall F

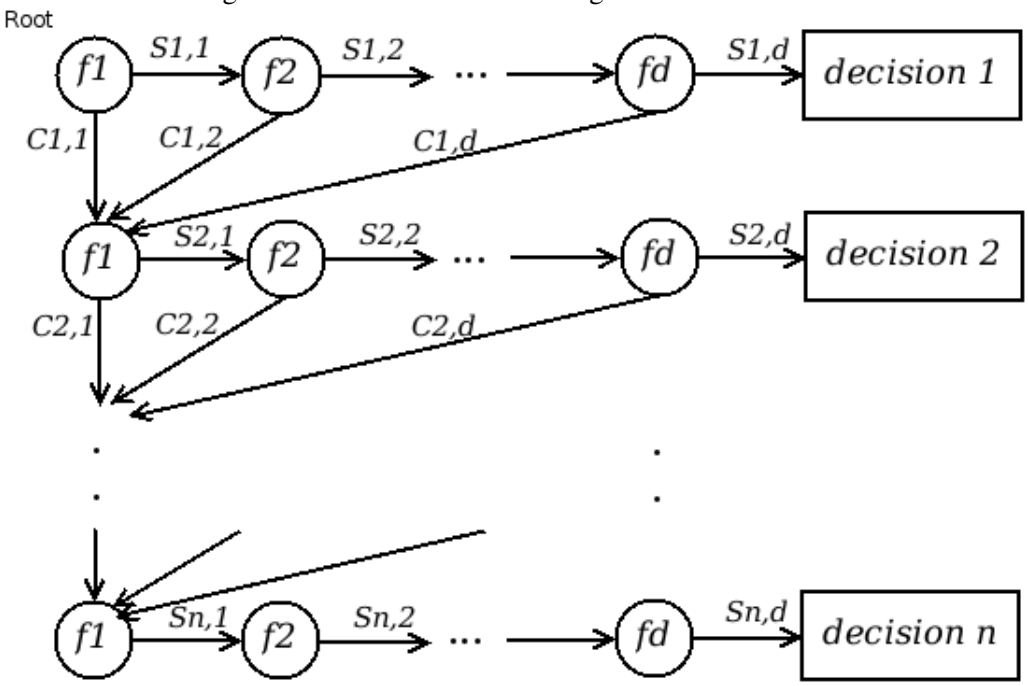
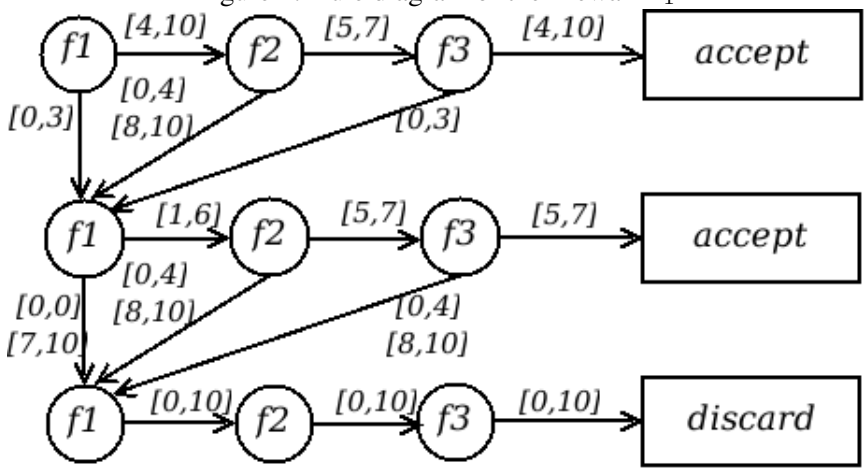


Figure 2: Rule diagram of the firewall F_1



As an example, the rule diagram of firewall F_1 in Section 2 is given in Figure 2.

Theorem 1. *The rule diagram of a firewall F , with n rules and d fields, has $n(d+1)$ nodes and $(2n-1)d$ edges.*

Proof. Each horizontal chain in the rule diagram has $d+1$ nodes - one for each field, and one for the decision. There are n chains - one for each rule. The total number of nodes is $n(d+1)$.

Similarly, each horizontal chain has d horizontal edges connecting the nodes in the chain. These constitute nd edges. Each chain, except the last, also has d edges which connect the nodes marked with the names of fields $(f_1, f_2, \ldots f_n)$ to the first node (f_1) of the next chain. These are another $(n-1)d$ edges. The total number of edges is $(2n-1)d$. □

In the next four sections, we present an algorithm that uses the rule diagram of a given firewall F to verify whether F satisfies a given property R. This algorithm has two passes.

1. Deterministic Pass: This pass produces any one of the following three outcomes:
   (a) F satisfies R.
   (b) F does not satisfy R.
   (c) No conclusion can be reached.
2. Probabilistic Pass: This pass is performed only when the deterministic pass has outcome (c). When the probabilistic pass is performed, it produces any one of the following two outcomes:
   (c_1) F satisfies R with a high probability.
   (c_2) F does not satisfy R.

The deterministic pass of our firewall verification algorithm is discussed in Sections 4 and 5, whereas the probabilistic pass is discussed in Sections 6 and 7.

4 Deterministic Pass

The deterministic pass (of our firewall verification algorithm) attempts to deterministically answer the question of whether a given firewall F satisfies a given property R. This pass is specified in Algorithm 1.

Applying the deterministic pass algorithm to firewall F_1 in Section 2 and to each of the properties R_1, R_2 and R_3 specified in Section 2, we conclude that F_1 satisfies R_1 , F_1 does not satisfy R_2 , and no conclusion can be reached on whether F_1 satisfies R_3 .

Theorem 2. *The time complexity of the deterministic pass algorithm, when applied to a firewall F and a property R, is $O(nd)$, where n is the number of rules in F and d is the number of fields in a rule (as well as in R).*

Proof. The algorithm computes the intersection of the labels $X_{i,j}$ and T_j (where $X_{i,j}$ is either $S_{i,j}$ or $C_{i,j}$) for every rule i and every field j. Each label consists of either one or two integer intervals. Computing the intersection of two integer intervals takes constant time. Hence in the worst case, as there are n rules and d fields in a rule, the time consumed for computation is $4nd$, which is $O(nd)$. □

Algorithm 1 Deterministic Pass

Input: A firewall F and a property R of F .

Output: Any one of the following outcomes:

(a) F satisfies R.

(b) F does not satisfy R.

(c) No conclusion can be reached.

1: Construct the rule diagram of firewall F , as shown in Figure 1.

2: Let the predicate of property R be:

$$f_1 \in T_1 \wedge f_2 \in T_2 \wedge .. \wedge f_d \in T_d$$

For every edge e, outgoing from a node labeled f_j , if the label of edge e is the set $X_{i,j}$ (where $X_{i,j}$ is either $S_{i,j}$ or $C_{i,j}$), and if the intersection of $X_{i,j}$ and T_j is empty, then remove edge e from the rule diagram.

3: If the earliest terminal node, reachable from the root, is labeled with a decision that differs from the decision of property R, then report that F does not satisfy R.

4: If all terminal nodes, reachable from the root, are labeled with the same decision as that of property R, then report that F satisfies R.

5: In all other cases, report that no conclusion can be reached.

5 Effectiveness of the Deterministic Pass

In this section, we show why the deterministic pass is not sufficient to reach a decision whether a firewall satisfies a property or not (decisions a and b respectively) in all cases. We also report three significant cases where the deterministic pass is sufficient to determine whether or not a firewall F satisfies a property R. These three cases are as follows.

1. R is a singleton property.
2. F is a conflict-free firewall.
3. F is an accept-discard firewall, and R has the decision discard.

We describe the three concepts of singleton properties, conflict-free firewalls, and accept-discard firewalls, in order after reporting the cases where a deterministic pass does not produce outcome a or b.

Consider property R to have the form

$$f_1 \in T_1 \wedge f_2 \in T_2 \wedge \ldots \wedge f_d \in T_d \rightarrow < decision >$$

and rule R_i to have the form

$$f_1 \in S_{i,1} \wedge f_2 \in S_{i,2} \wedge \ldots \wedge f_d \in S_{i,d} \rightarrow < decision_i >$$

Then a packet

$$f_1 = t_1 \wedge f_2 = t_2 \wedge \ldots \wedge f_d = t_d$$

is said to belong to the predicate of R iff $\wedge_{i=1}^{d} t_i \in T_i$.

We define a rule, whose decision is the same as that of the property R under verification, to be *compliant*, and a rule with the opposite decision to be *non-compliant*. A firewall practises first-match semantics : if a packet matches multiple rules ($R_i, R_{ii} \ldots$), the decision of the first rule it matches is the decision of the firewall. Iff the first rule matched by any packet which belongs to the predicate of property R is non-compliant, the firewall does not satisfy R.

In order to detect the rules which may be matched by packets which belong to the predicate of property R, we check the intersections of the labels $S_{i,j} \cap T_j$ for every field value j. If all the intersections are non-empty, rule R_i matches some packets in the predicate of R.

Further, to detect if a rule matches *every* packet in the predicate of R, we test the intersection of label $C_{i,j} \cap T_j$. If a single intersection is non-empty, there are packets in the predicate of R which rule R_i does not match. This is because $C_{i,j}$ is the complement of $S_{i,j}$: iff there is no value of T_j in common with $C_{i,j}$ then $T_j \subseteq S_{i,j}$. If $T_j \subseteq S_{i,j}$ for every field j, every packet in the predicate of R is matched by rule R_i. This means that the rules following R_i need not be checked to determine whether the given firewall satisfies property R. R_i is said to *completely mask* R.

However, suppose that a compliant rule R_i precedes the non-compliant rule R_{ii}. R_i does not match every packet in the predicate of R, but it does match every packet that R_{ii} matches. R_i is said to *mask* R from R_{ii}. As R_i does not completely mask R, the algorithm continues and detects that R_{ii} can match some packets in the predicate of R. It does not detect that no such packet will be matched by R_{ii} in practice, because R_i will match them first. R_{ii} is said to be a *pseudo-active* rule.

The deterministic pass can conclude that the firewall satisfies property R if all the rules detected to match packets R are compliant. However, the algorithm detects pseudo-active rules as well as the rules which actually match packets belonging to the predicate of R. It may be possible that the only non-compliant rules detected (as matching packets in the predicate of R) are pseudo-active. In such a case, the firewall satisfies property R, but the deterministic pass does not detect this.

The deterministic pass is only allowed to conclude that the firewall does not satisfy property R if the *first* rule R_{first}, that matches one or more packets belonging to the predicate of R, is non-compliant. Note that any packet p which R_{first} can (possibly) match, is guaranteed to be matched by R_{first}. There are no earlier rules matching packets belonging to the predicate of R, hence no earlier rule that can match p (and prevent the firewall from executing the decision of R_{first}). As R_{first} is non-compliant, this causes the firewall to reach the opposite decision from that specified in R in the case of the packet p, a packet that belongs to the predicate of R. This proves that the firewall does not satisfy R.

5.1 Singleton Properties

A *singleton property* is one that is only matched by a single packet, i.e. the property has a predicate of the form

$$f_1 \in T1 \wedge f_2 \in T_2 \wedge \ldots \wedge f_d \in T_d$$

where each interval T_j contains a single non-negative integer t_j. In this case, the predicate can also be expressed as

$$f_1 = t_1 \wedge f_2 = t_2 \wedge \ldots \wedge f_d = t_d$$

Theorem 3. *If the deterministic pass algorithm is applied to a firewall F and a singleton property R, then the algorithm is guaranteed to produce either outcome (a) or outcome (b).*

Proof. There is exactly one packet which belongs to the predicate of a singleton property R . Consequently, there is no way a rule R_i can match some of the packets in the predicate of R, but not others. If a rule R_i masks R from R_{ii}, it completely masks R. There are no pseudo-active rules. Therefore, the deterministic pass can decide with certainty whether R is satisified or not, and produces either outcome (a) or outcome (b). □

This result is the basis upon which the probabilistic pass of our algorithm, presented in Section 6, is founded.

5.2 Conflict-free Firewalls

A firewall F is said to be conflict-free if, for any two rules r_i and r_j in F , at least one of the following conditions holds:

1. No packet matches both r_i and r_j .
2. r_i and r_j have the same decision.

Theorem 4. *If the deterministic pass algorithm is applied to a conflict-free firewall F and a property R, then the algorithm is guaranteed to produce either outcome (a) or outcome (b).*

Proof. If a firewall is conflict-free, any packet p which a rule R_{any} can possibly match, is guaranteed to be matched by R_{any}. Note that no rule R_i preceding R_{any} can match p; hence, no R_i can mask R from R_{any}. Hence there are no pseudo-active rules. If R_{any} is non-compliant, then the firewall is proved to not satisfy the property R. If no R_{any} is non-compliant, then in any case the firewall satisfies property R: it is true for any firewall F that F satisfies R if all the rules detected to match packets belonging to the property of R are compliant.

□

5.3 Accept-discard firewalls

An accept-discard firewall is one in which all the rules with decision *accept* precede all rules with decision *discard*.

Theorem 5. *If the deterministic pass algorithm is applied to an accept-discard firewall F and a property R whose decision is* discard*, then the algorithm is guaranteed to produce either outcome (a) or outcome (b).*

Proof. The property R being verified is a *discard* property; this means that the discard rules in F are compliant, and the accept rules are non-compliant. As no compliant rule precedes a non-compliant rule, the possibility of a compliant R_i masking R from a non-compliant R_{ii} does not arise. Even if a non-compliant rule R_{ii} is pseudo-active, the rule R_i must itself be non-compliant. F cannot satisfy R. Hence, any non-compliant rule detected to match a packet belonging to the predicate of R is sufficient proof that F does not satisfy R. If there are no non-compliant rules thus detected, then F satisfies R anyway: it is true for any firewall F that F satisfies R if all the rules detected to match packets belonging to the property of R are compliant. Thus the deterministic pass always terminates with outcome (a) or outcome (b). □

This result is important because, in many practical cases, a firewall is designed by specifying packets to allow, followed by a default rule to discard all packets. Any such firewall is an accept-discard firewall. (Note that the deterministic pass algorithm is also guaranteed to produce outcome (a) or outcome (b) if it is applied to a *discard-accept* firewall, in which all the rules with decision *discard* precede all rules with decision *accept*.)

6 Probabilistic Pass

In the previous section, we discussed several instances of a firewall F and a property R, where the deterministic pass applied to F and R produces either outcome (a) or outcome (b). The deterministic pass, however, can produce outcome (c) in many instances of F and R. In order to handle these cases, we perform the probabilistic pass, which we detail in this section.

Theorem 6. *Any property R can be decomposed into an equivalent set Q of singleton properties, such that any firewall F satisfies R iff F satisfies each singleton property in Q.*

Proof. (Proof by construction) A property specifies a set of packets, and a decision dec. We take the predicate of property R, and for every packet $p : f_1 = x_1, f_2 = x_2 \ldots f_d = x_d$ that matches this predicate, construct a new singleton rule

$$f_1 = x_1, \ldots f_d = x_d \rightarrow dec$$

The set of these rules is Q. □

We refer to set Q in Theorem 6 as the *footprint* of property R.

As an example, assume that we need to verify whether or not a given firewall F satisfies the following property R:

$$f_1 \in [1, 2] \wedge f_2 \in [5, 5] \wedge f_3 \in [1, 2] \rightarrow discard$$

By Theorem 6, firewall F satisfies R iff F satisfies each of the following singleton properties in the footprint of R:

$$f_1 = 1 \wedge f_2 = 5 \wedge f_3 = 1 \rightarrow discard$$
$$f_1 = 1 \wedge f_2 = 5 \wedge f_3 = 2 \rightarrow discard$$
$$f_1 = 2 \wedge f_2 = 5 \wedge f_3 = 1 \rightarrow discard$$
$$f_1 = 2 \wedge f_2 = 5 \wedge f_3 = 2 \rightarrow discard$$

Now, by Theorem 3, each singleton property in the footprint of R can be verified using the deterministic pass, producing either outcome (a) or outcome (b).

It follows from this example that we can determine whether a firewall F satisfies a property R by decomposing R into the singleton properties of its footprint and verifying whether F satisfies each of these singleton properties.

However, the number of singleton properties in the footprint of a property can be very large. For example, there are 40000 singleton properties in the footprint of the following property:

$$f_1 \in [1, 200] \wedge f_2 \in [5, 5] \wedge f_3 \in [1, 200] \rightarrow discard$$

To get around this problem, we selectively verify some of the properties in the footprint of R, not all of them. In our probabilistic pass algorithm, discussed below, we take some small number, namely 4, of values for each field f_j as the sample set of field f_j. (Note that our choice of the number 4 was simply because in real life, firewalls have 5 to 7 fields; the number of properties tested in the probabilistic pass is 4^d, as we consider 4 values for each of d fields, and we wanted to ensure that at least $1,000$ random properties were tested. $4^5 = 1,024$, so 4 was the smallest choice satisfying our criterion in the worst case.)

The selection of values is done so as to divide the range of f_j in R into $4 - 1 = 3$ slices, as nearly equal as possible. In the above example, the range of both f_1 and f_3 is $[1, 200]$. We begin by selecting the two "corner" values 1 and 200. The other two values that we select are

$$1 + \text{ floor } (\frac{(200 - 1)}{3}) = 67 \quad (1)$$

$$1 + \text{ floor } (2 \times \frac{(200 - 1)}{3}) = 133 \quad (2)$$

Then the sample set for fields f_1 and f_3 is $\{1, 67, 133, 200\}$.

Algorithm 2 Probabilistic Pass

Input: A firewall F and a property R of F .

Output: Any one of the following outcomes:

(c_1) F satisfies R with a high probability.(See Section 7 below.)

(c_2) F does not satisfy R.

1: For each field f_j do a) Let property R contain: $f_j \in [x, x']$ b) Define X_j to be the set $\{x_0, x_1, x_2, x_3\}$ where, for each i in the range $0..3$,

$$x_i = x + floor(i\frac{(x' - x)}{3})$$

2: Define p to be the Cartesian product of the X_j sets :

$$p = X_1 \times \ldots \times X_d$$

(Each element in p is a packet of firewall F .)

3: For each packet $(p_1, p_2, \ldots, p_d)$ in p, compute a corresponding singleton property as follows:

$$f_1 = p_1 \wedge \ldots \wedge fd = pd \rightarrow < decision >$$

where $< decision >$ is the decision of property R. Let q be the set of the singleton properties thus computed. Note that set q is a subset (usually proper) of Q, the footprint of property R.

4: Use the deterministic pass algorithm to verify whether or not F satisfies every singleton property in set q.

5: If F satisfies every singleton property in set q, then report that F satisfies property R with high probability. Otherwise, report that F does not satisfy property R.

Finally, we verify only the properties, in the footprint, such that the values of their fields are drawn from the sample sets of the respective fields. For the above property, we now need to verify only 16 singleton properties instead of 40000. (Note that this pattern of sampling provides random sampling with the guarantee that no packet is sampled more than once.)

In the probabilistic pass, we need to verify a maximum of 4^d singleton properties. By Theorem 2, verifying each of these takes $O(nd)$ time. We can now state the following theorem:

Theorem 7. *The time complexity of the probabilistic pass algorithm, when applied to a firewall F and a property R, is $O(4^d nd)$, where n is the number of rules in F and d is the number of fields checked in F (as well as in R).*

Proof. The proof is provided by the derivation above. □

Note that in practice the value of d is around 5. Thus $4^d = 1024$, a constant factor, and the complexity of the probabilistic pass is $O(nd)$.

7 Effectiveness of the Probabilistic Pass

In this section, we show that when the probabilistic pass reports that the given firewall F satisfies the given property R, then the probability that F actually satisfies R is high.

Specifically, let A be the event that F satisfies R, and let B be the event that the probabilistic pass reports that F satisfies R. Thus, we need to prove that the conditional probability $P(A|B)$ is high. A derivation of the conditional probability $P(A|B)$ follows:

$$\begin{aligned}
P(A|B) &= \frac{P(AB)}{P(B)} \\
&= \frac{P(AB)}{P(AB) + P(\bar{A}B)} \\
&= \frac{P(A)}{P(A) + P(\bar{A}B)} && \text{, as } P(AB) = P(A) \\
&= \frac{P(A)}{P(A) + P(\bar{A})P(B|\bar{A})} \\
&= \frac{P(A)}{P(A) + (1 - P(A))P(B|\bar{A})}
\end{aligned}$$

Thus, to compute $P(A|B)$, we need to estimate the two quantities $P(A)$ and $P(B|\bar{A})$.

First, we assume that $P(A)$ has the unbiased value of $\frac{1}{2}$. (Note that there are two possibilities - either the given firewall satisfies the given property, or not. Thus, this assumpion is based on the view that these two possibilities are equally likely.) In practice, $P(A)$ is close to 1, but we assume $\frac{1}{2}$ and show that the algorithm is highly reliable even under this severe assumption.

Second, we estimate that the value of $P(B|\bar{A})$ is at most E^{4^d}, where E is the fraction of singleton properties in the footprint of R that are satisfied by F ,and 4^d is the number of singleton properties that are selected from the footprint of R and shown to be satisfied by F in the probabilistic pass.

It follows that

$$\begin{aligned}
P(A|B) &\geq \frac{\frac{1}{2}}{\frac{1}{2} + (1 - \frac{1}{2})E^{4^d}} \\
&= \frac{1}{1 + E^{4^d}}
\end{aligned}$$

This derivation proves the following theorem:

Theorem 8. *If the probabilistic pass reports that F satisfies R, the probability that F satisfies R is at least $\frac{1}{1+E^{4^d}}$.*

Proof. The proof is provided by the derivation above. □

In practice, the number of fields checked in a firewall is usually between 5 and 7. Thus, for E as high as 99.5%, and the lowest (worst) value of $d = 5$, the probability $P(A|B)$ is at least 99.41%. (In the more practically reasonable case where $P(A) = 0.99$, $P(A|B)$ rises to over 99.99%. Even $E = 99.95\%$ yields $P(A|B)$ of at least 99.40%.)

8 Experimental Results

In the previous sections, we have presented a linear time firewall verification algorithm, whose time complexity is $O(nd)$, where n is the number of rules in the given firewall and d is the number of fields checked in the firewall (or the given property). The verification algorithm consists of a deterministic pass followed by a probabilistic pass (which is run only on the cases where the deterministic pass fails to decide whether or not the given firewall satisfies the given property). Still, three important questions remain to be answered. They are:

1. What is the speed of a working implementation of our verification algorithm?
2. The deterministic pass runs 1000 times faster than the probabilistic pass. What percentage of properties is decided by the deterministic pass alone?
3. There is a probability of error in the case where the probabilistic pass produces outcome (c_1). How frequent are these cases?

8.1 Experiment 1

In our first experiment, we generated firewalls with n rules each, where n is varied from 100 to 2000 in steps of 100. For each value of n, we generated 100 random firewalls. Each generated firewall checks five fields, and the domain of values for each field is the integer interval $[0, 2^{16} - 1]$. For each generated firewall, we used the first 10 rules in the firewall as properties, and applied our deterministic and probabilistic passes in order to verify whether the generated firewall satisfies each of these 10 properties.

The results of the first experiment are summarized in Table 1. Note that each line in Table 1 represents the results of 1000 applications of our deterministic and probabilistic passes to verify whether a randomly generated firewall satisfies a property (which happens to be one of the first 10 rules in the generated firewall). For instance, the first line in Table 1 indicates that out of 1000 applications,

- 509 produced outcome (a) of the deterministic pass
- 381 produced outcome (b) of the deterministic pass
- 3 produced outcome (c_1) of the probabilistic pass
- 107 produced outcome (c_2) of the probabilistic pass

The average execution time for each application is 0.0 seconds.

# Rules	Verification Results				Time (sec)
	a(%)	b(%)	c_1 (%)	c_2 (%)	
100	50.9	38.1	0.3	10.7	0.0
200	54.7	34.8	0.6	9.9	0.1
300	53.0	35.9	0.4	10.7	0.1
400	52.0	37.7	0.5	9.8	0.1
500	53.7	35.6	0.6	10.1	0.2
600	54.6	35.1	0.5	9.8	0.2
700	51.0	36.6	0.3	12.1	0.3
800	52.7	34.7	0.7	11.9	0.3
900	53.1	35.2	0.9	10.8	0.4
1000	55.8	34.5	0.4	9.3	0.3
1100	54.6	34.2	0.8	10.4	0.3
1200	52.1	35.8	0.4	11.7	0.4
1300	50.7	38.4	0.2	10.7	0.4
1400	54.2	34.2	0.3	11.3	0.4
1500	54.0	35.0	0.2	10.8	0.4
1600	52.7	36.6	0.9	9.8	0.5
1700	53.4	37.5	1.0	8.1	0.6
1800	54.3	35.3	0.3	10.1	0.5
1900	55.5	34.1	0.2	10.2	0.6
2000	54.2	35.4	0.5	9.9	0.6

Table 1: Results where the first 10 rules in the firewall are used as properties

8.2 Experiment 2

The second experiment is similar to the first experiment, except that for each generated firewall, we used the last 10 rules in the firewall as properties, and applied our deterministic and probabilistic passes in order to verify whether the generated firewall satisfies each of these 10 properties.

The results of the second experiment are summarized in Table 2. Again each line in Table 2 represents the results of 1000 applications of our deterministic and probabilistic passes to verify whether a randomly generated firewall satisfies a property (which happens to be one of the last 10 rules in the generated firewall). For instance, the first line in Table 2 indicates that out of 1000 applications,

- 2 produced outcome (a) of the deterministic pass
- 552 produced outcome (b) of the deterministic pass
- 4 produced outcome (c_1) of the probabilistic pass
- 442 produced outcome (c_2) of the probabilistic pass

The average execution time for each application is 0.1 seconds.

8.3 Experiments 3 and 4

In Experiment 3, as in Experiment 1, the properties we verified were obtained from the first 10 rules. However, this time the rules were *flipped* before being used as properties; the predicates of the properties were left unaltered, but the decision was changed to *accept* if the original decision of the rule was *discard*, and vice versa.

# Rules	Verification Results				Time (sec)
	a(%)	b(%)	c_1 (%)	c_2 (%)	
100	0.2	55.2	0.4	44.2	0.1
200	0.6	56.4	0.0	43.0	0.1
300	0.4	59.7	0.2	39.7	0.2
400	0.1	60.6	0.3	39.0	0.2
500	0.3	56.1	0.3	43.3	0.3
600	1.1	54.7	0.1	44.1	0.4
700	0.3	59.7	0.1	39.9	0.4
800	0.3	55.2	0.5	44.0	0.6
900	0.2	58.7	0.2	40.9	0.6
1000	0.4	57.7	0.6	41.3	0.7
1100	0.3	54.2	0.3	45.2	0.8
1200	0.6	55.6	0.1	43.7	0.8
1300	0.5	57.9	0.3	41.3	0.7
1400	0.4	57.8	0.4	41.4	0.9
1500	0.2	55.5	0.2	44.1	0.9
1600	0.4	55.2	0.1	44.3	1.0
1700	0.7	57.5	0.4	41.4	1.2
1800	0.4	57.3	0.1	42.2	1.1
1900	0.8	55.9	0.4	42.9	1.2
2000	0.6	57.6	0.4	41.4	1.3

Table 2: Results where the last 10 rules in the firewall are used as properties

Expt #	Verification results			
	a(%)	b(%)	c_1 (%)	c_2 (%)
1	54.4	36.1	0.4	10.1
2	0.5	54.5	0.3	44.2
3	0.1	76.1	0.0	23.8
4	0.3	67.5	0.1	32.1

Table 3: Summary of results

Similarly, Experiment 4 used the last 10 rules, like Experiment 2, but this time the rules were flipped. All other details (number of firewalls, number of fields, number of properties) were identical in all four experiments.

The results from our algorithm are stable with regard to the length of the firewalls considered, after $80,000$ trials, and are summarized in Table 3. From this table, we can answer the three questions which we posed at the beginning of this section.

1. The execution time of our working implementation, to verify that a given firewall satisfies a given property, is about 0.5 seconds.

2. A majority of the properties (between 55% and 80%) are verified using the deterministic pass only.

3. The percentage of times that our verification algorithm produces outcome (c_1) is less than 0.5%.

From Table 3, and from our conclusion of the previous section that the probability of error when our verification algorithm produces outcome (c_1) is less than .6%, we conclude that the probability of error our verification algorithm is less than $.005 \times .006 = 3 \times 10^{-5}$.

Besides verifying the efficiency of the algorithm, our experiments produced an unexpected useful result. We observe that, unless a property is deterministically proved to be satisfied, the chance of its being satisfied (and this being found in the probabilistic pass) is extremely low. As the only case where the algorithm can make an error is for properties that satisfy these conditions, the chance of an erroneous result can be seen to be vanishingly small. We propose the heuristic of declaring that, if the firewall is not shown to satisfy a given property in the deterministic pass, it does not satisfy the property. Of course, the probabilistic pass can be employed where better accuracy is required.

9 Verification of general properties

So far, we have presented algorithms to verify whether a given firewall F satisfies a given property R , provided that R has the special form:

$$f_1 \in T_1 \wedge .. \wedge f_d \in T_d \rightarrow < decision >$$

In this form, the predicate of property R is a conjunction of d terms, each term is of the form ($f_j \in T_j$), and each T_j is an interval taken from the domain of field f_j . In this section, we argue that our algorithms can still be applied to verify whether F satisfies R even if the predicate of R is incomplete or has disjunction and negation operators. For convenience, we carry our argument using the firewall example F_1 in Section 2. Recall that firewall F_1 has only three fields f_1, f_2, and f_3 , and that the domain of each one of these three fields is the integer interval $[0, 10]$.

9.1 Incomplete Predicates

Assume that we need to verify whether firewall F_1 satisfies the following property R, whose predicate is incomplete:

$$f_2 \in [5, 8] \rightarrow < decision >$$

In this case, we can use our algorithms to verify whether F_1 satisfies the following equivalent property R' , whose predicate is complete:

$$f_1 \in [0, 10] \wedge f_2 \in [5, 8] \wedge f_3 \in [0, 10] \rightarrow < decision >$$

9.2 Disjunctive Predicates

Assume that we need to verify whether firewall F_1 satisfies the following property R, whose predicate has a disjunction:

$$f_1 \in [2,7] \vee f_2 \in [5,8] \rightarrow < decision >$$

In this case, we can use our algorithms to verify whether F_1 satisfies the following two properties R' and R'', whose predicates have no disjunctions:

$$f_1 \in [2,7] \wedge f_2 \in [0,10] \wedge f_3 \in [0,10] \rightarrow < decision >$$
$$f_1 \in [0,10] \wedge f_2 \in [5,8] \wedge f_3 \in [0,10] \rightarrow < decision >$$

If it is concluded that F_1 satisfies both R' and R'' , we conclude that F_1 satisfies the original property R. Otherwise, we conclude that F_1 does not satisfy R.

9.3 Negated Predicates

Assume that we need to verify whether firewall F_1 satisfies the following property R, whose predicate has a negation:

$$\neg f_2 \in [5,8] \rightarrow < decision >$$

In this case, we can use our algorithms to verify whether F_1 satisfies the following two properties R' and R'', whose predicates have no negations:

$$f_1 \in [0,10] \wedge f_2 \in [0,4] \wedge f_3 \in [0,10] \rightarrow < decision >$$
$$f_1 \in [0,10] \wedge f_2 \in [9,10] \wedge f_3 \in [0,10] \rightarrow < decision >$$

If it is concluded that F_1 satisfies both R' and R'' , we conclude that F_1 satisfies the original property R. Otherwise, we conclude that F_1 does not satisfy R.

9.4 General Predicates

Assume that we need to verify whether firewall F_1 satisfies the following property R, whose predicate has conjunction, disjunction and negation:

$$f_1 \in [2,7] \wedge \neg(f_2 \in [5,8] \vee \neg f_3 \in [3,6]) \rightarrow < decision >$$

In this case, we can use our algorithms to verify whether F_1 satisfies the following two properties R' and R'', whose predicates have conjunction only:

$$f_1 \in [2,7] \wedge f_2 \in [0,4] \wedge f_3 \in [3,6] \rightarrow < decision >$$
$$f_1 \in [2,7] \wedge f_2 \in [9,10] \wedge f_3 \in [3,6] \rightarrow < decision >$$

If it is concluded that F_1 satisfies both R' and R'', we conclude that F_1 satisfies the original property R. Otherwise, we conclude that F_1 does not satisfy R.

10 Related Work

The problem of ensuring that a given firewall is correct is of great importance. This important problem has been addressed using four approaches: firewall testing, firewall analysis, firewall verification, and firewall design. We briefly discuss these four approaches next.

1. Firewall Testing: To test a given firewall F , one generates many packets for which the "expected" decisions of F , accept or discard, are known a priori. The generated packets are then sent to F , and the actual decisions of F for these packets are observed. If the expected decision for each generated packet is the same as the actual decision for the packet, one concludes that the given firewall F is correct. Otherwise, the given firewall F has errors. Different methods of firewall testing differ in how the testing packets are generated. For instance, the test packets can be hand-generated by domain experts to target specific vulnerabilities in the given firewall F, or generated from the formal specifications of the security policy of the given firewall F, as in [4]. A scheme for targeting test packets for better fault coverage is given in [5]. [6] is a framework to generate packets for testing.

2. Firewall Analysis: To analyze a given firewall F, one applies an algorithm to identify (some or all of the) vulnerabilities, conflicts, anomalies, and redundancies in the given firewall F . A systematic method for analyzing firewalls is presented in [7]. The concept of conflicts between rules in a firewall is due to [8] and [9]. A classification of anomalies, as well as algorithms to detect them, may be found in [10] and [11]. (This analysis works for verifying the security policies in IPsec and VPN as well [12].) A framework for understanding the vulnerabilities in a single firewall is outlined in [13], and an analysis of these vulnerabilities presented in [14]. [15] is a quantitative study of configuration errors for a firewall. An example of an efficient firewall analysis algorithm is given in FIREMAN [16].

3. Firewall Verification: To verify a given firewall F against a given property R, one applies an algorithm (similar to the one discussed in this paper) to verify whether or not F satisfies R. The question of how to query a given firewall and obtain the answer (whether or not it satisfies a given property) is discussed in [7] and [1]. These algorithms are proved to be $O(n^d)$ in [2]. Our paper presents a new, $O(nd)$ algorithm.

4. Firewall Design: To ensure a firewall does not have vulnerabilities or other problems, it can be designed from the outset using structured algorithms. Such algorithms, that can generate a firewall from its specification, are provided in [3].

11 Concluding Remarks

We presented in this paper a linear time algorithm for verifying whether a given firewall satisfies a given property. The time complexity of this algorithm is $O(nd)$, where n is the number of rules in the firewall and d is the number of fields that are checked in the firewall or the property. Because verifying whether a given firewall satisfies a given property requires that each conjunct in each rule in the given firewall be compared with the given property, and because the given firewall has nd such conjuncts, the time complexity of any other algorithm for firewall verification can not be less than $O(nd)$, the time complexity of our algorithm.

Our verification algorithm consists of two passes: a deterministic pass followed by a probabilistic pass. The probabilistic pass is executed only when the deterministic pass has the outcome (c), i.e.. it cannot reach a definite conclusion as to whether the given firewall satisfies the given property or not. There are two possible outcomes of the probabilistic pass: (c_1) and (c_2). The (c_1) outcome is that the given firewall satisfies the given property with a high probability, and the (c_2) outcome is that the given firewall does not satisfy the given property. Thus, when our verification algorithm produces outcome (c_1), there is a possibility that this

outcome is erroneous. In Section 7, we showed by analysis that the probability of error when our algorithm produces outcome (c_1) is less than .005. Then in Section 8, we showed by simulation that the probability of our algorithm producing outcome (c_1) is less than .006. It follows that the probability of our algorithm producing an erroneous result is less than 3×10^{-5} .

This means that if our algorithm is used to verify that a given firewall satisfies $100,000$ given properties, then the results of at most 3 of these properties are wrong.

The algorithm presented in this paper is limited to verifying a single firewall. However, we can use this algorithm in conjunction with the techniques described in [2] in order to verify enterprise networks that have tens or even hundreds of firewalls.

The task of verifying whether a given firewall satisfies a given property is a fundamental one, and it can be used to perform other tasks such as the task of checking whether a particular rule in a given firewall is redundant. The problem, of whether our linear time verification algorithm can lead to a linear time redundancy checking algorithm, remains open and merits further research.

References

[1] A. X. Liu, M. G. Gouda, H. H. Ma, and A. H. H. Ngu, "Firewall queries," in *OPODIS*, pp. 197–212, 2004.

[2] M. Gouda, A. X. Liu, and M. Jafry, "Verification of distributed firewalls," in *Proceedings of the IEEE Global Communications Conference (GLOBECOM)*, 2008.

[3] A. X. Liu and M. G. Gouda, "Diverse firewall design," in *DSN*, pp. 595–604, 2004.

[4] J. Jürjens and G. Wimmel, "Specification-based testing of firewalls," in *PSI '02: Revised Papers from the 4th International Andrei Ershov Memorial Conference on Perspectives of System Informatics*, (London, UK), pp. 308–316, Springer-Verlag, 2001.

[5] A. El-Atawy, K. Ibrahim, H. Hamed, and E. Al-Shaer, "Policy segmentation for intelligent firewall testing," *Secure Network Protocols, 2005. (NPSec). 1st IEEE ICNP Workshop on*, pp. 67–72, Nov. 2005.

[6] D. Hoffman and K. Yoo, "Blowtorch: a framework for firewall test automation," in *ASE '05: Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, (New York, NY, USA), pp. 96–103, ACM, 2005.

[7] A. J. Mayer, A. Wool, and E. Ziskind, "Fang: A firewall analysis engine," in *IEEE Symposium on Security and Privacy*, pp. 177–187, 2000.

[8] D. Eppstein and S. Muthukrishnan, "Internet packet filter management and rectangle geometry," in *SODA*, pp. 827–835, 2001.

[9] H. Adiseshu, S. Suri, and G. M. Parulkar, "Detecting and resolving packet filter conflicts," in *INFOCOM*, pp. 1203–1212, 2000.

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

[11] 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.

[12] H. H. Hamed, E. S. Al-Shaer, and W. Marrero, "Modeling and verification of ipsec and vpn security policies," in *ICNP*, pp. 259–278, 2005.

[13] M. Frantzen, F. Kerschbaum, E. E. Schultz, and S. Fahmy, "A framework for understanding vulnerabilities in firewalls using a dataflow model of firewall internals," *Computers & Security*, vol. 20, no. 3, pp. 263–270, 2001.

[14] S. Kamara, S. Fahmy, E. E. Schultz, F. Kerschbaum, and M. Frantzen, "Analysis of vulnerabilities in internet firewalls," *Computers & Security*, vol. 22, no. 3, pp. 214–232, 2003.

[15] A. Wool, "A quantitative study of firewall configuration errors," *IEEE Computer*, vol. 37, no. 6, pp. 62–67, 2004.

[16] L. Yuan, J. Mai, Z. Su, H. Chen, C.-N. Chuah, and P. Mohapatra, "Fireman: A toolkit for firewall modeling and analysis," *Security and Privacy, IEEE Symposium on*, vol. 0, pp. 199–213, 2006.

[17] H. B. Acharya and M. G. Gouda, "Tr-09-01: Linear-time verification of firewalls," tech. rep., University of Texas, Austin, Feb. 2009. http://www.cs.utexas.edu/research/publications/ncstrl/ncstrl2html.cgi.

[18] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Trans. Computers*, vol. 35, no. 8, pp. 677–691, 1986.

[19] K. Strehl and L. Thiele, "Interval diagrams for efficient symbolic verification of processnetworks," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 19, no. 8, pp. 939–956, 2000.

[20] J. García-Alfaro, F. Cuppens, and N. Cuppens-Boulahia, "Analysis of policy anomalies on distributed network security setups," in *ESORICS*, pp. 496–511, 2006.