

# A Model of Stateful Firewalls and its Properties

Mohamed G. Gouda and Alex X. Liu<sup>1</sup>

Department of Computer Sciences,  
The University of Texas at Austin,  
Austin, Texas 78712-1188, U.S.A.  
Email: {gouda, alex}@cs.utexas.edu

## Abstract

We propose the first model of stateful firewalls. In this model, each stateful firewall has a variable set called the state of the firewall, which is used to store some packets that the firewall has accepted previously and needs to remember in the near future. Each stateful firewall consists of two sections: a stateful section and a stateless section. Upon receiving a packet, the firewall processes it in two steps. In the first step, the firewall augments the packet with an additional field called the tag, and uses the stateful section to compute the value of this field according to the current state of the firewall. In the second step, the firewall compares the packet together with its tag value against a sequence of rules in the stateless section to identify the first rule that the packet matches: the decision of this rule determines the fate of the packet. Our model of stateful firewalls has several favorable properties. First, despite its simplicity, it can express a variety of state tracking functionalities. Second, it allows us to inherit the rich results in stateless firewall design and analysis. Third, it provides backward compatibility such that a stateless firewall can also be specified using our model. This paper goes beyond proposing this stateful firewall model itself. A significant portion of this paper is devoted to analyzing the properties of stateful firewalls that are specified using our model. We outline a method for verifying whether a firewall is truly stateful. The method is based on the three properties of firewalls: conforming, grounded, and proper. We show that if a firewall satisfies these three properties, then the firewall is truly stateful.

## 1 Introduction

Serving as the first line of defense against unauthorized and potentially malicious traffic, firewalls 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 so that all incoming and outgoing packets have to pass through it. The function of a firewall is to map each incoming or outgoing packet to one of a set of predefined decisions, such as *accept* or *discard*. Based on how a decision is made for every packet, firewalls are categorized into 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 packets 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*. Using a stateful firewall to protect a private network, one can achieve finer access control by tracking the communication state between the private network and the outside Internet. For example, a stateful firewall can refuse to accept any packet from a remote host to a local host unless the local host has previously sent a packet to the remote host.

Although a variety of stateful firewall products have been available and deployed on the Internet for some time, such as Cisco PIX Firewalls [4], Cisco Reflexive ACLs [5], CheckPoint FireWall-1 [3] and Netfilter/IPTables [13], no model for specifying stateful firewalls exists. The lack of such a model constitutes a significant impediment for further development of stateful firewall technologies. First, without a model, it is difficult to conduct research on stateful firewalls. This explains why so little research on stateful firewalls has been done so far. In contrast, benefiting from the well-established rule based model of stateless firewalls, the research results for stateless firewalls have been numerous. People have known how to design stateless firewalls [2, 7, 8, 10] and how to analyze stateless firewalls [1, 6, 9, 11, 12, 16]. But the question of how to design and analyze stateful firewalls remains unanswered. Second, because there is no specification model for stateful firewalls, in existing stateful firewall products, state tracking func-

<sup>1</sup>Alex X. Liu is the corresponding author of this paper.

functionalities have been hard coded and different vendors hard code different state tracking functionalities. For example, the Cisco PIX Firewalls do not track the state for ICMP packets. Consequently, it is hard for the administrator of such a firewall to track the Ping [14] protocol. Last, without a specification model, it is difficult to analyze the properties of stateful firewalls. For example, it is difficult to analyze the properties of existing stateful firewalls because some of the functions of these firewalls are hard coded while others are specified by their administrators. All in all, a specification model for stateful firewalls is greatly needed.

In this paper, we propose the first stateful firewall model. In our firewall model, each firewall has a variable set called the *state* of the firewall, which is used to store some packets that the firewall has accepted previously and needs to remember in the near future. Each firewall consists of two sections: a *stateful section* and a *stateless section*. Each section consists of a sequence of rules. For every packet, the stateful section is used to check whether the state has a previous packet that may affect the fate of the current packet. To store this checking result, we assume that each packet has an additional field called the tag. The stateless section is used to decide the fate of each packet based on the information in the packet itself and its tag value.

Our stateful firewall model has the following favorable properties. First, it can express a variety of state tracking functionalities. Using a set of packets to record communication state provides a great deal of flexibility in expressing state tracking functionalities since the state of a communication protocol is characterized by packets. In a sense, our stateful firewall model captures the essence of communication states. Second, because we separate a firewall into a stateful section and a stateless section, we can inherit the existing rich results in designing and analyzing stateless firewalls because a stateless section alone is in fact a full-fledged stateless firewall. Third, our model is simple, easy to use, easy to understand, and easy to implement. Last, our model is a generalization of the current stateless firewall model. Although our model is intended to specify stateful firewalls, it can also be used to specify stateless firewalls, simply by leaving the stateful section empty and keeping the state empty.

This paper goes beyond proposing the stateful firewall model itself. A significant portion of this paper is devoted to analyzing the properties of stateful firewalls that are specified using our model. We outline a method for verifying that a firewall is truly stateful. The method is based on three properties of firewalls: conforming, grounded, and proper. We show that if a firewall satisfies these three properties, then the firewall is truly stateful.

The rest of this paper proceeds as follows. In Section 2, we introduce the syntax and semantics of our firewall model. In Section 3, we give two examples of stateful fire-

walls that are specified using our model. In Section 4, we discuss how to remove packets that are no longer needed from the state of a firewall. In Section 5, we study the issues related to firewall states. In Section 6, we present a method for verifying that a firewall is truly stateful. In Section 7, we give concluding remarks.

For simplicity, in the rest of this paper, we use “firewall” to mean “stateful firewall” unless otherwise specified.

## 2 Firewall Model

In this section, we introduce our firewall model through an example of a simple firewall that resides on the gateway router depicted in Figure 1. This router has two interfaces: interface 0, which connects the router to the outside Internet, and interface 1, which connects the router to a private network.

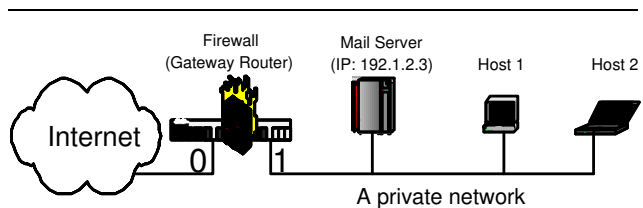


Figure 1. A firewall for a private network

This firewall tracks the Ping protocol (Packet Internet Groper Protocol) [14] to counter “smurf” attacks. The Ping protocol is used by a host to determine whether another host is up. When a host A wants to test whether a host B is up, A sends to B a series of ICMP (Internet Control Message Protocol) ping (i.e., echo request) packets. All of these ping packets have the same ID but different sequence numbers. When B receives from A a ping packet with ID  $x$  and sequence number  $y$ , B sends back to A a pong (i.e., echo reply) packet with the same ID  $x$  and the same sequence number  $y$ . The “smurf” attack, a type of Denial of Service attack, works as follows. An attacker sends a ping packet, whose source IP address has been forged to be the IP address of a victim host, to the broadcast address of a subnetwork. Subsequently, every host on the subnetwork will send a pong packet to the victim host.

One way to counter “smurf” attacks for a private network is to use a firewall to discard every incoming pong packet unless the packet corresponds to a previous ping packet sent from the private network. Suppose that we want to configure the firewall in Figure 1 in such a fashion. When a pong packet arrives, the firewall needs to check whether it has

**Stateful Section:**

$$R_1 : I \in \{0\} \wedge P \in \{icmp\} \wedge T \in \{pong\} \wedge S = D' \wedge D = S' \wedge ID = ID' \wedge SN = SN' \rightarrow tag := 1$$

**Stateless Section:**

$$r_1 : I \in \{1\} \wedge P \in \{icmp\} \wedge T \in \{ping\} \wedge tag \in all \rightarrow accept; insert$$

$$r_2 : I \in \{1\} \wedge P \in all \wedge T \in all \wedge tag \in all \rightarrow accept$$

$$r_3 : I \in \{0\} \wedge P \in \{icmp\} \wedge T \in \{pong\} \wedge tag \in \{1\} \rightarrow accept$$

$$r_4 : I \in \{0\} \wedge P \in \{icmp\} \wedge T \in \{pong\} \wedge tag \in \{0\} \rightarrow discard$$

$$r_5 : I \in \{0\} \wedge P \in all \wedge T \in all \wedge tag \in all \rightarrow accept$$

**Figure 2. Tracking the Ping protocol**

seen the corresponding ping packet. This requires the firewall to remember the ping packets sent from the private network to the outside. In our firewall model, each firewall has a variable set called the state. The state of a firewall contains the packets that the firewall has accepted previously and needs to remember in the near future. In this firewall example, we store in the state of the firewall the ping packets that are sent from the private network to the outside Internet.

In our firewall model, each firewall consists of two sections: a *stateful section* and a *stateless section*. The stateful section is used to check each packet against the state. The stateless section is used to decide the fate of a packet after the packet has been checked against the state. To store the checking result of the stateful section for each packet, we assume that each packet has an additional field called the *tag*. The value of the tag field of a packet is an integer, whose initial value is zero. The domain of this tag field depends on how many possible tag values that a firewall needs. In the above firewall example, when a packet arrives, if it is a pong packet and its corresponding ping packet is in the state, then the tag field of the packet is assigned 1; otherwise the tag field of the packet retains the initial value of 0. Therefore, the domain of the tag field in this example is  $[0, 1]$ .

We define a *packet* over the fields  $F_1, \dots, F_d$  to be a  $d$ -tuple  $(p_1, \dots, p_d)$  where each  $p_i$  is in the domain  $D(F_i)$  of field  $F_i$ , and each  $D(F_i)$  is an interval of nonnegative integers. For example, the domain of the source address in an IP packet is  $[0, 2^{32})$ .

The stateful section of a firewall consists a sequence of rules where each rule is called a *stateful rule*. A stateful rule is of the form

$$P(F_1, \dots, F_d, F'_1, \dots, F'_d, tag') \rightarrow tag := x$$

where  $P(F_1, \dots, F_d, F'_1, \dots, F'_d, tag')$  is a predicate over  $F_1, \dots, F_d, F'_1, \dots, F'_d, tag'$ . A packet  $(p_1, \dots, p_d)$  matches the above rule iff (if and only if) there exists a packet  $(p'_1, \dots, p'_d)$  with tag value  $t'$  in the state of the firewall such that  $P(p_1, \dots, p_d, p'_1, \dots, p'_d, t')$  is true. The meaning of this stateful rule is as follows. Given a packet  $p$

such that  $p$  matches this stateful rule (but  $p$  does not match any other stateful rules listed before this rule), the tag value of this packet  $p$  is changed from its initial value 0 to the new value  $x$ .

The stateless section of a firewall also consists a sequence of rules where each rule is called a *stateless rule*. A stateless rule is of the form

$$F_1 \in S_1 \wedge \dots \wedge F_d \in S_d \wedge tag \in S_t \rightarrow \langle decision \rangle$$

where each  $S_i$  is a nonempty subset of the domain of  $F_i$  for  $0 \leq i \leq d$ , and  $S_t$  is a nonempty subset of the domain of the tag field, and the  $\langle decision \rangle$  is “accept”, or “accept; insert”, or “discard”. For each  $i$  ( $1 \leq i \leq d$ ), if  $S_i = D(F_i)$ , we can replace  $F_i \in S_i$  by  $F_i \in all$ , or remove the conjunct  $F_i \in D(F_i)$  from the rule. A packet  $(p_1, \dots, p_d)$  with tag value  $t$  matches the above rule iff the condition  $p_1 \in S_1 \wedge \dots \wedge p_d \in S_d \wedge t \in S_t$  holds. The meaning of this stateless rule is as follows. Given a packet  $p$  such that  $p$  matches this stateless rule (but  $p$  does not match any other stateless rules listed before this rule), the *decision* for this packet is executed. If the *decision* is “accept”, then the packet  $p$  is allowed to proceed to its destination. If the *decision* is “accept; insert”, then the packet  $p$  is allowed to proceed to its destination and additionally packet  $p$  (together with its tag value) is inserted into the state of the firewall. If the *decision* is “discard”, then the packet  $p$  is discarded by the firewall.

In the firewall example in Figure 1, we assume that each packet has the following seven fields. For simplicity, in this paper we assume that each packet has a field containing the identification of the network interface on which a packet arrives. Figure 2 shows this firewall specified using our model.

name	meaning	domain
I	Interface	$[0, 1]$
S	Source IP address	$[0, 2^{32})$
D	Destination IP address	$[0, 2^{32})$
P	Protocol Type	$\{tcp, udp, icmp\}$
T	echo packet type	$\{ping, pong\}$
ID	echo packet ID	$[0, 2^{16})$
SN	echo packet sequence number	$[0, 2^{16})$

In this firewall example, the stateful section consists of one rule:  $I \in \{0\} \wedge P \in \{icmp\} \wedge T \in \{pong\} \wedge S = D' \wedge D = S' \wedge ID = ID' \wedge SN = SN' \rightarrow tag := 1$ . The meaning of this rule is as follows: if a packet  $p$  is an incoming pong packet (indicated by  $I \in \{0\} \wedge P \in \{icmp\} \wedge T \in \{pong\}$ ), and there exists a packet  $p'$  in the state such that the following four conditions hold:

1. the source address of  $p$  equals the destination address of  $p'$  (denoted  $S = D'$ ),
2. the destination address of  $p$  equals the source address of  $p'$  (denoted  $D = S'$ ),
3. the ID of  $p$  equals the ID of  $p'$  (denoted  $ID = ID'$ ),
4. the sequence number of  $p$  equals the sequence number of  $p'$  (denoted  $SN = SN'$ ),

then the tag field of packet  $p$  is assigned 1; otherwise the tag field of packet  $p$  retains its initial value 0. In this firewall example, the stateless section consists of five rules whose function is to map every packet with a certain tag value to one of predefined decisions. Note that the meaning of the rule  $r_1$  is as follows. Given a packet over the seven fields (namely I, S, D, P, T, ID, SN), if the packet matches rule  $r_1$ , then the firewall allows this packet to proceed to its destination and additionally the packet (which is a tuple over the seven fields) together with its tag value is inserted into the state of the firewall.

Note that when a firewall inserts a packet  $(p_1, \dots, p_d)$  with a tag value into the state of the firewall, the firewall may not need to insert all the  $d$  fields of the packet. For example, considering the above firewall example in Figure 2, its stateful section consists of one rule  $I \in \{0\} \wedge P \in \{icmp\} \wedge T \in \{pong\} \wedge S = D' \wedge D = S' \wedge ID = ID' \wedge SN = SN' \rightarrow tag := 1$ . This rule only examines four fields of the packets in the state: S, D, ID and SN. Therefore, instead of inserting a packet of all the seven fields (namely I, S, D, P, T, ID, SN) together with the tag value of the packet into the state, we only need to insert a tuple over the above four fields of S, D, ID and SN.

Two stateless rules *conflict* iff there exists at least one packet that matches both rules and the two rules have different decisions. For example, rule  $r_1$  and rule  $r_2$  in the stateless section of the firewall in Figure 2 conflict. Two stateful rules *conflict* iff in a reachable state of the firewall there exists at least one packet that matches both rules and the two rules have different decisions. In our firewall model, for both the stateful section and the stateless section, we follow the convention that stateless firewalls use to resolve conflicts: a packet is mapped to the decision of the first rule that the packet matches.

A set of rules is *comprehensive* iff for any packet there is at least one rule in the set that the packet matches. The set

of all the rules in the stateless section of a firewall must be comprehensive because each packet needs to be mapped to a decision. Note that the set of all the rules in the stateful section of a firewall does not need to be comprehensive. This is because the function of a stateful section is to assign nonzero values to the tag fields of some packets, but not all packets.

Given a packet to a firewall specified using our model, Figure 3 describes how the firewall processes this packet.

---

#### Step 1. Checking in the stateful section:

**If**  $P(F_1, \dots, F_d, F'_1, \dots, F'_d, tag') \rightarrow tag := x$   
 is the first stateful rule that the given packet matches  
**then** the tag of the packet is assigned value  $x$ ;  
**else** the tag of the packet retains value 0.

#### Step 2. Checking in the stateless section:

**If**  $F_1 \in S_1 \wedge \dots \wedge F_d \in S_d \wedge tag \in \langle decision \rangle$   
 is the first stateless rule that the given packet matches  
**then** the  $\langle decision \rangle$  is executed for the packet.

---

### Figure 3. Processing a given packet

---

By separating a firewall into a stateful section and a stateless section, we can inherit existing research results of stateless firewalls because a stateless section alone is in fact a full-fledged stateless firewall. For example, existing stateless firewall design methods [2,7,8,8], and stateless firewall analysis methods [1,6,9,11,12,16], are still applicable to the design and analysis of a stateless section. In addition, existing packet classification algorithms for stateless firewalls can still be used to map a packet with a certain tag value to the first rule that the packet matches in the stateless section.

## 3 Firewall Examples

In this section, we show two more examples of stateful firewalls.

### 3.1 Example I: Tracking Outgoing Packets

Suppose that the requirements for the firewall in Figure 1 are as follows:

1. Any packet from the outside malicious domain 192.168.0.0/16 should be discarded.
2. The mail server, with IP address 192.1.2.3, should be able to send and receive emails, but non-email traffic is not allowed to proceed to the mail server.

**Stateful Section:**

$$R_1 : I \in \{0\} \wedge S = D' \wedge D = S' \wedge SP = DP' \wedge DP = SP' \wedge P = P' \rightarrow tag := 1$$

**Stateless Section:**

$r_1 : I \in \{1\} \wedge S \in \{192.1.2.3\}$	$\wedge D \in all$	$\wedge DP \in all$	$\wedge P \in all$	$\wedge tag \in all \rightarrow accept$
$r_2 : I \in \{1\} \wedge S \in all$	$\wedge D \in all$	$\wedge DP \in all$	$\wedge P \in all$	$\wedge tag \in all \rightarrow accept; insert$
$r_3 : I \in \{0\} \wedge S \in [192.168.0.0, 192.168.255.255]$	$\wedge D \in all$	$\wedge DP \in all$	$\wedge P \in all$	$\wedge tag \in all \rightarrow discard$
$r_4 : I \in \{0\} \wedge S \in all$	$\wedge D \in \{192.1.2.3\}$	$\wedge DP \in \{25\}$	$\wedge P \in \{tcp\}$	$\wedge tag \in all \rightarrow accept$
$r_5 : I \in \{0\} \wedge S \in all$	$\wedge D \in \{192.1.2.3\}$	$\wedge DP \in all$	$\wedge P \in all$	$\wedge tag \in all \rightarrow discard$
$r_6 : I \in \{0\} \wedge S \in all$	$\wedge D \in all$	$\wedge DP \in all$	$\wedge P \in all$	$\wedge tag \in \{1\} \rightarrow accept$
$r_7 : I \in \{0\} \wedge S \in all$	$\wedge D \in all$	$\wedge DP \in all$	$\wedge P \in all$	$\wedge tag \in \{0\} \rightarrow discard$

**Figure 4. Tracking outgoing packets**

3. Any packet from a remote host to a local host, which is not the mail server, is discarded unless the local host has already sent a packet to the remote host earlier. In other words, the communication between a local host and a remote host can only be initiated by the local host.

In this example, we assume that each packet has six fields. Four of them have been discussed earlier: I (interface), S (source IP address), D (destination IP address), and P (protocol type). The remaining two are as follows:

name	meaning	domain
SP	Source Port	$[0, 2^{16})$
DP	Destination Port	$[0, 2^{16})$

Figure 4 shows the specification of this firewall. Its stateful section consists of one rule  $I \in \{0\} \wedge S = D' \wedge D = S' \wedge SP = DP' \wedge DP = SP' \wedge P = P' \rightarrow tag := 1$ . The meaning of this rule is as follows: if a packet  $p$  is an incoming packet (denoted  $I \in \{0\}$ ), and there exists a packet  $p'$  in the state such that the following five conditions hold:

1. the source address of  $p$  equals the destination address of  $p'$  (denoted  $S = D'$ ),
2. the destination address of  $p$  equals the source address of  $p'$  (denoted  $D = S'$ ),
3. the source port number of  $p$  equals the destination port number of  $p'$  (denoted  $SP = DP'$ ),
4. the destination port number of  $p$  equals the source port number of  $p'$  (denoted  $DP = SP'$ ),
5. the protocol type of  $p$  equals that of  $p'$  (denoted  $P = P'$ ),

then the tag field of packet  $p$  is assigned 1; otherwise the tag field of packet  $p$  retains value 0.

The stateless section of this firewall consists of seven rules from  $r_1$  to  $r_7$ . Note that the meaning of rule  $r_2$  is as follows. Any outgoing packet from a local host other than

the mail server is allowed to proceed to its destination, and additionally this packet, which is a tuple of the six fields (namely I, S, D, P, SP, DP), together with its tag value, is inserted into the state of the firewall. Since the stateful section of this firewall only examines the five fields (namely S, D, P, SP, and DP) of the packets in the state of this firewall, we only need to insert these five fields of a packet into the state.

**3.2 Example II: Tracking FTP Protocol**

In this section, we show an example of a firewall that tracks the FTP protocol. File Transfer Protocol (FTP) [15] is an application protocol that is used to transfer files between two hosts. We assume that the firewall in Figure 1 allows any local host to initiate an FTP connection to a remote host, but any remote host cannot initiate an FTP connection to a local host. For simplicity, we assume that non-FTP traffic is discarded.

What complicates the tracking of FTP is its dual-connection feature. FTP uses two TCP connections to transfer files between two hosts: a control connection and a data connection. When a client wants to connect to a remote FTP server, the client uses one of its available port numbers, say  $x$ , to connect to the server on the well-known port 21. This connection, between the client's port  $x$  and the server's port 21, is called the control connection. FTP uses the control connection to transfer FTP commands such as CWD (change working directory) and PORT (specify the port number that the client will use for the data connection). After this control connection is built between the client and the server, the client sends a PORT command with a value  $y$ , where  $y$  is an available port on the client, to the server via this control connection. After this PORT command is received, the server uses its well-known port 20 to connect back to the port  $y$  of the client. This connection, between the client's port  $y$  and the server's port 20, is called the data connection. Note that the control connection is initiated by the FTP client and the data connection is initiated by the FTP server. This dual-connection feature of the FTP proto-

**Stateful Section:**

$$R_1 : I \in \{0\} \wedge SP \in \{21\} \wedge P \in \{tcp\} \wedge S = D' \wedge D = S' \wedge DP = SP' \wedge DP' \in \{21\} \rightarrow tag := 1$$

$$R_2 : I \in \{0\} \wedge SP \in \{20\} \wedge P \in \{tcp\} \wedge S = D' \wedge D = S' \wedge T' = 1 \wedge DP = A' \wedge DP' \in \{21\} \rightarrow tag := 1$$

$$R_3 : I \in \{1\} \wedge DP \in \{20\} \wedge P \in \{tcp\} \wedge S = D' \wedge D = S' \wedge SP = DP' \wedge SP' \in \{20\} \rightarrow tag := 1$$

**Stateless Section:**

$$r_1 : I \in \{1\} \wedge SP \in all \wedge DP \in \{21\} \wedge P \in \{tcp\} \wedge tag \in all \rightarrow accept; insert$$

$$r_2 : I \in \{1\} \wedge SP \in all \wedge DP \in \{20\} \wedge P \in \{tcp\} \wedge tag \in \{1\} \rightarrow accept$$

$$r_3 : I \in \{1\} \wedge SP \in all \wedge DP \in all \wedge P \in all \wedge tag \in all \rightarrow discard$$

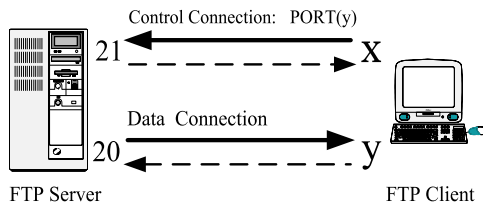
$$r_4 : I \in \{0\} \wedge SP \in \{20\} \wedge DP \in all \wedge P \in \{tcp\} \wedge tag \in \{1\} \rightarrow accept; insert$$

$$r_5 : I \in \{0\} \wedge SP \in \{21\} \wedge DP \in all \wedge P \in \{tcp\} \wedge tag \in \{1\} \rightarrow accept$$

$$r_6 : I \in \{0\} \wedge SP \in all \wedge DP \in all \wedge P \in all \wedge tag \in all \rightarrow discard$$

**Figure 5. Tracking the FTP protocol**

col is illustrated in Figure 6.



**Figure 6. FTP Protocol**

This firewall is specified in Figure 5. In this example, we assume that each packet has eight fields. Six of them have been discussed earlier: I (interface), S (source IP address), D (destination IP address), P (protocol type), SP (source port) and DP (destination port). The remaining two are as follows:

name	meaning	domain
T	Application Type	[0, 1]
A	Application Data	[0, 2 <sup>16</sup> )

For a packet, if the value of its field T is 1, then the value of its field A is the port number of a port command; otherwise field A contains another FTP control command.

In this example, the firewall only possibly accepts the following four types of packets: outgoing TCP packets to port 21, incoming TCP packets from port 21, incoming TCP packets from port 20, and outgoing TCP packets to port 20. Next we discuss each of these four types of packets.

1. Outgoing TCP packets to port 21: Any packet  $p$  of this type is accepted and inserted into the state. See rule  $r_1$  in Figure 5.
2. Incoming TCP packets from port 21: A packet  $p$  of this type is accepted iff there exists a packet  $p'$  in the state such that  $p$ 's source IP address equals  $p'$ 's destination IP address,  $p$ 's destination IP address equals  $p'$ 's

source IP address,  $p$ 's destination port number equals  $p'$ 's source port number, and  $p$ 's destination port number is 21. See the three rules  $r_1$ ,  $R_1$ , and  $r_5$  in Figure 5.

3. Incoming TCP packets from port 20: A packet  $p$  of this type is accepted iff there exists a packet  $p'$  in the state such that  $p$ 's source IP address equals  $p'$ 's destination IP address,  $p$ 's destination IP address equals  $p'$ 's source IP address,  $p$ 's destination port number is 21,  $p'$  contains a PORT command and  $p$ 's destination port equals the port number in this PORT command of  $p'$ . See the three rules  $r_1$ ,  $R_2$ , and  $r_4$  in Figure 5.
4. Outgoing TCP packets to port 20: A packet  $p$  of this type is accepted iff there exists a packet  $p'$  in the state such that  $p$ 's source IP address equals  $p'$ 's destination IP address,  $p$ 's destination IP address equals  $p'$ 's source IP address,  $p$ 's source port number equals  $p'$ 's destination port number, and  $p$ 's source port number is 20. See the three rules  $r_4$ ,  $R_3$ , and  $r_2$  in Figure 5.

**4 Removing Packets from Firewall State**

After a packet is inserted into the state of a firewall, the packet should be removed when it is no longer needed, otherwise security could be breached. We show this point by the firewall example in Figure 2 that tracks the Ping protocol. Suppose a local host named A sends a ping packet to a remote host named B. According to the specification of this firewall in Figure 2, this ping packet is inserted into the state of this firewall. When the corresponding pong packet comes back from host B, it is accepted by the firewall because of the stored ping packet, and additionally this stored ping packet should be removed from the state of the firewall. Otherwise, an attacker could replay the pong packet for an unlimited number of times and each of the replayed pong packets would be incorrectly allowed to proceed to the victim host A.

---

**Stateful Section:**
$$R_1 : I \in \{0\} \wedge P \in \{icmp\} \wedge T \in \{pong\} \wedge S = D' \wedge D = S' \wedge ID = ID' \wedge SN = SN' \rightarrow tag := 1$$
**Stateless Section:**
$$r_1 : I \in \{1\} \wedge P \in \{icmp\} \wedge T \in \{ping\} \wedge tag \in all \rightarrow accept; insert(10)$$
$$r_2 : I \in \{1\} \wedge P \in all \wedge T \in all \wedge tag \in all \rightarrow accept$$
$$r_3 : I \in \{0\} \wedge P \in \{icmp\} \wedge T \in \{pong\} \wedge tag \in \{1\} \rightarrow accept; remove$$
$$r_4 : I \in \{0\} \wedge P \in \{icmp\} \wedge T \in \{pong\} \wedge tag \in \{0\} \rightarrow discard$$
$$r_5 : I \in \{0\} \wedge P \in all \wedge T \in all \wedge tag \in all \rightarrow accept$$

---

**Figure 7. Tracking the Ping protocol (with packets removal)**

---

A new command, “*remove*”, is used to remove the packets that are no longer needed from the state of a firewall. Therefore, there are two more possible decisions that a stateless rule may use: “*accept; remove*” and “*accept; insert; remove*”, in addition to the three decisions (namely “*accept*”, “*accept; insert*”, and “*discard*”) that we have seen earlier. The meaning of a stateless rule with decision “*accept; remove*” is as follows. Given a packet  $p$ , if  $p$  matches this rule (but  $p$  does not match any stateless rule listed before this rule), then  $p$  is accepted. Moreover, if the state has a packet  $p'$  such that  $p$  satisfies the predicate of the first stateful rule that  $p$  matches using  $p'$ , then packet  $p'$  is removed from the state. Similarly for the meaning of a rule with decision “*accept; insert; remove*”. Consider the example of the firewall in Figure 2 that tracks the Ping protocol. When a ping packet is sent from a local host to a remote host, the ping packet is inserted into the state of the firewall by the stateless rule  $r_1 : I \in \{1\} \wedge P \in \{icmp\} \wedge T \in \{ping\} \wedge tag \in all \rightarrow accept; insert$ . When the corresponding pong packet comes back from the remote host, it is accepted by the stateless rule  $r_3$  and it should also trigger the removal of the stored ping packet. Therefore, a “*remove*” command should be added to rule  $r_3$ . In other words, rule  $r_3$  should be  $I \in \{0\} \wedge P \in \{icmp\} \wedge T \in \{pong\} \wedge tag \in \{1\} \rightarrow accept; remove$ .

Usually the packet that *initiates* the “conversation” between two hosts is stored in the state of a firewall, and the packet that *terminates* the “conversation” triggers the removal of the stored packet. Examples of the packets that can initiate a conversation are ping packets and TCP SYN packets. Examples of the packets that can terminate a conversation are pong packets and TCP FIN packets.

To remove the packets that are no longer needed in the state of a firewall, we cannot only rely on some packets to trigger the removal for two reasons. First, these triggering packets may get lost on their way. Second, the processes that are supposed to send triggering packets may abnormally terminate before sending out the triggering packets. In either case, the packets that should be removed still remain in the state. To deal with these two cases, when a packet is inserted into the state of a firewall, it is assigned a TTL (Time To Live) value. The TTL value of every packet

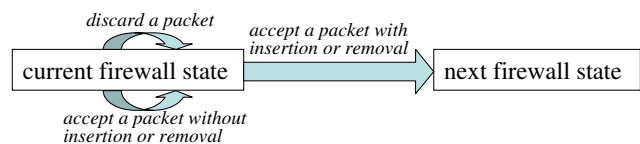
in the state decreases as time goes by. When the TTL value of a packet expires, the packet is automatically removed from the state.

Different packets may need different TTL values. Therefore, the “*insert*” command has a parameter  $t$ , which is the TTL value for the packet to be inserted into the state of a firewall. The meaning of a stateless rule with decision “*accept; insert( $t$ )*” is as follows. Given a packet  $p$  such that  $p$  matches this rule (but  $p$  does not match any stateless rule listed before this rule), provided that  $p$  is not an element of the state, then  $p$  is inserted into the state with TTL value  $t$ . On the other hand, if  $p$  already exists in the state, then the TTL value of  $p$  in the state is reassigned the value  $t$ .

Figure 7 shows the complete firewall for tracking the Ping protocol after we incorporate the TTL extension to the “*insert*” command in rule  $r_1$  and add the “*remove*” command to rule  $r_3$ . In this example, the TTL value in the “*insert*” command is 10 seconds.

## 5 Firewall States

Recall that each firewall has a variable set named the state of the firewall. Initially, the state of a firewall is empty. The transition between two states of a firewall is illustrated in Figure 8.

**Figure 8. Firewall state transition**

---

A *history* of a firewall is a finite sequence  $S.1, p.1, S.2, p.2, \dots, S.n$  such that the following three conditions hold.

- i. Each  $S.i$  is a state of the firewall. Note that  $S.1$  is the initial state of the firewall, which is an empty set.

- ii. Each  $p.i$  is a packet.
- iii. For every  $i$  ( $1 \leq i < n$ ), if the firewall is in state  $S.i$  and receives packet  $p.i$ , then the firewall accepts  $p.i$  and the state of the firewall becomes  $S.(i + 1)$ .

Note that in a firewall history,  $S.1, p.1, S.2, p.2, \dots, S.n$ , for every  $i$  ( $1 \leq i < n$ ), we have

$$\begin{cases} S.i \neq S.(i + 1) & \text{if in state } S.i, p.i \text{ is accepted, and} \\ & p.i \text{ is inserted into the state or } p.i \\ & \text{triggers the removal of a packet;} \\ S.i = S.(i + 1) & \text{otherwise} \end{cases}$$

A state of a firewall is called a *reachable state* iff the state is in a history of the firewall.

## 5.1 Truly Stateful and Truly Stateless Firewalls

Before we define truly stateful firewalls, we first define two important concepts associated with each firewall: the accepted set and the acceptable set.

A packet is called an *accepted packet* of a firewall iff the packet can be accepted in every reachable state of the firewall. The set of all accepted packets of a firewall is called the *accepted set* of the firewall. For a firewall  $f$ , we use  $f.a$  to denote its accepted set.

A packet is called an *acceptable packet* of a firewall iff the packet can be accepted in some (possibly every) reachable state of the firewall. The set of all acceptable packets of a firewall is called the *acceptable set* of the firewall. For a firewall  $f$ , we use  $f.b$  to denote its acceptable set.

Note that a stateless firewall can also be specified using our model. When we specify a stateless firewall, we leave the stateful section empty and specify no “insert” command in any rule in the stateless section. In this case, the state of the firewall remains empty and the firewall is therefore stateless. For a stateless firewall  $f$ , we use  $f.a$  to denote the set of all accepted packets of  $f$  and use  $f.b$  to denote the set of all acceptable packets of  $f$ . From the definition of stateful firewalls and stateless firewalls, we have the following theorem:

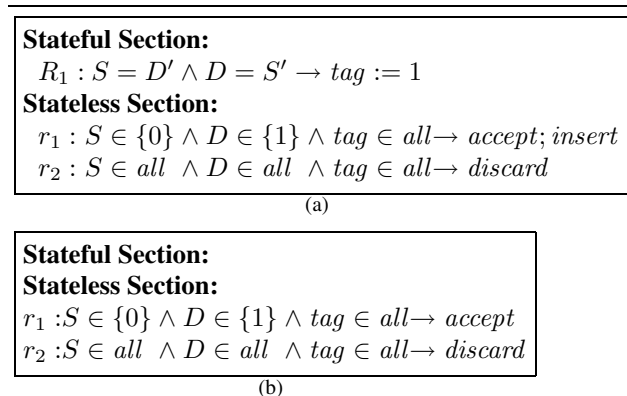
**Theorem 1** Let  $f$  be a firewall.

- i.  $f.a$  is a subset of  $f.b$  ( $f.a \subseteq f.b$ )
- ii. If  $f$  is stateless, then  $f.a = f.b$ .

A firewall  $f$  is *truly stateful* iff  $f.a$  is a proper subset of  $f.b$ ; i.e.,  $f.a \subset f.b$ . A firewall  $f$  is *truly stateless* iff  $f.a = f.b$ . Clearly, a stateless firewall is truly stateless, but a stateful firewall can either be truly stateful or be truly stateless. A stateful firewall that is truly stateless can be simplified, without changing its function, by making its

stateful section empty and removing the “insert” command from every rule in its stateless section.

As an example, consider the firewall in Figure 9(a). This firewall accepts each packet where  $S \in \{0\}$  and  $D \in \{1\}$  in each reachable state, and discards all other packets in each reachable state. Thus, this firewall is truly stateless (although it is syntactically stateful). Therefore, this firewall can be simplified as shown in Figure 9(b).



**Figure 9. A truly stateless firewall and its simplified version**

## 5.2 Stateless Derivatives

It is important that if a firewall designer designs a stateful firewall  $f$ , then he should verify that  $f$  is truly stateful. This is because if  $f$  is truly stateless, then  $f$  can be simplified into a stateless firewall. In this section, we identify a sufficient condition for verifying that a firewall is truly stateful. But first we introduce the concept of a stateless derivative of a firewall.

The *stateless derivative* of a firewall  $f$  is the firewall obtained after making the stateful section of  $f$  empty and removing the “insert” command from every rule in the stateless section of  $f$ . For example, Figure 9(b) shows the stateless derivative of the firewall in Figure 9(a).

The relationship between a firewall and its stateless derivative is stated in the following theorem, whose proof is presented in the appendix.

**Theorem 2** Let  $f$  be a firewall and  $g$  be its stateless derivative,

- i.  $f.a \subseteq g.a$
- ii.  $g.a = g.b$
- iii.  $g.b \subseteq f.b$

Recall that a firewall  $f$  is truly stateful iff  $f.a \subset f.b$ . By Theorem 2, one way to prove that a firewall  $f$ , whose stateless derivative is denoted  $g$ , is truly stateful is to prove that the following two conditions hold:

- i.  $f.a = g.a$ ;
- ii.  $g.b \subset f.b$

We call firewalls that satisfy the first condition *conforming firewalls*; and call firewalls that satisfy the second condition *proper firewalls*.

## 6 Firewall Properties

In this section, we discuss how to verify that a firewall is conforming or proper.

### 6.1 Conforming Firewalls

Before we give a theorem on how to verify that a firewall is conforming, we need to introduce the two concepts of complementary rules and accepting rules.

Let rule  $r$ , that appears in the stateless section of some firewall, be of the form

$$F_1 \in S_1 \wedge \dots \wedge F_d \in S_d \wedge tag \in S_t \rightarrow \langle decision \rangle$$

Rule  $r$  is *complementary* iff the set  $S_t$  does not contain the value 0. Rule  $r$  is *accepting* iff the  $\langle decision \rangle$  of  $r$  contains the command “accept”.

The following theorem can be used to verify that a firewall is conforming. The proof of this theorem is presented in the appendix.

**Theorem 3** A firewall  $f$  is conforming if every complementary rule in the stateless section of  $f$  is accepting.

As an example, we use Theorem 3 to prove that the firewall in Figure 2 is conforming as follows. This firewall has only one complementary rule, which is rule  $r_3 : I \in \{0\} \wedge P \in \{icmp\} \wedge T \in \{pong\} \wedge tag \in \{1\} \rightarrow accept$ . And rule  $r_3$  is an accepting rule. Therefore, this firewall is conforming.

### 6.2 Proper Firewalls

Based on our experience in designing firewalls, most firewalls are conforming. By Theorem 2, a conforming firewall is truly stateful iff it is proper. Next we discuss how to verify that a firewall is proper.

A firewall is proper iff its acceptable set is a proper superset of the acceptable set of its stateless derivative. For a firewall to be proper, we first need to make sure that its state does not remain empty forever. We call such firewalls

grounded. More precisely, grounded firewalls are defined as follows.

Let  $f$  be a firewall whose stateless section consists of  $n$  rules  $r_1, r_2, \dots, r_n$ :

$$\begin{aligned} r_1 : P_1 &\rightarrow \langle decision_1 \rangle \\ r_2 : P_2 &\rightarrow \langle decision_2 \rangle \\ &\dots \\ r_n : P_n &\rightarrow \langle decision_n \rangle \end{aligned}$$

A rule  $r_k$ , where  $1 \leq k \leq n$ , is called a *ground rule* iff the following three conditions hold:

- i.  $r_k$  is non-complementary;
- ii.  $\langle decision_k \rangle$  is “accept; insert” or “accept; insert; remove”;
- iii.  $\sim P_1 \wedge \sim P_2 \wedge \dots \wedge \sim P_{k-1} \wedge P_k$  is satisfiable by at least one packet.

A firewall is *grounded* iff it has a ground rule.

A ground rule of a grounded firewall guarantees that in the initial state of the firewall, there exists at least one packet that can be accepted and inserted into the state of the firewall.

To test whether a firewall is grounded, we can go through each rule and test whether it is a ground rule according to the above definition. Once we find a ground rule in a firewall, we know that the firewall is grounded. For example, consider the firewall in Figure 4. The second rule in the stateless section of this firewall is a ground rule because (1) it is non-complementary; (2) its decision is “accept; insert”; and (3)  $\sim P_1 \wedge P_2$  is satisfiable. Note that  $\sim P_1 \wedge P_2 = I \in \{1\} \wedge S \in [0, \alpha - 1] \cup [\alpha + 1, 2^{32}] \wedge D \in all \wedge DP \in all \wedge P \in all \wedge tag \in all$ , where  $\alpha$  denotes the integer formed by the four bytes of the IP address 192.1.2.3. Therefore, this firewall is grounded.

For a grounded firewall to be proper, we need to show that there exists at least one packet, denoted  $p$ , such that (1)  $p$  is discarded by the stateless derivative of the firewall, (2)  $p$  can be accepted by the firewall in some state. As an example, we show how to verify that a grounded firewall is proper by examining the firewall example in Figure 2 as follows. For this firewall, we assume that each packet consists of the fields of I, S, D, P, T, ID, and SN. Consider the two packets  $p'$  and  $p$  in the following table. It is straightforward to verify that packet  $p$  is discarded by the stateless derivative of this firewall (because of rule  $r_4$ ). At any state of this firewall,  $p'$  is accepted and inserted into the state because of rule  $r_1$ . Because of the stateful rule  $R_1$  and the stateless rule  $r_3$ , as long as  $p'$  is in the state, packet  $p$  is accepted. Therefore, this firewall is proper.

	I	S	D	P	T	ID	SN
$p'$	1	192.1.2.4	192.32.1.2	icmp	ping	10	200
$p$	0	192.32.1.2	192.1.2.4	icmp	pong	10	200

## 7 Conclusions and Future Work

We consider our paper to be the first step in designing and analyzing stateful firewalls. Our contributions in this paper are two-fold. First, we propose the first model for specifying stateful firewalls, which henceforth opens doors to new research on stateful firewalls. Our model of stateful firewalls has several favorable properties. It is simple but can express a variety of state tracking functionalities. It allows us to inherit the rich results in stateless firewall design and analysis. Moreover, it provides backward compatibility such that a stateless firewall can also be specified using our model. Second, we present methods for analyzing stateful firewalls that are specified using our model. We outline a method for verifying whether a firewall is truly stateful.

Several issues related to our stateful firewall model are left for future work, for example, how to efficiently implement this model and how to use it to analyze other properties of stateful firewalls.

## References

- [1] E. Al-Shaer and H. Hamed. Discovery of policy anomalies in distributed firewalls. In *IEEE INFOCOM'04*, pages 2605–2616, March 2004.
- [2] Y. Bartal, A. J. Mayer, K. Nissim, and A. Wool. Firmato: A novel firewall management toolkit. *Technical Report EES2003-1, Dept. of Electrical Engineering Systems, Tel Aviv University*, 2003.
- [3] CheckPoint FireWall-1. <http://www.checkpoint.com/>. Date of access: March 25, 2005.
- [4] Cisco PIX Firewalls. <http://www.cisco.com/>. Date of access: March 25, 2005.
- [5] Cisco Reflexive ACLs. <http://www.cisco.com/>. Date of access: March 25, 2005.
- [6] M. Frantzen, F. Kerschbaum, E. Schultz, and S. Fahmy. A framework for understanding vulnerabilities in firewalls using a dataflow model of firewall internals. *Computers and Security*, 20(3):263–270, 2001.
- [7] M. G. Gouda and A. X. Liu. Firewall design: consistency, completeness and compactness. In *Proc. of the 24th IEEE International Conference on Distributed Computing Systems (ICDCS'04)*, pages 320–327.
- [8] J. D. Guttman. Filtering postures: Local enforcement for global policies. In *Proc. of IEEE Symp. on Security and Privacy*, pages 120–129, 1997.
- [9] S. Kamara, S. Fahmy, E. Schultz, F. Kerschbaum, and M. Frantzen. Analysis of vulnerabilities in internet firewalls. *Computers and Security*, 22(3):214–232, 2003.
- [10] A. X. Liu and M. G. Gouda. Diverse firewall design. In *Proc. of the International Conference on Dependable Systems and Networks (DSN'04)*, pages 595–604, June 2004.
- [11] A. X. Liu, M. G. Gouda, H. H. Ma, and A. H. Ngu. Firewall queries. In *Proc. of the 8th International Conference on Principles of Distributed Systems (OPODIS-04)*, pages 124–139, December 2004.
- [12] A. Mayer, A. Wool, and E. Ziskind. Fang: A firewall analysis engine. In *Proc. of IEEE Symp. on Security and Privacy*, pages 177–187, 2000.
- [13] Netfilter/IPTables. <http://www.netfilter.org/>. Date of access: March 25, 2005.
- [14] J. Postel. Internet control message protocol. *RFC 792*, 1981.
- [15] J. Postel and J. Reynolds. File transfer protocol. *RFC 959*, 1985.
- [16] A. Wool. Architecting the lumeta firewall analyzer. In *Proc. of the 10th USENIX Security Symposium*, pages 85–97, August 2001.

## Appendix

### A Proof of Theorem 2

Proof of i: This assertion holds because  $f.a$  is the set of all the packets where each packet can be accepted in every reachable state of  $f$  and  $g.a$  is the set of all the packets that can be accepted in the initial state of  $f$ .

Proof of ii: Note that  $g$  is a stateless firewall. By Theorem 1, this assertion holds.

Proof of iii: This assertion holds because  $g.b$  is the set of all the packets that can be accepted in the initial state of  $f$ , and  $f.a$  is the set of all the packets where each packet can be accepted in some reachable state of  $f$ .

### B Proof of Theorem 3

Given a firewall  $f$  and its stateless derivative  $g$ , we know  $f.a \subseteq g.a$  according to Theorem 2. Next we prove that if every complementary rule of  $f$  is accepting, then  $g.a \subseteq f.a$ . For any packet  $p \in g.a$ , there is an accepting rule  $r$  whose predicate is of the form

$$F_1 \in S_1 \wedge \dots \wedge F_d \in S_d \wedge tag \in S_t$$

such that  $0 \in S_t$ , and the packet  $p$  with tag value being 0 matches  $r$  but does not match any rule listed above  $r$ . Because every complementary rule is an accepting rule, every packet with a certain tag value that satisfies

$$F_1 \in S_1 \wedge \dots \wedge F_d \in S_d \wedge tag \in (D(tag) - S_t)$$

is accepted by the firewall. Here  $D(tag)$  denotes the domain of tag. So, no matter what the tag value of  $p$  is,  $p$  is accepted by  $f$ . Therefore,  $p \in f.a$ .

## Acknowledgements

We would like to thank David Taylor and the anonymous reviewers for their constructive comments on the early version of this paper.