Available online at www.sciencedirect.com
ScienceDirect
Computer Networks 51 (2007) 1106–1120
Computer Networks
www.elsevier.com/locate/comnet

Structured firewall design ☆

Mohamed G. Gouda, Alex X. Liu *

Department of Computer Sciences, The University of Texas at Austin, Austin, TX 78712-0233, United States

Received 6 September 2005; received in revised form 13 April 2006; accepted 17 June 2006
Available online 15 August 2006

Responsible Editor: D. Frincke

Abstract

A firewall is a security guard placed at the point of entry between a private network and the outside Internet such that all incoming and outgoing packets have to pass through it. The function of a firewall is to examine every incoming or outgoing packet and decide whether to accept or discard it. This function is conventionally specified by a sequence of rules, where rules often conflict. To resolve conflicts, the decision for each packet is the decision of the first rule that the packet matches. The current practice of designing a firewall directly as a sequence of rules suffers from three types of major problems: (1) the consistency problem, which means that it is difficult to order the rules correctly; (2) the completeness problem, which means that it is difficult to ensure thorough consideration for all types of traffic; (3) the compactness problem, which means that it is difficult to keep the number of rules small (because some rules may be redundant and some rules may be combined into one rule).

To achieve consistency, completeness, and compactness, we propose a new method called *structured firewall design*, which consists of two steps. First, one designs a firewall using a firewall decision diagram instead of a sequence of often conflicting rules. Second, a program converts the firewall decision diagram into a compact, yet functionally equivalent, sequence of rules. This method addresses the consistency problem because a firewall decision diagram is conflict-free. It addresses the completeness problem because the syntactic requirements of a firewall decision diagram force the designer to consider all types of traffic. It also addresses the compactness problem because in the second step we use two algorithms (namely FDD reduction and FDD marking) to combine rules together, and one algorithm (namely firewall compaction) to remove redundant rules. Moreover, the techniques and algorithms presented in this paper are extensible to other rule-based systems such as IPsec rules.

Keywords: Network security; Firewall; Firewall design; Firewall policy

☆ This material is based upon work supported by the National Science Foundation under Grant No. 0520250.
* Corresponding author. Tel.: +1 512 471 9711; fax: +1 512 471 9536.
E-mail addresses: gouda@cs.utexas.edu (M.G. Gouda), alex@cs.utexas.edu (A.X. Liu).

1389-1286/$ - see front matter
doi:10.1016/j.comnet.2006.06.015

1. Introduction

1.1. Firewall basics

Firewalls are crucial elements in network security, and have been widely deployed in most businesses and institutions for securing private networks. A firewall is placed at the point of entry between a private network and the outside Internet such that all incoming and outgoing packets have to pass through it. The function of a firewall is to examine every incoming or outgoing packet and decide whether to accept or discard it. A packet can be viewed as a tuple with a finite number of fields such as source IP address, destination IP address, source port number, destination port number, and protocol type. The function of a firewall is conventionally specified as a sequence of rules. Each rule in a firewall is of the form

$\langle predicate \rangle \rightarrow \langle decision \rangle$.

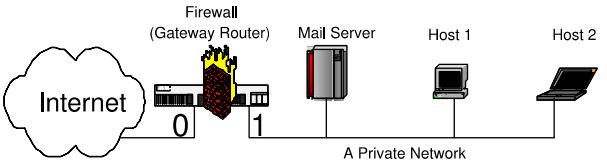
The $\langle predicate \rangle$ of a rule is a boolean expression over some packet fields together with the physical network interface on which a packet arrives. For simplicity, we assume that each packet has a field containing the identification of the network interface on which a packet arrives. The $\langle decision \rangle$ of a rule can be *accept*, or *discard*, or a combination of these decisions with other options such as a logging option. For simplicity, we assume that the $\langle decision \rangle$ of a rule is either *accept* or *discard*.

A packet *matches* a rule if and only if (*iff*) the packet satisfies the predicate of the rule. The rules in a firewall often conflict. Two rules in a firewall *conflict* iff they overlap and also have different decisions. Two rules in a firewall *overlap* iff there is at least one packet that can match both rules. Due to conflicts among rules, a packet may match more than one rule in a firewall, and the rules that a packet matches may have different decisions. To resolve conflicts, the decision for each packet is the decision of the first (i.e., highest priority) rule that the packet matches. Consequently, the rules in a firewall are order sensitive. To ensure that every packet has at least one matching rule in a firewall, the predicate of the last rule in a firewall is usually a tautology. The last rule of a firewall is usually called the *default rule* of the firewall.

1.2. Consistency, completeness and compactness

Because of the conflicts and order sensitivity of firewall rules, designing a firewall directly as a sequence of rules suffers from these three problems: the consistency problem, the completeness problem, and the compactness problem. Next, we expatiate on these three problems via a simple firewall example shown in Fig. 1. This firewall resides on a gateway router that connects a private network to the outside Internet. The gateway router has two interfaces: interface 0, which connects the router to the outside Internet, and interface 1, which connects the router to the private network. In this example, we assume that every packet has the following five fields.

Name	Meaning
I	Interface
S	Source IP address
D	Destination IP address
N	Destination port number
P	Protocol type



1. Rule r_1: $(\mathbf{I = 0}) \wedge (\mathbf{S = any}) \wedge (\mathbf{D = Mail\ Server}) \wedge (\mathbf{N = 25}) \wedge (\mathbf{P = tcp}) \rightarrow \mathbf{accept}$
   (This rule allows incoming SMTP packets to proceed to the mail server.)
2. Rule r_2: $(\mathbf{I = 0}) \wedge (\mathbf{S = Malicious\ Hosts}) \wedge (\mathbf{D = any}) \wedge (\mathbf{N = any}) \wedge (\mathbf{P = any}) \rightarrow \mathbf{discard}$
   (This rule discards incoming packets from previously known malicious hosts.)
3. Rule r_3: $(\mathbf{I = 1}) \wedge (\mathbf{S = any}) \wedge (\mathbf{D = any}) \wedge (\mathbf{N = any}) \wedge (\mathbf{P = any}) \rightarrow \mathbf{accept}$
   (This rule allows any outgoing packet to proceed.)
4. Rule r_4: $(\mathbf{I = any}) \wedge (\mathbf{S = any}) \wedge (\mathbf{D = any}) \wedge (\mathbf{N = any}) \wedge (\mathbf{P = any}) \rightarrow \mathbf{accept}$
   (This rule allows any incoming or outgoing packet to proceed.)

Fig. 1. A firewall example.

A firewall on the Internet typically consists of hundreds or thousands of rules. Here for simplicity, this firewall example only has four rules. Although this firewall is small, it exemplifies all the following three problems.

1. *Consistency problem*: It is difficult to order the rules in a firewall correctly. This difficulty mainly comes from conflicts among rules. Because rules often conflict, the order of the rules in a firewall is critical. The decision for every packet is the decision of the first rule that the packet matches. In the firewall example in Fig. 1, rule r_1 and r_2 conflict since the SMTP packets from previously known malicious hosts to the mail server match both rules and the decisions of r_1 and r_2 are different. Because r_1 is listed before r_2 and the decision of rule r_1 is "*accept*", the SMTP packets from previously known malicious hosts are allowed to proceed to the mail server. However, such packets probably should be prohibited from reaching the mail server because they originate from malicious hosts. Therefore, rules r_1 and r_2 probably should be swapped.Because of the conflicts, the net effect of a rule cannot be understood by the literal meaning of the rule. The decision of a rule affects the fate of the packets that match this rule but does not match any rule listed before this rule. To understand one single rule r_i, one needs to go through all the rules from r_1 to r_{i-1}, and for every rule r_j, where $1 \leqslant j \leqslant i-1$, one needs to figure out the logical relationship between the predicate of r_j and that of r_i. In the firewall example in Fig. 1, the net effect of rule r_2 is not to "discard all packets originated from previously known malicious hosts", but rather is to "discard all non-SMTP packets originated from previously known malicious hosts". The difficulty in understanding firewall rules in turn makes the design and maintenance of a firewall error-prone. Maintenance of a firewall usually involves inserting, deleting or updating rules, and reporting the function of the firewall to others such as managers. All of these tasks require precise understanding of firewalls, which is difficult, especially when the firewall administrator is forced to maintain a legacy firewall that is not originally designed by him.
2. *Completeness problem*: It is difficult to ensure that all possible packets are considered. To ensure that every packet has at least one matching rule in a firewall, the common practice is to make the predicate of the last rule a tautology. This is clearly not a good way to ensure the thorough consideration of all possible packets. In the firewall example in Fig. 1, due to the last rule r_4, non-email packets from the outside to the mail server and email packets from the outside to the hosts other than the mail server are accepted by the firewall. However, these two types of traffic probably should be blocked. A mail server is usually dedicated to email service only. When a host other than the mail server starts to behave like a mail server, it could be an indication that the host has been hacked and it is sending out spam. To block these two types of traffic, the following two rules should be inserted immediately after rule r_1 in the above firewall:
   (a) $(I = 0) \wedge (S = any) \wedge (D = Mail\ Server) \wedge (N = any) \wedge (P = any) \rightarrow discard$
   (b) $(I = 0) \wedge (S = any) \wedge (D = any) \wedge (N = 25) \wedge (P = tcp) \rightarrow discard$
3. *Compactness problem*: A poorly designed firewall often has redundant rules. A rule in a firewall is redundant iff removing the rule does not change the function of the firewall, i.e., does not change the decision of the firewall for every packet. In the above firewall example in Fig. 1, rule r_3 is redundant. This is because all the packets that match r_3 but do not match r_1 and r_2 also match r_4, and both r_3 and r_4 have the same decision. Therefore, this firewall can be made more compact by removing rule r_3.

The consistency problem and the completeness problem cause firewall errors. An error in a firewall means that the firewall either accepts some malicious packets, which consequently creates security holes on the firewall, or discards some legitimate packets, which consequently disrupts normal businesses. Given the importance of firewalls, such errors are not acceptable. Unfortunately, it has been observed that most firewalls on the Internet are poorly designed and have many errors in their rules [25].

The compactness problem causes low firewall performance. In general, the smaller the number of rules that a firewall has, the faster the firewall can map a packet to the decision of the first rule the packet matches. Reducing the number of rules is especially useful for the firewalls that use TCAM (Ternary Content Addressable Memory). Such firewalls use $O(n)$ space (where n is the number of rules) and constant time in mapping a packet to a decision. Despite the high performance of such

TCAM-based firewalls, TCAM has very limited size and consumes much more power as the number of rules increases. Size limitation and power consumption are the two major issues for TCAM-based firewalls.

1.3. Structured firewall design

To achieve consistency, completeness, and compactness, we propose a new method called *structured firewall design*, which consists of two steps. First, one designs a firewall using a firewall decision diagram (FDD for short) instead of a sequence of often conflicting rules. Second, a program converts the FDD into a compact, yet functionally equivalent, sequence of rules. This method addresses the consistency problem because an FDD is conflict-free. It addresses the completeness problem because the syntactic requirements of an FDD force the designer to consider all types of traffic. It also addresses the compactness problem because in the second step we use two algorithms (namely FDD reduction and FDD marking) to combine rules together, and one algorithm (namely firewall compaction) to remove redundant rules.

In some sense, our method of structured firewall design is like the method of structured programming, and the method of designing a firewall directly as a sequence of conflicting rules is like the method of writing a program with many goto statements. In late 1960s, Dijkstra pointed out that goto statements are considered harmful [10] because a program with many goto statements is very difficult to understand and therefore writing such a program is very error prone. Similarly, a firewall of a sequence of conflicting rules is very difficult to understand and writing a sequence of conflicting rules directly is extremely error prone.

Using the method of structured firewall design, the firewall administrator only deals with the FDD that uniquely represents the semantics of a firewall. The FDD is essentially the formal specification of a firewall. Since an FDD can be converted to an equivalent sequence of rules, our method does not require any modification to any existing firewall, which takes a sequence of rules as its configuration. Whenever the firewall administrator wants to change the function of his firewall, he only needs to modify the FDD and then use programs to automatically generate a new sequence of rules. This process is like a programmer first modifying his source code and then compiling it again.

Note that this paper is primarily on how to design stateless firewalls. There are two types of firewalls: stateless firewalls and stateful firewalls. If a firewall decides the fate of every packet solely by examining the packet itself, then the firewall is called a *stateless firewall*. If a firewall decides the fate of some packet not only by examining the packet itself but also by examining the packets that the firewall has accepted previously, then the firewall is called a *stateful firewall*. Studying how to design stateless firewalls is particularly important for two major reasons. First, many firewalls deployed on the Internet are stateless. Second, as we have shown in [13], the methods for designing stateless firewalls can also be used to assist the design of stateful firewalls. In other words, studying the design of stateless firewalls is the foundation for further exploration of the design of stateful firewalls. Although this paper concerns the design of firewalls, the techniques and algorithms presented are extensible to other security policy systems such as the one discussed in [22].

The rest of this paper proceeds as follows. In Section 2, we examine related work. In Section 3, we introduce firewall decision diagrams. In Section 4, we present Algorithm 1 whose function is to reduce the size of a user-specified FDD. In Section 5, we present Algorithm 2 whose function is to do some marking on the reduced FDD. In Section 6, we present Algorithm 3 whose function is to generate firewall rules with the help of the marking information produced by Algorithm 2. In Section 7, we present Algorithm 4 whose function is to remove redundant rules from the firewall rules generated by Algorithm 3. In Section 8, we present Algorithm 5 whose function is to simplify firewall rules. In Section 10, we give concluding remarks.

2. Related work

It has been observed that most firewalls on the Internet are poorly designed and have many configuration errors in their rules [25,9]. There are two approaches to reduce firewall design errors. The first approach is to prevent errors from happening when designing firewalls. The second approach is to detect errors after firewalls have been designed.

With the first approach, people have tried to invent high-level languages that can be used to specify firewall rules. Examples of such languages are the simple model definition language in [6], the Lisp-like language in [15], the declarative predicate

language in [7], and the high-level firewall language in [1]. These high-level firewall languages are helpful for designing firewalls because otherwise people have to use vendor specific languages to describe firewall rules. However, a firewall specified using these high-level firewall languages is still a sequence of rules and the rules may still conflict. The three problems of consistency, completeness and compactness that are inherent in designing a firewall by a sequence of rules still remain. Our firewall design method belongs to the first approach, but we propose a new firewall design paradigm that addresses the three problems of consistency, completeness and compactness. Our method is a complementary and prior step to those high-level firewall languages.

With the second approach, two methods have been proposed previously. The first method is to analyze the function of a firewall by issuing firewall queries. A firewall query is a question regarding the function of a firewall. For example, the firewall administrator may want to ask the question "Which computers in the private network can receive packets from a known malicious host in the outside Internet?" The concept of firewall queries was introduced in [20,17]. A language for describing firewall queries and an algorithm for processing firewall queries are presented in [19].

The second method is conflict detection. The basic idea of this method is as follows. First, the firewall administrator uses a program to detect all pairs of conflicting rules. Second, he manually examines every pair of conflicting rules to see whether the two rules need to be swapped or a new rule needs to be added. How to efficiently detect all pairs of conflicting rules in a firewall has been discussed in [16,11,21,5]. Similar to conflict detection, six types of "anomalies" were defined in [3,4].

Patz et al. presented in [22] a "decorrelation algorithm" that takes policy rules and breaks them down into decorrelated simple expressions. The sprit of our paper is the opposite: we should design a firewall in a consistent (i.e., decorrelated) and complete fashion for correctness purposes, and then automatically generate a compact but possibly inconsistent (i.e., correlated) firewall rules for efficiency purposes.

Firewall vulnerabilities are discussed and classified in [18,12]. However, the focus of [18,12] are the vulnerabilities of the packet filtering software and the supporting hardware part of a firewall, not the configuration of a firewall.

3. Firewall decision diagrams

A *field* F_i is a variable whose domain, denoted $D(F_i)$, is a finite interval of non-negative integers. For example, the domain of the source address in an IP packet is $[0, 2^{32} - 1]$.

A *packet* over fields $F_1, \ldots, F_d$ is a d-tuple $(p_1, \ldots, p_d)$ where each p_i $(1 \leqslant i \leqslant d)$ is an element of $D(F_i)$. We use Σ to denote the set of all packets over fields $F_1, \ldots, F_d$. It follows that Σ is a finite set and $|\Sigma| = |D(F_1)| \times \cdots \times |D(F_d)|$, where $|\Sigma|$ denotes the number of elements in set Σ and each $|D(F_i)|(1 \leqslant i \leqslant d)$ denotes the number of elements in set $D(F_i)$.

A *firewall decision diagram* (FDD) f over fields $F_1, \ldots, F_d$ is an acyclic and directed graph that has the following five properties:

1. There is exactly one node in f that has no incoming edges. This node is called the *root* of f. The nodes in f that have no outgoing edges are called *terminal* nodes of f.
2. Each node v in f is labeled with a field, denoted $F(v)$, such that
$$F(v) \in \begin{cases} \{F_1, \ldots, F_d\} & \text{if } v \text{ is non-terminal} \\ \{accept, discard\} & \text{if } v \text{ is terminal.} \end{cases}$$
3. Each edge e in f is labeled with a non-empty set of integers, denoted $I(e)$, such that if e is an outgoing edge of node v, then we have
$I(e) \subseteq D(F(v))$.
4. A directed path in f from the root to a terminal node is called a *decision path*. No two nodes on a decision path have the same label.
5. The set of all outgoing edges of a node v in f, denoted $E(v)$, satisfies the following two conditions:
   (a) *Consistency*: $I(e) \cap I(e') = \emptyset$ for any two distinct edges e and e' in $E(v)$.
   (b) *Completeness*: $\cup_{e \in E(v)} I(e) = D(F(v))$.

Fig. 2 shows an example of an FDD over two fields F_1 and F_2. The domain of each field is the interval $[1, 10]$. Note that in labelling the terminal nodes, we use letter "*a*" as a shorthand for "*accept*" and letter "*d*" as a shorthand for "*discard*". These two notations are carried through the rest of this paper.

In this paper, the label of an edge in an FDD is always represented by the minimum number of non-overlapping integer intervals whose union equals the label of the edge. For example, one

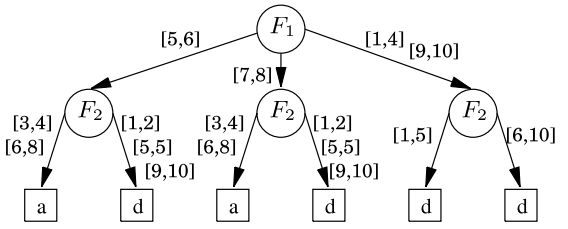


Fig. 2. An FDD example.

outgoing edge of the root is labeled with the set $\{1,2,3,4,9,10\}$, which is represented by the two intervals $[1,4]$ and $[9,10]$.

For brevity, in the rest of this paper, we assume that all packets and all FDDs are over the d fields $F_1,\cdots,F_d$ unless otherwise specified.

A firewall decision diagram maps each packet to a decision by testing the packet down the diagram from the root to a terminal node, which indicates the decision of the firewall for the packet. Each non-terminal node in a firewall decision diagram specifies a test of a packet field, and each edge descending from that node corresponds to some possible values of that field. Each packet is mapped to a decision by starting at the root, testing the field that labels this node, then moving down the edge whose label contains the value of the packet field; this process is then repeated for the sub-diagram rooted at the new node.

A decision path in an FDD is represented by $\langle v_1 e_1 \cdots v_k e_k v_{k+1}\rangle$ where v_1 is the root, v_{k+1} is a terminal node, and each $e_i\,(1 \leqslant i \leqslant k)$ is a directed edge from node v_i to node v_{i+1}.

A decision path $\langle v_1 e_1 \cdots v_k e_k v_{k+1}\rangle$ in an FDD represents the following rule:

$$F_1 \in S_1 \wedge \cdots \wedge F_d \in S_d \rightarrow \langle decision\rangle,$$

where

$$S_i = \begin{cases} I(e_j) & \text{if there is a node } v_j \text{ in the decision} \\ & \text{path that is labeled with field } F_i \\ D(F_i) & \text{otherwise.} \end{cases}$$

and $\langle decision\rangle$ is the label of the terminal node v_{k+1} in the path.

For an FDD f, we use $f.rules$ to denote the set of all rules that are represented by all the decision paths of f. For any packet p, there is one and only one rule in $f.rules$ that p matches because of the consistency and completeness properties of an FDD. For example, the rules represented by all the decision paths of the FDD in Fig. 2 are listed in Fig. 3. Taking the example of the packet $(7,9)$, it matches only rule r_4 in Fig. 3.

The semantics of an FDD f is defined as follows: for any packet p, f maps p to the decision of the rule (in fact the only rule) that p matches in $f.rules$. More precisely, a packet $(p_1,\ldots,p_d)$ is *accepted by* an FDD f iff there is a rule of the form

$$F_1 \in S_1 \wedge \cdots \wedge F_d \in S_d \rightarrow accept$$

in $f.rules$ such that the condition $p_1 \in S_1 \wedge \cdots \wedge p_d \in S_d$ holds. Similarly, a packet $(p_1,\ldots,p_d)$ is *discarded by* an FDD f iff there is a rule of the form

$$F_1 \in S_1 \wedge \cdots \wedge F_d \in S_d \rightarrow \ discard$$

in $f.rules$ such that the condition $p_1 \in S_1 \wedge \cdots \wedge p_d \in S_d$ holds. For example, the packet $(6,8)$ is discarded by the FDD in Fig. 2 because the rule that this packet matches is rule r_4 in Fig. 3 and the decision of this rule is "*discard*".

Let f be an FDD. The *accept set* of f, denoted $f.accept$, is the set of all packets that are accepted by f. Similarly, the *discard set* of f, denoted $f.discard$, is the set of all packets that are discarded by f. These two sets associated with an FDD precisely define the semantics of the FDD.

Based on the definitions of accept set and discard set, we have the following theorem. (Recall that Σ denotes the set of all packets over the fields $F_1,\ldots,F_d$.)

Theorem 1 (Theorem of FDDs). *For any FDD f, the following two conditions hold*:

1. $f.accept \cap f.discard = \emptyset$, *and*
2. $f.accept \cup f.discard = \Sigma$.

Two FDDs f and f' are *equivalent* iff they have identical accept sets and identical discard sets, i.e., $f.accept = f'.accept$ and $f.discard = f'.discard$.

There are some similarities between the structure of firewall decision diagrams and that of interval decision diagrams [24], which are mainly used in

$$\begin{aligned}
r_1&: F_1 \in [5,6] \wedge F_2 \in [3,4] \cup [6,8] && \rightarrow a\\
r_2&: F_1 \in [5,6] \wedge F_2 \in [1,2] \cup [5,5] \cup [9,10] && \rightarrow d\\
r_3&: F_1 \in [7,8] \wedge F_2 \in [3,4] \cup [6,8] && \rightarrow a\\
r_4&: F_1 \in [7,8] \wedge F_2 \in [1,2] \cup [5,5] \cup [9,10] && \rightarrow d\\
r_5&: F_1 \in [1,4] \cup [9,10] \wedge F_2 \in [1,5] && \rightarrow d\\
r_6&: F_1 \in [1,4] \cup [9,10] \wedge F_2 \in [6,10] && \rightarrow d
\end{aligned}$$

Fig. 3. All rules represented by FDD in Fig. 2.

formal verification. However, there are two major differences. First, in a firewall decision diagram, the label of a non-terminal node must have a finite domain; while in an interval decision diagram, the label of a non-terminal node may have an infinite domain. Second, in a firewall decision diagram, the label of an edge is a set of integers which could be the union of several non-continuous intervals; while in an interval decision diagram, the label of an edge is limited to only one interval. In broader sense, the structure of firewall decision diagrams is also similar to other types of decision diagrams such as the binary decision diagrams [8] and decision trees [23]. But note that the optimization goal of reducing the total number of simple rules generated is unique to firewall decision diagrams, which will be explored next.

4. FDD reduction

In this section, we present an algorithm for reducing the number of decision paths in an FDD. This reduction helps to reduce the number of rules generated from an FDD. First, we introduce two concepts: isomorphic nodes in an FDD and reduced FDDs.

Two nodes v and v' in an FDD are *isomorphic* iff v and v' satisfy one of the following two conditions:

1. Both v and v' are terminal nodes with identical labels.
2. Both v and v' are non-terminal nodes and there is a one-to-one correspondence between the outgoing edges of v and the outgoing edges of v' such that every pair of corresponding edges have identical labels and they both point to the same node.

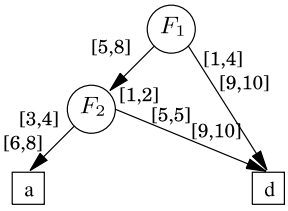


Fig. 5. A reduced FDD.

An FDD f is *reduced* iff it satisfies all of the following three conditions:

1. No node in f has only one outgoing edge.
2. No two nodes in f are isomorphic.
3. No two nodes have more than one edge between them.

Algorithm 1 (FDD reduction) in Fig. 4 takes any FDD and outputs an equivalent but reduced FDD. The correctness of this algorithm follows directly from the semantics of FDDs. Note that this algorithm for reducing an FDD is similar to the one described in [8] for reducing a BDD.

As an example, if we apply Algorithm 1 to the FDD in Fig. 2, we get the reduced FDD in Fig. 5. Note that the FDD in Fig. 2 consists of six decision paths, whereas the FDD in Fig. 5 consists of three decision paths.

5. FDD marking

A firewall rule of the form $F_1 \in S_1 \wedge \cdots \wedge F_d \in S_d \rightarrow \langle decision \rangle$ is *simple* iff every S_i $(1 \leqslant i \leqslant d)$

Algorithm 1 (FDD Reduction)
Input : An FDD f
Output : A reduced FDD that is equivalent to f
Steps:

Repeatedly apply the following three reductions to f until none of them can be applied any further.

1. If there is a node v that has only one outgoing edge e, assuming e points to node v', then remove both node v and edge e, and let all the edges that point to v point to v'.
2. If there are two nodes v and v' that are isomorphic, then remove v' together with all its outgoing edges, and let all the edges that point to v' point to v.
3. If there are two edges e and e' that both are between the same pair of nodes, then remove e' and change the label of e from $I(e)$ to $I(e) \cup I(e')$. (Recall that $I(e)$ denotes the label of edge e.)

Fig. 4. Algorithm 1 (FDD reduction).

is an interval of consecutive non-negative integers. Because most firewalls require simple rules, we want to minimize the number of simple rules generated from an FDD. The number of simple rules generated from a "marked version" of an FDD is less than or equal to the number of simple rules generated from the original FDD. Next, we define a marked FDD.

A *marked version* f' of an FDD f is the same as f except that exactly one outgoing edge of each non-terminal node in f' is marked "*all*". Since the labels of the edges that are marked "*all*" do not change, the two FDDs f and f' have the same semantics, i.e., f and f' are equivalent. A marked version of an FDD is also called a *marked FDD*.

Fig. 6 shows two marked versions f' and f'' of the FDD in Fig. 5. In f', the edge labeled $[5,8]$ and the edge labeled $[1,2] \cup [5,5] \cup [9,10]$ are both marked *all*. In f'', the edge labeled $[1,4] \cup [9,10]$ and the edge labeled $[1,2] \cup [5,5] \cup [9,10]$ are both marked *all*.

The *load of a non-empty set of integers S*, denoted *load*(S), is the minimal number of non-overlapping integer intervals that cover S. For example, the load of the set $\{1,2,3,5,8,9,10\}$ is 3 because this set is covered by the three integer intervals $[1,3]$, $[5,5]$ and $[8,10]$, and this set cannot be covered by any two intervals.

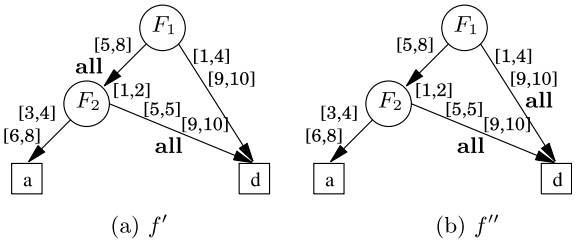


Fig. 6. Two marked FDDs.

The *load of an edge e* in a marked FDD, denoted *load*(e), is defined as follows:

$$load(e) = \begin{cases} 1 & \text{if } e \text{ is marked } all \\ load(I(e)) & \text{otherwise.} \end{cases}$$

The *load of a node v* in a marked FDD, denoted *load*(v), is defined recursively as follows:

$$load(v) = \begin{cases} 1 & \text{if } v \text{ is terminal} \\ \sum_{i=1}^{k} (load(e_i) \times load(v_i)) & \text{if } v \text{ is non-terminal: suppose } v \text{ has } k \\ & \text{outgoing edges } e_1, \ldots, e_k, \text{ which point to} \\ & \text{nodes } v_1, \ldots, v_k \text{ respectively} \end{cases}$$

Algorithm 2 (FDD Marking)
Input : An FDD f
Output : A marked version f' of f such that for every marked version f'' of f, $load(f') \leq load(f'')$
Steps:

1. Compute the load of each terminal node v in f as follows: $load(v) := 1$
2. **while** there is a node v whose load has not yet been computed, suppose v has k outgoing edges $e_1, \cdots, e_k$ and these edges point to nodes $v_1, \cdots, v_k$ respectively, and the loads of these k nodes have been computed
   do
   (a) Among the k edges $e_1, \cdots, e_k$, choose an edge e_j with the largest value of $(load(e_j) - 1) \times load(v_j)$, and mark edge e_j with "*all*".
   (b) Compute the load of v as follows: $load(v) := \sum_{i=1}^{k} (load(e_i) \times load(v_i))$.
   end

Fig. 7. Algorithm 2 (FDD marking).

The *load of a marked FDD f*, denoted *load*(*f*), equals the load of the root of *f*.

Different marked versions of the same FDD may have different loads. Fig. 6 shows two marked versions f' and f'' of the same FDD in Fig. 5. The load of f' is 5, whereas the load of f'' is 4.

As we will see in Section 8, for any two marked versions of the same FDD, the one with the smaller load will generate a smaller number of simple rules. Therefore, we should use the marked version of FDD f that has the minimal load to generate rules.

Algorithm 2 (FDD marking) in Fig. 7 takes any FDD and outputs a marked version that has the minimal load.

As an example, if we apply Algorithm 2 to the reduced FDD in Fig. 5, we get the marked FDD in Fig. 6(b).

The correctness of Algorithm 2 is stated in Theorem 2, whose proof is presented in the Appendix A.

Theorem 2. *The load of an FDD marked by Algorithm* 2 (*FDD Marking*) *is minimal.*

6. Firewall generation

In this section, we present an algorithm for generating a sequence of rules, which form a firewall, from a marked FDD such that the firewall has the same semantics as the marked FDD. First, we introduce the semantics of a firewall.

A packet $(p_1,\ldots,p_d)$ matches a rule $F_1 \in S_1 \wedge \cdots \wedge F_d \in S_d \rightarrow \langle decision \rangle$ iff the condition $p_1 \in S_1 \wedge \cdots \wedge p_d \in S_d$ holds. A *firewall* consists of a sequence of rules such that for any packet there is at least one rule that the packet matches. A firewall maps every packet to the decision of the first rule that the packet matches. Let f be a firewall of a sequence of rules. The set of all packets accepted by f is denoted $f.accept$, and the set of all packets discarded by f is denoted $f.discard$. The next theorem follows from these definitions. Recall that Σ denotes the set of all packets over the fields $F_1,\ldots,F_d$.

Theorem 3 (Theorem of firewalls). *For a firewall f of a sequence of rules,*

1. $f.accept \cap f.discard = \emptyset$, *and*
2. $f.accept \cup f.discard = \Sigma$

Based on Theorem 1 and 3, we now extend the equivalence relations on FDDs to incorporate the firewalls. Given f and f', where each is an FDD or a firewall, f and f' are *equivalent* iff they have identical accept sets and identical discard sets, i.e., $f.accept = f'.accept$ and $f.discard = f'.discard$. This equivalence relation is symmetric, reflexive, and transitive. We use $f \equiv f'$ to denote the equivalence relation between f and f'.

To generate an equivalent firewall from a marked FDD f, we basically make a depth-first traversal of f such that for each non-terminal node v, the outgoing edge marked "*all*" of v is traversed after all the other outgoing edges of v have been traversed. Whenever a terminal node is encountered, assuming $\langle v_1 e_1 \cdots v_k e_k v_{k+1} \rangle$ is the decision path where for every i $(1 \leqslant i \leqslant k)$ e_i is the most recently traversed outgoing edge of node v_i, output a rule r as follows:

$$F_1 \in S_1 \wedge \cdots \wedge F_d \in S_d \rightarrow F(v_{k+1}),$$

where

$$S_i = \begin{cases} I(e_j) & \text{if the decision path has a node } v_j \\ & \text{that is labeled with field} \\ & F_i \text{ and } e_j \text{ is not marked "}all\text{"} \\ D(F_i) & \text{otherwise.} \end{cases}$$

Note that the first rule generated by the above procedure is the first rule in the resulting firewall, the second rule generated is the second rule in the resulting firewall, and so on.

For the above rule r, the predicate $F_1 \in S_1 \wedge \cdots \wedge F_d \in S_d$ is called the *matching predicate* of r.

The rule represented by the path $\langle v_1 e_1 \cdots v_k e_k v_{k+1} \rangle$ is $F_1 \in T_1 \wedge \cdots \wedge F_d \in T_d \rightarrow F(v_{k+1})$, where

$$T_i = \begin{cases} I(e_j) & \text{if the decision path has a node } v_j \\ & \text{that is labeled with field } F_i \\ D(F_i) & \text{otherwise.} \end{cases}$$

We call the predicate $F_1 \in T_1 \wedge \cdots \wedge F_d \in T_d$ the *resolving predicate* of the above rule r. Note that if a packet satisfies the resolving predicate of r, r is the first rule that the packet matches in the firewall generated. If a packet satisfies the resolving predicate of rule r in firewall f, we say the packet is *resolved* by r in f.

Algorithm 3 (Firewall Generation)
Input : A marked FDD f
Output : A firewall that is equivalent to f. For each rule r, $r.mp$ and $r.rp$ is computed
Steps:

Depth-first traverse f such that for each nonterminal node v, the outgoing edge marked "*all*" of v is traversed after all other outgoing edges of v have been traversed. Whenever a terminal node is encountered, assuming $\langle v_1 e_1 \cdots v_k e_k v_{k+1} \rangle$ is the decision path where each e_i is the most recently traversed outgoing edge of node v_i, output a rule r together with its matching predicate $r.mp$ and its resolving predicate $r.rp$ as follows:

r is the rule $F_1 \in S_1 \wedge \cdots \wedge F_d \in S_d \ \rightarrow \ F(v_{k+1})$, where

$$S_i = \begin{cases} I(e_j) & \text{if the decision path has a node } v_j \text{ that is labeled with field } F_i \\ & \text{and } e_j \textit{ is not marked "all"} \\ D(F_i) & \text{otherwise} \end{cases}$$

$r.mp$ is the predicate of rule r.
$r.rp$ is the predicate $F_1 \in T_1 \wedge \cdots \wedge F_d \in T_d$, where

$$T_i = \begin{cases} I(e_j) & \text{if the decision path has a node } v_j \text{ that is labeled with field } F_i \\ D(F_i) & \text{otherwise} \end{cases}$$

Fig. 8. Algorithm 3 (firewall generation).

Algorithm 3 (firewall generation) in Fig. 8 takes any marked FDD and outputs an equivalent firewall. Recall that the ith rule output by Algorithm 3 is the ith rule in the firewall generated. The correctness of this algorithm follows directly from the semantics of FDDs and firewalls. In Algorithm 3, for every rule generated, we also generate its matching predicate and its resolving predicate. In the next section, we will see that these two predicates associated with each rule play important roles in removing redundant rules.

As an example, if we apply Algorithm 3 to the marked FDD in Fig. 6(b), we get the firewall in Fig. 9.

$$\begin{aligned} r_1 &= F_1 \in [5,8] \wedge F_2 \in [3,4] \cup [6,8] \ \rightarrow \ a, \\ r_1.mp &= F_1 \in [5,8] \wedge F_2 \in [3,4] \cup [6,8] \\ r_1.rp &= F_1 \in [5,8] \wedge F_2 \in [3,4] \cup [6,8] \end{aligned}$$

$$\begin{aligned} r_2 &= F_1 \in [5,8] \wedge F_2 \in [1,10] \ \rightarrow \ d, \\ r_2.mp &= F_1 \in [5,8] \wedge F_2 \in [1,10] \\ r_2.rp &= (F_1 \in [5,8] \wedge F_2 \in [1,2] \cup [5,5] \cup [9,10]) \end{aligned}$$

$$\begin{aligned} r_3 &= F_1 \in [1,10] \wedge F_2 \in [1,10] \ \rightarrow \ d, \\ r_3.mp &= F_1 \in [1,10] \wedge F_2 \in [1,10] \\ r_3.rp &= F_1 \in [1,4] \cup [9,10] \wedge F_2 \in [1,10] \end{aligned}$$

Fig. 9. A generated firewall.

7. Firewall compaction

Firewalls often have redundant rules. A rule in a firewall is redundant iff removing the rule does not change the semantics of the firewall, i.e., does not change the accept set and the discard set of the firewall. Removing redundant rules from a firewall produces an equivalent firewall but with fewer rules. For example, the rule r_2 in Fig. 9 is redundant. Removing this rule yields an equivalent firewall with two rules, which are shown in Fig. 10.

In this section, we present an efficient algorithm for discovering redundant rules. Algorithm 4 (firewall compaction) in Fig. 11 takes any firewall and outputs an equivalent but more compact firewall.

In Algorithm 4, "$r_i.rp$ implies $r_k.mp$" means that for any packet p, if p satisfies $r_i.rp$, then p satisfies $r_k.mp$. Checking whether $r_i.rp$ implies $r_k.mp$ is simple. Let $r_i.rp$ be $F_1 \in T_1 \wedge F_2 \in T_2 \wedge \cdots \wedge F_d \in T_d$ and let $r_k.mp$ be $F_1 \in S_1 \wedge F_2 \in S_2 \wedge \cdots \wedge F_d \in S_d$. Then, $r_i.rp$ implies $r_k.mp$ iff for every j, where $1 \leqslant j \leqslant d$, the condition $T_j \subseteq S_j$ holds.

Checking whether no packet satisfies both $r_i.rp$ and $r_j.mp$ is simple. Let $r_i.rp$ be $F_1 \in T_1 \wedge F_2 \in$

1. $F_1 \in [5,8] \wedge F_2 \in [3,4] \cup [6,8] \rightarrow \ a$,
2. $F_1 \in [1,10] \wedge F_2 \in [1,10] \rightarrow \ d$

Fig. 10. A firewall with no redundant rules.

Algorithm 4 (Firewall Compaction)
Input : A firewall $\langle r_1, \cdots, r_n \rangle$
Output : An equivalent but more compact firewall
Steps:
1. **for** $i = n$ **to** 1 **do**
   $redundant[i] :=$ **false**.
2. **for** $i = n$ **to** 1 **do**
   if there exist a rule r_k in the firewall, where $i < k \leq n$, such that the following four conditions hold
   (1) $redundant[k] = false$.
   (2) r_i and r_k have the same decision.
   (3) $r_i.rp$ implies $r_k.mp$.
   (4) for every rule r_j, where $i < j < k$, at least one of the following three conditions holds:
   (a) $redundant[j] = true$.
   (b) r_i and r_j have the same decision.
   (c) no packet satisfies both $r_i.rp$ and $r_j.mp$.
   then $redundant[i] :=$ **true**.
   else $redundant[i] :=$ **false**.
3. **for** $i = n$ **to** 1 **do**
   if $redundant[i] = true$ **then** remove r_i from the firewall.

Fig. 11. Algorithm 4 (firewall compaction).

Algorithm 5 (Firewall Simplification)
Input : A firewall f
Output : A simple firewall f' where f' is equivalent to f
Steps:

while f has a rule of the form $F_1 \in S_1 \wedge \cdots \wedge F_i \in S_i \wedge \cdots \wedge F_d \in S_d \rightarrow \langle decision \rangle$ where some S_i is represented by $[a_1, b_1] \cup \cdots \cup [a_k, b_k]$ where $k \geq 2$.

do
replace this rule by the following k non-overlapping rules:
$F_1 \in S_1 \wedge \cdots \wedge F_i \in [a_1, b_1] \wedge \cdots \wedge F_d \in S_d \rightarrow \langle decision \rangle$,
$F_1 \in S_1 \wedge \cdots \wedge F_i \in [a_2, b_2] \wedge \cdots \wedge F_d \in S_d \rightarrow \langle decision \rangle$,
$\vdots$
$F_1 \in S_1 \wedge \cdots \wedge F_i \in [a_k, b_k] \wedge \cdots \wedge F_d \in S_d \rightarrow \langle decision \rangle$
end

Fig. 12. Algorithm 5 (firewall simplification).

$T_2 \wedge \cdots \wedge F_d \in T_d$ and let $r_j.mp$ be $F_1 \in S_1 \wedge F_2 \in S_2 \wedge \cdots \wedge F_d \in S_d$. We have $r_i.rp \wedge r_j.mp = F_1 \in (T_1 \cap S_1) \wedge F_2 \in (T_2 \cap S_2) \wedge \cdots \wedge F_d \in (T_d \cap S_d)$. Therefore, no packet satisfies both $r_i.rp$ and $r_j.mp$ iff there exists j, where $1 \leqslant j \leqslant d$, such that $T_j \cap S_j = \emptyset$.

The correctness of Algorithm 4 is stated in Theorem 4, whose proof is presented in Appendix A.

Theorem 4. *If we apply Algorithm* 4 *to a firewall* f *and get the resulting firewall* f', *then* f *and* f' *are equivalent.*

As an example, if we apply Algorithm 4 to the firewall in Fig. 9, we get the compact firewall in Fig. 10.

Let n be the number of rules in a firewall and d be the number of packet fields that a rule checks, the computational complexity of Algorithm 4 is $O(n^2 * d)$. Note that d can be regarded as a constant because d is usually small. Most firewalls checks five packet fields: source IP address, destination IP address, source port number, destination port number, and protocol type.

8. Firewall simplification

Most firewall implementations, such as Linux's ipchains [2], requires each firewall rule to be simple. Recall that a firewall rule of the form $F_1 \in S_1 \wedge \cdots \wedge F_d \in S_d \rightarrow \langle decision \rangle$ is *simple* iff every S_i $(1 \leqslant i \leqslant d)$ is an interval of consecutive non-negative integers. A firewall is *simple* iff all its rules are simple.

Algorithm 5 (firewall simplification) in Fig. 12 takes any firewall and outputs an equivalent firewall in which each rule is simple. The correctness of this algorithm follows directly from the semantics of firewalls.

As an example, if we apply Algorithm 5 to the firewall in Fig. 10, we get the firewall in Fig. 13.

What we get from Algorithm 5 is a simple firewall. For each rule $F_1 \in S_1 \wedge \cdots \wedge F_i \in S_i \wedge \cdots \wedge F_d \in S_d \rightarrow \langle decision \rangle$, S_i is an interval of non-negative integers. Some existing firewall products, such as Linux's ipchains [2], require that S_i be represented in a prefix format such as 192.168.0.0/16, where 16 means that the prefix is the first 16 bits of 192.168.0.0 in a binary format. In this paper we stop the level of discussion at simple rules because an integer interval can be converted to multiple prefixes algorithmically. For example, integer interval $[2,8]$ can be converted to three prefixes: $001*, 01*$, 1000. A w-bit integer interval can be converted to at most $2w$-2 prefixes [14].

9. Summary of structured firewall design

In this section, we summarize our structured firewall design method. Fig. 14 shows the five steps of this method.

Our firewall design method starts by a user specifying an FDD f. The consistency and completeness properties of f can be verified automatically based on the syntactic requirements of an FDD. After an FDD is specified, it goes through the following five steps, and we get a simple firewall that is equivalent to the FDD. The first step is to apply Algorithm 1 (FDD reduction) to the user-specified FDD. We then get an equivalent but reduced FDD, which has a smaller number of decision paths. The second step is to apply Algorithm 2 (FDD marking) to the reduced FDD. We then get an equivalent FDD where each non-terminal node has exactly one outgoing edge that is marked *all*. The third step is to apply Algorithm 3 (FDD generation) to the marked FDD. We then get an equivalent firewall. The fourth step is to apply Algorithm 4 (firewall compaction) to the generated firewall. We then get an equivalent firewall with a smaller number of rules. The fifth step is to apply Algorithm 5 (firewall simplification) to this firewall. We then get the final result: a simple firewall that is equivalent to the user-specified FDD.

1. $F_1 \in [5,8] \wedge F_2 \in [3,4] \rightarrow a$,
2. $F_1 \in [5,8] \wedge F_2 \in [6,8] \rightarrow a$,
3. $F_1 \in [1,10] \wedge F_2 \in [1,10] \rightarrow d$,

Fig. 13. A simple firewall.

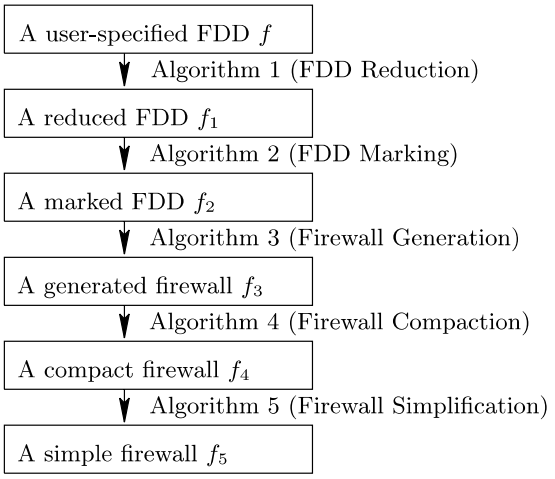


Fig. 14. Five steps of our firewall design method ($f \equiv f_1 \equiv f_2 \equiv f_3 \equiv f_4 \equiv f_5$).

Three of the above five algorithms, namely Algorithm 1 (FDD reduction), Algorithm 2 (FDD marking) and Algorithm 4 (Firewall compaction), are for the purpose of reducing the number of rules in the final simple firewall. Algorithm 1 (FDD reduction) does so by reducing the number of decision paths in the user-specified FDD. Algorithm 2 (FDD marking) does so by reducing the load of some edges in the FDD. Algorithm 4 (firewall compaction) does so by removing some redundant rules from the generated firewall. These three algorithms could reduce the number of simple rules dramatically. Consider the running example illustrated in Figs. 2–13. If we directly generate and simplify our firewall from the FDD in Fig. 2, ignoring Algorithm 1, 2, and 4, we would have ended up with a simple firewall that has 14 rules. However, with the help of these three algorithms, we end up with a simple firewall that has only three rules.

10. Conclusions

Our contribution in this paper is twofold. First, we propose the structured firewall design method that addresses the consistency problem, the completeness problem, and the compactness problem. These three problems are inherent in the current practice of designing a firewall directly as a sequence of (possibly conflicting) rules. Our method starts with a decision diagram that ensures the consistency and completeness properties, and ends up with a sequence of rules that ensures the compactness property. In this process, the user only deals with a firewall decision diagram, which is the formal specification of a firewall. Converting a decision diagram to a compact sequence of rules is automatically carried by a series of five algorithms presented in this paper. Second, we present three optimization techniques, namely FDD reduction, FDD marking, and firewall compaction, for reducing the total number of rules generated from a firewall decision diagram.

In this paper, for ease of presentation, we assume that a firewall maps every packet to one of two decisions: accept or discard. Most firewall software supports more than two decisions such as accept, accept-and-log, discard, and discard-and-log. Our firewall design method can be straightforwardly extended to support more than two decisions.

The design method discussed in this paper is not limited to just firewalls. Rather, the techniques and algorithms presented in this paper are extensible to other rule-based systems such as general packet classifiers and IPsec rules.

Appendix A. Proof of Algorithm 2

Consider an FDD f. Let f' be the version marked by algorithm 2, and let f'' be an arbitrary marked version. Next we prove that $load(f') \leqslant load(f'')$.

Consider a node v, which has k outgoing edges $e_1, e_2, \ldots, e_k$ and these edges point to $v_1, v_2, \ldots, v_k$ respectively, such that the loads of $v_1, v_2, \ldots, v_k$ in f' is the same as those in f''. Clearly such node v exists because the load of any terminal node is constant 1.

Let e_i be the edge marked ALL in f' and e_j be the edge marked ALL in f''. Suppose $i \neq j$. We use $load'(v)$ to denote the load of node v in f' and $load''(v)$ to denote the load of node v in f''. We then have

$$load'(v) = \sum_{t=1}^{i-1}(load(e_t) \times load(v_t)) + load(v_i) + \sum_{t=i+1}^{k}(load(e_t) \times load(v_t))$$

$$load''(v) = \sum_{t=1}^{j-1}(load(e_t) \times load(v_t)) + load(v_j) + \sum_{t=j+1}^{k}(load(e_t) \times load(v_t))$$

$$load'(v) - load''(v) = (load(e_j) - 1) \times load(v_j) - (load(e_i) - 1) \times load(v_i)$$

According to Algorithm 2, $(load(e_j) - 1) \times load(v_j) \leqslant (load(e_i) - 1) \times load(v_i)$. So, $load'(v) \leqslant load''(v)$.

Apply the above argument to any node v in f, we have $load'(v) \leqslant load''(v)$. So, the load of an FDD marked by Algorithm 2 is minimal.

Appendix B. Proof of Theorem 4

Suppose for the rule r_i in firewall $\langle r_1, \ldots, r_n \rangle$, there exist a rule r_k in this firewall, where $i < k \leqslant n$, such that the following four conditions hold:

1. $redundant[k] = false$.
2. r_i and r_k have the same decision.
3. $r_i.rp$ implies $r_k.mp$.
4. for every rule r_j, where $i < j < k$, at least one of the following three conditions holds:
   (a) $redundant[j] = true$.
   (b) r_i and r_j have the same decision.
   (c) no packet satisfies both $r_i.rp$ and $r_j.mp$.

If we remove rule r_i from firewall $\langle r_1, \ldots, r_n \rangle$, the packets whose decision may be affected are the packets that are resolved by r_i in $\langle r_1, \ldots, r_n \rangle$, i.e., the packets that satisfy $r_i.rp$. Let S be the set of all the packets that satisfy $r_i.rp$. Because $r_i.rp$ implies $r_k.mp$ and $redundant[k] = false$, if we remove rule r_i, the packets in S will be resolved by the rules from r_{i+1} to r_k in $\langle r_1, \ldots, r_{i-1}, r_{i+1}, \ldots, r_n \rangle$. Consider a rule r_j where $i < j < k$. If $redundant[j] = true$, we assume r_j has been removed; therefore, rule r_j does not affect the decision of any packet in S. If the two rules r_i and r_j have the same decision, then rule r_j does not affect the decision of any packet in S. If

no packet satisfies both $r_i.rp$ and $r_j.mp$, then any packet in S does not match rule r_j; therefore, rule r_j does not affect the decision of any packet in S. Note that r_i and r_k have the same decision. Therefore, for any packet p in S, the decision that the firewall $\langle r_1, \ldots, r_{i-1}, r_{i+1}, \ldots, r_n \rangle$ makes for p is the same as the decision that the firewall $\langle r_1, \ldots, r_{i-1}, r_i, r_{i+1}, \ldots, r_n \rangle$ makes for p. So rule r_i is redundant.

Suppose we apply Algorithm 4 to a firewall f. Since any rule removed by Algorithm 4 is redundant, the resulting firewall f' is equivalent to the original firewall f.

References

[1] High level firewall language. Available from: <http://www.hlfl.org/>.

[2] ipchains. Available from: <http://www.tldp.org/howto/ipchains-howto.html>.

[3] E. Al-Shaer, H. Hamed, Discovery of policy anomalies in distributed firewalls, in: IEEE INFOCOM'04, 2004, pp. 2605–2616.

[4] E. Al-Shaer, H. Hamed, R. Boutaba, M. Hasan, Conflict classification and analysis of distributed firewall policies, IEEE Journal on Selected Areas in Communications (JSAC) 23 (10) (2005) 2069–2084.

[5] F. Baboescu, G. Varghese, Fast and scalable conflict detection for packet classifiers, in: Proceedings of the 10th IEEE International Conference on Network Protocols, 2002.

[6] Y. Bartal, A.J. Mayer, K. Nissim, A. Wool, Firmato: a novel firewall management toolkit, in: Proceeding of the IEEE Symposium on Security and Privacy, 1999. pp. 17–31.

[7] A. Begel, S. McCanne, S.L. Graham, BPF+: exploiting global data-flow optimization in a generalized packet filter architecture, in: Proceedings of ACM SIGCOMM '99, 1999.

[8] R.E. Bryant, Graph-based algorithms for boolean function manipulation, IEEE Transactions on Computers 35 (8) (1986) 677–691.

[9] CERT, Test the firewall system. Available from: <http://www.cert.org/security-improvement/practices/p060.html>.

[10] E.W. Dijkstra, Goto statement considered harmful, Communications of the ACM 11 (3) (1968) 147–148.

[11] D. Eppstein, S. Muthukrishnan, Internet packet filter management and rectangle geometry, in: Symposium on Discrete Algorithms, 2001, pp. 827–835.

[12] M. Frantzen, F. Kerschbaum, E. Schultz, S. Fahmy, A framework for understanding vulnerabilities in firewalls using a dataflow model of firewall internals, Computers and Security 20 (3) (2001) 263–270.

[13] M.G. Gouda, A.X. Liu, A model of stateful firewalls and its properties, in: Proceedings of the IEEE International Conference on Dependable Systems and Networks (DSN-05), 2005, pp. 320–327.

[14] P. Gupta, N. McKeown, Algorithms for packet classification, IEEE Network 15 (2) (2001) 24–32.

[15] J.D. Guttman, Filtering postures: local enforcement for global policies, in: Proceedings of IEEE Symposium on Security and Privacy, 1997, pp. 120–129.

[16] A. Hari, S. Suri, G.M. Parulkar, Detecting and resolving packet filter conflicts, in: Proceedings of IEEE INFOCOM, 2000, pp. 1203–1212.

[17] S. Hazelhurst, A. Attar, R. Sinnappan, Algorithms for improving the dependability of firewall and filter rule lists, in: Proceedings of the Workshop on Dependability of IP Applications, Platforms and Networks, 2000.

[18] S. Kamara, S. Fahmy, E. Schultz, F. Kerschbaum, M. Frantzen, Analysis of vulnerabilities in internet firewalls, Computers and Security 22 (3) (2003) 214–232.

[19] A.X. Liu, M.G. Gouda, H.H. Ma, A.H. Ngu, Firewall queries, in: T. Higashino (Ed.), Proceedings of the 8th International Conference on Principles of Distributed Systems, Lecture Notes in Computer Science, 3544, Springer-Verlag, 2004, pp. 124–139.

[20] A. Mayer, A. Wool, E. Ziskind, Fang: A firewall analysis engine, in: Proceedings of IEEE Symposium on Security and Privacy, 2000, pp. 177–187.

[21] J.D. Moffett, M.S. Sloman, Policy conflict analysis in distributed system management, Journal of Organizational Computing 4 (1) (1994) 1–22.

[22] G. Patz, M. Condell, R. Krishnan, L. Sanchez, Multidimensional security policy management for dynamic coalitions, in: Proceedings of the DARPA Information Survivability Conference and Exposition (DISCEX II), 2001.

[23] J. Quinlan, Induction of decision trees, Machine Learning 1 (1) (1986) 81–106.

[24] K. Strehl, L. Thiele, Interval diagrams for efficient symbolic verification of process networks, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 19 (8) (2000) 939–956.

[25] A. Wool, A quantitative study of firewall configuration errors, IEEE Computer 37 (6) (2004) 62–67.

Mohamed G. Gouda was born in Egypt. His first B.Sc. was in Engineering and his second was in Mathematics; both are from Cairo University. Later, he obtained M.A. in Mathematics from York University and Masters and Ph.D. in Computer Science from the University of Waterloo. He worked for the Honeywell Corporate Technology Center in Minneapolis 1977–1980. In 1980, he joined the University of Texas at Austin where he currently holds the Mike A. Myers Centennial Professorship in Computer Sciences. He spent one summer at Bell labs in Murray Hill, one summer at MCC in Austin, and one winter at the Eindhoven Technical University in the Netherlands.

His research areas are distributed and concurrent computing and network protocols. In these areas, he has been working on abstraction, formality, correctness, non-determinism, atomicity, reliability, security, convergence, and stabilization. He has published over 60 journal papers, and over 80 conference and workshop papers. He has supervised 19 Ph.D. Dissertations.

He was the founding Editor-in-Chief of the Springer-Verlag journal Distributed Computing 1985–1989. He served on the editorial board of Information Sciences 1996–1999, and he is

currently on the editorial boards of Distributed Computing and the Journal of High Speed Networks.

He was the program committee chairman of ACM SIGCOMM Symposium in 1989. He was the first program committee chairman of IEEE International Conference on Network Protocols in 1993. He was the first program committee chairman of IEEE Symposium on Advances in Computers and Communications, which was held in Egypt in 1995. He was the program committee chairman of IEEE International Conference on Distributed Computing Systems in 1999. He is on the steering committee of the IEEE International Conference on Network Protocols and on the steering committee of the Symposium on Self-Stabilizing Systems, and was a member of the Austin Tuesday Afternoon Club from 1984 till 2001.

He is the author of the textbook "Elements of Network Protocol Design", published by John-Wiley & Sons in 1998. This is the first ever textbook where network protocols are presented in an abstract and formal setting. He also coauthored, with Tommy M. McGuire, the monograph "The Austin Protocol Compiler", published by Springer in 2005.

He is the 1993 winner of the Kuwait Award in Basic Sciences. He was the recipient of an IBM Faculty Partnership Award for the academic year 2000–2001 and again for the academic year 2001–2002 and became a Fellow of the IBM Center for Advanced Studies in Austin in 2002. He won the 2001 IEEE Communication Society William R. Bennet Best Paper Award for his paper "Secure Group Communications Using Key Graphs", coauthored with C.K. Wong and S.S. Lam and published in the February 2000 issue of the IEEE/ACM Transactions on Networking (vol. 8, Number 1, pp. 16–30). In 2004, his paper "Diverse Firewall Design", coauthored with Alex X. Liu and published in the proceedings of the International Conference on Dependable Systems and Networks, won the William C. Carter award.

Alex X. Liu received the B.S. degree in computer sciences from Jilin University, China, in 1996. He received the M.S. degree in computer sciences from the University of Texas at Austin in 2002, where he is currently working toward a Ph.D. degree in computer sciences. He has published 14 refereed conference and journal papers on a variety of network security topics. He won the 2004 IEEE& IFIP William C. Carter Award, the 2004 National Outstanding Overseas Students Award sponsored by the Ministry of Education of China, the 2005 George H. Mitchell Award for Excellence in Graduate Research in the University of Texas at Austin, and the 2005 James C. Browne Outstanding Graduate Student Fellowship in the University of Texas at Austin. His research interests include network security, computer security, networks protocols, and distributed systems.