

Stateful Firewalls

THE FOCUS OF THIS CHAPTER IS ON STATEFUL firewalls, a type of firewall that attempts to track the state of network connections when filtering packets. The stateful firewall's capabilities are somewhat of a cross between the functions of a packet filter and the additional application-level protocol intelligence of a proxy. Because of this additional protocol knowledge, many of the problems encountered when trying to configure a packet-filtering firewall for protocols that behave in nonstandard ways (as mentioned in Chapter 2, "Packet Filtering") are bypassed.

This chapter discusses stateful filtering, stateful inspection, and deep packet inspection, as well as state when dealing with various transport and application-level protocols. We also demonstrate some practical examples of how several vendors implement state tracking as well as go over examples of such firewalls.

How a Stateful Firewall Works

The stateful firewall spends most of its cycles examining packet information in Layer 4 (transport) and lower. However, it also offers more advanced inspection capabilities by targeting vital packets for Layer 7 (application) examination, such as the packet that initializes a connection. If the inspected packet matches an existing firewall rule that permits it, the packet is passed and an entry is added to the state table. From that point forward, because the packets in that particular communication session match an existing state table entry, they are allowed access without call for further application layer inspection. Those packets only need to have their Layer 3 and 4 information (IP address and TCP/UDP port number) verified against the information stored in the state table to confirm that they are indeed part of the current exchange. This method increases overall firewall performance (versus proxy-type systems, which examine all packets) because only initiating packets need to be unencapsulated the whole way to the application layer.

Conversely, because these firewalls use such filtering techniques, they don't consider the application layer commands for the entire communications session, as a proxy firewall would. This equates to an inability to really control sessions based on application-level

traffic, making it a less secure alternative to a proxy. However, because of the stateful firewall's speed advantage and its ability to handle just about any traffic flow (as opposed to the limited number of protocols supported by an application-level proxy), it can be an excellent choice as the only perimeter protection device for a site or as a role player in a more complex network environment.

Note

Using a single perimeter protection device is often a financial necessity for smaller sites. However, despite the fact that only a single firewall is being implemented, other defense-in-depth options such as intrusion detection systems (IDSs), logging and monitoring servers, and host-level protection should also be used for a more secure network implementation.

Now that we have discussed the stateful firewall, for a better understanding of its function, let's discuss the meaning of *state* and how it is tracked in network communications.

Using a Firewall as a Means of Control

An important point that should be considered when discussing perimeter security is the concept of a firewall as a network chokepoint. A *chokepoint* is a controllable, single entry point where something is funneled for greater security. However, as the name implies, this area of limited entry also can be a place where bandwidth is restricted. A good example of a chokepoint in the real world is a metal detector at an airport. Imagine if the metal detector was the size of an entire hallway in the airport, and 20 or more people could walk through a single gate at one time. If the detector goes off, it would be difficult for the inspectors to determine which party had triggered it and to be able to stop that person to examine him or her further. More fine-grained traffic control is needed in such a situation. That is why the concept of a chokepoint is necessary in such a case; it allows one inspector to watch one party go through one metal detector at a time. The chokepoint offers additional control of the parties entering the airport. Like other chokepoints, this channeling of people for additional control can also lead to slowdowns in the process; therefore, lines often form at airport metal detectors.

Similar to an airport metal detector, a firewall offers a chokepoint for your network segment. All traffic that enters or leaves your network needs to pass through it for inspection. This additional control not only helps protect inbound and outbound traffic flows but also allows a single point for examining and logging such traffic, verifying that if a breach exists, it is recorded.

The Concept of State

One confusing concept to understand when discussing firewall and TCP/IP communications is the meaning of *state*. The main reason this term is so elusive is that it can mean different things in different situations. Basically, state is the condition of being of a given communication session. The definition of this condition of being for a given host or session can differ greatly, depending on the application with which the parties are communicating and the protocols the parties are using for the exchange.

Devices that track state most often store the information as a table. This state table holds entries that represent all the communication sessions of which the device is aware. Every entry holds a laundry list of information that uniquely identifies the communication session it represents. Such information might include source and destination IP address information, flags, sequence and acknowledgment numbers, and more. A state table entry is created when a connection is started out through the stateful device. Then, when traffic returns, the device compares the packet's information to the state table information to determine whether it is part of a currently logged communication session. If the packet is related to a current table entry, it is allowed to pass. This is why the information held in the state table must be as specific and detailed as possible to guarantee that attackers will not be able to construct traffic that will be able to pass the state table test.

Firewall Clustering and Tracking State

It is possible to cluster firewalls together for redundancy, or to allow more bandwidth than a single firewall can handle on its own. In this clustered state, any of the firewall partners could possibly receive any part of a traffic flow. Therefore, although the initial SYN packet for a connection might be received on firewall 1, the final ACK response might come back to firewall 2. To be able to handle traffic statefully when firewalls are clustered, a single shared state table must be available to all the cluster members. This facilitates the complete knowledge of all traffic that other cluster members have seen. It is often accomplished using a dedicated communications cable between the members as a sort of direct link, solely for the sharing of vital state information. Such a mechanism affords an efficient means for the propagation of said state table information, allowing even the fastest communication links to operate without the problem of an incompletely updated state table.

The only other means to implement clustered firewalls without having to share state is by placing the firewalls in a "sandwich" between load-balancers. This way, a given traffic stream will always hit the same firewall it was initiated through. For more information on design choices for firewall clustering, take a look at Chapter 17, "Tuning the Design for Performance."

Transport and Network Protocols and State

Transport protocols can have their connection's state tracked in various ways. Many of the attributes that make up a communication session, including IP address and port pairings, sequence numbers, and flags, can all be used to fingerprint an individual connection. The combination of these pieces of information is often held as a hash in a state table for easy comparison. The particulars depend on the vendor's individual implementation. However, because these protocols are different, so are the ways the state of their communications can be effectively tracked.

TCP and State

Because TCP is a connection-oriented protocol, the state of its communication sessions can be solidly defined. Because the beginning and end of a communication session is

well defined in TCP and because it tracks the state of its connections with flags, TCP is considered a stateful protocol. TCP's connection is tracked as being in one of 11 states, as defined in RFC 793. To truly understand the stateful tracking of TCP, it is important to realize the many stages a TCP connection goes through, as detailed in the following list:

- **CLOSED**—A “non-state” that exists before a connection actually begins.
- **LISTEN**—The state a host is in when waiting for a request to start a connection. This is the true starting state of a TCP connection.
- **SYN-SENT**—The time after a host has sent out a SYN packet and is waiting for the proper SYN-ACK reply.
- **SYN-RCVD**—The state a host is in after receiving a SYN packet and replying with its SYN-ACK reply.
- **ESTABLISHED**—The state a connection is in after its necessary ACK packet has been received. The initiating host goes into this state after receiving a SYN-ACK, as the responding host does after receiving the lone ACK.

During the process of establishing a TCP connection, a host goes through these states. This is all part of the three-way handshake, as shown in Figure 3.1.

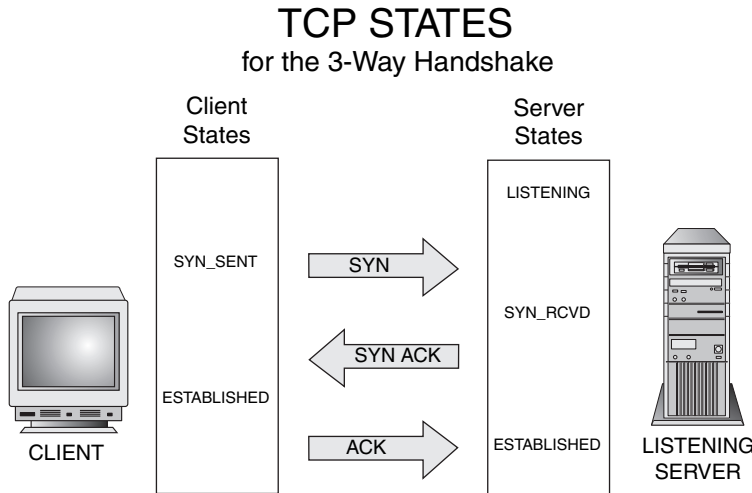


Figure 3.1 The TCP three-way handshake connection establishment consists of five well-defined states.

The remaining 6 of the 11 TCP connection states describe the tearing down of a TCP connection. The first state is used during an active close by the initiator and a passive close by the receiver, as shown in Figure 3.2.

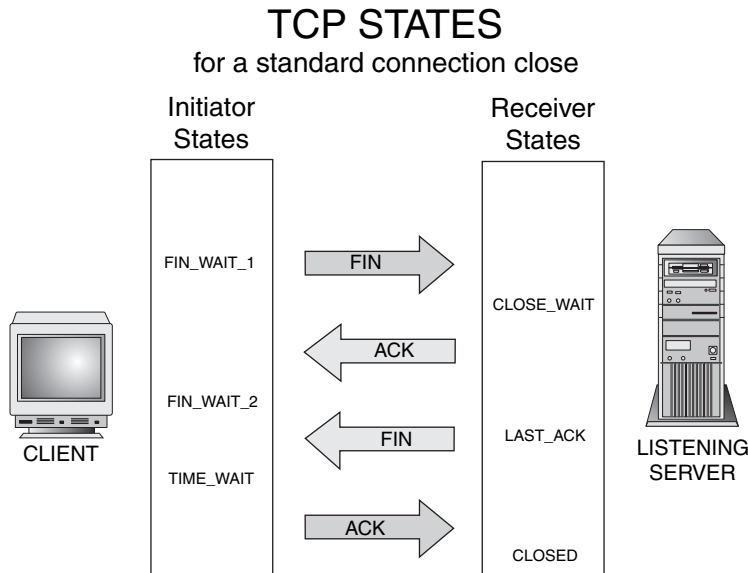


Figure 3.2 The active/passive closing of a normal TCP connection consists of six states.

- **FIN-WAIT-1**—The state a connection is in after it has sent an initial FIN packet asking for a graceful close of the TCP connection.
- **CLOSE-WAIT**—The state a host's connection is in after it receives an initial FIN and sends back an ACK to acknowledge the FIN.
- **FIN-WAIT-2**—The connection state of the host that has received the ACK response to its initial FIN, as it waits for a final FIN from its connection partner.
- **LAST-ACK**—The state of the host that just sent the second FIN needed to gracefully close the TCP connection back to the initiating host while it waits for an acknowledgment.
- **TIME-WAIT**—The state of the initiating host that received the final FIN and has sent an ACK to close the connection. Because it will not receive an acknowledgment of its sent ACK from the connection partner, it has to wait a given time period before closing (hence, the name TIME-WAIT); the other party has sufficient time to receive the ACK packet before it leaves this state.

Note

The amount of time the TIME-WAIT state is defined to pause is equal to the Maximum Segment Lifetime (MSL), as defined for the TCP implementation, multiplied by two. This is why this state is also called 2MSL.

- **CLOSING**—A state that is employed when a connection uses the nonstandard simultaneous close. The connection is in this state after receiving an initial FIN and sending an ACK. After receiving an ACK back for its FIN, the connection will go into the TIME-WAIT state (see Figure 3.3).

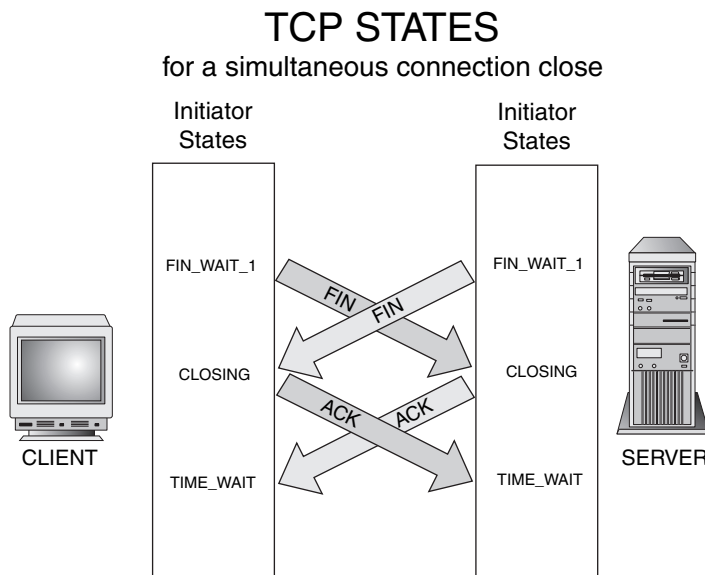


Figure 3.3 The simultaneous close of a TCP connection, where both parties close actively, consists of six states.

You can determine the state of the TCP connection by checking the flags being carried by the packets, as alluded to by the various descriptions of the TCP states. The tracking of this flag information, in combination with the IP address/port address information for each of the communicating parties, can paint a pretty good picture of what is going on with the given connection. The only other pieces of the puzzle that might be needed for clarity are the sequence and acknowledgment numbers of the packets. This way, if packets arrive out of order, the dialog flow of the communication can be more easily discerned, and the use of replay attacks against a device tracking state will be less likely to succeed.

Entries for TCP communication sessions in a state table are removed when the connection is closed. To prevent connections that are improperly closed from remaining in the state table indefinitely, timers are also used. While the three-way handshake is transpiring, the initial timeout value used is typically short (under a minute), so network scans and the like are more quickly cleared from the state table. The value is lengthened considerably (to as long as an hour or more) after the connection is established, because a properly initiated session is more likely to be gracefully closed.

It would seem from what we have just covered that the state of any TCP connection is easily definable, concrete, and objective. However, when you're tracking the overall communication session, these rules might not always apply. What if an application that employs nonstandard communication techniques was being used? For example, as discussed in Chapter 2, standard FTP uses an atypical communication exchange when initializing its data channel. The states of the two individual TCP connections that make up an FTP session can be tracked in the normal fashion. However, the state of the FTP connection obeys different rules. For a stateful device to be able to correctly pass the traffic of an FTP session, it must be able to take into account the way that standard FTP uses one outbound connection for the control channel and one inbound connection for the data channel. We will cover this issue in greater detail in the "Application-Level Traffic and State" section, later in this chapter.

UDP and State

Unlike TCP, UDP is a connectionless transport protocol. This makes the tracking of its state a much more complicated process. In actuality, a connectionless protocol has no state; therefore, a stateful device must track a UDP connection in a pseudo-stateful manner, keeping track of items specific to its connection only. Because UDP has no sequence numbers or flags, the only items on which we can base a session's state are the IP addressing and port numbers used by the source and destination hosts. Because the ephemeral ports are at least somewhat random, and they differ for any connection coming from a given IP address, this adds a little bit of credence to this pseudo-stateful method of session tracking. However, because the UDP session is connectionless, it has no set method of connection teardown that announces the session's end. Because of this lack of a defined ending, a state-tracking device will typically be set up to clear a UDP session's state table entries after a preconfigured timeout value (usually a minute or less) is reached. This prevents entries from filling the table.

Another point of concern with UDP traffic is that because it cannot correct communication issues on its own, it relies entirely on ICMP as its error handler, making ICMP an important part of a UDP session to be considered when tracking its overall state.

For example, what if during a UDP communication session a host can no longer keep up with the speed at which it is receiving packets? UDP offers no method of letting the other party know to slow down transmission. However, the receiving host can send an ICMP source quench message to let the sending host know to slow down transmission of packets. However, if the firewall blocks this message because it is not part of the normal UDP session, the host that is sending packets too quickly does not know that an issue has come up, and it continues to send at the same speed, resulting in lost packets at the receiving host. Stateful firewalls must consider such "related" traffic when deciding what traffic should be returned to protected hosts.

ICMP and State

ICMP, like UDP, really isn't a stateful protocol. However, like UDP, it also has attributes that allow its connections to be pseudo-statefully tracked. The more complicated part of

tracking ICMP involves its one-way communications. The ICMP protocol is often used to return error messages when a host or protocol can't do so on its own, in what can be described as a "response" message. ICMP response-type messages are precipitated by requests by other protocols (TCP, UDP). Because of this multiprotocol issue, figuring ICMP messages into the state of an existing UDP or TCP session can be confusing to say the least. The other, easier-to-track way in which ICMP is used is in a request/reply-type usage. The most popular example of an application that uses this request/reply form is ping. It sends echo requests and receives echo reply messages. Obviously, because the given stimulus in these cases produces an expected response, the session's state becomes less complicated to track. However, instead of being tracked based on source and destination addresses, the ICMP message can be tracked on request message type and reply message type. This tracking method is about the only way ICMP can enter into a state table.

Another issue with ICMP is that, like UDP, it is connectionless; therefore, it must base the retention of a state table entry on a predetermined timeout because ICMP also does not have a specific mechanism to end its communication sessions.

Application-Level Traffic and State

We have covered in some detail the ways that state can be tracked at the transport and network protocol levels; however, things change when you are concerned about the state of the entire session. When a stateful device is deciding which traffic to allow into the network, application behaviors must be taken into account to verify that all session-related traffic is properly handled. Because the application might follow different rules for communication exchanges, it might change the way that state has to be considered for that particular communication session. Let's look at an application that uses a standard communication style (HTTP) and one that handles things in a nonstandard way (FTP).

HTTP and State

HTTP is the one of the main protocols used for web access, and it's the most commonly used protocol on the Internet today. It uses TCP as its transport protocol, and its session initialization follows the standard way that TCP connections are formed. Look at the following tcpdump trace:

```
21:55:46.1 Host.1096 > maverick.giac.org.80: S 489703169:489703169(0)
➡ win 16384 <mss1460,nop,nop,sackOK> (DF)
21:55:46.2 maverick.giac.org.80 > Host.1096: S 3148360676:3148360676(0)
➡ ack 489703170win 5840 <mss 1460,nop,nop,sackOK> (DF)
21:55:46.5 Host.1096 > maverick.giac.org.80: . ack 1 win 17520 (DF)
```

This tcpdump trace shows the three-way handshake between a contacting client named Host and the SANS GIAC web server, Maverick. It is a standard TCP connection establishment in all aspects.

The following packet lists the first transaction after the TCP connection was established. Notice that in the payload of the packet, the `GET / HTTP/1.1` statement can be clearly made out (we truncated the output for display purposes):

```
21:55:46.6 Host.1096 > maverick.giac.org.80: P 1:326(325) ack 1 win 17520 (DF)
E..m."@...6.....!...H.P.0G...+P.Dpe$.GET./..HTTP/1.1..Accept:.image/gif,.image
```

This is the first HTTP command that a station issues to receive a web page from a remote source.

Let's look at the next packet, which is truncated for display purposes:

```
21:55:46.8 maverick.giac.org.80 > Host.1096: P 1:587(586) ack 326 win 6432 (DF)
➤E..r..@.2..6.!.....P.H...+..0HGP.....HTTP/1.1.301.Moved.Permanently..
➤Date:.Wed,.06.Feb.2002.02:56:03.GMT..Server:.Apache..Location:
➤.http://www.sans.org/newlook/home.php..Kee
```

Notice that this reply packet begins to return the home page for the SANS GIAC website at `http://www.sans.org`. As shown in the preceding example, protocols such as HTTP that follow a standard TCP flow allow an easier definition of the overall session's state. Because it uses a single established connection from the client to the server and because all requests are outbound and responses inbound, the state of the connection doesn't differ much from what would be commonly tracked with TCP. If tracking only the state of the TCP connection in this example, a firewall would allow the HTTP traffic to transpire as expected. However, there is merit in also tracking the application-level commands being communicated. We cover this topic more in the section "Problems with Application-Level Inspection," later in this chapter. Next, we look at how this scenario changes when dealing with applications that use a nonstandard communication flow, such as standard FTP traffic.

File Transfer Protocol and State

File Transfer Protocol (FTP) is a popular means to move files between systems, especially across the Internet. FTP in its standard form, however, behaves quite differently from most other TCP protocols. This strange two-way connection establishment also brings up some issues with the tracking of state of the entire connection. Would a firewall that only tracks the state of the TCP connections on a system be able to pass standard FTP traffic? As seen in Chapter 2, the answer is no. A firewall cannot know to allow in the SYN packet that establishes an FTP data channel if it doesn't take into account the behavior of FTP. For a stateful firewall to be able to truly facilitate all types of TCP connections, it must have some knowledge of the application protocols being run, especially those that behave in nonstandard ways.

When the application-level examination capabilities of a stateful inspection system are being used, a complicated transaction like that used by a standard FTP connection can be dissected and handled in an effective and secure manner.

The stateful firewall begins by examining all outbound traffic and paying special attention to certain types of sessions. As we know from Chapter 2, an FTP control session can be established without difficulty; it is the inbound data-channel initialization that is problematic. Therefore, when a stateful firewall sees that a client is initializing an outbound FTP control session (using TCP port 21), it knows to expect the server being contacted to initiate an inbound data channel on TCP port 20 back to the client. The firewall can dynamically allow an inbound connection from the IP address of the server on port 20 to the address of the client. However, for utmost security, the firewall should also specify the port on which the client will be contacted for this exchange.

The firewall discovers on which port the client is contacted through the use of application inspection. Despite the fact that every other piece of information we have needed thus far in this exchange has been Layer 4 or lower, the port number used by the server initializing the data channel is actually sent to it in an FTP `port` command from the client. Therefore, by inspecting the traffic flow between client and server, the firewall also picks up the port information needed for inbound data channel connection. This process is illustrated in Figure 3.4.

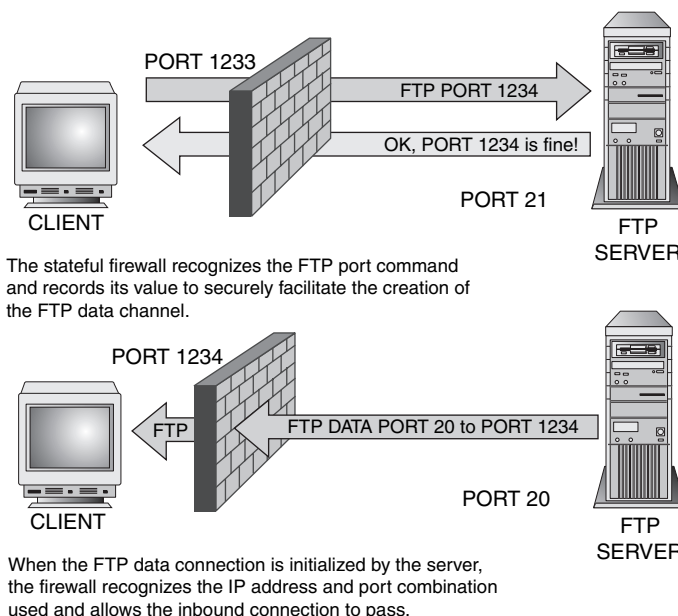


Figure 3.4 The stateful firewall examines the FTP `port` command to determine the destination port for the establishment of the FTP data channel.

Multimedia Protocols and the Stateful Firewall

Multimedia protocols work similarly to FTP through a stateful firewall—just with more connections and complexity. The widespread use of multimedia communication types, such as H.323, Real Time Streaming Protocol (RTSP), CUSeeME, Microsoft's NetShow, and more, have demanded a secure means to allow such traffic to pass into the networks of the world.

All these protocols rely on at least one TCP control channel to communicate commands and one or more channels for multimedia data streams running on TCP or UDP. The control channels are monitored by the stateful firewall to receive the IP addresses and port numbers used for the multimedia streams. This address information is then used to open secure conduits to facilitate the media streams' entrance into the network, as shown in Figure 3.5.

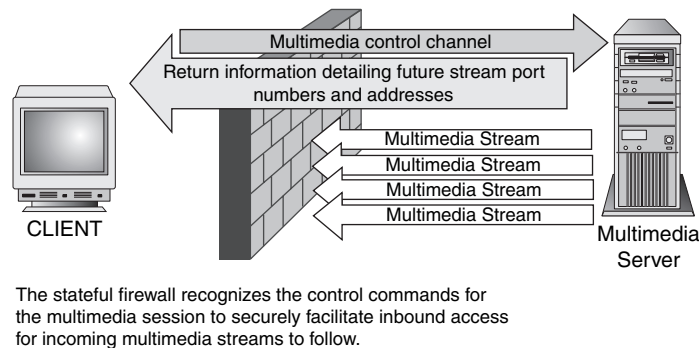


Figure 3.5 The stateful firewall tracks the multimedia protocol's communication channel to facilitate the passing of incoming media streams.

Note

Stateful firewalls now allow the use of multistream multimedia applications, such as H.323, in conjunction with Port Address Translation (PAT). In the not-so-distant past, this was a long-time problem with multistream protocols because the multiple ports used per connection could easily conflict with the PAT translation's port dispersal.

Problems with Application-Level Inspection

Despite the fact that many stateful firewalls by definition can examine application layer traffic, holes in their implementation prevent stateful firewalls from being a replacement for proxy firewalls in environments that need the utmost in application-level control. The main problems with the stateful examination of application-level traffic involve the abbreviated examination of application-level traffic and the lack of thoroughness of this examination, including the firewall's inability to track the content of said application flow.

To provide better performance, many stateful firewalls abbreviate examinations by performing only an application-level examination of the packet that initiates a communication session, which means that all subsequent packets are tracked through the state table using Layer 4 information and lower. This is an efficient way to track communications, but it lacks the ability to consider the full application dialog of a session. In turn, any deviant application-level behavior after the initial packet might be missed, and there are no checks to verify that proper application commands are being used throughout the communication session.

However, because the state table entry will record at least the source and destination IP address and port information, whatever exploit was applied would have to involve those two communicating parties and transpire over the same port numbers. Also, the connection that established the state table entry would not be properly terminated, or the entry would be instantly cleared. Finally, whatever activity transpired would have to take place in the time left on the timeout of the state table entry in question. Making such an exploit work would take a determined attacker or involve an accomplice on the inside.

Another issue with the way stateful inspection firewalls handle application-level traffic is that they typically watch traffic more so for triggers than for a full understanding of the communication dialog; therefore, they lack full application support. As an example, a stateful device might be monitoring an FTP session for the `port` command, but it might let other non-FTP traffic pass through the FTP port as normal. Such is the nature of a stateful firewall; it is most often reactive and not proactive. A stateful firewall simply filters on one particular command type on which it must act rather than considering each command that might pass in a communication flow. Such behavior, although efficient, can leave openings for unwanted communications types, such as those used by covert channels or those used by outbound devious application traffic.

In the previous example, we considered that the stateful firewall watches diligently for the FTP `port` command, while letting non-FTP traffic traverse without issue. For this reason, it would be possible in most standard stateful firewall implementations to pass traffic of one protocol through a port that was being monitored at the application level for a different protocol. For example, if you are only allowing HTTP traffic on TCP port 80 out of your stateful firewall, an inside user could run a communication channel of some sort (that uses a protocol other than the HTTP protocol) to an outside server listening for such communications on port 80.

Another potential issue with a stateful firewall is its inability to monitor the content of allowed traffic. For example, because you allow HTTP and HTTPS out through your firewall, it would be possible for an inside user to contact an outside website service such as <http://www.gotomypc.com>. This website offers users the ability to access their PC from anywhere via the web. The firewall will not prevent this access, because their desktop will initiate a connection to the outside Gotomypc.com server via TCP port 443 using HTTPS, which is allowed by your firewall policy. Then the user can contact the Gotomypc.com server from the outside and it will “proxy” the user’s access back to his desktop via the same TCP port 443 data flow. The whole communication will transpire over HTTPS. The firewall won’t be able to prevent this obvious security breach because

the application inspection portion of most stateful firewalls really isn't meant to consider content. It is looking for certain trigger-application behaviors, but most often (with some exceptions) not the lack thereof. In the case of <http://www.gotomypc.com>, application-level inspection has no means to decipher that this content is inappropriate.

Note

Despite the fact that standard stateful examination capabilities of most such firewalls could not catch deviant traffic flows such as the covert channels based on commonly open ports, many vendors also offer content filtering or Deep Packet Inspection features on their stateful firewall products to prevent such issues. FireWall-1 and the PIX both offer varying levels of content filtering, for example. However, such features are often not enabled by default or need to be purchased separately and must be configured properly to be effective.

Another popular example of traffic that sneaks out of many otherwise secure networks involves programs such as AOL Instant Messenger, Kazaa, and other messaging and peer-to-peer file-sharing programs. These programs have the potential to transmit through any port, and because most stateful firewalls have at least one outbound port open, they will find their way out. Like the aforementioned “covert channel” example, the standard stateful firewall does not differentiate this type of traffic; it allows the traffic to pass as long as it is using one of the available ports. Content-level filtering or a true proxy firewall that considers all application-level commands could be used to prevent such traffic.

Deep Packet Inspection

Some of the biggest problems security professionals face today are allowed through their firewalls by design. As mentioned in the previous section, covert channels, nefarious content traversing known ports, and even malicious code carried on known protocols are some of the most damaging security threats your business will be exposed to. It is true that even a defense mechanism as simple as a packet filter could block most of these threats if you blocked the port they were carried on, but the real issue is that they travel over protocols you want to allow into your network and are required for your business! For example, many of the most widespread worms travel over NetBIOS, HTTP, or SQL-related protocols—all of which can be an important part of your Internet or network business. Obviously, it is not good form to allow NetBIOS or SQL into your network from the Internet, but if an attack is launched from an email attachment received at a user's PC, it is very likely that you might allow these protocols to traverse security zones on your network. How can we prevent issues carried by protocols that our businesses require to function? The answer is Deep Packet Inspection.

Deep Packet Inspection devices are concerned with the content of the packets. The term Deep Packet Inspection is actually a marketing buzzword that was recently coined for technology that has been around for some time; content examination is not something new. Antivirus software has been doing it at the host and mail server level, and network IDSs have been doing it on the wire for years. However, these products have limited visibility and capability to deal with the malicious payloads they find. A major disadvantage of content filtering at these levels is that the worm, Trojan horse, or malicious packet has

already entered your network perimeter. Firewalls offering Deep Packet Inspection technology have the ability to detect and drop packets at the ingress point of the network. What more appropriate place to stop malicious traffic than at the firewall?

In the past, you have been able to use router or firewall content-filtering technologies to enter the signature of a worm or other malicious event and block it at the exterior of your network. However, what newer Deep Packet Inspection devices bring to the table are preloaded signatures, similar to those used by an antivirus solution. This way, your firewall is aware of and able to detect and remove malicious content as it arrives at your network. Also, because the packet's content is being considered at the application layer, traffic anomalies representative of an attack or worm can also be considered and filtered even if a specific signature isn't available for it. For example, if some attack uses a command that is considered nonstandard for a particular protocol, the device doing Deep Packet Inspection would be able to recognize it and drop the malicious content.

Note

The Deep Packet Inspection technology used in many popular firewall solutions is very similar to the content examination capabilities inherent in Intrusion Prevention Systems (IPSs). However, despite the fact that the technology is similar, the firewall-based solutions lack the volume of signatures and the thoroughness of analysis that a true IPS offers. Firewall-based Deep Packet Inspection could be considered "IPS-Lite." For more information on IPS, take a look at Chapter 11, "Intrusion Prevention Systems."

A Deep Packet Inspection firewall is responsible for performing many simultaneous functions. The entire content of a packet's application layer information needs to be reviewed against a list of attack signatures as well as for anomalous traffic behaviors. These firewalls also have to perform all the standard functions a stateful firewall typically handles. Therefore, advanced hardware is required to perform all these processes in a timely manner. This advanced hardware integration (typically dedicated "processors" just for this task) is what has set Deep Packet Inspection firewalls apart from their predecessors. It enables the swift processing and removal of anomalous traffic, with the added advantage of the stateful firewall's perspective on the overall communication flow of the network. This offers a major edge when determining which traffic is malicious and which is not.

Note

It is important to remember that for Deep Packet Inspection to work on SSL encrypted traffic flows, some means to decrypt the traffic must be employed. SSL certificates must also be loaded on the Deep Packet Inspection device and SSL flows must be decrypted, reviewed, and reencrypted before they are sent on to their destination. This process will cause some network latency and requires additional processing power to achieve efficient communications.

Most vendors are either already offering or are considering to offer solutions that incorporate this type of Deep Packet Inspection technology. Major vendors, including Check

Point, Cisco, and Juniper, are using some form of Deep Packet Inspection in their products and are constantly advancing it to help handle the new attacks that arrive at our networks on a daily basis.

As heavy-hitting worms such as SQL-Slammer, Blaster, Code-Red, and Welchia pound on our networks, transported via protocols that we use on a regular basis, the need for devices that consider the content of packets as well as its application become more and more urgent. Deep Packet Inspection is an excellent method to shut down some of the most used attack vectors exploited by malicious content today.

Stateful Filtering and Stateful Inspection

The definition of stateful filtering seems to vary greatly among various product vendors and has developed somewhat, as time has gone on. Stateful filtering can mean anything, from the ability to track and filter traffic based on the most minute of connection details to the ability to track and inspect session information at the application level. With this loose interpretation in mind, let's define these terms for the purpose of this chapter.

Stateful filtering has been used to define the stateful tracking of protocol information at Layer 4 and lower. Under this definition, stateful filtering products exhibit no knowledge of application layer protocols. At the most basic level, such products use the tracking of the IP addresses and port numbers of the connecting parties to track state. As mentioned previously, this is the only way that connectionless protocols can be tracked, but at best, this is only "pseudo-stateful." What about using this same method of stateful filtering for the tracking of the connection-oriented TCP? As mentioned previously, this method does not in any way track the TCP flags. TCP's flags define its connection states; therefore, although this method might be tracking some information from the various communication sessions, it is not truly tracking the TCP connection state.

More advanced forms of stateful filtering can also track sequence and acknowledgment numbers and the TCP packet flags. With the addition of these criteria, we can get truly stateful connection tracking for TCP, although we still lack the ability to differentiate traffic flows at the application level.

Stateful inspection, in contrast, has come to be used as a description of the devices that track state using all the Layer 4-type information listed previously, as well as the tracking of application-level commands. All this information can be combined to offer a relatively strong definition of the individual connection's state. Also, because Layer 7 information is being examined, extra insight into nonstandard protocol behaviors is available. This allows normally troublesome protocols such as FTP and H.323 to be securely passed by the device without complication.

Note

Stateful inspection is a term originally coined by the security product manufacturer Check Point, the maker of FireWall-1, for the way FireWall-1 handles the tracking of state information. It comprises both the tracking of state using Layer 4 protocol information and the tracking of application-level traffic commands.¹

In both stateful filtering and stateful inspection, the tracked state information is most often recorded into a state table that tracks the information until a connection is torn down (as with TCP) or until a preconfigured timeout is reached (TCP, UDP, and ICMP). Every vendor has its own implementation of these methods, and in the next several sections, we will look at some vendors' definitions of stateful filtering/stateful inspection as used in their products.

Stateful Firewall Product Examples

As stated previously, various firewall products handle the tracking of state in many different ways. This section lists some popular firewall products and provides explanations of how they handle state. We also show examples of each product's state table and examine a sample configuration of a stateful firewall.

Netfilter/IPTables

Netfilter and IPTables are the two main pieces of the most recent incarnation of a firewall product that is freely available for Linux distributions. IPTables is the construct that is used to build the firewall rule sets. Netfilter is the bridge between the Linux kernel and the IPTables rule structure. Netfilter/IPTables is the successor of the ipfwadm and IPChains products, with an ever-increasing list of features and functionality. Now thanks to its connection-tracking feature, IPTables offers stateful filtering capability.²

Connection tracking records the state of a connection based mostly on protocol-specific information. Administrators create rules specifying what protocols or specific traffic types should be tracked. When a connection is begun using a tracked protocol, IPTables adds a state table entry for the connection in question. This state table entry includes such information as the following:

- The protocol being used for the connection
- The source and destination IP addresses
- The source and destination ports
- A listing with source and destination IP addresses and ports reversed (to represent response traffic)
- The time remaining before the rule is removed
- The TCP state of the connection (for TCP only)
- The connection-tracking state of the connection

Following is an example of a state table entry for IPTables:

```
tcp 6 93 SYN_SENT src=192.168.1.34 dst=172.16.2.23 sport=1054 dport=21 [UNREPLIED]
↪src=172.16.2.23 dst=192.168.1.34 sport=21 dport=1054 use=1
```

The first line starts out listing the protocol in question, followed by the protocol's numerical designation (6 for TCP). The next value, 93, represents the time remaining before the entry is automatically cleared from the state table. Then is shown the state that

the TCP connection is in. The source and destination IP addresses follow, and then the source and destination ports are listed. Because this is an initial connection (as demonstrated by the connection's TCP state), this line lists that IPTables sees this connection as [UNREPLIED] and hasn't increased its timeout value yet. Next in the listing, we see a reversal of the original source and destination address and port information to allow return traffic. After the connection is established, the state table entry is altered, as you can see in the next example:

```
tcp 6 41294 ESTABLISHED src=192.168.1.34 dst=172.16.2.23 sport=1054 dport=21
➡src=172.16.2.23 dst=192.168.1.34 sport=21 dport=1054 [ASSURED] use=1
```

The [UNREPLIED] marker is removed after the first return packet. Upon establishment of the connection, the [ASSURED] marker is placed on the entry, and the timeout value (41294) is greatly increased.

Now let's consider the rules of IPTables.

Note

The following rule examples are basic and for demonstration purposes only. They do not take into account egress filtering or the lockdown or allowance of specific services. For optimum security, rules that specifically designate only those individual applications allowed would be more appropriate.

To begin, we'll look at the syntax and how it works. This first sample rule is considered an *output* rule because it defines which traffic can leave through the firewall (-A specifies that this rule will be appended to already existing rules):

```
iptables -A OUTPUT -p tcp -m state --state NEW,ESTABLISHED -j ACCEPT
```

This output rule determines which outbound communication will be accepted (as specified by the -j option). This particular rule deals only with the TCP protocol, as specified by the -p tcp option.

Note

IPTables and Netfilter now support IPv6. All you need is kernel version 2.4.x or above and all necessary modules and kernel patches loaded. Then you can use the `ip6tables` command for creating rules for IPv6, which supports the new 128-bit addresses. The -p protocol switch supports both ICMPv6 and IPv6. For more information on whether your system supports IPv6 or how to set up IP6Tables, check out the Linux Documentation Project site at <http://www.tldp.org/HOWTO/Linux+IPv6-HOWTO/>.

It specifies in the --state section that NEW and ESTABLISHED traffic is allowed out of our network. This rule, as listed, allows no egress protection. All new outbound TCP traffic will be allowed because the NEW option is specified. NEW tells the firewall to watch for packets with a lone SYN flag that are initiating a connection and to create entries in the state table for every such occurrence. The ESTABLISHED option allows traffic that is part of an existing session that has previously been recorded in the state table to pass as well, which means that any standard TCP communications will be able to leave the network.

Another part of the command worth mentioning is `-m state`. The `-m` denotes what module should be used for the rule in question—in this case, the standard `state` module that comes with IPTables. Now let's examine the rule that will allow the return traffic for our connection back into our network:

```
iptables -A INPUT -p tcp -m state --state ESTABLISHED -j ACCEPT
```

This command appears identical to the preceding one, except that it is an *input* rule, and only `ESTABLISHED` is listed under the `--state` section of the command. This means that only return traffic will be allowed inbound to our network, as defined by the state table. IPTables determines whether incoming traffic is return traffic for the connection entered into the state table by checking it against the reversed connection information located in the state table entry. No new connections will be able to enter our network from the outside.

Even though most of the requirements of TCP stateful tracking are available in IPTables, one exception to this is the tracking of sequence and acknowledgment numbers, which can be added with the `tcp-window-tracking` patch.³

From our previous definition of the items held in the state table, you can see that the items needed to do a pseudo-stateful job of tracking ICMP and UDP are present. Examples of basic UDP output and input rules would be as follows:

```
iptables -A OUTPUT -p udp -m state --state NEW,ESTABLISHED -j ACCEPT
iptables -A INPUT -p udp -m state --state ESTABLISHED -j ACCEPT
```

These rules appear identical to those specified for TCP, except for the `-p udp` option listing.

ICMP rules look about the same:

```
iptables -A OUTPUT -p icmp -m state --state NEW,ESTABLISHED,RELATED -j ACCEPT
iptables -A INPUT -p icmp -m state --state ESTABLISHED,RELATED -j ACCEPT
```

The main differences are the `-p icmp` specification for protocol and a new entry in the `--state` section: `RELATED`.

The `RELATED` option is the means by which IPTables allows traffic that is already in some way associated with an established traffic flow to initiate a new connection in the state table and be passed through the firewall. This related traffic might be an ICMP error message that is returned for a UDP or TCP connection already held in the state table. It could also be the initialization of an inbound FTP data channel on TCP port 20, after state table information had already been logged for an inside station starting a control channel connection on TCP port 21.

As listed, our rule allows ICMP traffic inbound and outbound that is related to existing `ESTABLISHED` traffic flows. Therefore, errors returned in response to existing TCP and UDP connections will pass. Because the `NEW` option is listed for outbound traffic, requests from ICMP programs such as `ping` will be able to leave our network, and the `ESTABLISHED` option specified for inbound traffic will allow the replies to said traffic to return back through. However, inbound `ping` requests will not be allowed in because the `NEW` option is not specified inbound.

The rules of conduct for defining related traffic are included in connection-tracking modules. They facilitate the examination of application-specific commands, such as the way the `ip_conntrack_ftp` module facilitates the inspection of FTP's `port` command to allow the secure handling of standard FTP traffic. (For more information on how stateful firewalls handle FTP traffic, see the "File Transfer Protocol and State" section, earlier in this chapter.) These modules can be added on as new protocols are used in your environment.

To implement a module such as `ip_conntrack_ftp` to allow standard outbound FTP communications to be properly initialized through our IPTables firewall, it first has to be loaded with a command such as the following:

```
modprobe ip_conntrack_ftp
```

Next, a specific rule has to be created to inspect the related traffic. This can be accomplished in the case of FTP by making an `INPUT` rule that allows inbound TCP port 20 traffic with the state option of `RELATED`. This will allow the inbound port 20 traffic to connect if the inspection process deems it related to an existing connection in the state table. Here is a listing of such a rule:

```
iptables -A INPUT -p tcp --sport 20 -m state --state ESTABLISHED,RELATED -j ACCEPT
```

An `OUTPUT` rule will be needed as well to allow response traffic to return:

```
iptables -A OUTPUT -p tcp --dport 20 -m state --state ESTABLISHED -j ACCEPT
```

Notice that the `--sport 20` option representing the source port in the `INPUT` rule has changed to the `--dport 20` (or destination port) option in the `OUTPUT` rule. This change is due to the reversal of communication roles for outbound versus inbound traffic.

Check Point FireWall-1

The Check Point FireWall-1 (FW-1) is one of the most popular stateful firewalls in use today. It is software based and can be loaded onto hardware server solutions of various platform types, including Windows, Solaris, and Red Hat Linux. It is also offered as a hardware appliance solution by Nokia. FireWall-1 uses a state table for the basic tracking of connections at the protocol level and an `INSPECT` engine for more complicated rules involving application layer traffic and nonstandard protocol behavior.

When deciding whether to allow a packet to pass, FireWall-1 tests it against the following data structures, in the order specified:

- First, FireWall-1 checks to see whether a connection is already logged in the state table for this particular incoming packet. If so, it is forwarded without further scrutiny.
- Next, if the state table did not contain an entry for the packet, the packet is compared against the security policy. If a rule allows the packet to pass, it will be forwarded on, and an entry for its communication session will be added to the state table.

TCP traffic is handled at a protocol level, much like previously shown examples. When a communication ensues, because the first packet of a connection will not be reflected in

the state table, it is tested against the security policy. If it is accepted based on one of the rules, it is added into the state table.

Tip

For the most complete stateful protection of TCP communication flows, be sure to use the latest vendor-recommended version and feature pack of FireWall-1. In this text, all commands and examples use FW-1 NG. Also, for the highest level of security protection, be sure that all suggested hot fixes are applied.

The rules that might allow traffic to pass are either one of the implied rules set up in the FireWall-1 section of the Global Properties of SmartDashboard or are part of the rulebase created and maintained in FireWall-1's SmartDashboard GUI interface. For an example of a rule's listing and what the SmartDashboard interface looks like, refer to Figure 3.6.

Implied Rules

Be aware that even though FW-1's implied rules are not seen by default when you are viewing a firewall policy, they will allow certain types of traffic through your firewall. To ease your troubleshooting efforts, you may want to check the Log Implied Rules box in the FireWall-1 section of Global Properties. Also, to keep yourself cognizant of the implied rules when building your rulebase, you can check the Implied Rules option under the View menu of SmartDashboard so these rules appear when you view your firewall policy.

As shown in Figure 3.6, designing a rule set for firewall products such as FireWall-1 can be less demanding than some of the less user-friendly choices, such as IPTables. FireWall-1 allows you to use a GUI interface to represent your networks and systems as objects. You can elect to allow or disallow traffic for specific services by selecting appropriate options and referencing relevant objects. Rule order is one of the most critical things to keep in mind when designing such a rule set. It is vital to list specific rules at the top of the rule list before more general rules that might inadvertently apply to unwanted traffic types.

Note

For more information on building a FireWall-1 rulebase, see Lance Spitzner's paper titled "Building Your Firewall Rulebase" at <http://www.spitzner.net/rules.html>.

FireWall-1 enforces timeouts for TCP connections to ensure that improperly terminated sessions that lack the common FIN packet exchange do not remain in the state table indefinitely. The initial timeout on a half-open connection (before the three-way handshake has been completed) is logged at 60 seconds by default. Upon completion of the three-way handshake, this timeout is increased to 60 minutes to allow for latency in communications. After the closing of the connection is initiated with a FIN packet, the timeout is dropped to 50 seconds to ensure that the state table entry is more quickly cleared if the graceful FIN exchange is not completed successfully.

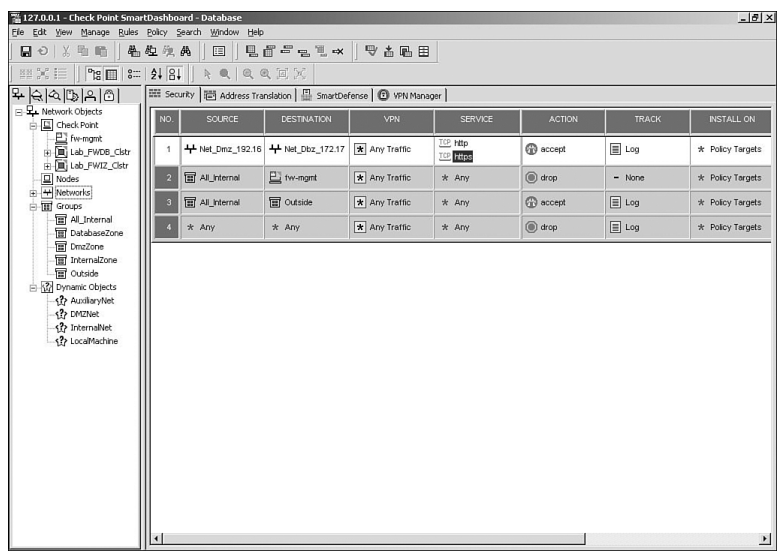


Figure 3.6 Check Point FireWall-1 NG offers a user-friendly GUI interface called SmartDashboard (formerly Policy Editor in previous versions) for editing its rule set.

Note

The 60-minute timeout setting for TCP connections, as well as the default UDP timeout, can be adjusted in the Stateful Inspection section of the Check Point NG Global Properties dialog box, as shown in Figure 3.7. In Check Point NG, all TCP and UDP services will use the shown timeout values by default, or you can manually configure a specific service with its own timeout value by clicking the Advanced button in the properties box for that particular service.

FireWall-1 handles UDP traffic in much the same way that other stateful firewalls do. It uses a pseudo-stateful method of tracking outbound UDP connections, and it allows inbound UDP packets that match one of the currently recorded communication flows. This process is accomplished through the recording of the IP addressing and port numbers that the communication partners use. A timer is used to remove the session from the state table after a predetermined amount of inactivity (see Figure 3.7).

For a better understanding of FireWall-1’s tracking of state, look at its state table. Listing 3.1 is the Check Point FireWall-1 state table as decoded (it normally appears as rows of difficult-to-decipher numbers) by a Perl script (fwtable.pl) available from Lance Spitzner’s website at <http://www.spitzner.net/fwtable.txt>.⁴

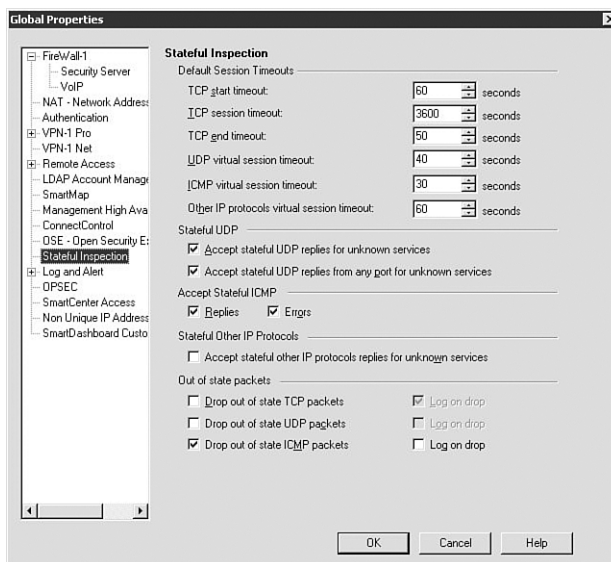


Figure 3.7 The Stateful Inspection section of the Global Properties dialog box for FireWall-1 NG contains many settings that define how FireWall-1 handles state.

Listing 3.1 A Check Point FireWall-1's State Table as Translated by fwtable.pl

Src_IP	Src_Prt	Dst_IP	Dst_Prt	IP_prot	Kbuf	Type	Flags	Timeout
192.168.1.202	1783	192.168.1.207	137	17	0	16386	ffffff00	18/40
192.168.1.202	1885	192.168.1.207	80	6	0	28673	ffffff00	43/50
192.168.1.202	1884	192.168.1.207	80	6	0	28673	ffffff00	43/50
192.168.1.202	1797	192.168.1.207	23	6	0	16385	ffffff00	35/50
192.168.1.202	1796	192.168.1.207	22	6	0	16385	ffffff00	35/50
192.168.1.202	1795	192.168.1.207	21	6	0	16385	ffffff10	35/50
192.168.1.202	1798	192.168.1.207	25	6	0	16385	ffffff00	35/50
192.168.1.202	1907	192.168.1.207	80	6	0	28673	ffffff00	43/50

IP addresses, port numbers, and even timeout values can be clearly seen in the FireWall-1 state table represented by fwtable.pl.

Tip

Dr. Peter Bieringer has updated Lance Spitzner's fwtable script to support versions of FW-1 through NG. For a copy, check out <http://www.fw-1.de/aerasesec/download/fw1-tool/fw1-tool.pl>.

FW-1 supports the stateful inspection of many popular protocols. These include the following TCP protocols: H.323, FTP-BIDIR, RTSP, IIOP, SQLNET2, ENC-HTTP, Netshow, DNS_TCP, SSH2, FW1_CVP, HTTP, FTP-Port, PNA, SMTP, FTP-PASV,

FTP_BASIC, SSL_V3, Winframe, CIFS, FTP, INSPECT, CitrixICA, and RSHELL. It also pseudo-statefully inspects the following UDP protocols: CP-DHCP-reply, SNMP Reads, SIP, H.323 RAS, NBDatagram, DNS, CP-DHCP-request, NBName, and Freetel. FW-1 has additional capabilities to track RPC traffic of many varieties. It has the capability to allow many nonstandard protocols, including not only RPC, but also H.323, FTP (standard), and SQLNET2. For example, FireWall-1 statefully handles traffic based on remote procedure calls (RPCs), such as Network File System (NFS), whose ports are randomly generated by the portmapper service. The portmapper service runs on TCP and UDP ports 111 and handles the automatic generation of RPC programs' access ports. Because these ports are generated randomly, it would be nearly impossible to write a secure rule that could effectively permit such traffic. FireWall-1 solves this problem by tracking all specified portmapper traffic and actually caching the port numbers that portmapper maps to the RPC programs in use. This way, such traffic can be effectively tracked.

Configuring an FW-1 Service for Stateful Inspection

It should be mentioned that FW-1 does not automatically realize that it should track a service statefully by its port. The object representing the service must be configured for the protocol type in question. If you chose one of the predefined service objects from Check Point, this should already be done for you. However, if you created your own service object for some reason (different naming convention, alternate port setting, and so on), you need to manually configure the protocol that the object represents. For example, if your company runs standard FTP over port TCP/1021 (instead of TCP/21), it would seem that creating a TCP object and assigning it a name and port 1021 would be enough. However, FW-1 would handle this as a standard single-session TCP service and would not allow the return data channel to ensue. To configure the service object for FTP protocol, edit the object, click the Advanced button, and change the protocol type drop-down box to FTP-PORT. To make this change take place, you will need to reinstall the policy containing the object.

A new feature of FW-1 NG is the Smart Defense component. It is available in Feature Pack 3 and higher or can be loaded as a hotfix with Feature Pack 2. It allows advanced application layer examination (akin to Deep Packet Inspection) for a list of known attacks, worms, and types of malicious behavior. Applying this additional level of support to your existing FW-1 policy is as easy as switching to the SmartDefense tab in SmartDashboard and checking the protection options you want to employ. For additional information on the many elements of SmartDefense, check out the SmartDefense Technical White Paper on Check Point's website (http://www.checkpoint.com/products/downloads/smartdefense_whitepaper.pdf).

The Cisco PIX Firewall

The PIX firewall statefully inspects traffic using Cisco's Adaptive Security Algorithm (ASA). The ASA is used to make a representative hash of each outgoing TCP and UDP packet and then store it in a state table. When the TCP and UDP traffic return, because a representative entry is recorded in the state table, the traffic will be allowed to pass. ICMP traffic is a different matter. Inbound ICMP traffic is denied through the outside interface of the PIX, and a specific access list must be created to allow any such traffic.

Outbound ICMP is allowed, but it will not work by default, because the inbound responses will be blocked, like the `echo-reply` response to a `ping` command. Here's an example of an access list that will let ICMP traffic from a given test address cross through the PIX (as commonly used for troubleshooting new PIX installations):

```
access-list ICMP-ACL permit icmp test address inside address range
access-group ICMP-ACL in interface outside
```

The first command creates an access list called `ICMP-ACL` that permits ICMP traffic from a specified test address to our inside address range. The second line applies that ACL inbound on the outside interface.

The command that the PIX firewall uses to configure the stateful examination of traffic flows for a given protocol is the `fixup` command. The `fixup` command starts an advanced application-specific examination of outbound traffic of the protocol type listed to the designated port. The Cisco PIX firewall supports this application-level examination of traffic for the following protocols through the standard `fixup` command: CTIQBE, ESP-IKE, FTP, HTTP, H.323 (now supporting version 3 and 4), ICMP ERROR, ILS, MGCP, PPTP, RSH, RTSP, SIP, SIP UDP, SKINNY (now supporting PAT), SMTP, SNMP, SQLNET, and TFTP. The fixups for these protocols can be added or removed from a PIX configuration and reconfigured for various port specifications. They are considered extraneous to the operations of the PIX.

The PIX also offers built-in `fixup` support for these protocols: RTSP, CUSEEME, DNS, SUNRPC, XDMCP, H.323 RAS, TCP, and UDP. These integrated fixups are not seen in the PIX's configuration. They work in the background in conjunction with the normal fixups for the advanced inspection of these particular types of traffic. Even if a fixup is not installed for a particular TCP or UDP traffic type, the PIX will still track the session in its state table and allow its return traffic re-admittance to the network.

Not all fixups are created equal. Each fixup tracks application layer information at different levels. This level of inspection might vary from making sure the traffic passes through NAT successfully to the monitoring for specific application-level commands. For example, the SMTP fixup is the most stringent of them all. Since PIX software version 4.2, the SMTP fixup has supplied a protection feature called "mailguard." This fixup allows only SMTP commands to pass through it successfully. Non-SMTP traffic commands are dropped, but the PIX still returns an OK to the sender as if the information were passed. This helps protect poorly defended mail servers from outside attacks. Other fixups, such as FTP and H.323, allow the return of nonstandard communication traffic by monitoring the application-level commands that control the formation of their data channels.

Because the standard `fixup` command allows the specifying of the port number to be examined for the protocol, alternative configurations are supported. (This is not true for the built-in fixups.) For example, if you need to access a web server that is running on port 8080, use the following command:

```
Pixprompt(config)#fixup protocol http 8080
```


Such a `fixup` command will allow the creation of state table information for the listed outbound traffic type. Multiple `fixups` can be listed if more than one port number is used per protocol. The PIX's state tables contain ASA hashes based on the source and destination addresses, port numbers, sequence numbers, and TCP flags. Because PIX firewalls use truly random TCP sequence number generation, the connection is kept more secure.⁵

When the reply returns, the PIX checks the response against the state table and information that it knows about the behavior of the protocol in question. If the information checks out, it is allowed to pass. All other information is dropped unless it is specifically allowed using an access list.

The table listing connection state for a Cisco PIX can be viewed using the `show conn` command. Such a table can be seen in Listing 3.2.

Listing 3.2 The Output from a Cisco PIX Firewall's `show conn` Command

```
TCP out xx.yy.zz.129:5190 in 172.16.1.33:1960 idle 629:25:50 Bytes 6737 flags UIO
TCP out xx.yy.zz.254:23 in 172.16.1.88:1053 idle 0:11:33 Bytes 226696 flags UIO
TCP out xx.yy.zz.254:23 in 172.16.1.76:1146 idle 256:09:15 Bytes 78482 flags UIO
TCP out xx.yy.zz.254:23 in 172.16.1.100:1660 idle 145:21:19 Bytes 9657 flags UIO
TCP out xx.yy.zz.254:23 in 172.16.1.100:1564 idle 641:51:05 Bytes 132891 flags UIO
UDP out xx.yy.zz.12:137 in 172.16.1.12:137 idle 0:00:03 flags
```

Notice that standard IP address and port information is tracked, along with the time that entries will remain in the table. Also notice on the last entry for a UDP connection that no flags are listed. Despite the fact that this output shows current connections and can give you a good idea of what information is in your PIX's state table, this is not a true dump of the state table because it lacks the information provided by the stored ASA hash. You will notice, for example, that sequence numbers are not listed in this output.

To learn more about the way the PIX firewall operates and to better understand the configuration of a stateful firewall, we will look at a PIX firewall configuration using software version 6.3(4). The configuration will only include those items that have to do with passing standard protocol information.

First, in the PIX configuration listing are commands that define the interfaces:

```
nameif ethernet0 outside security0
nameif ethernet1 inside security100
```

This is a simple configuration with only two interfaces: an inside interface and an outside interface. Notice the security levels (shown as `security0` and `security100`). By default, all traffic can flow from a higher security numbered interface to a lower one on a PIX, but none can flow from a lower interface to a higher one. By default, this PIX cannot receive inbound traffic connections, but it can send anything out. These default behaviors can be adjusted by using NAT and access lists.

To allow an inbound connection, two criteria must be met:

- A static NAT mapping must be configured to allow the inbound traffic flow to bypass NAT translation, assuming that NAT is used in your environment. If it is not, this criterion can be ignored.
- An access list must be made to allow the type of traffic in question. For highest security, inbound traffic should only be allowed when using a firewall with a DMZ port; public servers can be placed on their own screened subnet.

Because NAT will not be an issue, you only need to add an access list to disallow outbound connections. This prevents the traffic types that you want to disallow. You can also create an egress filter to verify that only authentic local traffic is leaving your network.

Note

This configuration as listed does not include support for egress protection. For optimum security, egress protection of some sort is suggested. For more information on egress filters, see Chapter 2.

The `nameif` interface commands are followed by the backbone of the PIX configuration: the `fixup` commands (as mentioned earlier in the section). These commands list the protocols and their associated port numbers that the PIX will inspect. Listed next are some popular choices:

```
fixup protocol ftp 21
fixup protocol http 80
fixup protocol h323 1720
fixup protocol rsh 514
fixup protocol smtp 25
fixup protocol sqlnet 1521
fixup protocol sip 5060
```

The next lines of this listing show the IP addresses and subnet masks assigned to both the inside and outside ports. These are displayed here as a point of reference for the NAT-related commands to follow:

```
ip address outside 192.168.2.178 255.255.255.240
ip address inside 172.16.1.10 255.255.0.0
```

In the next portion of the listing, we create two address pools of outside addresses for our NAT pool, reflected by the first line, `(outside) 2`, and for PAT, `(outside) 1`. If we were only using PAT, the first line would not be necessary.

```
global (outside) 2 192.168.2.180-192.168.2.190 netmask 255.255.255.240
global (outside) 1 192.168.2.179
```

Next, we define the inside addresses for pool 2 and pool 1. The first statement lists the pool of inside addresses that will be NAT address translated. All other IP addresses will be forced to use PAT.

```
nat (inside) 2 172.16.1.96 255.255.255.248 0 0
nat (inside) 1 0.0.0.0 0.0.0.0 0 0
```

The second line demonstrates a wildcard so that any IP address (other than those listed on the previous line) will be PAT translated. We know that this is a PAT translation command because it maps to the previous `global (outside) 1` command, which only has one public address.

PIX firewalls allow return traffic in conjunction with the NAT statement. The NAT and state tables combine to let the firewall know which traffic is returning as part of an already started conversation.

The next statements are the defaults of a PIX configuration and were not added in this example. However, they are displayed to show the default timeouts for the NAT translation table (`xlate`), the connection listings (state table, `conn`), and user authentication traffic listings (`uauth`):

```
timeout xlate 3:00:00
timeout conn 1:00:00 half-closed 0:10:00 udp 0:02:00 rpc 0:10:00 h323 0:05:00 sip
➡0:30:00 sip_media 0:02:00
timeout uauth 0:05:00 absolute
```

Note

When you're troubleshooting broken connections through a PIX firewall, one good step is to raise the `xlate` timeout. If out-of-state traffic is seen getting dropped in the firewall logs, the `conn` timeout values may need adjusted.

These timeouts can be adjusted at any time by simply retyping the preceding commands with new timeout values.

The PIX Device Manager (PDM) has simplified PIX firewall management. The PDM is a GUI interface that's used to edit most of the PIX firewall's settings. It allows the editing of firewall access rules, NAT translation settings, host and network objects, and general firewall configuration changes. It also has a Monitoring section (see Figure 3.8) that allows the viewing of statistical information about the PIX and its performance as well as provides the ability to generate attractive exportable graphs.

To use the PDM, you simply have to add the appropriate PDM image to your PIX. This is done by copying the PDM file to the PIX's flash memory with the following command:

```
copy tftp://ipaddress/pdmfilename flash:pdm.
```

Here, *ipaddress* is the IP of the TFTP server holding the PDM image you are copying, and *pdmfilename* is the name of the PDM file. If there is already a PDM file on your PIX, it will be erased. Your PIX will need to be configured to allow HTTP access from any clients needing to do PDM administration. This is done as follows:

```
Pix(config)#http 10.0.0.1 255.255.255.255 inside
```

Here, 10.0.0.1 is the station you want to manage the PIX with. This is followed by a 24-bit mask so that it is the only station allowed to make contact. This could be any legal IP address mask allowing access from one to an entire network segment of addresses. Finally, the statement is ended with the name of the interface you want to manage the PIX through—preferably the inside interface!

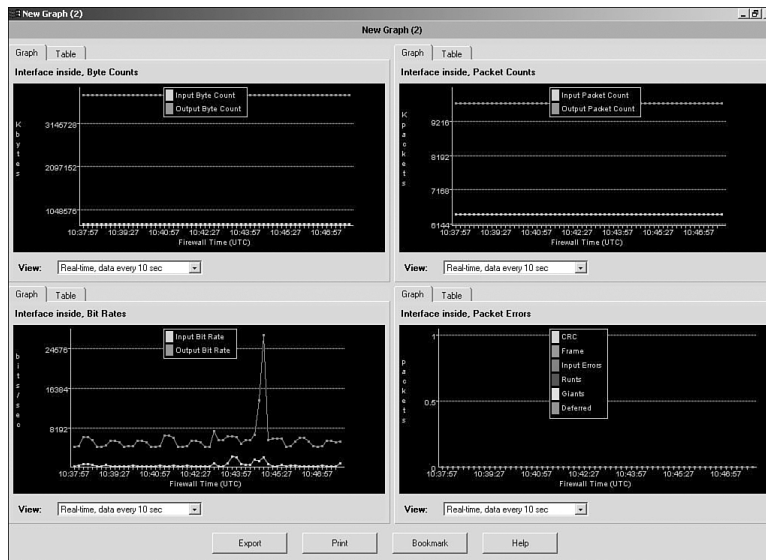


Figure 3.8 The PIX Device Manager's Monitoring tab allows the generation of aesthetically pleasing graphs that can be exported.

Now all you have to do is type `HTTPS://10.0.0.100` into the URL line of any web browser (where 10.0.0.100 is the IP address of the interface you specified in the HTTP access line). Be sure to specify HTTPS in your web browser! Trying to access the PIX via HTTP is disallowed and will return a “Page cannot be displayed” message. If your configuration was successful, you should be prompted to accept a certificate and then receive a pop-up window asking for authentication information. You can use any username and password you already have configured in your PIX for management, or if none are configured you can leave the Username box blank and type the PIX's enable password in the password box. You should be greeted with the PDM's home startup screen, as shown in Figure 3.9.

If you click the Configuration button, you will be taken to the area in the PDM where most of the important firewall rule and NAT management takes place. The Access Rules tab (see Figure 3.10) is where the firewall rules configured in your PIX can be viewed and edited.

Note

The first time you access the PDM, the configuration screen information may not be populated. If so, go to the File menu and choose Refresh PDM with the Running Configuration on the Firewall. You will be prompted that the PDM needs to query the PIX for the first time to get the information it needs to populate the PDM tabs. Thereafter, all information should appear as expected.



Figure 3.9 The PDM home page shows many useful pieces of information about your PIX, including version, license, and status information.

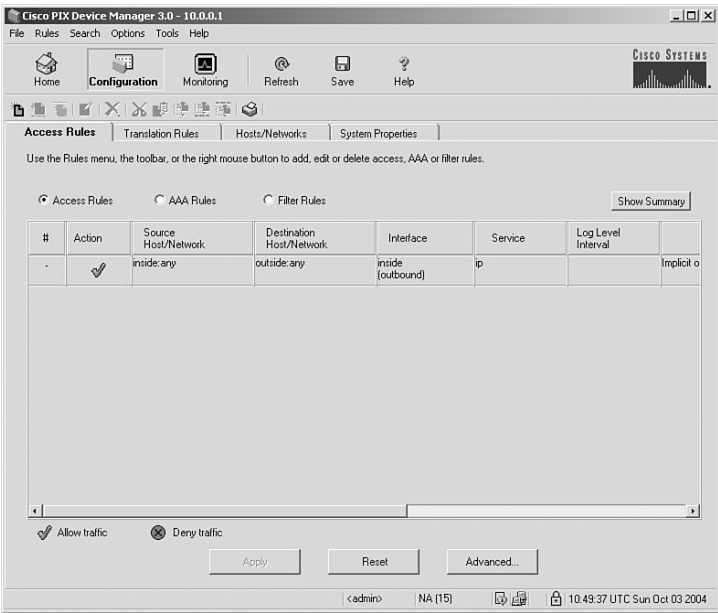


Figure 3.10 The Configuration section of the PDM allows viewing and editing of the access rules that make up the firewall policy.

If no specific rules have been added to the PIX, the Access Rules tab will appear, as shown in Figure 3.10, with the implicit permit rule for outbound traffic.

The Translation Rules tab shows the NAT configuration for the PIX firewall, per interface (see Figure 3.11). When the radio button Translation Rules is enabled, all NAT commands specified in the PIX and their associated global commands are displayed. Clicking the Manage Pools button allows you to edit the NAT pools for the given interface.

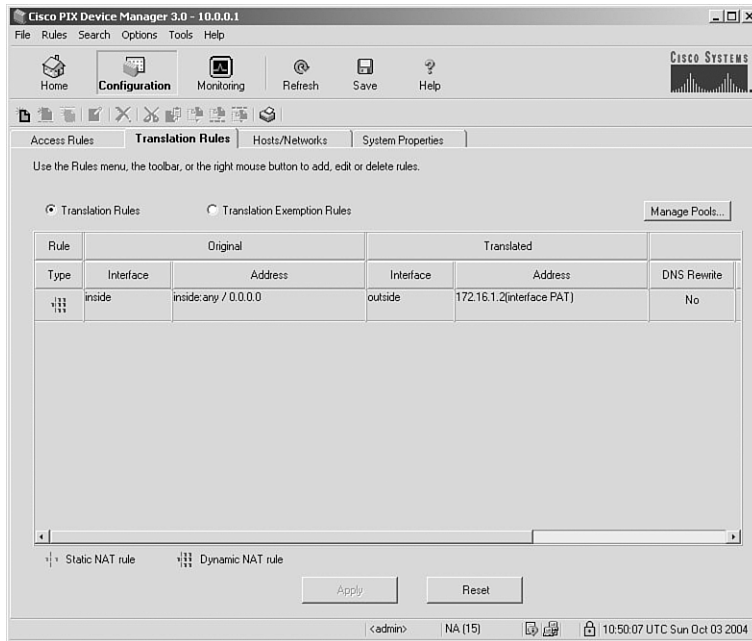


Figure 3.11 The PDM Translation Rules screen shows all NAT information for the PIX.

If the radio button Translation Exemption Rules is selected, the configuration of any NAT 0 commands in the PIX are displayed.

The Hosts/Networks tab is where objects can be viewed and edited for use in the PDM. These objects will be used to populate the access rules when creating a firewall policy.

Finally, you can perform many basic PIX administration tasks on the System Properties tab, including interface configuration, DHCP server and client configuration, logging and AAA settings, various management settings, and more. The PDM is an excellent tool to ease the administrative burden of the firewall for a PIX novice—or even a seasoned professional.

Now that 10Gb networking is being incorporated into the enterprise and Internet connection speeds are getting faster, the speed with which a firewall can statefully process information is becoming increasingly important. An exciting addition to the PIX line is the FireWall Services Module (FWSM). This is a full PIX firewall on a card that fits into an available slot in the 6500 series Cisco enterprise switches. It supports most of the features of the standard PIX but takes advantage of the port density and speed of the 6500. The bandwidth for connectivity is supplied by the backplane of the 6500 (which by default supports 32Gbps), letting the FWSM support an astounding throughput of up to 5.5Gbps! If that's not enough throughput for you, up to three more FWSMs can be added to the 6500 series chassis for a combined throughput of 20Gbps. The FWSM uses the VLAN interfaces of the 6500 for ingress and egress of traffic. In turn, the FWSM can support as many as 250 virtual interfaces!

Note

Be sure to use an up-to-date version of the FWSM code. Major vulnerabilities that could cause a DoS condition were announced in late 2003 (documented as CSCeb16356 and CSCeb88419). Both problems have been corrected in software version 1.1.3.

Virtual firewalls can be configured, allowing management of separate policies by different groups of administrators. The FWSM supports active-passive and active-active configurations as well as management via the PDM. When considering a means to protect intra-VLAN communication on a 6500 series switch or considering a solution in an enterprise environment that requires the maximum in throughput, you would be remiss not to take into account the power and flexibility of the FWSM.

High-Speed NetScreen Firewall Appliance

As network bandwidth requirements grow higher and content gets richer, the need for faster firewalls becomes greater. That is the focus of the Juniper Networks NetScreen firewall, which is an appliance-based stateful firewall that is particularly well-regarded due to its fast performance. Using specialized microchips called Application-Specific Integrated Circuits (ASICs), rather than relying solely on a central microprocessor, NetScreen is able to achieve very high throughput, especially on its carrier-class model. ASICs that are designed to perform a particular task are much more efficient and much faster than a processor running code to do the same task.

The core functions that NetScreen offers are typical for what you would expect of a stateful firewall aimed at the enterprise market. Along with the standard access control features, NetScreen also includes basic QoS capabilities, and integrated high speed VPN support (6 Gb/s throughput with 3DES as of this writing).

NetScreen is also able to screen for and block some of the more popular network attacks such as the Ping of Death, Land and Teardrop attacks, as well as port scans and various types of network floods. Despite all of these features, the filtering and logging capabilities of the NetScreen still leave some room for improvement. Never-the-less, the NetScreen firewall's performance is demonstrative of ASIC-based firewall appliances becoming an integral part of network access control.

Summary

The firewall provides a secured method of controlling what information moves in to or out of a defined ingress/egress point of your network. This concept of a network “chokepoint” allows increased control and a single target for the monitoring and logging of network traffic. This extra control does come at a price: an overall cost in performance.

The stateful firewall adds intelligence to the packet-filtering method of network communication control. Stateful filtering has been popularly used to define the filtering of the state of packet flows based on information from Layers 4 and below. This definition is ambiguous because the amount of protocol information that is considered in the filtering can deviate among vendor implementations. Items such as source and destination IP addresses and port numbers, sequence and acknowledgment numbers, as well as flags and other Layer 4 information can all be considered.

Stateful inspection also monitors Layer 4 information (just like stateful filtering) and adds application-level examination to provide insight into the communication session. This offers a secure means to handle nonstandard TCP/IP traffic flows. Stateful inspection offers a much more secure environment than a “dumb” packet filter as well as performance advantages over a proxy firewall, making it an excellent compromise between the two technologies. However, the same features that give stateful application inspection a performance advantage over a proxy firewall also make it less secure in environments where all aspects of application-level communication must be considered.

In any case, the stateful firewall is an excellent fit as a single perimeter security solution for smaller environments. It performs well as a role player in larger or more complex environments where multiple firewall technologies are implemented. Clearly, the stateful firewall is a solid choice and a strong performer in the current network landscape. In the next chapter, we examine a way to filter network traffic by taking advantage of application-level restraints that can be implemented using proxy firewalls.

References

- 1 Check Point. “Stateful Inspection Technology Tech Note.” http://www.checkpoint.com/products/security/whitepapers/firewall-1_statefulinspection.pdf. March 2002.
- 2 Netfilter/IPTables Documentation. “What Is Netfilter?” <http://www.iptables.org/documentation>. March 2002.
- 3 Fabrice Marie. “Netfilter Extensions HOWTO.” <http://netfilter.samba.org/documentation/HOWTO//netfilter-extensions-HOWTO.html>. March 2002.
- 4 Lance Spitzner. “Understanding the FW-1 State Table.” November 29, 2000. <http://www.spitzner.net/fwtable.html>. March 2002.
- 5 Cisco Systems, Inc. “Cisco’s PIX Firewall Series and Stateful Firewall Security.” http://www.cisco.com/warp/public/cc/pd/fw/sqfw500/tech/nat_wp.pdf. March 2002.