CHAPTER 6

Controlling Access Through the Firewall

A firewall's main function is to provide effective security between pairs of its interfaces. To do this, all of the traffic destined to pass through it must undergo a variety of operations, inspections, translations, filters, and special handling. You must configure each aspect of these actions in order to thoroughly enforce the security policies that apply to your network.

All of the features related to controlling user access *through* the firewall have been collected in this chapter. The size of the chapter is a testament to the broad range of security policy tools available to you, as a firewall administrator.

6-1: Routed and Transparent Firewall Modes

Traditionally, Cisco firewalls have operated by performing Layer 3 (IP address) operations. Naturally, the stateful inspection process can look at higher layers within the IP packets being examined. But the firewall itself has maintained its own interface IP addresses and acted as a router or gateway to the networks that connect to it. As well, all of the traffic inspection and forwarding decisions are based on Layer 3 (IP address) parameters. This is known as the *routed firewall mode*. Each firewall interface must be connected to a different IP subnet, and be assigned an IP address on that subnet. When a routed firewall is installed or inserted into a network for the first time, the network must become segmented across the firewall's interfaces. For example, where a single IP subnet used to be, the inside and outside interfaces now form the boundary of two separate subnets.

This can make the installation difficult, as some readdressing must take place. The easiest approach is to keep the original IP addressing on the firewall's inside interface, where the majority of protected hosts reside. The outside interface can take on an address from a new subnet that is shared between the firewall and the next-hop router. In other words, the outside of the firewall usually has a lesser number of directly connected hosts to readdress to a new subnet.

Routed firewalls can also participate in IP routing by using a dynamic routing protocol such as RIP or OSPF. The firewall can coexist with other routers in the network and maintain a dynamic routing table.

By default, the Adaptive Security Appliance (ASA) and Firewall Services Module (FWSM) platforms are configured for routed mode. You can configure IP addresses on the firewall interfaces, IP routing, and other Layer 3 features by following the guidelines found in Chapter 3, "Building Connectivity."

TIP You can see the current firewall mode by using the following command:

```
Firewall# show firewall
```

The result is either "Firewall mode: Router" or "Firewall mode: Transparent". If you need to configure a firewall back to routed mode, use the following command:

```
Firewall(config)# firewall router
```

A Cisco firewall can also be configured to operate in *transparent firewall mode*. The firewall appears to operate as a Layer 2 device, without becoming a router hop or a gateway to the connected networks. This is also known as a *Layer 2 firewall* or a *stealth firewall*, because its interfaces have no IP addresses and cannot be detected or manipulated. Only a single management address can be configured on the firewall.

As a Layer 2 device, a transparent mode firewall can be dropped or wedged into an existing network, separating the inside and outside without changing any existing IP addresses. This is commonly called *bump-in-the-wire* because the firewall does not break or segment the IP subnet along a wire—it more or less becomes part of the wire. This makes a new installation very straightforward.

TIP Cisco firewalls running PIX release 6.3 or earlier operate solely in the routed firewall mode. Beginning with FWSM 2.2(1) and PIX 7.0, you can configure a firewall to operate in either the routed or transparent firewall mode, but not both.

You can think of a transparent mode firewall as a type of transparent bridge, where packets are bridged from one interface to another based only on their MAC addresses. The firewall maintains a MAC address table from received packets, containing the source MAC address and the source interface. The firewall is able to forward a packet by knowing the location or the egress interface of the destination MAC address.

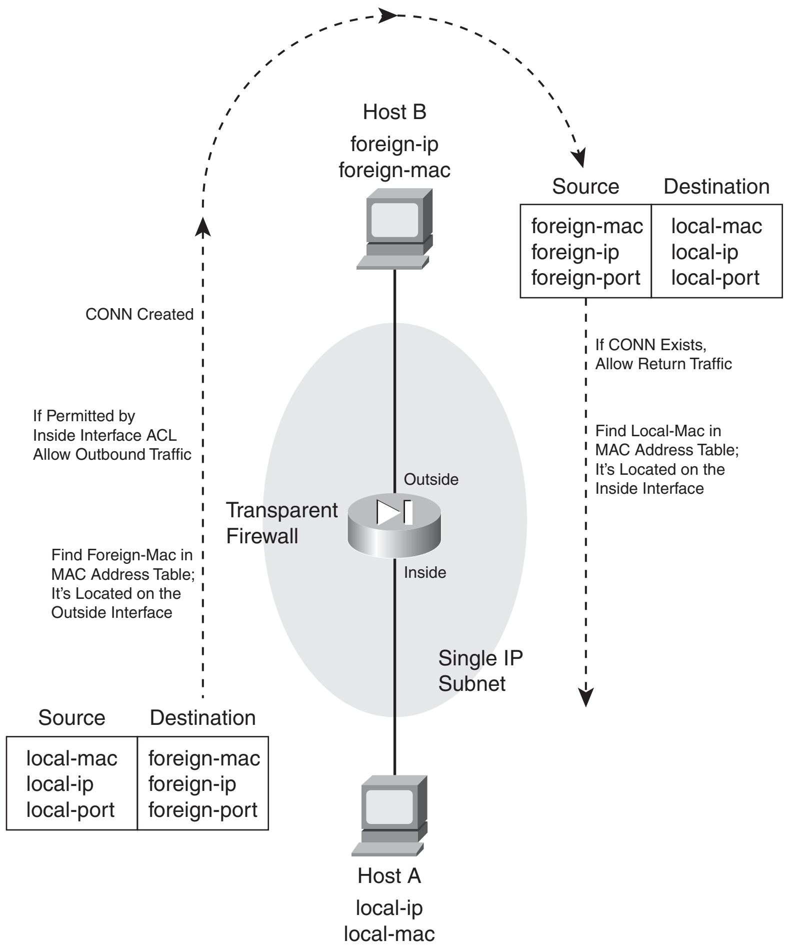
Figure 6-1 illustrates the transparent firewall process. Host A sends a packet to Host B. Notice that Hosts A and B are located on the same IP subnet. Their *local-ip* and *foreign-ip* addresses only designate that "local" is on the inside of the firewall and "foreign" is on the outside. When the firewall builds a conn table entry, the host addresses are shown in this fashion.

TIP In transparent mode, a firewall can support only two interfaces—the *inside* and the *outside*. If your firewall supports more than two interfaces from a physical and licensing standpoint, you can assign the inside and outside to two interfaces arbitrarily. As soon as those interfaces are configured, the firewall does not permit a third interface to be configured.

Some platforms also support a dedicated management interface, which can be used for all firewall management traffic. However, the management interface cannot be involved in accepting or inspecting user traffic.

In multiple context mode, each context also has two interfaces (inside and outside). However, each context must use a pair of interfaces that is different than any other context. In other words, contexts cannot share inside and outside interfaces because of the bridging operation.

Figure 6-1 *Transparent Firewall Operation*



When a firewall enters transparent mode, it no longer supports or has a need for address translation. After all, that is a function based on having different IP subnets on different firewall interfaces. In transparent mode, both interfaces must share the same IP subnet. However, IP packets are still inspected without the Layer 2 limitation. Full extended access lists (IP protocol and port numbers) are used to evaluate traffic policies, and the firewall's application inspection engines are able to interpret IP activity at any layer.

If a transparent firewall acts as a Layer 2 device, is it limited to only IP traffic? From a traffic inspection standpoint, it is; only IP traffic can be inspected and policies enforced. However, from a bridging perspective, the firewall can transparently bridge non-IP packets from one interface to another. This is done by configuring the permitted EtherTypes explicitly in a special interface access list.

A transparent firewall can also maintain an Address Resolution Protocol (ARP) table, where Media Access Control (MAC) and Internet Protocol (IP) addresses are associated as a pair. These are learned from ARP replies that are overheard on an interface. Normally, the ARP table is used only for management traffic to and from the transparent firewall, when the firewall itself needs to send packets to a destination.

You can configure the firewall to support ARP inspection, which can detect and prevent ARP spoofing. ARP requests and replies are forwarded through the firewall by default, as if the two firewall interfaces were bridged. This creates the potential for a malicious host to send spoofed ARP replies with its own MAC address. By doing so, the malicious host can advertise itself as a router's IP address, causing other hosts to send their traffic to it instead of the router.

ARP inspection uses static ARP entries as the basis for its inspection process. The firewall examines each ARP reply packet it overhears and compares the source IP and MAC addresses, as well as the source interface, to known static entries in its own ARP table. If the ARP reply matches an existing entry, it is allowed to pass through the firewall. If any of its information conflicts with an existing entry, the firewall assumes the ARP reply contains spoofed addresses and drops the packet. If an existing ARP entry cannot be found, the firewall can be configured to transmit or drop the ARP reply packet.

Configuring a Transparent Firewall

Use the following steps to configure a firewall for transparent mode.

1. Enter transparent firewall mode:

   ```
   Firewall(config)# firewall transparent
   ```

   By default, a firewall operates in the routed mode. You can use this command to initiate the transparent firewall mode. Transparent mode begins immediately and does not require a firewall reload.

   Because transparent and routed modes use different approaches to network security, the running configuration is cleared as soon as transparent mode begins. The idea is to enter transparent mode and build an appropriate configuration from scratch.

   For that reason, you should save the routed mode running configuration to Flash memory or to an external server. That way, you have a copy in case you need to revert back to routed mode or you need to refer to some portion of that configuration.

You can always display the current firewall mode with the **show firewall** command. As an example, a firewall is configured for transparent mode in the following command sequence:

```
Firewall# show firewall
Firewall mode: Router
Firewall# configure terminal

Firewall(config)# firewall transparent
Switched to transparent mode
Firewall(config)# exit
Firewall#

Firewall# show firewall
Firewall mode: Transparent
Firewall#
```

2. Configure an interface

In transparent mode, none of the firewall interfaces can support an IP address. You still have to configure the necessary interface media parameters (speed and duplex), any virtual LAN (VLAN) information, a logical name, and a security level. These parameters are configured in the following steps:

a. Define a physical interface.

FWSM	N/A

ASA	`Firewall(config)# interface hardware-id`
	`Firewall(config-if)# speed {auto | 10 | 100 | nonegotiate}`
	`Firewall(config-if)# duplex {auto | full | half}`
	`Firewall(config-if)# [no] shutdown`

The interface is referenced by its *hardware-id*, such as **GigabitEthernet0** or **Ethernet0**. (The actual name is not case sensitive when you enter it.)

You can set the interface speed with one of the **auto** (negotiate the speed; the default), **10** (10 Mbps only), **100** (100 Mbps only), or **nonegotiate** (fiber optic interfaces only) keywords. The duplex mode can be set using one of the following keywords: **auto** (autonegotiate; the default)**, full** (full-duplex only), or **half** (half-duplex only).

By default, interfaces are administratively shut down with the **shutdown** command. You can enable an interface by using the **no shutdown** interface configuration command.

b. Define a VLAN interface.

FWSM	`Firewall(config)# interface vlan vlan_id`

ASA	`Firewall(config)# interface hardware_id[.subinterface]`
	`Firewall(config-subif)# vlan vlan_id`

On a FWSM platform, VLAN interfaces are inherent, as it has no physical interfaces at all. Only the VLAN number *vlan_id* needs to be provided.

On an ASA, logical VLAN interfaces must be carried over a physical trunk interface, identified as *hardware-id* (**GigabitEthernet0**, for example). A *subinterface* number is added to the physical interface name to create the logical interface. This is an arbitrary number that must be unique for each logical interface. The VLAN number is specified as *vlan-id* in a separate **vlan** subinterface configuration command.

Packets being sent out a logical VLAN interface are tagged with the VLAN number as they enter the physical trunk link. The VLAN number tag is stripped off at the far end of the trunk, and the packets are placed onto the corresponding VLAN. The same process occurs when packets are sent toward the firewall on a VLAN.

The trunk encapsulation used is always IEEE 802.1Q, and the tagging encapsulation and unencapsulation are automatically handled at each end of the trunk. Make sure the far end switch is configured to trunk unconditionally.

c. Name the interface and assign a security level.

FWSM	`Firewall(config)#` **`nameif`** *`vlan-id if_name`* **`security`***`level`*

ASA	`Firewall(config-if)#` **`nameif`** *`if_name`*
	`Firewall(config-if)#` **`security-level`** *`level`*

On a FWSM platform, the interface is identified by its *vlan-id* (**vlan5** for example; the word **vlan** is always present). If multiple security context mode is being used, the *vlan-id* could be an arbitrary name that has been mapped to the context by the **allocate-interface** command in the system execution space.

The interface is given the arbitrary name *if_name* (1 to 48 characters) that other firewall commands can use to refer to it. For example, you could name an interface as “inside”, “outside”, or something completely different.

A security level is also assigned to the interface as a *level* (a number 0 to 100, from lowest to highest). On a FWSM, the level must be given immediately following the **security** keyword, as in **security** *level*. Security levels 0 and 100 are reserved for the “outside” and “inside” interfaces, respectively. Other perimeter interfaces should have levels between 1 and 99.

As an example, the outside interface could be configured as follows:

FWSM	`Firewall(config)#` **`nameif vlan10 outside security0`**

ASA	`Firewall(config)#` **`interface gigabitethernet0`**
	`Firewall(config-if)#` **`nameif outside`**
	`Firewall(config-if)#` **`security-level 0`**

TIP Security levels are only used to determine how the firewall inspects and handles traffic. For example, traffic passing from a higher security interface toward a lower one is assumed to be going toward a less secure area. Therefore, it is forwarded with less stringent policies and traffic coming in toward a higher security area.

In addition, firewall interfaces must have different security levels, unless the **same-security-traffic permit inter-interface** global configuration command has been used.

3. Configure a management address:

```
Firewall(config)# ip address ip_address subnet_mask
```

The firewall can support only a single IP address for management purposes. The address is not bound to an interface, as in routed mode. Rather, it is assigned to the firewall itself, accessible from either of the bridged interfaces.

The management address is used for all types of firewall management traffic, such as Telnet, SSH, HTTP, SNMP, Syslog, TFTP, FTP, and so on.

TIP A transparent firewall can also support multiple security contexts. In that case, interface IP addresses must be configured from the respective context. The system execution space uses the admin context interfaces and IP addresses for its management traffic.

4. (Optional) Configure routing information for management purposes:

```
Firewall(config)# route if_name foreign_network foreign_mask gateway [metric]
```

You can configure a static route to allow the firewall to send management traffic to addresses that are not on the local IP subnet. Dynamic routing protocols are not supported in the transparent firewall mode.

The remote network can be found on the firewall interface named *if_name* (**outside**, for example), and has network address *foreign_network* and subnet mask *foreign_mask*. The next-hop router address is given as *gateway*. You can also specify a distance *metric*, which is the number of router hops until the gateway is reached. If you omit the *metric*, it defaults to one hop.

You can repeat this command to define other stable, static routes.

You do not have to configure a static route for the subnet directly connected to the firewall interfaces. However, you should define one static route as a default route toward the outside public network. For that, use **0** for the *foreign_network* and *foreign_mask* values. If you have other IP subnets active on the inside of the firewall, you need to define static routes for each of them.

As an example, a firewall is located on the network 10.1.0.0 255.255.0.0. Its next-hop router on the outside interface is 10.1.1.1. A router exists at 10.1.1.2 on the inside of the firewall, where subnets 192.168.1.0/24 and 192.168.100.0/24 can be found. The following commands can be used to configure this routing information on a transparent firewall:

```
Firewall(config)# route outside 0 0 10.1.1.1 1
Firewall(config)# route inside 192.168.1.0 255.255.255.0 10.1.1.2 1
Firewall(config)# route inside 192.168.100.0 255.255.255.0 10.1.1.2 1
```

5. (Optional) Manipulate the MAC address learning process:

By default, a firewall learns MAC addresses as they are received on any interface. You can use the following commands to display the current MAC address learning state and the current MAC address table entries, respectively:

```
Firewall# show mac-learn
Firewall# show mac-address-table [count] [static] [if_name]
```

You can specify a single interface as *if_name* on an ASA or as **interface** *if_name* on a FWSM platform.

As an example, the following output indicates that MAC address learning is enabled on both firewall interfaces, and the MAC address table has been populated with a few entries:

```
Firewall# show mac-learn
interface                         mac learn
-------------------------------------------
 outside                          enabled
 inside                           enabled
Firewall#
Firewall# show mac-address-table
interface     mac  address       type       Age(min)
------------------------------------------------------
outside       00a0.c900.0201     dynamic    5
inside        0050.e2c6.f680     dynamic    4
outside       0008.20f7.fbfc     dynamic    4
Firewall#
```

You can use any of the following steps to modify the MAC address learning process:

a. (Optional) Set the MAC address table aging time:

```
Firewall(config)# mac-address-table aging-time minutes
```

As new addresses are learned, they are placed in the MAC address table. If no traffic is seen to include a MAC address that is already in the table, that entry is aged out and removed after *minutes* (5 to 720 minutes, default 5).

b. (Optional) Define a static MAC address table entry:

```
Firewall(config)# mac-address-table static if_name mac_address
```

You might define a static entry for a MAC address that can never be learned. This might occur if a host or server always listens to traffic using one MAC address and transmits using another. In that case, the transmit address can be learned, while the receiver address cannot.

After a static entry is configured, the firewall always expects that MAC address to be found on the specified interface. If the same MAC address appears on a different firewall interface, the firewall drops those packets.

As an example, the following commands configure two static entries in the MAC address table:

```
Firewall(config)# mac-address-table static inside 0006.5b02.a841
Firewall(config)# mac-address-table static outside 0040.9646.6cf6
```

c. (Optional) Disable MAC address learning on an interface:

```
Firewall(config)# mac-learn if_name  disable
```

By default, MAC address learning is enabled on every firewall interface. You can disable it on a firewall interface named *if_name* (**outside**, for example) so that only static MAC address table entries are used during a packet forwarding operation.

This might be handy in a scenario where you want to keep tight control over the MAC addresses that can be learned dynamically. However, static entries would need to be added for each new host, increasing the amount of firewall administration needed.

6. (Optional) Use ARP inspection

a. Add a static ARP entry:

```
Firewall(config)# arp if_name ip_address mac_address
```

The IP address will be associated with the MAC address (dotted triplet *nnnn.nnnn.nnnn* format). These are expected to be found on the firewall interface named *if_name* (**outside**, for example).

Static ARP entries never age out. To clear a static ARP entry, repeat the command by beginning with the **no** keyword.

TIP The **arp** command syntax also includes an optional **alias** keyword. In routed firewall mode, this makes the firewall use proxy ARP when it gets an ARP request for that address, rather than forward the request on through to the actual target.

In transparent mode, this keyword has no effect. The firewall will not interact with other hosts using Layer 3 mechanisms, so proxy ARP is not possible.

As an example, the following commands configure static ARP entries in preparation for ARP inspection:

```
Firewall(config)# arp inside 192.168.198.199 0006.5b02.a841
Firewall(config)# arp outside 172.21.4.9 0040.9646.6cf6
```

b. Enable ARP inspection on an interface:

```
Firewall(config)# arp-inspection if_name enable [flood | no-flood]
```

By default, ARP inspection is disabled on all firewall interfaces. When enabled, ARP replies received on the interface named *if_name* (**outside** for example) are inspected and matched against known static ARP entries. One of the following actions is taken:

— If the MAC address and the IP address are both found in the ARP table in a single entry, the ARP reply must be valid and is allowed to pass through the firewall.

— If either the MAC address or the IP address is found in the ARP table, but not both in a single entry, the ARP reply contains invalid or spoofed information. Therefore, it is dropped and not forwarded through the firewall.

— If neither MAC nor IP address is found in the ARP table, you can select the action as one of: **flood** (forward or flood the ARP packet out the other firewall interface so it can reach its destination) or **no-flood** (drop the ARP packet without forwarding it). By default, the **flood** action is assumed.

TIP ARP inspection is only effective in handling ARP packets that need to traverse the firewall. In other words, the ARP requester and responder are located on different firewall interfaces. If both hosts are located on the same firewall interface, the ARP reply can be answered directly without having to pass through the firewall.

You can display the ARP inspection status on each interface with the following command:

```
Firewall# show arp-inspection
```

For example, the firewall in the following output has ARP inspection enabled on both interfaces, but only the inside interface is configured to flood unknown ARP replies:

```
Firewall# show arp-inspection
interface                arp-inspection        miss
----------------------------------------------------
outside                  enabled               no-flood
inside                   enabled               flood
Firewall#
```

TIP If you are having trouble getting ARP inspection to work properly, make sure that all of the addresses involved in ARP requests and replies are configured as static ARP entries. If the firewall receives an ARP packet and finds a corresponding entry in its ARP table that was learned dynamically, the ARP packet is dropped. Only static ARP entries are used with ARP inspection.

For example, ARP inspection has been enabled on the outside interface of a firewall. The firewall has already learned the MAC and IP addresses for a router on the outside, and has created a dynamic ARP entry for it. When the router sends an ARP reply toward the outside firewall interface, ARP inspection rejects it. The following syslog output is generated when this happens:

```
%ASA-3-322002: ARP inspection check failed for arp response received from host
0008.20f7.fbfc on interface outside. This host is advertising MAC Address
0008.20f7.fbfc for IP Address 192.168.93.129, which is dynamically bound to
MAC Address 0008.20f7.fbfc
Firewall# show arp
        outside 192.168.93.129 0008.20f7.fbfc
        outside 192.168.93.130 0008.20f7.fb00
        inside 192.168.93.134 0050.e2c6.f680
Firewall#
```

Although it looks like everything matches up with the outside host's addresses, the problem is that the ARP entry was dynamically learned and created.

7. Configure interface access lists.

Before traffic can be forwarded through the firewall, make sure to configure the appropriate security policies by applying an access list to each interface. Access lists are covered in detail in Section 6-3, "Controlling Access with Access Lists."

8. Configure a forwarding policy for non-IP protocols.

a. Define an access list:

FWSM	`Firewall(config)#` **`access-list`** *`acl_id`* **`ethertype`** `{`**`permit`** `|` **`deny`**`} {`**`unicast`** `|` **`multicast`** `|` **`broadcast`** `|` *`ethertype`*`}`
ASA	`Firewall(config)#` **`access-list`** *`acl_id`* **`ethertype`** `{`**`permit`** `|` **`deny`**`} {`**`any`** `|` **`bpdu`** `|` **`ipx`** `|` **`mpls-unicast`** `|` **`mpls-multicast`** `|` *`ethertype`*`}`

By default, only IP packets are allowed to pass through a firewall (providing the packets are permitted by the various inspection processes). You can create an access list that defines a policy on whether non-IP protocols can be forwarded through the firewall.

This command defines an Access Control Entry (ACE) for the access list named *acl_id*. You can **permit** or **deny** the EtherType that is specified. There is always an implicit **deny**

all ACE at the end of every access list. Therefore, all non-IP EtherTypes are dropped unless they are explicitly permitted in an access list.

The EtherType value can be one of the following: **any** (any non-IP packet), **bpdu** (bridge protocol data units, used for Spanning Tree Protocol operation), **ipx** (Novell IPX, 0x8137 and 0x8138), **mpls-unicast** (MPLS unicast, 0x8847), or **mpls-multicast** (MPLS multicast, 0x8848). You can also specify a numeric *ethertype* value, which is a 16-bit hex number greater than 0x600.

TIP Well-known EtherType values are assigned and maintained by the IEEE. You can search or download the most current list of values at standards.ieee.org/regauth/ethertype/index.shtml. Many other values are not publicly registered, and are not shown in that listing.

You can repeat the **access-list** *acl_id* **ethertype** command to add more EtherTypes to be permitted or denied.

b. Apply the EtherType access list to an interface:

```
Firewall(config)# access-group acl_id {in | out} interface if_name
```

The EtherType access list named *acl_id* is applied to the firewall interface named *if_name* (**outside**, for example). Generally this should be done in the **in** (inbound) direction, as the EtherType should be evaluated as packets are received, before being inspected.

You can apply one EtherType access list and one extended IP access list to the same interface.

TIP A Cisco firewall can inspect only IP packets; no inspection is possible for non-IP EtherTypes. This means you have to explicitly permit those EtherTypes in an access list and apply the list to both the inbound and outbound interfaces in the inbound direction.

For example, the following commands configure a firewall to permit IEEE 802.1d Spanning Tree BPDU and Novell IPX traffic, while denying all other non-IP packets. The access list is applied to the inbound direction on both sides of the firewall, allowing bidirectional forwarding.

```
Firewall(config)# access-list MyEthertypes ethertype permit bpdu
Firewall(config)# access-list MyEthertypes ethertype permit ipx
Firewall(config)# access-group MyEthertypes interface in outside
Firewall(config)# access-group MyEthertypes interface in inside
```

6-2: Address Translation

Cisco firewalls provide security policies and traffic inspection using two basic principles:

- **Address translation**—When a host on one firewall interface initiates a connection to a host on a different interface, the firewall must provide a way to translate the IP addresses across itself appropriately. Even if the IP addresses should appear identically on both sides of the firewall, a translation must still occur.

  One exception to this is when the **same-security-traffic** command is used to allow traffic to pass between interfaces with an identical security level. In that case, address translation can still be configured if it is needed, but it is not required. The other exception is when the **no nat-control** command is used. This is the default beginning with ASA 7.0 and FWSM 3.1(1), which allows hosts to initiate connections through the firewall without requiring address translation.
- **Access control**—After an address translation is established, traffic is inspected and allowed only if the appropriate interface access lists permit it.

This section covers the address translation process, which forms the basis for the routed firewall mode, as relationships between inside and outside IP addresses are built as needed.

TIP Address translation is inherent when a firewall is configured for routed mode. Beginning with ASA 8.0, address translation can be used in transparent mode as well.

Defining Access Directions

A firewall differentiates its interfaces by providing more security to some and less security to others. Therefore, it is important to understand how the interfaces relate to each other and how access is provided as traffic moves through a firewall.

TIP By default, all firewall interfaces must be assigned a unique security level value, causing some interfaces to have more security while others have less. Beginning with ASA 7.2(1) and FWSM 2.2(1), you can use the **same-security-traffic permit inter-interface** to configure a firewall such that its interfaces have the same relative level of security. This command is discussed in the "Same-Security Access" section in this chapter.

Outbound Access

Outbound access is defined as connections that are initiated from a higher security interface toward a lower security interface. In other words, users on a more secure network want to connect to something on a less secure network.

Examples of outbound access are connections from the inside (higher security) to the outside (lower security).

The firewall can limit the number of simultaneous connections that are used by an address translation, as well as how many embryonic (not fully initialized) connections can be formed.

You must configure two firewall mechanisms to allow outbound connections:

- **Address translation**—Local (more secure) addresses must be mapped to global (less secure) addresses across two firewall interfaces.
- **Outbound access**—The firewall only builds outbound connections that meet security policy requirements configured as an access list. (ASA and PIX platforms allow outbound connections to be initiated without an access list, by default. The FWSM requires an access list to permit outbound connections.)

Inbound Access

Inbound access is defined as connections that are initiated from a lower security interface toward a higher security interface. In other words, users on a less secure network want to connect to something on a more secure network.

Examples of inbound access are connections from the outside to the inside.

The firewall can limit the number of simultaneous connections that are used by an address translation, as well as how many embryonic (not fully initialized) connections can be formed.

You must configure two firewall mechanisms to allow inbound connections:

- **Address translation**—Local (more secure) addresses must be mapped to global (less secure) addresses across two firewall interfaces.
- **Inbound access**—The firewall allows only inbound connections that meet security policy requirements configured as an access list. You must apply an access list to the lower security interface to permit only the specific inbound connections that are to be allowed.

Same-Security Access

ASA 7.0 and FWSM 2.2(1) introduced the capability to configure multiple interfaces with the same level of security. In this case, it is not easy to classify the traffic passing between same-security interfaces as inbound or outbound.

Why would you ever want to define two or more interfaces as having the same level of security? Perhaps the interfaces support groups of users or resources that should be allowed to freely exchange information. In other words, the user communities are equally trusted and are under the same administrative control.

In addition, Cisco firewalls have a finite number of unique security levels that you can assign to interfaces. Security levels 0 to 100 can be used, representing the lowest to the highest security, respectively. On some firewall platforms, you can arbitrarily define logical firewall interfaces. If your environment needs to support more than 100 different firewall interfaces, you will not be able to assign more than 100 unique security levels. Some of the interfaces will have to be configured with identical security levels.

Same-security access has the following characteristics:

- **Address translation**—You can choose to use or not use address translation between same-security interfaces.
- **Access**—Where many of the firewall inspection features normally limit, filter, or inspect traffic in one direction (inbound or outbound), the same operations can occur in both directions between same-security interfaces.

As well, traffic between same-security interfaces is inherently permitted without any requirement for access lists. To enable traffic to pass between interfaces that have the same security level, use the following global configuration command:

```
Firewall(config)# same-security-traffic permit inter-interface
```

Sometimes you might want to allow traffic to enter and exit the same firewall interface. This can be handy for VPN peers that have tunnels built to the firewall, but need traffic to pass back out to other VPN peers or other networks connected to the same interface.

Firewalls do not normally allow traffic to “hairpin” or come back out the same interface. Beginning with ASA 7.2(1) and FWSM 2.3(1), you can use the following global configuration command to permit hairpin traffic:

```
Firewall(config)# same-security-traffic permit intra-interface
```

In this case, the interface itself is considered to have the same security level in both directions, hence the **intra-interface** keyword.

Types of Address Translation

A firewall can translate the IP addresses of hosts on one interface to identical or different addresses on another interface. This translation does not have to occur in the same fashion for all hosts on all interfaces. In fact, the firewall can be very flexible with address translation, depending on the needs of the hosts, their applications, or the security policies required.

As the firewall builds address translations, it maintains entries in a translation database. These are known as *xlate* entries, and can be displayed by the **show xlate** command. (This command is more fully explained in Chapter 11, Section “11-3: Verifying Firewall Connectivity.”) An xlate entry must exist before inbound connections will be permitted to reach an inside host through an outside address.

For example, in the following output, the firewall is performing two different types of address translation. In the two lines that begin with **Global**, static NAT is being used. The local or inside address is always translated to the same global or outside address, regardless of what protocol or port number is being used.

In the lines that begin with **PAT**, Port Address Translation (PAT) is being used to allow multiple local or inside hosts to be translated to one or more global or outside addresses. The translation is performed dynamically, on a per-connection basis. Each local address and port number used in a connection is translated to the global address, but with a unique global port number. The port numbers are shown in parentheses.

```
Firewall# show xlate
22499 in use, 24492 most used
Global 10.1.1.17 Local 192.168.1.11
Global 10.1.1.16 Local 192.168.1.10
PAT Global 10.1.2.1(10476) Local 192.168.40.251(4705)
PAT Global 10.1.2.1(10382) Local 192.168.48.11(3134)
PAT Global 10.1.2.1(10372) Local 192.168.236.69(1716)
[output omitted]
```

Fully initialized connections are also kept in a connection database. These are known as *conn* entries, shown by the **show conn** command. Before two hosts can communicate through a firewall, an xlate entry must be created, a connection must be permitted by an access list (if one is required on an interface), and a conn entry must be created.

To continue the previous example, the following output from the **show conn** command displays any active connections currently being inspected by the firewall. (This command is more fully explained in Chapter 11, Section "11-3: Verifying Firewall Connectivity.")

```
Firewall# show conn
UDP out 195.242.2.2:53 in 192.168.48.11:3134 idle 0:00:10 flags d
TCP out 207.46.245.60:80 in 192.168.236.69:1716 idle 0:06:18 Bytes 937 flags UIO
[output omitted]
```

The **in** addresses shown in these two lines correspond to the **Local** addresses of the last two lines in the xlate table of the previous example. Inside host 192.168.48.11 is using its UDP port 3134 to open a DNS request with outside host 195.242.2.2 on UDP port 53.

In the second line, inside host 192.168.236.69 has opened a connection to TCP port 80 on outside host 207.46.245.60. Notice that each entry in the conn table also has an idle timer and connection flags. The TCP conn entry also has a byte count, showing the total amount of data that has been sent or received over that connection.

Table 6-1 lists the types of address translation supported by Cisco firewalls, along with the respective configuration commands that can be used. Each translation type inherently allows connections to be initiated in the inbound, outbound, or both directions, as shown in the rightmost column.

Table 6-1 *Address Translation Types Supported by Cisco Firewalls*

Translation Type	Application	Basic Command	Direction Connections Can Be Initiated
Static NAT	Real source addresses (and ports) are translated to mapped addresses (and ports)	**static**	Inbound or outbound
Policy NAT	Conditionally translates real source address (and port) to a mapped address	**static access-list**	Inbound or outbound
Identity NAT	No translation of real source addresses	**nat 0**	Outbound only
NAT Exemption	No translation of real source addresses matched by access list	**nat 0 access-list**	Inbound or outbound
Dynamic NAT	Translates real source addresses to a pool of mapped addresses	**nat** *id* **global** *id address-range*	Outbound only
PAT	Translates real source addresses to a single mapped address with dynamic port numbers	**nat** *id* **global** *id address*	Outbound only

The **static** command creates a persistent translation between a real and a mapped address. This sets the stage to allow both outbound and inbound connections to be initiated. The actual xlate entries are created when the **static** command is entered.

In each of the **nat** command forms shown, the translation is used for *outbound* connections only, initiated by an inside host. Inbound traffic is then permitted only if it is return traffic from an outbound connection or if it is explicitly permitted by an inbound access list applied to the outside interface. One exception is NAT exemption, which allows connections to be initiated in the outbound *and* inbound directions.

Sometimes you might need to use a form of the **nat** command to translate the source addresses of outside hosts that are allowed to initiate connections. You can apply each of the **nat** translation processes in reverse, by adding the **outside** keyword.

TIP In all forms of inside address translation (without the **outside** keyword), only the pertinent addresses on the higher security interface are subject to translation. In other words, the inside source address is translated in the outbound direction, while the inside destination address is translated in the inbound direction.

The **outside** keyword reverses this—only the addresses on the lower security interface are subject to translation. This is often called *Outside NAT* or *Bidirectional NAT*.

You can also configure a firewall to allow two or more firewall interfaces to have an equal security level. In this case, no higher or lower security boundary exists between the two interfaces. Therefore, address translation does not apply between them.

If you configure several address translation operations, you might have some overlap between them. For example, the same local address might appear in more than one NAT definition. To resolve any ambiguity, the firewall evaluates the various types of NAT in the following order before creating an xlate entry:

1. NAT exemptions (**nat 0 access-list** commands)
2. Policy NAT (**static access-list** commands)
3. Static NAT (**static** commands without port numbers)
4. Static PAT (**static** commands with port numbers)
5. Policy NAT (**nat** *nat_id* **access-list** commands)
6. Dynamic NAT and PAT (**nat** *nat_id* commands)

If multiple commands of the same translation type are configured, they are evaluated in sequential order until the first match occurs.

Each type of address translation is described in more detail in the sections that follow.

TIP Be aware that the interface names, address, and port designations have changed in the firewall command syntax related to address translation. In legacy PIX 6.x commands, the terms *local* and *global* are relative to inside and outside interfaces, respectively.

Beginning with FWSM 2.2 and ASA 7.0, address translation is configured using the terms *real* and *mapped*, referring to parameters before and after translation, respectively. Although *real* and *mapped* are shown in this chapter, they can be used interchangeably with *local* and *global*.

Handling Connections Through an Address Translation

Once an address translation is set up across two firewall interfaces, hosts have the potential to open connections through the firewall. Hopefully, hosts that are permitted to traverse the firewall will be on their good behavior and attempt to open only the legitimate connections they need. But if one connection can be initiated, multitudes more might follow, especially if some malicious intent is involved. Fortunately, Cisco firewalls have the capability to enforce connection limits on hosts passing through.

Both the **static** and **nat** commands have parameters that can be configured to define connection limits. You can use the following parameter syntax, which can be found in each form of the **static** and **nat** commands presented in this chapter:

ASA, FWSM	`... [`**`norandomseq`**`] [[`**`tcp`**`]` *`max_conns`* `[`*`emb_limit`*`]] [`**`udp`** `udp_`*`max_conns`*`]`

UDP and TCP Connection Limits

By default, a firewall allows an unlimited number of outbound connections to be opened across an address translation. If this situation is abused, it is possible to open so many connections that cause firewall resources and destination host resources to become exhausted.

On all firewall platforms, you can limit this to *max_conns* (1 to 65535 simultaneous connections, default 0 or unlimited). This becomes the combined total of UDP and TCP connections that are initiated from the inside hosts using the address translation.

You can also define separate UDP and TCP connection limits for each host using the address translation. You can specify the maximum number of TCP connections with the **tcp** keyword followed by *max_conns* (1 to 65535 simultaneous connections, 0 for unlimited). The maximum number of UDP "connections" can be set with the **udp** keyword followed by *udp_max_conns* (1 to 65535, 0 for unlimited). Because UDP is a connectionless protocol, the firewall views each unique pair of host addresses and unique UDP port numbers as a separate connection that is using a conn table entry.

TIP As soon as the connection limit is reached for a host, any subsequent connection attempt is dropped. However, any connections that have already been built are subject to a connection idle timeout. If the firewall does not see any data passing over a connection for a specified time period, that connection is automatically closed.

Separate idle timers are maintained for UDP and TCP connections. You can display the idle timer thresholds with the **show running-config timeout** command, as in the following example. The TCP idle timer is shown as *conn*, while the UDP idle timer is *udp*:

```
Firewall# show running-config timeout
timeout xlate 0:06:00
timeout conn 1:00:00 half-closed 0:10:00 udp 0:02:00 icmp 0:00:02 sunrpc
0:10:00 h323
0:05:00 h225 1:00:00 mgcp 0:05:00 mgcp-pat 0:05:00 sip 0:30:00 sip_media
0:02:00
timeout uauth 0:05:00 absolute
Firewall#
```

You can set the idle timers with the following global configuration command:

```
Firewall(config)# timeout [conn hh:mm:ss] [udp hh:mm:ss]
```

You can set the TCP **conn** timer from 00:05:00 (5 minutes) to 1192:59:59, or 0:00:00 for an unlimited time. The UDP **udp** timer can be set from 00:01:00 (1 minute) to 1192:59:59, or 0:00:00 for an unlimited time. The TCP and UDP timers default to 1 hour and 2 minutes, respectively.

Limiting Embryonic Connections

By default, a firewall allows an unlimited number of TCP connections to be initiated to a target host across an address translation. Recall that a TCP connection has a three-way handshake (SYN-SYN/ACK-ACK) that must be completed between two hosts before the connection can be established. If the handshake sequence is not yet completed, the connection is called an *embryonic* (initialized but not yet formed) connection.

An embryonic connection can result from a handshake that is delayed or lost. Therefore, under normal conditions, hosts maintain the initiated connection while they wait for the handshake completion. A malicious user can also abuse this by attempting to initiate multitudes of embryonic connections to a target host, as a denial-of-service attack. None of the SYN packets used to initiate the connections are ever answered by the malicious user; rather, the idea is to overwhelm the target with too many potential connections while it waits for the originator to answer with the handshake.

A firewall can limit the number of embryonic TCP connections initiated to a host across a translated address. This only applies to *inbound* connections, where outside hosts are initiating TCP connections to inside hosts.

Until this limit is reached, the firewall inspects each SYN packet, adds a new conn table entry (marked as embryonic), and forwards the SYN on to the destination host. If the inside host replies with a SYN/ACK, followed by an ACK from the outside host to complete the TCP connection handshake, then the firewall updates its conn table entry (marked as open connections) and allows the connection to form. If the three-way handshake is not completed within 30 seconds, the firewall deletes the connection entry because of the “SYN timeout”.

However, while the limit is reached or exceeded, the firewall begins to intercept each new SYN packet and answers on behalf of the target inside host. This is not added as an entry in the firewall’s conn table. Instead, an “empty” SYN/ACK packet is returned to the outside host, as if the inside host had sent it. If the originating host actually replies with an ACK, the handshake is completed between the firewall and the inside host and the connection is built.

The firewall keeps track of the initial SYN packets and the SYN/ACK replies it sends by keeping a table of *SYN cookies*, or unique identifiers. In this fashion, the firewall acts as a connection proxy, absorbing the effects of an excessive amount of TCP connection requests.

For address translation with the **static** command, this applies only to *inbound* connections aimed at higher security interfaces. The limit, *emb_limit* (0 to 65535), defaults to 0 (unlimited number of embryonic connections) and is ignored for outbound connections.

The opposite is true for the same embryonic connection limit *emb_limit* parameter in the **nat** command, which is applied to *outbound* connections aimed toward lower security interfaces. Here, you can limit the potential for denial-of-service attacks initiated by inside hosts.

TCP Initial Sequence Numbers

By default, when the firewall creates new *outbound* TCP connections, it assigns a randomized TCP initial sequence number (ISN). This is useful to prevent outside users from being able to predict or guess the sequence number and hijack a connection.

Normally, hosts provide their own random ISNs when they initiate new TCP connections. However, the TCP/IP protocol stack in some operating systems has a weak implementation of this, allowing the ISN to be predicted. The firewall maintains the original ISN for use with the originating host and overwrites this value for use with the destination host. Therefore, neither the originating nor target host is aware that the ISN has been altered or further randomized.

Sometimes this additional ISN operation interferes with a protocol that is passing through a firewall. For example, some protocols such as BGP use a packet authentication method such as MD5 to preserve the integrity of a message. The originating host computes a hash value over the whole TCP packet, including the original ISN, and includes this within the packet payload. To authenticate the message, the receiving host should be able to recompute the hash and get the same value.

However, if the ISN has been randomized after the original hash value was computed, a different hash value will result and the packet authentication will fail. You can use the **norandomseq** keyword to keep the local firewall from changing the ISN, so that only one firewall is randomizing it.

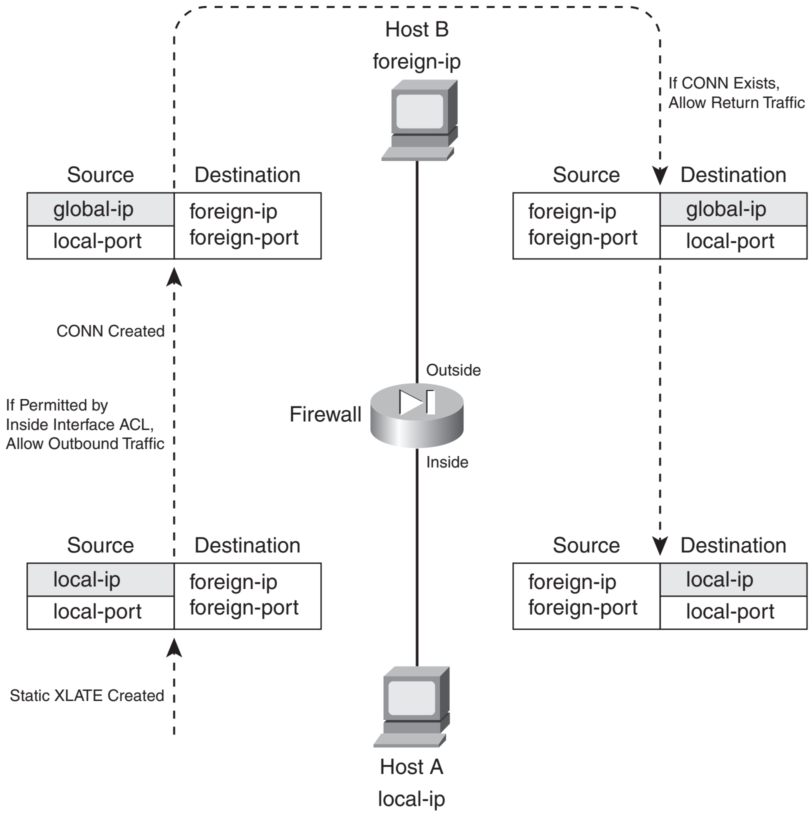
TIP You should always depend on the additional security provided by a firewall's ISN randomization, unless you notice that it is creating problems with a protocol or application. Only then, consider using the nondefault **norandomseq** setting to disable the randomization.

Static NAT

Static NAT can be used when an internal or real host needs to have the same mapped address for every outbound connection that is initiated. As well, inbound connections can also be initiated to the internal host, if they are permitted by security policies.

Address translation occurs on a one-to-one persistent basis. Each static translation that is configured causes a static xlate entry to be created. Figure 6-2 illustrates static NAT operation during an outbound connection. Inside Host A is initiating a connection to outside Host B. Notice that only the real (local) IP address is being translated, as the source address in the outbound direction, and as the destination address in the inbound direction for return traffic.

Figure 6-2 *Static NAT Operation Across a Firewall*



You can use the following command to configure a static NAT entry:

ASA, FWSM	`Firewall(config)#` **`static (`**`real_ifc`**`,`**`mapped_ifc`**`)`** `{`*`mapped_ip`* `|` **`interface`**`} {`*`real_ip`* `[`**`netmask`** *`mask`*`]} [`**`dns`**`] [`**`norandomseq`**`] [[`**`tcp`**`]` *`max_conns`* `[`*`emb_limit`*`]] [`**`udp`** *`udp_max_conns`*`]`
PIX 6.3	`Firewall(config)#` **`static (`**`real_ifc`**`,`**`mapped_ifc`**`)`** `{`*`mapped_ip`* `|` **`interface`**`} {`*`real_ip`* `[`**`netmask`** *`mask`*`]} [`**`dns`**`] [`**`norandomseq`**`] [`*`max_conns`* `[`*`emb_limit`*`]]`

A static NAT entry is created across the firewall interfaces named *real_ifc* (**inside** for example) and *mapped_ifc* (**outside** for example). The real IP address *real_ip* is translated to the mapped IP address *mapped_ip* only when the firewall needs to forward a packet between the *real_ifc* and *mapped_ifc* interfaces. The addresses can be a single IP address (use **netmask 255.255.255.255**) or an entire

IP subnet address (use **netmask** with the correct subnet mask). By default, if the **netmask** keyword is omitted, a host mask is assumed.

This command causes the address translation to be carried out regardless of the IP protocol or port number being used. If you need a static translation only for a specific UDP or TCP port number, you can define a static PAT entry with the following command:

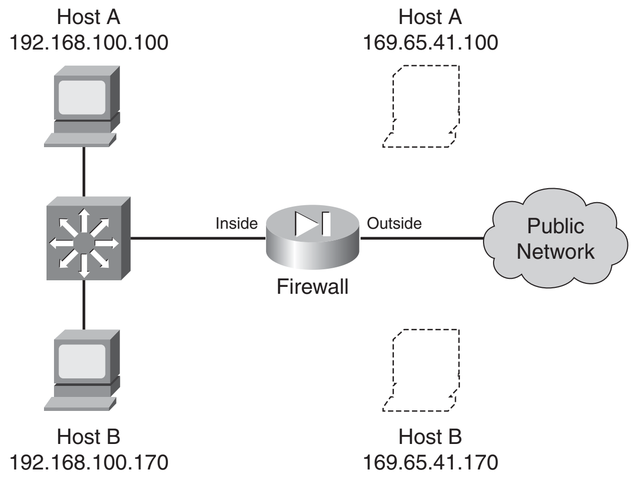
ASA, FWSM	`Firewall(config)#` **`static (`**`real_ifc`**`,`**`mapped_ifc`**`)`** `{`**`tcp`** `|` **`udp`**`} {`*`mapped_ip`* `|` **`interface`**`}` *`mapped_port`* `{`*`real_ip real_port`* `[`**`netmask`** *`mask`*`]} [`**`dns`**`] [`**`norandomseq`**`] [[`**`tcp`**`]` *`max_conns`* `[`*`emb_limit`*`]] [`**`udp`** *`udp_max_conns`*`]`
PIX 6.3	`Firewall(config)#` **`static (`**`real_ifc`**`,`**`mapped_ifc`**`)`** `{`**`tcp`** `|` **`udp`**`} {`*`mapped_ip`* `|` **`interface`**`}` *`mapped_port`* `{`*`real_ip real_port`* `[`**`netmask`** *`mask`*`]} [`**`dns`**`] [`**`norandomseq`**`] [`*`max_conns`* `[`*`emb_limit`*`]]`

Now the firewall translates the *real_ip* and *real_port* to the *mapped_ip* and *mapped_port* values.

The firewall can inspect and alter DNS packets if the **dns** keyword is added. If the real address is found in the packet, it is rewritten with the mapped address.

As an example, consider two hosts that reside on the inside of a firewall, using private IP addresses 192.168.100.100 and 192.168.100.170. Outbound connections from these hosts should appear as 169.65.41.100 and 169.65.41.170, respectively. Because the hosts must always receive the same mapped addresses, static NAT should be used. Figure 6-3 shows a network diagram for this scenario.

Figure 6-3 *Network Diagram for the Static NAT Example*



The static NAT entries could be configured with the following commands:

```
Firewall(config)# static (inside,outside) 169.65.41.100 192.168.100.100 netmask
255.255.255.255 0 0
Firewall(config)# static (inside,outside) 169.65.41.170 192.168.100.170 netmask
255.255.255.255 0 0
```

The netmask is given as a host mask (255.255.255.255), because each translation is applied to a single host address.

To extend this example further, suppose inbound SMTP and HTTP connections to 169.65.41.100 could be sent to two separate inside hosts, each handling one of the two types of connections. Static PAT is a good solution for this scenario. With the following commands, SMTP connections are translated to inside host 192.168.100.100, while HTTP connections are translated to 192.168.100.200. An access list is also applied to the outside interface, to permit inbound SMTP and HTTP connections.

```
Firewall(config)# static (inside,outside) tcp 169.65.41.100 smtp 192.168.100.100 smtp
netmask 255.255.255.255 0 0
Firewall(config)# static (inside,outside) tcp 169.65.41.100 www 192.168.100.200 www
netmask 255.255.255.255 0 0
Firewall(config)# access-list acl_outside permit tcp any host 169.65.41.100 eq smtp
Firewall(config)# access-list acl_outside permit tcp any host 169.65.41.100 eq www
Firewall(config)# access-group acl_outside in interface outside
```

TIP Notice that the order of the two addresses is reversed from the order of the two interfaces, implying that the IP addresses should be different. If an inside host has an IP address that can appear on the outside without being translated, you can enter *real_ip* and *mapped_ip* as the same address.

However, you should do that only if you intend to permit inbound connections to that address. A static NAT defined with identical addresses creates an xlate entry that can allow hosts on the outside to access the inside host.

If no address translation is needed, a better solution is to use identity NAT (the **nat 0** command) or NAT exemption (the **nat 0 access-list** command).

If inbound connections are not needed, you should define an identity NAT. Xlate entries are only created when connections are initiated from the inside, offering a more secure solution. If inbound and outbound connections should be allowed to initiate, NAT exemption is the better choice.

You can also configure a static NAT entry based on the mapped (global) firewall interface, even if its address is not known ahead of time. In that case, you can use the **interface** keyword to translate the address it pulled from a DHCP server. For example, the following command translates the outside interface address to the inside host address 192.168.100.100. No matter what IP address the outside interface has, the translation takes place with the correct value.

```
Firewall(config)# static (inside,outside) interface 192.168.100.100 netmask
255.255.255.255
```

Policy NAT

You can use policy NAT when real addresses need to be translated to several different mapped addresses, depending on a policy decision. An access list is used to trigger the address translation only when a match is permitted. Policy NAT can be configured in two ways:

- A conditional **static** command, where inside real addresses are translated to predictable mapped addresses, depending on the outcome of an access list. This form of translation can be used if inbound connections are expected and permitted to the inside hosts.
- A conditional **nat** command, where inside real addresses are translated to different mapped addresses defined in **global** commands, depending on the outcome of an access list. Use this form if the inside hosts are only expected to initiate outbound connections; inbound connections to those hosts will not be allowed.

You can use the following steps to configure a policy NAT:

1. Identify the translation policy:

   ```
   Firewall(config)# access-list acl_name permit ip real-ip real_mask  foreign_ip
   foreign_mask
   ```

   An access list named *acl_name* is used to identify traffic by source (*real_ip real_mask)* and destination (*foreign_ip foreign_mask*). When an outbound packet triggers the **permit** statement, a matching policy NAT **static** or **nat** command is also triggered. The *real_ip* address given here is the address that is ultimately translated. You can substitute the **host** *real_ip* keyword pair if the source is a single host.

   You use *foreign_ip* and *foreign_mask* to define a destination host or a whole subnet on the public network. You can also substitute the **host** *foreign_ip* keyword pair if the destination is a single host.

   You can repeat this **access-list** command to define other source/destination combinations that trigger a matching **static** command.

2. (**static** Only) Define the **static** command translation:

ASA, FWSM	`Firewall(config)# static (real_ifc,mapped_ifc) mapped_ip access-list acl_name [dns] [norandomseq] [[tcp] max_conns [emb_limit]] [udp udp_max_conns]`
PIX 6.3	`Firewall(config)# static (real_ifc,mapped_ifc) mapped_ip access-list acl_name [dns] [norandomseq] [max_conns [emb_limit]]`

   A conditional static NAT or policy NAT translation is defined across the firewall interfaces named *real_ifc* and *mapped_ifc*. Here, the *mapped_ip* address replaces the *real_ip* address matched in the access list named *acl_name*.

   You can repeat this command to define multiple NAT policies. Each **static** command should reference a different access list.

3. (**nat** Only) Define the **nat** command translation.

a. Configure a global address:

```
Firewall(config)# global (mapped_ifc) nat_id {global_ip [-global_ip] [netmask
global_mask]} | interface
```

Global IP addresses are used as mapped or translated addresses, and are defined as a single address (*global_ip*) or a range of addresses (*global_ip-global_ip*). The global definition must be identified with a NAT ID *nat_id* (1 to 2,147,483,647), which is linked to **nat** commands with the same value.

The destination or mapped interface is given as *mapped_ifc* (**outside**, for example), complete with surrounding parentheses. Therefore, NAT occurs for traffic that matches a policy and also exits this interface.

You can specify a subnet mask as *global_mask*, so that the firewall automatically excludes the network and broadcast addresses from the range of global addresses given. You can also use the **interface** keyword to use the mapped interface's IP address as the global address. In this case, the translation is performed using PAT, as many real IP addresses could become translated to the single interface address.

b. Configure a NAT translation:

ASA, FWSM	`Firewall(config)# nat (real_ifc) nat_id access-list acl_name [dns] [outside] [[tcp] max_conns [emb_limit]] [norandomseq] [udp udp_max_conns]`
PIX 6.3	`Firewall(config)# nat (real_ifc) nat_id access-list acl_name [dns] [outside] [norandomseq] [max_conns [emb_limit]]`

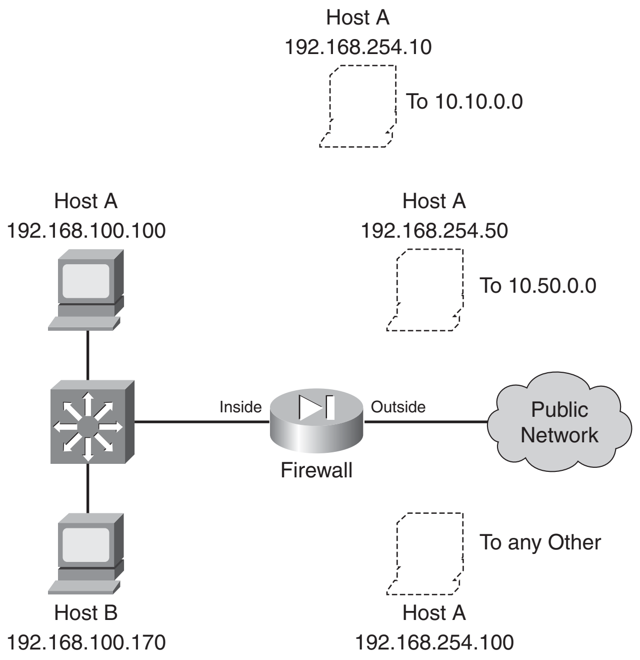
Define the NAT translation to occur at the local or real interface named *real_ifc* (**inside**, for example). The address translation will use mapped addresses defined in **global** commands using the same NAT ID *nat_id* as is given here.

The translation only occurs if the extended access list *acl_name* matches a permit statement. You can match against source and destination addresses and port numbers.

As an example, suppose two hosts reside on the inside of a firewall, using private IP addresses 192.168.100.100 and 192.168.100.170. Outbound connections from Host A should appear as different global addresses, depending on the destination of the connection. The inside network interfaces with several different external business partners, each expecting Host A to reside in a different address space. This is a good application for *policy NAT*, also called *conditional NAT*.

If Host A opens a connection to the 10.10.0.0/16 network, it should appear as global address 192.168.254.10. If Host A opens a connection to the 10.50.0.0/16 network, it should appear as 192.168.254.50. Lastly, if Host A opens a connection to any other destination, it should appear as global address 192.168.254.100. Figure 6-4 shows a network diagram for this scenario.

Figure 6-4 *Network Diagram for the Policy NAT Example*



First, policy NAT using **static** commands is considered. The policy NAT entries could be configured with the following commands:

```
Firewall(config)# access-list hostApolicy10 permit ip host 192.168.100.100
10.10.0.0 255.255.0.0
Firewall(config)# static (inside,outside) 192.168.254.10 access-list
hostApolicy10 0 0
Firewall(config)# access-list hostApolicy50 permit ip host 192.168.100.100
10.50.0.0 255.255.0.0
Firewall(config)# static (inside,outside) 192.168.254.50 access-list
hostApolicy50 0 0
Firewall(config)# static (inside,outside) 192.168.254.100 192.168.100.100 netmask
255.255.255.255 0 0
```

If ACL **hostApolicy10** matches and permits traffic, Host A is translated to 192.168.254.10. If ACL **hostApolicy50** matches and permits traffic, Host A is translated to 192.168.254.50.

You might be inclined to define a third policy access list that denies the other two conditions, and then permits everything else. However, policy NAT will not accept an access list that contains **deny** statements; the idea is to permit the traffic where you need a translation. Instead, you can use a regular static NAT (without an access list) to define the third condition. Now the inside address 192.168.100.100 has been defined in three different address translation commands. This works because the more specific static translations (policy NAT) are evaluated first, followed by regular static NAT.

Finally, policy NAT with **nat** commands is used. You could use the following configuration commands:

```
Firewall(config)# access-list hostApolicy10 permit ip host 192.168.100.100
10.10.0.0 255.255.0.0
Firewall(config)# global (outside) 1 192.168.254.10 255.255.255.255
Firewall(config)# nat (inside) 1 access-list hostApolicy10
Firewall(config)# access-list hostApolicy50 permit ip host 192.168.100.100
10.50.0.0 255.255.0.0
Firewall(config)# global (outside) 2 192.168.254.50 255.255.255.255
Firewall(config)# nat (inside) 2 access-list hostApolicy50
Firewall(config)# access-list hostApolicy100 permit ip host 192.168.100.100 any
Firewall(config)# global (outside) 3 192.168.254.100 255.255.255.255
Firewall(config)# nat (inside) 3 access-list hostApolicy100
```

Traffic passing from host 192.168.100.100 to the 10.10.0.0/16 subnet, for example, matches the permit statement in access list *hostApolicy10*, which triggers the **nat** command with ID 1. This causes the inside host address to be translated to the address defined in global ID 1, 192.168.254.10.

Identity NAT

Identity NAT can be used when the real host and the mapped address are identical. In other words, the same IP subnet appears on both sides of the firewall. This is useful if you have registered IP addresses on the inside, and you have no need to translate them on the outside.

You can use the following command to configure an identity NAT:

ASA, FWSM	`Firewall(config)# nat (real_ifc) 0 real_ip real_mask [dns] [norandomseq] [[tcp] max_conns [emb_limit]] [udp udp_max_conns]`
PIX 6.3	`Firewall(config)# nat (real_ifc) 0 real_ip real_mask [dns] [norandomseq] [max_conns [emb_limit]]`

Notice that the *nat_id* here is always zero. This is a special case of the translation policy, one that does not require a corresponding **global** command.

Recall that the **static** command can also set up an identity NAT, where the real address appears unchanged on the mapped side. In other words, no real NAT is taking place.

What is the difference between using **static** and using **nat 0**, if both prevent NAT from occurring? When the **static** command defines an identity NAT, connections involving the real address can be initiated in *both* directions through the firewall (assuming the connections are permitted by access lists).

As soon as the **static** command is entered, the firewall creates static **xlate** entries when the real hosts attempt outbound connections that are permitted through the firewall. Likewise, it is also possible for outside hosts to reach the real hosts in the inbound direction, because the xlate entries will still be created.

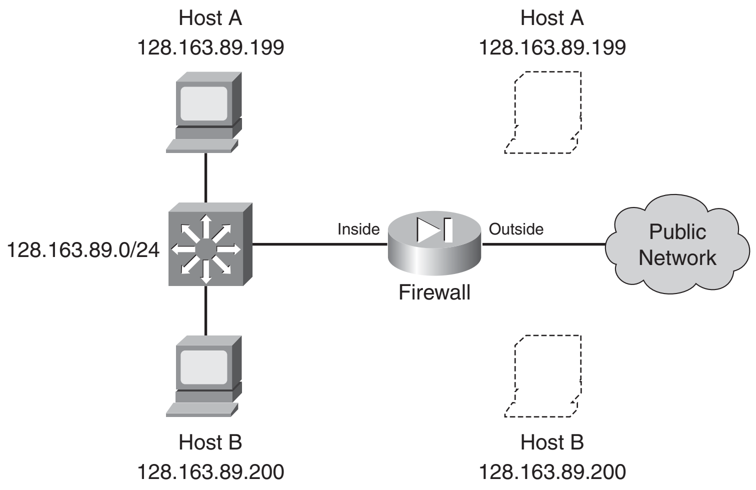
If you define the same real host with a **nat 0** command, that host can only initiate *outbound* connections. No inbound connections are allowed. Therefore, the **nat 0** command sets up a one-way path without translating the real address.

As a last note, you should avoid configuring both **static** and **nat 0** commands for the same real addresses. It might seem logical to define both to prevent NAT from occurring, but the two methods are really mutually exclusive.

As an example, consider two hosts that reside on the inside of a firewall. The inside network uses a publicly registered IP subnet of 128.163.89.0/24. For this reason, no address translation is necessary, as the inside hosts can appear on the outside with publicly routable addresses.

An identity NAT can be used in this case. For example, Host A at 128.163.89.199 on the inside will also appear as 128.163.89.199 on the outside. In fact, the whole subnet will be defined in a similar manner. Figure 6-5 shows a network diagram for this scenario.

Figure 6-5 *Network Diagram for the Static Identity NAT Example*



If both outbound *and* inbound connections should be possible, you should consider using the **nat 0 access-list** command to define a NAT exemption. This is discussed fully in the next section "NAT Exemption." The NAT exemption entries for the whole subnet could be configured as follows:

```
Firewall(config)# access-list ExemptList permit ip 128.163.89.0 255.255.255.0 any
Firewall(config)# nat (outside) 0 access-list ExemptList
```

You should also configure the appropriate access lists to permit the inbound and outbound connections for this subnet.

TIP Legacy PIX 6.3 platforms create xlate entries for each individual host contained in the identity NAT subnet. For large subnets, the number of static xlate entries can grow quite large. Beginning with ASA 7.0, the firewall builds a single xlate entry representing an entire subnet.

To configure the identity NAT for outbound use only, you could use the following command:

```
Firewall(config)# nat (inside) 0 128.163.89.0 255.255.255.0
```

The subnet mask, given as 255.255.255.0, defines the extent of the addresses that have identity translation entries. It also allows the firewall to prevent translations from being built for the network (128.163.89.0) and broadcast (128.163.89.255) addresses.

NAT Exemption

Sometimes you might have specific real (local) IP addresses that need to bypass NAT and appear untranslated. This might be needed only for individual IP addresses or for unique traffic flows. NAT exemption is similar to an Identity NAT, where the real and mapped IP addresses are identical. However, NAT exemption uses an access list to define a policy for bypassing translation.

Unlike identity NAT, which allows connections to be initiated only in the *outbound* direction, NAT exemption allows connections to be initiated in *either* the inbound or outbound direction.

NAT exemption is most often used in conjunction with VPN connections. Inside addresses might normally be translated for all outbound connections through a firewall. If a remote network is reachable through a VPN tunnel, the inside hosts might need to reach remote VPN hosts without being translated. NAT exemption provides the policy mechanism to conditionally prevent the address translation.

You can use the following steps to configure NAT exemption:

1. Define the policy with an access list:

ASA, FWSM	`Firewall(config)# access-list acl_name [extended] permit ip local_ip local_mask foreign_ip foreign_mask`
PIX 6.3	`Firewall(config)# access-list acl_name permit ip local_ip local_mask foreign_ip foreign_mask`

Local addresses that are permitted by an entry in the access list are exempted from translation. Normally, you should only configure **permit** statements as a part of the NAT exemption access list. (Although **deny** statements are allowed, you would really be defining conditions to deny when NAT should be denied!)

In addition, only the **ip** protocol is allowed in the access list. NAT exemption is evaluated based on source and destination addresses—not on IP protocols or port numbers.

2. Add the access list to the policy:

ASA, FWSM	`Firewall(config)# nat (real_ifc) 0 access-list acl_name [dns] [outside] [[tcp] max_conns [emb_limit] [norandomseq]] [udp udp_max_conns]`
PIX 6.3	`Firewall(config)# nat (real_ifc) 0 access-list acl_name [dns] [outside] [max_conns [emb_limit] [norandomseq]]`

Packets permitted by the access list named *acl_name* are exempted from translation. In other words, those packets are passed on out a different firewall interface with the original real address unchanged.

Notice that the *nat_id* here is always zero. This is a special case of the translation policy, one that does not require a corresponding **global** command.

As a last note, you should avoid configuring both **static** and **nat 0** commands for the same real addresses. It might seem logical to define both to prevent NAT from occurring, but the two methods are really mutually exclusive.

As an example, a firewall is configured to use PAT on all outbound traffic. However, inside addresses in the 192.168.1.0/24 subnet should not be translated when connections are initiated to the 192.168.77.0/24 and 192.168.100.0/24 subnets. The following commands can be used to configure both PAT and NAT exemption:

```
Firewall(config)# nat (inside) 1 0 0
Firewall(config)# global (outside) 1 interface
Firewall(config)# access-list exempt1 permit ip 192.168.1.0 255.255.255.0
192.168.77.0 255.255.255.0
Firewall(config)# access-list exempt1 permit ip 192.168.1.0 255.255.255.0
192.168.100.0 255.255.255.0
Firewall(config)# nat (inside) 0 access-list exempt1
```

Although two different address translation methods are configured, no conflict exists between them regarding the translation of 192.168.1.0/24 hosts. This is because of the order that NAT operations are performed. NAT exemption is always evaluated before any other translation type.

Dynamic Address Translation (NAT or PAT)

Dynamic address translation can be used to allow hosts with real addresses to share or “hide behind” one or more common mapped addresses. Address translation occurs on a many-to-one basis, in a dynamic fashion. This can be accomplished in two ways:

- **Dynamic NAT**—Inside host addresses are translated to values pulled from a pool of mapped addresses. Each inside address gets exclusive use of the mapped address it is assigned, for the duration of any active connections. As soon as all of a host’s connections are closed, that mapped address is returned to the pool.

  This means that all inside hosts must compete for the use of the mapped addresses. If the mapped address pool is too small, some hosts could be denied because their translations could not be set up.

- **Dynamic PAT**—Inside host addresses are translated to a single mapped address. This is possible because the inside port numbers can be translated to a dynamically assigned port number used with the mapped address.

  Because port numbers are used as part of the translation, each dynamic PAT entry can support only a single connection (protocol and port number) from a single inside host. In other words, if one inside host initiates two outbound connections, two PAT entries are created—each using a unique port number with the mapped address.

  When a connection is closed, its dynamic PAT entry is deleted after 30 seconds. The mapped port number becomes available for use again.

  Each mapped address has the potential to provide up to 65,535 dynamic PAT entries for a single IP protocol (UDP or TCP, for example), because that many unique port numbers are available. Additional mapped addresses can be used, adding 65,535 more port numbers to the pool. As soon as the port numbers from one mapped address have been exhausted, the next configured mapped address will be used.

Figure 6-6 illustrates dynamic NAT, where Host A is initiating a connection to Host B. Notice that the *real-ip* is translated to *mapped-ip-n,* which is one of a possible pool of mapped addresses. The real-port is not translated, however, because it is still unique to the mapped address. For return traffic, the firewall must translate the destination address back to the original *real-ip.*

Figure 6-7 illustrates dynamic PAT. Notice the procedure is almost identical to that of dynamic NAT in Figure 6-6. The difference is that a dynamic *mapped-port* value is used, rather than a dynamic *mapped-ip.* The combination of mapped IP and port numbers keep the connection unique so that it can be translated back to the real address and port for return traffic.

For dynamic translation (either NAT or PAT), you configure the mapped addresses that can be used, along with the translation policy that will trigger the translation.

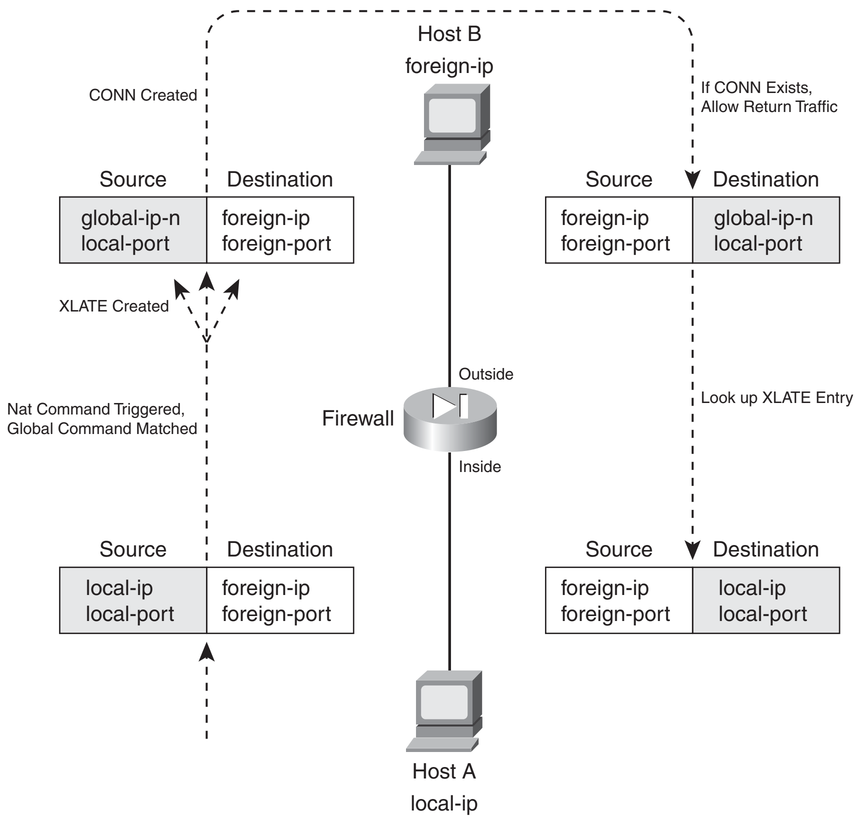
Mapped addresses are defined in groups, where *nat_id* (1 to 2,147,483,647) is a group index that corresponds to a matching translation policy. You can repeat the **global** commands that follow with the same *nat_id* to define more mapped addresses to use for the translation policy.

1. Define mapped addresses for NAT:

   ```
   Firewall(config)# global (mapped_ifc) nat_id global_ip[-global_ip] [netmask
   global_mask]
   ```

   You can use mapped addresses that are located on the firewall interface named *mapped_ifc* (**outside**, for example) for address translation. You can define a single *global_ip* address or a range of addresses as the starting and ending addresses *global_ip*-*global_ip*. A subnet mask can be given with the **netmask** keyword, where *global_mask* matches the mask in use on the global IP subnet. If the mask is given, it is used to determine and reserve the network and broadcast addresses so that they are not used for translation.

Figure 6-6 *Dynamic PAT Operation Across a Firewall*



As an example, the following command can be used to configure 10.1.2.1 through 10.1.2.254 as global (mapped) addresses on the outside interface for NAT ID 1. These addresses will be used for translation triggered by the corresponding NAT policy for NAT ID 1, with the **nat 1** command.

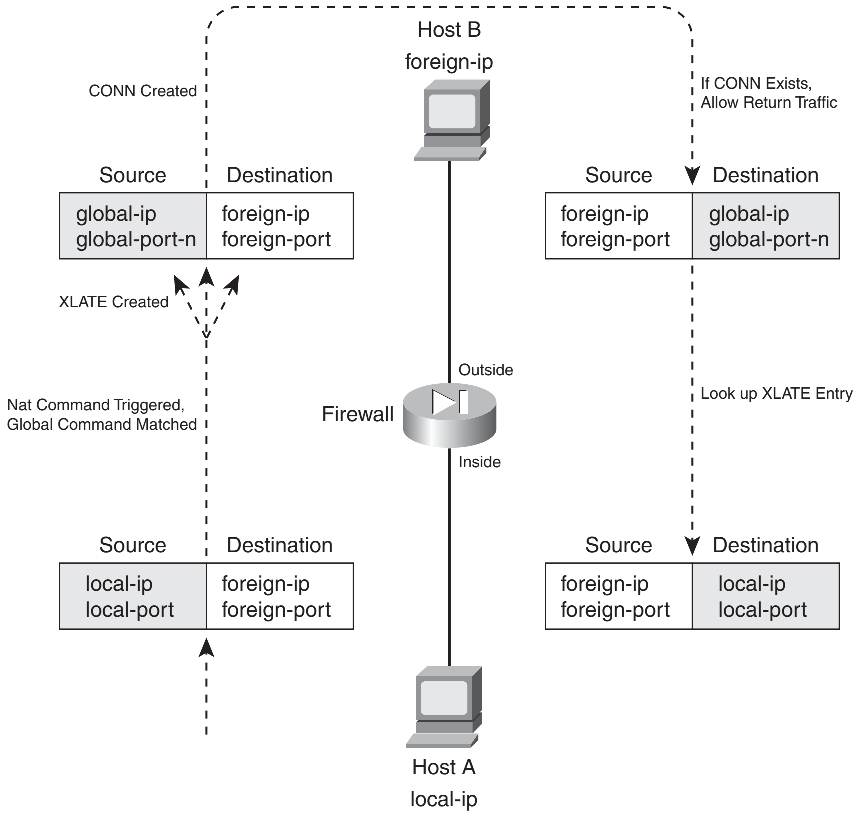
```
Firewall(config)# global (outside) 1 10.1.2.1-10.1.2.254 netmask 255.255.255.0
```

TIP The global addresses can be located on the IP subnet assigned to the firewall interface, although it is not required. You can also use other addresses, as long as devices on the outside network can route those addresses back to the firewall interface.

2. Define one or more mapped addresses for PAT:

```
Firewall(config)# global (mapped_ifc) nat_id {global_ip | interface}
```

Figure 6-7 *Dynamic PAT Operation Across a Firewall*



You can also specify single mapped addresses for PAT, where many real addresses are translated to or "hide behind" a single mapped address. To do this, use the **global** command with a single *global_ip* address. You can repeat the command to define other single mapped addresses to use for PAT. As soon as one mapped address is exhausted of its port numbers, the next mapped PAT address is used.

For example, the following commands set aside three mapped addresses for dynamic PAT usage in NAT group 2:

```
Firewall(config)# global (outside) 2 130.65.77.24
Firewall(config)# global (outside) 2 130.65.77.25
Firewall(config)# global (outside) 2 130.65.77.26
```

You can also use the **interface** keyword to use the IP address of the interface itself as a PAT address. This is handy when the firewall requests a dynamic IP address from a service provider and the address is not known ahead of time. The following command shows an example, where the outside interface address is added to the list of mapped PAT addresses in NAT group 2:

```
Firewall(config)# global (outside) 2 interface
```

TIP If a range of mapped addresses is defined, the firewall uses these addresses first to create new address translations. These are used only for dynamic NAT, where a single local address is translated to a single mapped address. As soon as the connections belonging to a translation close or time out, that mapped address is released back into the pool of available addresses.

If all of the dynamic NAT mapped addresses are in use, the firewall begins creating translations based on any single dynamic PAT mapped addresses that are defined.

3. Define a translation policy.

Outbound address translation occurs when a packet is sent from a real interface that has a **nat** policy to a mapped interface that has a **global** definition. The **nat** and **global** definitions must match by having the same *nat_id* index.

In the translation policy **nat** commands, the real firewall interface is named *real_ifc*. Do not forget the parentheses, as in **(inside)**.

The firewall can inspect and alter DNS packets if the **dns** keyword is added. If the real address is found in the packet, it is rewritten with the translated or mapped address.

The following command syntax defines a translation policy:

ASA, FWSM	`Firewall(config)#` **`nat (`***`real_ifc`***`)`** *`nat_id real_ip`* `[`*`mask`* `[`**`dns`**`] [`**`outside`**`][[`**`tcp`**`]` *`max_conns`* `[`*`emb_limit`*`] [`**`norandomseq`**`]] [`**`udp`** *`udp_max_conns`*`]`
PIX 6.3	`Firewall(config)#` **`nat (`***`real_ifc`***`)`** *`nat_id real_ip`* `[`*`mask`* `[`**`dns`**`] [`**`outside`**`] [[`**`norandomseq`**`] [`*`max_conns`* `[`*`emb_limit`*`]]]`

The real address is defined as *real_ip*, which can be a single IP address or a subnet address with a subnet *mask*.

As an example, the following command can be used to trigger dynamic NAT or PAT when inside addresses in the 192.168.100.0/24 subnet initiate outbound connections. This NAT policy uses NAT ID 1; the corresponding **global** command must also use NAT ID 1.

```
Firewall(config)# nat (inside) 1 192.168.100.0 255.255.255.0
```

You can also use an access list to trigger the dynamic NAT or PAT translation. This allows the translation to be conditional as well as dynamic. First, define an access list with the following command:

ASA, FWSM	`Firewall(config)#` **`access-list`** *`acl_name`* `[`**`extended`**`]` **`permit`** *`protocol real_ip real_mask`* `[`*`operator port`*`]` *`foreign_ip foreign_mask`* `[`*`operator port`*`]`
PIX 6.3	`Firewall(config)#` **`access-list`** *`acl_name`* **`permit`** *`protocol real_ip real_mask`* `[`*`operator port`*`]` *`foreign_ip foreign_mask`* `[`*`operator port`*`]`

The real source addresses will be candidates for dynamic NAT or PAT if they are matched by a **permit** statement in the access list. You can only use **permit** statements when you configure the access list.

However, you can be specific in the access list by specifying an IP *protocol* (**tcp** or **udp**, for example) and source and destination operators (**eq**, for example) and *port* numbers. These values are not used in the actual dynamic NAT or PAT operation; they are only used to define the traffic that triggers the translation.

As soon as the access list is configured, add it to the translation policy with the following command syntax:

ASA, FWSM	`Firewall(config)#` **`nat (`***`real_ifc`***`)`** *`nat_id`* **`access-list`** *`acl_name`* `[`**`dns`**`] [`**`outside`**`][[`**`tcp`**`]` *`max_conns`* `[`*`emb_limit`*`] [`**`norandomseq`**`]] [`**`udp`** *`udp_max_conns`*`]`
PIX 6.3	`Firewall(config)#` **`nat`** `[`**`(`***`local_interface`***`)`**`]` *`nat_id`* **`access-list`** *`acl_name`* `[`**`dns`**`] [`**`norandomseq`**`] [`*`max_conns`* `[`*`emb_limit`*`]]]`

The access list named *acl_name* is used by the translation policy to identify packets to be translated. For example, the following commands cause dynamic translation with NAT ID 1 to be used when inside hosts in 192.168.1.0/24 initiate connections to the 10.1.0.0/16 network, as matched by access list FlowA. When the same inside hosts initiate connections to the 10.2.0.0/16 network, matched by access list FlowB, NAT ID 2 is used.

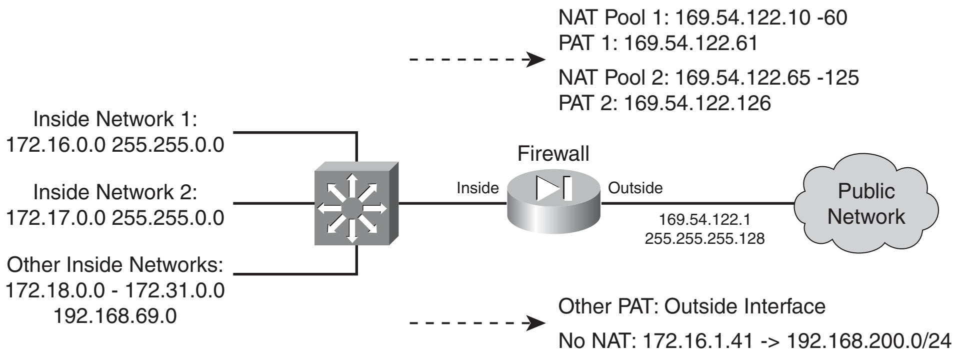
```
Firewall(config)# access-list FlowA permit ip 192.168.1.0 255.255.255.0 10.1.0.0
255.255.0.0
Firewall(config)# access-list FlowB permit ip 192.168.1.0 255.255.255.0 10.2.0.0
255.255.0.0
Firewall(config)# global (outside) 1 interface
Firewall(config)# global (outside) 2 interface
Firewall(config)# nat (inside) 1 access-list FlowA
Firewall(config)# nat (inside) 2 access-list FlowB
```

Dynamic NAT and PAT Example

A firewall connects to the outside network using IP address 169.54.122.1 255.255.255.128. The rest of that subnet is available for the firewall to use as dynamic NAT and/or PAT addresses. Several different IP subnets are on the inside of the firewall: 172.16.0.0/16, 172.17.0.0/16, and various others. (The firewall must have **route** commands defined or dynamic routing information to reach these inside networks, because they are not directly connected to it. Those commands are not shown here.)

Several different NAT and PAT definitions are needed in this scenario, as shown in Figure 6-8. Hosts on the inside network 172.16.0.0 255.255.0.0 are allowed to make outbound connections and will be translated using the global address pool 169.54.122.10 through 169.54.122.60. As long as these addresses are available, they will be assigned as dynamic NAT. If all of the addresses in the pool are in use, the next translation will use global address 169.54.122.61 for dynamic PAT. These are configured as nat/global group ID 1.

Figure 6-8 *Network Diagram for the Dynamic NAT and PAT Example*



A similar translation arrangement is needed for inside network 172.17.0.0 255.255.0.0. These use global pool 169.54.122.65 through 169.54.122.125 for NAT and global address 169.54.122.126 for PAT. These are configured as nat/global group ID 2.

For other inside networks, a default translation arrangement uses the firewall's outside interface address for dynamic PAT. The nat/global group ID 3 performs this function.

One other exception must be made to the address translation mechanism: When inside host 172.16.1.41 opens outbound connections to the 192.168.200.0/24 network, its address should not be translated at all. This is configured using nat group 0, as a NAT exemption identified by access list **acl_no_nat**.

Finally, an access list is applied to the inside interface, controlling the outbound traffic. By default, the firewall will allow outbound traffic from that interface even if no access list exists. The decision is made to use an access list, only to prevent hosts on the inside from spoofing source IP addresses that are different than the inside subnets. Access list **acl_inside** is used for this purpose.

TIP You can also use the **ip verify reverse-path interface inside** command to enable reverse path forwarding lookups. This feature verifies that each packet's source address did indeed come from a subnet that the firewall expects to be located on the source interface. In effect, spoofed addresses are detected. This command is covered in more detail in Chapter 3, "Building Connectivity."

You can configure the firewall for this scenario with the following commands:

```
Firewall(config)# global (outside) 1 169.54.122.10-169.54.122.60 netmask 255.255.255.128
Firewall(config)# global (outside) 1 169.54.122.61
Firewall(config)# nat (inside) 1 172.16.0.0 255.255.0.0 0 0
Firewall(config)# global (outside) 2 169.54.122.65-169.54.122.125 netmask 255.255.255.128
Firewall(config)# global (outside) 2 169.54.122.126
Firewall(config)# nat (inside) 2 172.17.0.0 255.255.0.0 0 0
Firewall(config)# global (outside) 3 interface
Firewall(config)# nat (inside) 3 access-list nat_0 0 0 0
Firewall(config)# access-list acl_no_nat permit ip host 172.16.1.41 192.168.200.0
255.255.255.0
Firewall(config)# nat (inside) 0 access-list acl_no_nat

Firewall(config)# access-list acl_inside permit ip 172.16.0.0 255.240.0.0 any
Firewall(config)# access-list acl_inside permit ip 192.168.69.0 255.255.255.0 any
Firewall(config)# access-list acl_inside deny ip any any
Firewall(config)# access-group acl_inside in interface inside
```

Controlling Traffic

A host on one firewall interface is allowed to create any type of connection to a host on a different firewall interface as long as an address translation can be made (if required) and any relevant interface access lists permit it.

As soon as address translation methods have been configured between pairs of firewall interfaces, you must also configure and apply access lists to the appropriate interfaces.

You can configure and use an access list to limit the types of traffic in a specific direction. When the ACL permits traffic, connections are allowed to pass; when it denies traffic, those packets are dropped at the firewall.

In addition, when an xlate entry is created for a new connection and the interface access lists permit the initial traffic, the return traffic specific to that connection is also permitted—only because the firewall has built the proper xlate and conn entries for it.

You can use the following sequence of steps to configure an access list:

1. Use an access list to permit allowed traffic:

ASA, FWSM	Firewall(config)# **access-list** *acl_id* [**line** *line-num*] [**extended**] {**permit** | **deny**}
	{*protocol* | **object-group** *protocol_obj_group*}
	{*source_addr source_mask* | **object-group** *network_obj_group*}
	[*operator sport* | **object-group** *service_obj_group*]
	{*destination_addr destination_mask* | **object-group** *network_obj_group*}
	[*operator dport* | **object-group** *service_obj_group*]
	[**log** [[**disable** | **default**] | [*level*]]] [**interval** *secs*]]
	[**time-range** *name*] [**inactive**]

PIX 6.3	Firewall(config)# **access-list** *acl_id* [**line** *line-num*] {**permit** | **deny**}
	{*protocol* | **object-group** *protocol_obj_group*}
	{*source_addr source_mask* | **object-group** *network_obj_group*}
	[*operator sport* | **object-group** *service_obj_group*]
	{*destination_addr destination_mask* | **object-group** *network_obj_group*}
	[*operator dport* | **object-group** *service_obj_group*]
	[**log** [[**disable** | **default**] | [*level*]]] [**interval** *secs*]]

Be aware that any source and destination addresses you specify are relative to any address translation that occurs on the interface where the access list is applied.

For example, suppose inside address 192.168.1.1 will be translated to outside address 204.152.16.1. If the access list will be applied to the outside interface to permit inbound connections, then you should use destination address 204.152.16.1 because the host is known by that address on the outside.

Likewise, if the access list will be applied to the inside interface to limit outbound traffic, you should use source address 192.168.1.1.

To configure the access list named **acl_id**, you should refer to Section 6-3, "Controlling Access with Access Lists," which covers ACLs in much greater detail.

If you are creating several access lists, you might consider assigning them meaningful names. For example, an ACL that will control access on the outside interface could be named **acl_outside**. Although it is not necessary to begin the name with **acl_**, that does provide a handy clue that an ACL is being referenced when you look through a large firewall configuration.

2. Apply the access list to a firewall interface:

ASA, FWSM	`Firewall(config)# access-group acl_id {in | out} interface interface_name [per-user-override]`
PIX 6.3	`Firewall(config)# access-group acl_id in interface interface_name [per-user-override]`

The access list named *acl_id* is applied to the interface named *interface_name* (**inside**, for example). The access list evaluates or filters traffic only in the direction specified: **in** (traffic arriving on the interface) or **out** (traffic leaving the interface).

If you use downloadable access lists from a RADIUS server, you can add the **per-user-override** keyword. This allows any downloaded ACLs to override the ACL applied to the interface. In other words, the per-user downloaded ACLs will be evaluated first, before the interface ACL.

Controlling Access with Medium Security Interfaces

So far, inbound and outbound access has been discussed in relation to two firewall interfaces—the inside and the outside. If your firewall has other "medium security" interfaces (security levels between 0 and 100), you face some additional considerations. These interfaces are usually used as demilitarized zone (DMZ) networks, where services are made available to the public networks while offering a certain level of security. DMZ networks are then isolated from the highest security inside networks, although their services can be accessed from the inside.

Outbound access from a medium security interface to a lower one is really no different than from the inside interface. You still need to configure the following:

- Address translation with the **static** command or with the **global** and **nat** commands. This allows hosts on the DMZ to appear on the outside with a valid address.

- An access list applied to the medium security interface. This allows hosts on the DMZ to be permitted to initiate inbound connections toward the inside interface. The same access list also controls outbound connections from the DMZ.

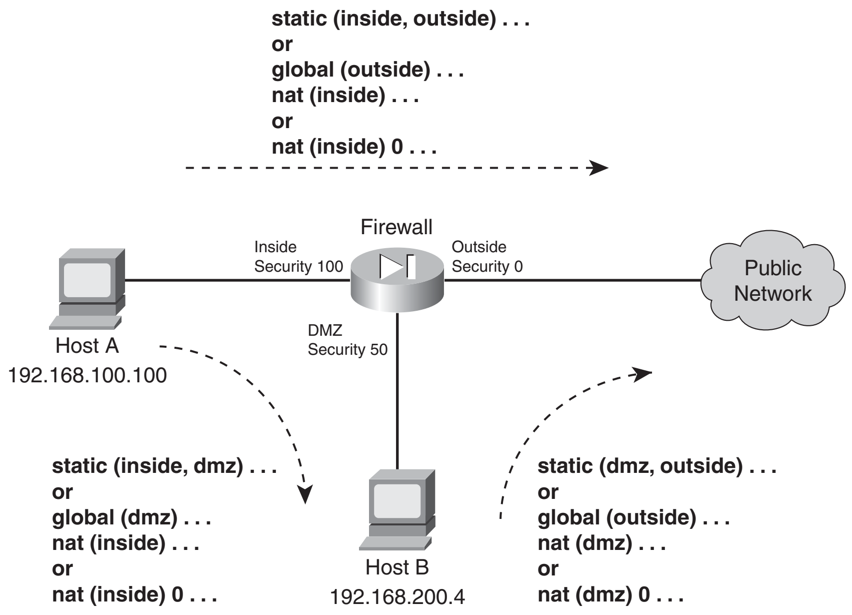
Figure 6-9 shows how outbound access can be configured on a firewall with three interfaces. Basically, you need to consider each interface separately and decide which other lower security interfaces will be involved in the outbound connections. For those interface pairs, configure address translation (if required) and make sure any interface access lists allow the outbound connections.

You should also consider inbound connections, made from a lower security interface toward a higher security DMZ interface. This could include connections from the outside interface toward a DMZ interface, or from a DMZ toward the inside interface.

Inbound access into a medium security interface is really no different than into the inside interface. You still need to configure address translation (if required) so that hosts on the higher security interface appear as a mapped address on the lower security interface.

An access list should also be applied to the interface with the lowest security level of the connection. For example, if an outside host is allowed to connect to a DMZ host, an ACL applied to the outside interface must permit the inbound connection.

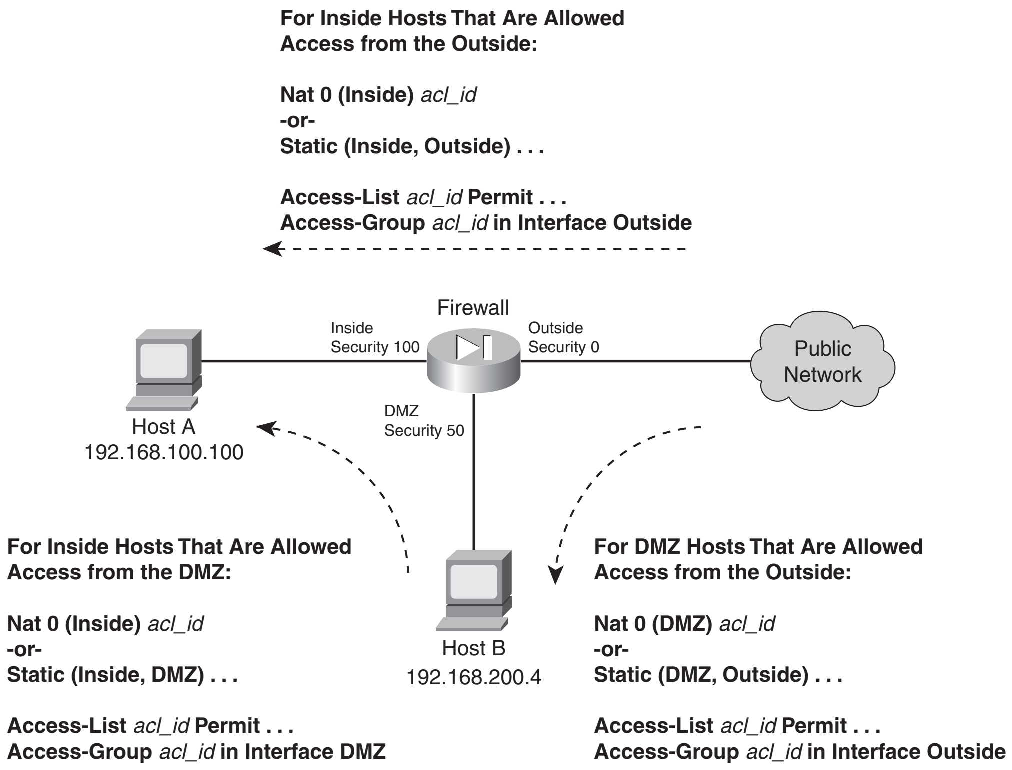
Figure 6-9 *Outbound Access on a Firewall with Three Interfaces*



Similarly, if a DMZ host is allowed to connect to an inside host, an ACL must be applied to the DMZ interface that permits the inbound connection. The ACL must be applied to the interface closest to the source of inbound connections.

Figure 6-10 shows how inbound access can be configured on a firewall with three interfaces. Basically, you need to consider each interface separately and decide which other higher security interfaces will accept inbound connections from it. For those interface pairs, configure address translation (if required) and make sure any interface access lists allow only specific inbound connections.

Figure 6-10 *Inbound Access on a Firewall with Three Interfaces*


For Inside Hosts That Are Allowed Access from the Outside:
Nat 0 (Inside) acl_id
-or-
Static (Inside, Outside) . . .
Access-List acl_id Permit . . .
Access-Group acl_id in Interface Outside
Firewall
Inside Security 100
Outside Security 0
Public Network
DMZ Security 50
Host A 192.168.100.100
Host B 192.168.200.4
For Inside Hosts That Are Allowed Access from the DMZ:
Nat 0 (Inside) acl_id
-or-
Static (Inside, DMZ) . . .
Access-List acl_id Permit . . .
Access-Group acl_id in Interface DMZ
For DMZ Hosts That Are Allowed Access from the Outside:
Nat 0 (DMZ) acl_id
-or-
Static (DMZ, Outside) . . .
Access-List acl_id Permit . . .
Access-Group acl_id in Interface Outside

An access list must be applied to each lower security interface so that specific inbound connections are permitted. This sounds straightforward, but some interesting implications must be considered.

In Figure 6-10, for example, the outside interface can accept inbound connections that are destined for the DMZ network, as well as for the inside network. Therefore, the access list applied to the outside interface should be configured to permit the necessary connections to the global addresses of the DMZ hosts as well as the inside hosts.

Now consider the DMZ interface. Hosts on the DMZ network might initiate inbound connections to the inside interface. Therefore, the access list applied to the DMZ interface should be configured to permit inbound traffic to the global addresses of the inside hosts.

Suppose DMZ hosts are also allowed to initiate outbound connections toward the outside network. The access list applied to the DMZ interface must also be configured to permit these outbound connections, in addition to any inbound traffic toward the inside. It is easy to forget that access lists applied to medium security interfaces should permit traffic destined for several locations.

6-3: Controlling Access with Access Lists

On a Cisco firewall, you can use access lists to filter traffic coming into or out of a firewall interface. Access lists that are applied to interfaces become an integral part of the traffic inspection mechanism.

Access lists can be defined using the familiar Cisco IOS Software ACL format. However, one important difference exists between the firewall and IOS ACL formats: Firewalls use real subnet masks (a 1-bit matches, a 0-bit ignores), while IOS platforms use a wildcard mask (a 0-bit matches, a 1-bit ignores). For example, the subnet 192.168.199.0/24 would be configured as **192.168.199.0 255.255.255.0** on an ASA or FWSM, and as **192.168.199.0 0.0.0.255** on an IOS platform.

Access lists are configured one line at a time; every line that makes up a single access list must have an identical ACL name. Each line of an access list is called an *Access Control Entry (ACE)*.

Cisco firewalls also offer an ACL configuration feature not found in the IOS Software. Access lists can be configured in a modular fashion, using defined *object groups* as a type of macro. Object groups are discussed in detail in the "Defining Object Groups" section.

Compiling Access Lists

Access lists are normally evaluated in sequential order, as they appear in the firewall configuration. As access lists grow in length, the amount of time needed to evaluate the ACEs in sequence can also grow. Fortunately, the ASA and FWSM platforms compile access lists into a more efficient *Turbo ACL* format. On a PIX platform, you can compile ACLs beginning with release 6.2. Once compiled, access lists can be evaluated in a deterministic fashion, without the need to work through each ACE in sequence. In other words, the time required to find and evaluate any specific ACE within the entire access list is more or less constant.

In PIX releases 6.2 and 6.3, access lists must have at least 19 ACEs before they can be compiled into the Turbo ACL format. If they have less than 19 entries, it becomes more efficient for the firewall to evaluate them sequentially. As well, compiled ACLs are stored in Flash memory so that they can be retrieved in their native compiled format even after a reboot.

On a PIX platform, TurboACLs can be compiled when they are first configured, and at most every time when they are edited thereafter. ASAs running 7.0 or later automatically compile access lists immediately after they are edited or configured—usually after every ACE is entered. This is important because any configuration changes you make to ACLs take effect right away. This process is completely hidden, so that you are never aware that an ACL is compiled or not. It simply exists in human-readable form in the firewall configuration.

A FWSM automatically compiles and applies ACLs in real-time by default too. However, you can configure it to use manual compilation instead.

Recompiling an ACL is a silent process, but can burden an already loaded firewall CPU. For this reason, you might choose to manually compile and apply ACL changes so that the only occur at predetermined times. For example, you might plan on making changes to large ACLs during a time of low traffic through the firewall or during a scheduled maintenance window.

On a FWSM, you can use the following command to see whether ACLs are compiled automatically or manually:

```
Firewall# show access-list mode
```

You can use the following command to configure whether access lists will be compiled automatically (the default) or manually:

```
Firewall(config)# access-list mode {auto-commit | manual-commit}
```

In the manual-commit mode, you have to manually compile and commit ACLs to FWSM memory after they are edited, in order for them to be updated and used. If you forget to commit the ACL changes, the FWSM keeps using the previously compiled ACL contents it already has in memory. The following configuration command can be used to commit *all* access lists configured on the FWSM:

```
Firewall(config)# access-list commit
```

Configuring an Access List

You can use the steps presented in this section to configure a firewall access list. The access list exists in the firewall configuration, but does not actively do anything until you apply it to a firewall interface or to some other firewall function.

Access lists are defined simply by entering ACE commands in global configuration mode. There is no need to define the access list name first; just the action of entering an ACE with an ACL ID *acl_id* (an arbitrary text name) is enough to make it a part of that access list. However, as soon as an access list is defined with at least one ACE, the order the ACEs are entered becomes important.

When you enter a new ACE into the configuration, it is always appended to the end or bottom of the access list. Therefore, the order that ACEs appear in an access list is important. This is because access lists are evaluated line-by-line, in sequential order. To designate an ACE's exact position within a whole access list, you can specify a line number as part of the ACE configuration.

In addition, every access list ends with an implicit, hidden **deny ip any any** ACE. Even though a newly configured ACE is appended to the bottom of the access list, the implicit deny ACE always comes after that. In effect, anything that is not explicitly permitted by an ACE somewhere in the access list will be denied by this final implicit ACE.

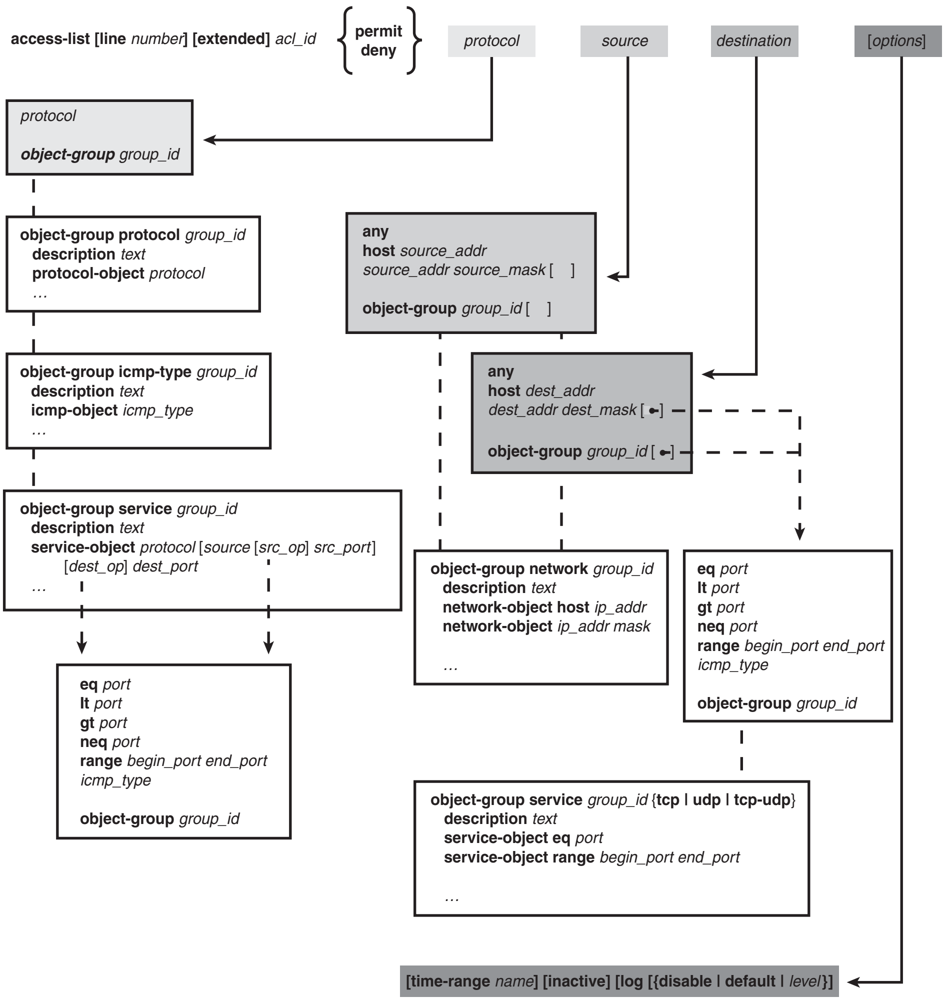
Adding an ACE to an Access List

You can define an access list entry with the following configuration command:

```
Firewall(config)# access-list acl_id [line line-num] [extended] {permit | deny} protocol
source_addr  source_mask [operator sport] destination_addr destination_mask  [operator
dport] [log [[disable | default] | [level]]] [interval secs]] [time-range name] [inactive]
```

Although the command syntax looks complex, the concept is simple: Either **permit** or **deny** traffic from a source (using an optional source port) to a destination (using an optional destination port). An access list can be built from many different keywords and parameters. You might find Figure 6-11 helpful, as it shows the basic ACE syntax and how each portion can be configured.

Figure 6-11 *Cisco Firewall ACE Structure and Composition*



access-list [line number] [extended] acl_id
permit
deny
protocol
source
destination
[options]
protocol
object-group group_id
object-group protocol group_id
description text
protocol-object protocol
...
object-group icmp-type group_id
description text
icmp-object icmp_type
...
object-group service group_id
description text
service-object protocol [source [src_op] src_port] [dest_op] dest_port
...
eq port
lt port
gt port
neq port
range begin_port end_port
icmp_type
object-group group_id
any
host source_addr
source_addr source_mask [ ]
object-group group_id [ ]
any
host dest_addr
dest_addr dest_mask [ ]
object-group group_id [ ]
object-group network group_id
description text
network-object host ip_addr
network-object ip_addr mask
...
eq port
lt port
gt port
neq port
range begin_port end_port
icmp_type
object-group group_id
object-group service group_id {tcp | udp | tcp-udp}
description text
service-object eq port
service-object range begin_port end_port
...
[time-range name] [inactive] [log [{disable | default | level}]

To simplify complex traffic definitions, you can also define groups of parameters as object groups; the object groups are then referenced in the ACE configurations. (Object groups are covered in the "Defining Object Groups" section in this chapter.)

TIP You can use a similar ACE command syntax to create an access list for IPv6 traffic. Use the following guidelines when you adapt the syntax for IPv6:

- Use the **ipv6 access-list** command keywords instead of **access-list**.
- Whenever an IP subnet is given with *source_addr source_mask* or *destination_addr destination_mask*, substitute the IPv6 address prefix as *source_ipv6_prefix/prefix_length* or *destination_ipv6_prefix/prefix_length.*
- Whenever specific host addresses are needed, substitute **host** *ip_address* with **host** *ipv6_address*.

First, begin by identifying the protocol of interest. The matched *protocol* can be **ip** (any IP protocol), **icmp** (1), **tcp** (6), **udp** (17), **ah** (51), **eigrp** (88), **esp** or **ipsec** (50), **gre** or **pptp** (47), **igmp** (2), **igrp** (9), **ipinip** (4), **nos** (94), **ospf** (89), **pim** (103), **pcp** (108), or **snp** (109). You can also specify the *protocol* as a decimal number (0-255) to identify a protocol that does not have a predefined keyword.

Source and destination addresses can be explicit IP addresses or subnets, and the masks are regular subnet masks.

TIP Cisco firewalls do not use the "inverted" masks required by routers running Cisco IOS Software. Instead, think of a firewall mask as a normal IP subnet mask, where a 1 bit matches a bit value and a 0 bit ignores it.

If you need to identify a specific host in an access list, you can give its IP address and a host mask (255.255.255.255). You can also specify the same thing by using the **host** keyword followed by the IP address.

To specify a wildcard or "any" IP address, you can use IP address 0.0.0.0 and mask 0.0.0.0 (0 bits in the mask ignore the value). You can also do the same thing by using the **any** keyword in place of an address and mask.

TIP For inbound firewall rules, ACEs are usually concerned with the destination address and destination port values. This would be useful to allow outside hosts to connect to inside web or email servers, for example. You should always define an ACE with as specific source and destination information as possible. The goal is to define the most strict security policy while allowing users to connect to necessary resources.

As well, you should always include access list rules that filter out attempts to spoof legitimate IP addresses (RFC2827) or use IP addresses set aside for private network use (RFC1918). For example, the following ACEs can be used to deny RFC1918 source addresses:

```
Firewall(config)# access-list anti_spoof deny ip 10.0.0.0 255.0.0.0 any
Firewall(config)# access-list anti_spoof deny ip 172.16.0.0 255.240.0.0 any
Firewall(config)# access-list anti_spoof deny ip 192.168.0.0 255.255.0.0 any
```

If you need to match against a source or destination port number, you can add one of the following keywords as an optional operator:

Less Than	**lt** *port*
Greater Than	**gt** *port*
Equal To	**eq** *port*
Not Equal To	**neq** *port*
Range	**range** *lower upper*

The operator compares the port number to the value given by *port* (a single decimal number; for a range, give two numbers for lower and upper limits). Port numbers can be given as predefined keywords or as decimal numbers. The keywords supported by Cisco firewalls are listed in Appendix A: "Well-Known Protocol and Port Numbers."

In the case of an ICMP (protocol **icmp**) ACE, no operator keyword is used. Instead, the ICMP message type is given alone, in place of the port number.

By default, each ACE is enabled and actively used when it is configured. However, individual ACEs can be disabled without removing them from the configuration. This might be handy if you need to troubleshoot or temporarily deactivate a firewall rule. To do this, reenter the ACE configuration command along with the **inactive** keyword. To reenable an ACE, reenter it without the **inactive** keyword.

You can also configure individual ACEs so that they are active and evaluated only during a predefined time range. Time-based ACEs can be useful if you have security policies that change, based on the time of day, day of the week, and so on. After an access list is configured, you can always remove an ACE or insert an ACE at a specific location within the access list. These tasks are covered in the next section.

TIP ASA and FWSM access lists are always assumed to use the "extended" format, where both source and destination addresses and ports can be specified. This is very similar to extended IP access lists on IOS router and switch platforms. However, the ASA and

FWSM firewalls can support both "standard" and "extended" forms, although standard ACLs are reserved for use with routing protocols.

You might be wise to get into the habit of using the **extended** keyword when you configure, edit, or delete ACE commands. Even if you do not specify the keyword when an ACE is entered, it is automatically inserted into the configuration. The **extended** keyword becomes important when you need to remove an ACE, as it must be given from the command line.

Remember that even though an access list is properly configured, it will not be used until it is applied to a firewall function. Access lists are most often used by applying them to firewall interfaces. Access lists can be applied to interfaces in the inbound and outbound directions independently.

Manipulating Access Lists

Every access list contains statements that are internally numbered. If you type in a new ACE without using a line number, it is simply added to the end of the list (just prior to the implicit **deny ip any any** ACE). If you do include a line number, the new ACE is inserted into the list just prior to the current ACE at that position. The current ACE is not replaced; rather, it and all ACEs below it are moved down one line to make room for the new ACE.

TIP Prior to FWSM 2.3, line numbers were not used at all. In that case, you would have to edit an ACL simply by adding new ACEs or deleting existing ones.

Typically, a firewall numbers ACEs in an ACL with incremental values: the first ACE is line 1, the second ACE is line 2, and so on. However, the ACL line numbers are not shown in the configuration. To see them, you can use the **show access-list** [*acl_id*] command. For example, suppose the following ACL has been configured in a firewall:

```
Firewall(config)# access-list test permit tcp any host 192.168.10.1 eq www
Firewall(config)# access-list test permit udp any host 192.168.10.2 eq domain
Firewall(config)# access-list test permit tcp any host 192.168.10.3 eq smtp
```

When the running-configuration is displayed, the access list is shown just as it was entered. Notice that the final implicit **deny any any** is not shown, although it is actually present.

```
Firewall# show running-config
[output omitted]
access-list test extended permit tcp any host 192.168.10.1 eq www
access-list test extended permit udp any host 192.168.10.2 eq domain
access-list test extended permit tcp any host 192.168.10.3 eq smtp
```

No line numbers are shown, though the **extended** keyword has been added even though the ACE lines were not manually entered that way. To see the ACL line numbers, use the **show access-list** command, as follows:

```
Firewall# show access-list test
access-list test line 1 extended permit tcp any host 192.168.10.1 eq www (hitcnt=1784)
access-list test line 2 extended permit udp any host 192.168.10.2 eq domain (hitcnt=37465)
access-list test line 3 extended permit tcp any host 192.168.10.3 eq smtp (hitcnt=43544)
```

If a new ACE is entered without specifying a line number, it simply goes to the end of the ACL:

```
Firewall(config)# access-list test permit udp any host 192.168.10.4 eq tftp
Firewall(config)# exit
Firewall# show access-list test
access-list test line 1 extended permit tcp any host 192.168.10.1 eq www
(hitcnt=1784)
access-list test line 2 extended permit udp any host 192.168.10.2 eq domain
(hitcnt=37465)
access-list test line 3 extended permit tcp any host 192.168.10.3 eq smtp (hitcnt=43544)
access-list test line 4 extended permit udp any host 192.168.10.4 eq tftp (hitcnt=0)
```

Finally, suppose another new ACE is configured so that it appears at ACL line 2. Notice how the ACEs that were previously at lines 2 and 3 are moved down in the following output, to make room for the new ACE at line 2:

```
Firewall(config)# access-list test line 2 deny udp 192.168.200.0 255.255.255.0 host
192.168.10.2 eq domain
Firewall(config)# exit
Firewall# show access-list test
access-list test line 1 extended permit tcp any host 192.168.10.1 eq www (hitcnt=1784)
access-list test line 2 extended deny udp 192.168.200.0 255.255.255.0 host 192.168.10.2
eq domain (hitcnt=0)
access-list test line 3 extended permit udp any host 192.168.10.2 eq domain (hitcnt=37465)
access-list test line 4 extended permit tcp any host 192.168.10.3 eq smtp (hitcnt=43544)
access-list test line 5 extended permit udp any host 192.168.10.4 eq tftp (hitcnt=0)
```

Beginning with ASA 7.0 and FWSM 2.2, you can make an existing ACE inactive without removing it from the ACL configuration. To do this, enter the complete ACE command again, followed by the **inactive** keyword. The ACE will still be included in the running configuration, but will not be evaluated in the ACL as long as it is inactive. To make the ACE active again, reenter the whole ACE command without the **inactive** keyword.

To remove an existing ACE, use the **no** keyword followed by the complete ACE command line. For example, suppose the following access list has been configured on a firewall (notice the **extended** keyword present in the output):

```
Firewall# show run access-list acl_outside
access-list acl_outside extended permit ip any any
access-list acl_outside extended permit tcp any host 10.3.3.3 eq www
access-list acl_outside extended permit tcp any host 10.3.3.4 eq smtp
```

Obviously, the first ACE could pose a security risk because it might allow all types of traffic to pass. Therefore, you could delete that ACE by using the following command:

```
Firewall# configure terminal
Firewall(config)# no access-list acl_outside extended permit ip any any
```

Remember to include the **extended** keyword, so that the entire ACE syntax is matched. No ACL line numbers can be specified when you delete an ACE.

You can also remove an entire access list, including each of its ACE lines, from the running configuration. However, it is a good idea to first make sure the access list is no longer referenced in any other firewall feature or applied to any firewall interface. This prevents a feature from trying to reference an access list that no longer exists.

Then you can use the following command syntax to remove the access list:

ASA, FWSM 2.3+	`Firewall(config)# clear configure access-list acl_name`

FWSM 2.2-, PIX 6.3	`Firewall(config)# no access-list acl_name`
	or
	`Firewall(config)# clear access-list acl_name`

Notice how the **clear configure access-list** command is quite different from the **no access-list** *acl_name* that you might have expected—the difference in syntax might just prevent you from accidentally deleting a very large or important access list someday!

Suppose you configure an ACL with a name, and you decide that you need to change its name at a later date. Normally, this would be a cumbersome process. You would have to create a new ACL by reentering each of the ACE lines along with the new ACL name and then delete the old ACL.

Beginning with ASA 8.0, you can easily rename an ACL with just one configuration command. Use the following command syntax to change the ACL name from *old_acl_name* to *new_acl_name*:

```
asa(config)# access-list old_acl_name rename new_acl_name
```

Adding Descriptions to an Access List

Access lists can become very large, especially if your network has many security policies, many hosts, and many types of traffic to protect with a firewall. Large access lists can be difficult to read (for a human) because it might not be clear what a particular ACE is doing or what its original purpose was.

You can add access list entries that contain text descriptions or remarks to provide some readable clues. Remark ACEs can be added anywhere within an access list, provided you can supply a line number when you configure them. As well, you can configure as many remark ACEs as necessary in an access list.

To add a remark ACE to an access list, use the following command:

```
Firewall(config)# access-list acl_id [line line-num] remark text
```

A new ACE is added to the access list named *acl_id* and contains only the word **remark** followed by the description *text* (arbitrary text string, 1 to 100 characters). If the **line** keyword is used, the remark is added just prior to the current ACE at line *line-num*.

TIP Remark ACEs are useful as descriptions or reminders of the purpose for the ACE or group of ACEs that follow them. Keep in mind that the remark lines are not tied to actual ACEs; they only occupy space in the ACL. If you remove or relocate an ACE, a remark line might be left behind in its original position. In other words, unless you remove or relocate remark lines and their associated ACEs, the remarks might become confusing or meaningless.

Defining a Time Range to Activate an ACE

Once configured, access list entries are active all the time. Beginning with ASA 7.0 and FWSM 3.1(1), you can use two mechanisms to control whether an ACE is actively being evaluated or not:

- Adding the **inactive** keyword to an ACE configuration disables it until the keyword is removed.
- Adding a defined time range to an ACE configuration, during which the ACE will be actively used. Outside the time range, the ACE will be inactive.

To control an ACE with a time range, you must configure the time range itself first. You can use the steps that follow to configure a time range. Once configured, the time range must also be applied to specific ACEs, as discussed in the section, "Adding an ACE to an Access List."

TIP Because a time range controls the state of access list entries, you should make sure the firewall's internal clock is set to the correct date and time. Otherwise, the firewall might not provide the proper security policies at the expected times.

To learn more about setting the firewall clock and synchronizing it with an accurate, external source, refer to Section "10-1: Managing the Firewall Clock," in Chapter 10, "Firewall Logging."

---

1. Create the time range definition:

ASA, FWSM	`Firewall(config)#` **`time-range`** *`name`*
PIX 6.3	N/A

The time range is referenced by its *name* (arbitrary text string, up to 64 characters).

2. (Optional) Define a recurring time period:

ASA, FWSM	`Firewall(config-time-range)#` **`periodic`** *`start-day hh:mm`* **`to`** *`end-day hh:mm`*
	or
	`Firewall(config-time-range)#` **`periodic`** *`days-of-the-week hh:mm`* **`to`** *`hh:mm`*

PIX 6.3	N/A

You can define the time period in two ways. In the first form of the command, a continuous time period is defined. This begins on *start-day* at time *hh:mm* (24-hour format) and ends on *end-day* at time *hh:mm* (24-hour format). The day parameters can be one of the following: **Sunday**, **Monday**, **Tuesday**, **Wednesday**, **Thursday**, **Friday**, or **Saturday**. These are not case sensitive and can be abbreviated, as long as they are not ambiguous.

In the second form, the period lasts from *hh:mm* to *hh:mm* (24-hour format, both within the same day), only on the days listed in *days-of-the-week*. This is one of the following keywords: **daily** (Monday-Sunday), **weekdays** (Monday-Friday), or **weekends** (Saturday-Sunday). You can also use a list of one or more of the following day keywords separated by spaces: **Sunday**, **Monday**, **Tuesday**, **Wednesday**, **Thursday**, **Friday**, or **Saturday**.

You can repeat either of these commands in the same time range definition to add more time periods.

As an example, a time range is configured to define "after hours". For the weekday (Monday through Friday) period between 5:00pm until 8:00am the following morning, two separate **periodic** commands must be used. This is because the time between the start and end times cannot cross midnight and into the following day. A third periodic range is given for weekends, from midnight Saturday until 11:59pm Sunday using the following commands:

```
Firewall(config)# time-range after_hours
Firewall(config-time-range)# periodic weekdays 17:00 to 23:59
Firewall(config-time-range)# periodic weekdays 00:00 to 08:00
Firewall(config-time-range)# periodic saturday 00:00 to sunday 23:59
Firewall(config-time-range)# exit
```

3. (Optional) Define an absolute time period:

ASA, FWSM	`Firewall(config-time-range)#` **`absolute`** `[`**`start`** *`hh:mm day month year`*`] [`**`end`** *`hh:mm day month year`*`]`
PIX 6.3	N/A

The time period has an absolute starting time and date and an ending time and date. The times are given as *hh:mm* (24-hour format), and dates as *day* (1-31), *month* name, and *year* (1993-2035).

If you omit the start limit, the time range begins immediately and runs until the end limit. If you provide a start limit, but omit the end limit, the time range runs indefinitely as soon as the start limit occurs.

As an example, in the configuration that follows, one time range named *Dec2010* is configured to define the entire month of December, 2010. A second time range named *End_2010* is configured to start immediately and end at the very end of December, 2010.

```
Firewall(config)# time-range Dec2010
Firewall(config-time-range)# absolute start 00:00 1 december 2010 end 23:59 30
december 2010
Firewall(config-time-range)# exit
Firewall(config)# time-range End_2010
Firewall(config-time-range)# absolute end 23:59 31 december 2010
Firewall(config-time-range)# exit
```

4. Reference the time range in an ACE.

   In the ACE configuration, add the **time-range** *name* keyword and argument to the **access-list** ACE command.

Access List Examples

A firewall connects an inside network (192.168.17.0/24) to an outside network. Inside hosts include a DNS server, a mail server, and two other servers that support extranet services. Assume that the inside hosts have a NAT exemption to the outside, such that they maintain their same IP addresses. The outside hosts include two extranet servers on the 172.22.10.0/24 network.

The following rules should be configured on the firewall, as shown by the actual configuration commands:

- ICMP traffic from the outside should be limited to only ICMP echo (ping) requests, ICMP time-exceeded replies (needed for traceroute support), and ICMP unreachables (needed for path MTU discovery).

```
Firewall(config)# access-list acl_outside permit icmp any 192.168.17.0 255.255.255.0
echo
Firewall(config)# access-list acl_outside permit icmp any 192.168.17.0 255.255.255.0
time-exceeded
Firewall(config)# access-list acl_outside permit icmp any 192.168.17.0 255.255.255.0
unreachable
```

- Only DNS traffic from the outside is permitted to inside host 192.168.17.21.

```
Firewall(config)# access-list acl_outside permit udp any host 192.168.17.21 eq domain
```

- Only SMTP, POP3, and NNTP traffic from the outside is permitted to inside host 192.168.17.22.

```
Firewall(config)# access-list acl_outside permit tcp any host 192.168.17.22 eq smtp
Firewall(config)# access-list acl_outside permit tcp any host 192.168.17.22 eq pop3
Firewall(config)# access-list acl_outside permit tcp any host 192.168.17.22 eq nntp
```

- Outside hosts 172.22.10.41 and 172.22.10.53 are allowed to send SQLnet and FTP packets toward the two inside hosts 192.168.17.100 and 192.168.17.114.

```
Firewall(config)# access-list acl_outside permit tcp host 172.22.10.41 host
192.168.17.100 eq sqlnet
Firewall(config)# access-list acl_outside permit tcp host 172.22.10.41 host
192.168.17.100 eq ftp
```

```
Firewall(config)# access-list acl_outside permit tcp host 172.22.10.53 host
192.168.17.100 eq sqlnet
Firewall(config)# access-list acl_outside permit tcp host 172.22.10.53 host
192.168.17.100 eq ftp
Firewall(config)# access-list acl_outside permit tcp host 172.22.10.41 host
192.168.17.114 eq sqlnet
Firewall(config)# access-list acl_outside permit tcp host 172.22.10.41 host
192.168.17.114 eq ftp
Firewall(config)# access-list acl_outside permit tcp host 172.22.10.53 host
192.168.17.114 eq sqlnet
Firewall(config)# access-list acl_outside permit tcp host 172.22.10.53 host
192.168.17.114 eq ftp
```

TIP Notice that only the FTP control port (TCP 21) is specified in the access list, even though a separate FTP data connection is opened on a different port. As long as FTP inspection is enabled with the **inspect ftp** command, you do not have to explicitly configure the data port in the access list. Instead, the firewall automatically permits the additional inbound FTP data connection when it is negotiated dynamically.

Finally, the access list must be applied to the outside interface, in the inward direction:

```
Firewall(config)# access-group acl_outside in interface outside
```

Most of the security policies or rules in this example can be easily configured. The final rule, however, requires a bit more work defining a one-to-one relationship between outside and inside hosts, along with a combination of TCP ports that will be permitted. The more hosts and services you have on each side, the more complicated and lengthy the access list gets. ACL complexity can be greatly reduced by using object groups, as discussed in the following section.

Defining Object Groups

Object groups can be thought of as a type of macro used within access lists. Object groups can contain lists of IP addresses, ICMP types, IP protocols, or ports. You can define several different types of object groups, each containing a list of similar values, as follows:

- **Network object group**—Contains one or more IP addresses.
- **Protocol object group**—Contains one or more IP protocols.
- **ICMP object group**—Contains one or more ICMP types.
- **Basic service object group**—Contains one or more UDP or TCP port numbers.
- **Enhanced service object group**—Beginning with ASA 8.0, service object groups can contain any mix of protocols, ICMP types, UDP ports, and TCP ports. Enhanced service object groups can be used for either source or destination services, or both, in the access list configuration.

Object groups can also be nested; that is, one object group can contain members that are themselves object groups of the same type.

TIP When object groups are configured, it is possible to use the same name for object groups of different types. For example, a network object group might be called "Engineering" and contain IP addresses of hosts in the engineering department. A service (application port) object group could also be called "Engineering", and contain a set of TCP ports needed in the engineering department.

While the firewall might be able to sort out the object groups into the correct types, name duplication can be confusing for human users. It is usually a good idea to add a little tag into each object group's name to help differentiate their purposes. For example, you could use "Engineering_hosts" and "Engineering_ports". In fact, beginning with ASA 7.0, all object groups are required to have unique names.

The steps needed to configure each type of object group are presented in the sections that follow. The command syntax is identical on the ASA and FWSM firewall platforms. As soon as an object group is defined or configured, it can be referenced within one or more access lists.

Defining Network Object Groups

You can use the following steps to configure a network object group containing a list of IP addresses.

1. Name the object group:

   ```
   Firewall(config)# object-group network group_id
   ```

   The group is named *group_id* (arbitrary text string, 1 to 64 characters).

2. (Optional) Add a description:

   ```
   Firewall(config-network)# description text
   ```

   You can add a descriptive string *text* (up to 200 characters) to help explain the purpose of the object group.

3. Add an IP address to the list:

   ```
   Firewall(config-network)# network-object host ip_addr
   ```

   or

   ```
   Firewall(config-network)# network-object ip_addr mask
   ```

   The IP address can be given as the **host** keyword with a single address *ip_addr*. If you have preconfigured a hostname with the **host** command, the hostname can be used here. You can also define an IP subnet with a subnet mask, if needed.

   You can repeat this command to define more IP addresses in the object group.

4. (Optional) Reference another network object group:

```
Firewall(config-network)# group-object group_id
```

Sometimes you might define a network object group for one purpose and need to include it in a larger object group. You can include another object group by referencing its group name *group_id*. This group must be configured before including it here.

As an example, suppose you need to configure the same access list statement for a list of three IP addresses (192.168.1.10, 192.168.1.20, and 192.168.1.30) and one IP subnet (192.168.2.0/24), all assigned to hosts in the Accounting department. The following network object group could be configured with the list of addresses, prior to applying it within an access list.

```
Firewall(config)# object-group network Accounting_addrs
Firewall(config-network)# description List of Accounting Dept IP addresses
Firewall(config-network)# network-object host 192.168.1.10
Firewall(config-network)# network-object host 192.168.1.20
Firewall(config-network)# network-object host 192.168.1.30
Firewall(config-network)# network-object 192.168.2.0 255.255.255.0
Firewall(config-network)# exit
```

Now suppose that list of addresses is also a part of a larger list containing the hosts of a remote site. A new object group could be configured for the remote site, containing other object groups defined for various departments. You could use the following commands:

```
Firewall(config)# object-group network RemoteSite_addrs
Firewall(config-network)# description List of IP addresses used in the Remote Site
Firewall(config-network)# group-object Accounting_addrs
```

Defining Protocol Object Groups

You can use the following steps to configure a protocol object group, containing a list of IP protocols.

1. Name the object group:

```
Firewall(config)# object-group protocol group_id
```

The group is named *group_id* (arbitrary text string, 1 to 64 characters).

2. (Optional) Add a description:

```
Firewall(config-protocol)# description text
```

You can add a descriptive string *text* (up to 200 characters) to help explain the purpose of the object group.

3. Add an IP protocol to the list:

```
Firewall(config-protocol)# protocol-object protocol
```

The IP protocol can be given as *protocol*, a decimal number (1 to 255), or one of the names listed in Table 6-2. To match any IP protocol, use the value **ip.**

Table 6-2 *IP Protocol Names/Numbers*

Protocol Name	Protocol Number
icmp	**1**
igmp	**2**
ipinip	**4**
tcp	**6**
igrp	**9**
udp	**17**
gre or **pptp**	**47**
esp or **ipsec**	**50**
ah	**51**
icmp6	**58**
eigrp	**88**
ospf	**89**
nos	**94**
pim	**103**
pcp	**108**
snp	**109**

You can repeat this command to define more IP protocols in the group.

4. (Optional) Reference another protocol object group:

```
Firewall(config-protocol)# group-object group_id
```

Sometimes you might define a protocol object group for one purpose and need to include it in a larger protocol object group. You can include another object group by referencing its group name *group_id*. You must configure this group prior to referencing it.

As an example, a protocol object group named Tunnels_proto is configured to contain a list of tunneling protocols (GRE, IP-in-IP, ESP, and AH). A second group named Routing_proto contains a list of routing protocols (IGRP, EIGRP, and OSPF). These object groups can then become a part of a larger list called Group1_proto, using the following commands:

```
Firewall(config)# object-group protocol Tunnels_proto
Firewall(config-protocol)# description Tunneling Protocols
Firewall(config-protocol)# protocol-object gre
Firewall(config-protocol)# protocol-object ipinip
Firewall(config-protocol)# protocol-object esp
Firewall(config-protocol)# protocol-object ah
Firewall(config-protocol)# exit
!
Firewall(config)# object-group protocol Routing_proto
Firewall(config-protocol)# description Routing Protocols
Firewall(config-protocol)# protocol-object igrp
Firewall(config-protocol)# protocol-object eigrp
Firewall(config-protocol)# protocol-object ospf
Firewall(config-protocol)# exit
!
Firewall(config)# object-group protocol Group1_proto
Firewall(config-protocol)# description Group1 list of protocols
Firewall(config-protocol)# group-object Tunnels_proto
Firewall(config-protocol)# group-object Routing_proto
```

Defining ICMP Type Object Groups

You can use the following steps to configure an ICMP type object group, containing a list of ICMP type values.

1. Name the object group:

   ```
   Firewall(config)# object-group icmp-type group_id
   ```

   The group is named *group_id* (arbitrary text string, 1 to 64 characters).

2. (Optional) Add a description:

   ```
   Firewall(config-icmp-type)# description text
   ```

   You can add a descriptive string *text* (up to 200 characters) to help explain the purpose of the object group.

3. Add an ICMP type to the list:

   ```
   Firewall(config-icmp-type)# icmp-object icmp_type
   ```

   The ICMP type can be given as *icmp_type*, a number from 0 to 255, or one of the decimal numbers or names listed in Table 6-3.

Table 6-3 *ICMP Type Names/Numbers*

ICMP Type Name	Number
echo-reply	**0**
unreachable	**3**
source-quench	**4**
redirect	**5**

continues

Table 6-3 *ICMP Type Names/Numbers (Continued)*

ICMP Type Name	Number
alternate-address	**6**
echo	**8**
router-advertisement	**9**
router-solicitation	**10**
time-exceeded	**11**
parameter-problem	**12**
timestamp-request	**13**
timestamp-reply	**14**
information-request	**15**
information-reply	**16**
mask-request	**17**
mask-reply	**18**
traceroute	**30**
conversion-error	**31**
mobile-redirect	**32**

You can repeat the **icmp-object** *icmp_type* command to define more ICMP types in the group.

4. (Optional) Reference another ICMP object group:

```
Firewall(config-icmp-type)# group-object group_id
```

Sometimes you might define an icmp-type object group for one purpose and need to include it in a larger object group. You can include another object group by referencing its group name *group_id*. You must configure this group before referencing it with the preceding command.

As an example, an icmp-type object group named "Ping" is configured to contain ICMP types echo and echo-reply. Then that object group is included as a part of a more encompassing icmp-type object group that also contains the unreachable, redirect, and time-exceeded types. You can use the following commands to accomplish this:

```
Firewall(config)# object-group icmp-type Ping_icmp
Firewall(config-icmp)# icmp-object echo
Firewall(config-icmp)# icmp-object echo-reply
Firewall(config-icmp)# exit
!
Firewall(config)# object-group icmp-type BiggerList_icmp
Firewall(config-icmp)# group-object Ping_icmp
Firewall(config-icmp)# icmp-object unreachable
Firewall(config-icmp)# icmp-object redirect
Firewall(config-icmp)# icmp-object time-exceeded
```

Defining Basic Service Object Groups

You can use the following steps to configure a service object group, containing a list of port numbers.

1. Name the object group:

   ```
   Firewall(config)# object-group service group_id {tcp | udp | tcp-udp}
   ```

   The group is named *group_id* (arbitrary text string, 1 to 64 characters). Ports defined in the group can be used in access list statements that match TCP ports (**tcp**), UDP ports (**udp**), or either TCP or UDP port numbers (**tcp-udp**).

   TIP Object groups configured for **tcp** can be used only in access list statements that match TCP port numbers. Likewise, **udp** object groups can match only UDP port numbers. If you need to match an application that uses the same port number over both TCP and UDP, you can use the **tcp-udp** object group. You still need to have separate TCP and UDP access list statements, but they can both reference the same object group.

2. (Optional) Add a description:

   ```
   Firewall(config-service)# description text
   ```

   You can add a descriptive string *text* (up to 200 characters) to help explain the purpose of the object group.

3. Add a port number to the list:

   ```
   Firewall(config-service)# port-object eq port
   ```

   or

   ```
   Firewall(config-service)# port-object range begin_port end_port
   ```

   A specific port number can be given with the **eq** keyword as *port*, a decimal number or name. With the **range** keyword, you can also specify a range of port values from *begin_port* to *end_port*. Refer to Appendix A for a complete list of well-known protocol and port number keywords supported by Cisco firewalls.

   You can repeat this command to define more IP addresses in the group.

4. (Optional) Reference another basic service object group:

   ```
   Firewall(config-service)# group-object group_id
   ```

Sometimes you might define a service object group for one purpose and need to include it in a larger object group. You can include another object group by referencing its group name *group_id*. You must configure this group prior to referencing it.

As an example, one object group is defined to contain the HTTP and HTTPS TCP ports for web services. A second group is defined to contain the SMTP, POP3, and IMAP4 TCP ports for e-mail services. A third object group is defined to contain both of the other groups, in addition to the TCP port range 2000 through 2002. You can use the following commands to accomplish this:

```
Firewall(config)# object-group service Web_ports tcp
Firewall(config-service)# description TCP ports used by web browsers
Firewall(config-service)# port-object eq www
Firewall(config-service)# port-object eq https
Firewall(config-service)# exit
!
Firewall(config)# object-group service Mail_ports tcp
Firewall(config-service)# description TCP ports used for email
Firewall(config-service)# port-object eq smtp
Firewall(config-service)# port-object eq pop3
Firewall(config-service)# port-object eq imap4
Firewall(config-service)# exit
!
Firewall(config)# object-group service Example_ports tcp
Firewall(config-service)# description A bunch of TCP ports
Firewall(config-service)# group-object Web_ports
Firewall(config-service)# group-object Mail_ports
Firewall(config-service)# port-object range 2000 2002
```

Defining an Enhanced Service Object Group

Beginning with ASA 8.0, you can define a single enhanced service object group that can contain any combination of protocols, ICMP types, UDP ports, and TCP ports. This is important because it greatly simplifies object group configuration and use.

Use the following steps to configure an enhanced service object group:

1. Name the object group:

   ```
   Firewall(config)# object-group service group_id
   ```

   The group is named *group_id* (an arbitrary string of 1 to 64 characters). Notice that no other keywords are used to identify the group as a specific type.

2. (Optional) Add a description:

   ```
   Firewall(config-service)# description text
   ```

You can add a descriptive string *text* (up to 200 characters) to help explain the purpose of the object group.

3. Add a service object to the list:

```
Firewall(config-service)# service-object  protocol [source [src_op] src_port]
[dest_op] dest_port
```

In one service object definition, you can identify the protocol, source port, and destination port. The source port information is optional; if you do not provide the **source** keyword, only the destination port is assumed. If you do provide the **source** keyword, you need to also supply the source *and* destination parameters.

You can use the keywords listed in Table 6-4 when you configure the protocol and port parameters:

Table 6-4 protocol/port/type *Parameter Keywords for Enhanced Service Objects*

***protocol* Keyword**	***port* or *type* Keyword**
ah	
eigrp	
esp (same as **ipsec**)	
gre	
icmp *type*	**0–255** **alternate-address**, **conversion-error**, **echo**, **echo-reply**, **information-reply**, **information-request**, **mask-reply**, **mask-request**, **mobile-redirect**, **parameter-problem**, **redirect**, **router-advertisement**, **router-solicitation**, **source-quench**, **time-exceeded**, **timestamp-reply**, **timestamp-request**, **traceroute**, **unreachable**
icmp6 *type*	**0–255** **echo**, **echo-reply**, **membership-query**, **membership-reduction**, **membership-report**, **neighbor-advertisement**, **neighbor-redirect**, **neighbor-solicitation**, **packet-too-big**, **parameter-problem**, **router-advertisement**, **router-renumbering**, **router-solicitation**, **time-exceeded**, **unreachable**
igmp	
igrp	
ip	
ipinip	
ipsec (same as **esp**)	
nos	
ospf	

continues

Table 6-4 protocol/port/type *Parameter Keywords for Enhanced Service Objects (Continued)*

***protocol* Keyword**	***port* or *type* Keyword**
pcp	
pim	
pptp	
snp	
udp *port*	**0–65535** **biff**, **bootpc**, **bootps**, **cifs**, **discard**, **dnsix**, **domain**, **echo**, **http**, **isakmp**, **kerberos**, **mobile-ip**, **nameserver**, **netbios-dgm**, **netbios-ns**, **nfs**, **ntp**, **pcanywhere-status**, **pim-auto-rp**, **radius**, **radius-acct**, **rip**, **secureid-udp**, **sip**, **snmp**, **snmptrap**, **sunrpc**, **syslog**, **tacacs**, **talk**, **tftp**, **time**, **who**, **www**, **xdmcp**
tcp *port*	**0–65535** **aol**, **bgp**, **chargen**, **cifs**, **citrix-ica**, **cmd**, **ctiqbe**, **daytime**, **discard**, **domain**, **echo**, **exec**, **finger**, **ftp**, **ftp-data**, **gopher**, **h323**, **hostname**, **http**, **https**, **ident**, **imap4**, **irc**, **kerberos**, **klogin**, **kshell**, **ldap**, **ldaps**, **login**, **lotusnotes**, **lpd**, **netbios-ssn**, **nfs**, **nntp**, **pcanywhere-data**, **pim-auto-rp**, **pop2**, **pop3**, **pptp**, **rsh**, **rtsp**, **sip**, **smtp**, **sqlnet**, **ssh**, **sunrpc**, **tacacs**, **talk**, **telnet**, **uucp**, **whois**, **www**
tcp-udp *port*	**0–65535** **cifs**, **discard**, **domain**, **echo**, **http**, **kerberos**, **nfs**, **pim-auto-rp**, **sip**, **sunrpc**, **tacacs**, **talk**, **www**

The source and destination operators, *src_op* and *dest_op*, can be given as one of the following keywords:

- Less Than: **lt** *port*
- Greater Than: **gt** *port*
- Equal To: **eq** *port*
- Not Equal To: **neq** *port*
- Range: **range** *lower upper*

4. (Optional) Reference another service object group:

```
Firewall(config-service)# group-object group_id
```

Sometimes you might define a service object group for one purpose and need to include it in a larger object group. You can include another object group by referencing its group name *group_id*. You must configure this group prior to referencing it.

As an example of an enhanced service object group, suppose you would like to identify packets containing ICMP echo and echo-reply, IPsec ESP (IP protocol 50), ISAKMP (UDP port 500), UDP port 10000, and HTTP. With object groups on a FWSM or an ASA prior to ASA 8.0, you would have to configure individual object groups for each type of traffic, as shown in the following example:

```
Firewall(config)# object-group icmp test-icmp
Firewall(config-icmp)# description ICMP types to identify
Firewall(config-icmp)# icmp-object echo
Firewall(config-icmp)# icmp-object echo-reply
Firewall(config-icmp)# exit
Firewall(config)# object-group protocol test-protocol
Firewall(config-protocol)# description Protocols to identify
Firewall(config-protocol)# protocol-object esp
Firewall(config-protocol)# exit
Firewall(config)# object-group service test-service1 udp
Firewall(config-service)# description UDP ISAKMP
Firewall(config-service)# port-object eq isakmp
Firewall(config-service)# exit
Firewall(config)# object-group service test-service2 udp
Firewall(config-service)# description UDP 10000
Firewall(config-service)# port-object eq 10000
Firewall(config-service)# exit
Firewall(config)# object-group service test-service3 tcp
Firewall(config-service)# description TCP 80
Firewall(config-service)# port-object eq www
Firewall(config-service)# exit
```

In contrast, with ASA 8.0, all of the traffic types can be identified using a *single* enhanced service object group as demonstrated in the configuration that follows. Notice how the configuration is smaller and that only a single object group needs to be referenced.

```
Firewall(config)# object-group service test
Firewall(config-service)# description Enhanced Service Obj Group
Firewall(config-service)# service-object icmp echo
Firewall(config-service)# service-object icmp echo-reply
Firewall(config-service)# service-object esp
Firewall(config-service)# service-object udp eq isakmp
Firewall(config-service)# service-object udp source 10000
Firewall(config-service)# service-object tcp eq www
Firewall(config-service)# exit
```

Using Object Groups in an Access List

After you have defined an object group, it must be referenced in an access list before it can be used. You can substitute the object group in place of the appropriate protocol, address, or port parameters within the **access-list** command syntax. In the complete command syntax that follows, notice the location of each of the regular object group keywords:

```
Firewall(config)# access-list acl_id [line line-num] [extended] {permit | deny}
   {protocol | object-group protocol_obj_group}
   {source_addr  source_mask | object-group network_obj_group}
   [operator sport | object-group service_obj_group]
   {destination_addr destination_mask | object-group network_obj_group}
   [operator dport | object-group service_obj_group]
   [log [[disable | default] | [level]]] [interval secs]]
   [time-range name] [inactive]
```

You can substitute the following regular ACE parameters with the object group syntax shown:

Regular ACE Parameters	Substitute This Object Group Syntax
protocol	**object-group** *protocol_grp_id*
icmp *icmp-type*	**object-group** *icmp_grp_id*
address mask	**object-group** *network_grp_id*
operator port	**object-group** *service_grp_id*

As you begin to use object groups, think of the common things in security policies that can be grouped together: ICMP packet types, inside and/or outside host addresses, protocols, and port numbers. Consider the following example, which is identical to the scenario presented in the "Access List Examples" section.

```
Firewall(config)# object-group icmp-type allowed-icmp
Firewall(config-icmp)# description ICMP traffic allowed inside
Firewall(config-icmp)# icmp-object echo
Firewall(config-icmp)# icmp-object time-exceeded
Firewall(config-icmp)# icmp-object unreachable
Firewall(config-icmp)# exit
Firewall(config)# object-group service mail-services tcp
Firewall(config-service)# description Email and News services to 192.168.17.22
Firewall(config-service)# port-object eq smtp
Firewall(config-service)# port-object eq pop3
Firewall(config-service)# port-object eq nntp
Firewall(config-service)# exit
Firewall(config)# object-group network extranet-hosts
Firewall(config-network)# description Extranet hosts allowed in for backend services
Firewall(config-network)# network-object host 172.22.10.41
Firewall(config-network)# network-object host 172.22.10.53
Firewall(config-network)# exit
Firewall(config)# object-group network extranet-targets
Firewall(config-network)# description Inside servers supporting extranet servers
Firewall(config-network)# network-object host 192.168.17.110
Firewall(config-network)# network-object host 192.168.17.114
Firewall(config-network)# exit
Firewall(config)# object-group service extranet-services tcp
Firewall(config-service)# description Extranet backend services allowed
Firewall(config-service)# port-object eq sqlnet
Firewall(config-service)# port-object eq ftp
Firewall(config-service)# exit
```

Now the actual access list is configured, including any references to the object groups that have been configured:

```
Firewall(config)# access-list acl_outside permit icmp any 192.168.17.0 255.255.255.0
object-group allowed-icmp
Firewall(config)# access-list acl_outside permit udp any host 192.168.17.21 eq domain
Firewall(config)# access-list acl_outside permit tcp any host 192.168.17.22 object-group
mail-services
Firewall(config)# access-list acl_outside permit tcp object-group extranet-hosts object-
group extranet-targets object-group extranet-services
```

This time, the number of object group commands seems to be lengthy but the access list is rather short. The idea is to make the ACL as short as possible to become more abstract and readable. If you have defined your object groups with meaningful names, you begin to see the access list defined with abstract functions or quantities.

You can always make adjustments to the security policies by adding or deleting lines from object groups. Even if the object group definitions become very lengthy, it does not really impact the firewall's performance—the overall access list is compiled so that it can match traffic in a consistent amount of time, regardless of the actual ACL length.

Although the access list is much shorter in the running-configuration when it includes object group references, this is primarily for a firewall administrator's benefit.

Internally, the firewall expands the object groups (even nested ones) out into the full access list configuration. If the firewall is configured to compile ACLs, then the expanded result is compiled and used. You can display the full access list with the **show access-list** command, as in the following example:

```
Firewall# show access-list acl_outside
access-list acl_outside; 15 elements
access-list acl_outside line 1 extended permit icmp any 192.168.17.0 255.255.255.0 object-
group allowed-icmp
access-list acl_outside line 1 extended permit icmp any 192.168.17.0 255.255.255.0 echo
(hitcnt=0)
access-list acl_outside line 1 extended permit icmp any 192.168.17.0 255.255.255.0 time-
exceeded (hitcnt=0)
access-list acl_outside line 1 extended permit icmp any 192.168.17.0 255.255.255.0
unreachable (hitcnt=0)
access-list acl_outside line 2 extended permit udp any host 192.168.17.21 eq domain
(hitcnt=0)
access-list acl_outside line 3 extended permit tcp any host 192.168.17.22 object-group
mail-services
access-list acl_outside line 3 extended permit tcp any host 192.168.17.22 eq smtp
(hitcnt=0)
access-list acl_outside line 3 extended permit tcp any host 192.168.17.22 eq pop3
(hitcnt=0)
access-list acl_outside line 3 extended permit tcp any host 192.168.17.22 eq nntp
(hitcnt=0)
access-list acl_outside line 4 extended permit tcp object-group extranet-hosts object-
group extranet-targets object-group extranet-services
access-list acl_outside line 4 extended permit tcp host 172.22.10.41 host 192.168.17.110
eq sqlnet (hitcnt=0)
access-list acl_outside line 4 extended permit tcp host 172.22.10.41 host 192.168.17.110
eq ftp (hitcnt=0)
access-list acl_outside line 4 extended permit tcp host 172.22.10.41 host 192.168.17.114
eq sqlnet (hitcnt=0)
access-list acl_outside line 4 extended permit tcp host 172.22.10.41 host 192.168.17.114
eq ftp (hitcnt=0)
access-list acl_outside line 4 extended permit tcp host 172.22.10.53 host 192.168.17.110
eq sqlnet (hitcnt=0)
```

```
access-list acl_outside line 4 extended permit tcp host 172.22.10.53 host 192.168.17.110
eq ftp (hitcnt=0)
access-list acl_outside line 4 extended permit tcp host 172.22.10.53 host 192.168.17.114
eq sqlnet (hitcnt=0)
access-list acl_outside line 4 extended permit tcp host 172.22.10.53 host 192.168.17.114
eq ftp (hitcnt=0)
Firewall#
```

Even though this access list was reported to have 15 elements (ACEs), notice what has happened with the line numbers. The line numbers follow the ACL in the running configuration, where the object groups have not been expanded. However, in this output, the line numbers do not increment as long as the ACEs are part of the same object-group expansion. This is shown by the shaded text, where "line 1" represents all of the ACEs from the object-group *allowed-icmp*.

The real beauty of object groups can be seen when you need to make changes to existing firewall rules. For example, suppose you have a host on the DMZ (10.1.1.1) that needs to communicate with another host on the inside (192.168.10.30). With an access list, you might configure these ACEs:

```
Firewall(config)# access-list acl_dmz extended permit tcp host 10.1.1.1 host
192.168.10.30 eq https
Firewall(config)# access-list acl_dmz extended permit tcp host 10.1.1.1 host
192.168.10.30 eq www
Firewall(config)# access-list acl_dmz extended permit tcp host 10.1.1.1 host
192.168.10.30 eq sqlnet
Firewall(config)# access-list acl_dmz extended permit tcp host 10.1.1.1 host
192.168.10.30 eq 5003
```

That seems straightforward enough, until you need to add a second DMZ host at a later date. You could replicate the four ACE command lines to permit traffic from the new DMZ server, as follows:

```
Firewall(config)# access-list acl_dmz extended permit tcp host 10.1.1.1 host
192.168.10.30 eq https
Firewall(config)# access-list acl_dmz extended permit tcp host 10.1.1.1 host
192.168.10.30 eq www
Firewall(config)# access-list acl_dmz extended permit tcp host 10.1.1.1 host
192.168.10.30 eq sqlnet
Firewall(config)# access-list acl_dmz extended permit tcp host 10.1.1.1 host
192.168.10.30 eq 5003
Firewall(config)# access-list acl_dmz extended permit tcp host 10.1.1.2 host
192.168.10.30 eq https
Firewall(config)# access-list acl_dmz extended permit tcp host 10.1.1.2 host
192.168.10.30 eq www
Firewall(config)# access-list acl_dmz extended permit tcp host 10.1.1.2 host
192.168.10.30 eq sqlnet
Firewall(config)# access-list acl_dmz extended permit tcp host 10.1.1.2 host
192.168.10.30 eq 5003
```

Replicating ACEs might become cumbersome if you continue to add even more DMZ hosts that need the same rules. Even worse, what if you begin adding hosts on the inside that need to receive the same types of traffic from the DMZ hosts?

Instead, you can use object groups to work smarter. Define one network object group for the DMZ hosts and another for the inside hosts:

```
Firewall(config)# object-group network dmz_hosts
Firewall(config-network)# network-object host 10.1.1.1
Firewall(config-network)# network-object host 10.1.1.2
Firewall(config-network)# exit
Firewall(config)# object-group network inside_hosts
Firewall(config-network)# network-object host 192.168.10.30
Firewall(config-network)# exit
Firewall(config)# access-list acl_dmz extended permit tcp object-group dmz_hosts object-
group inside_hosts eq https
Firewall(config)# access-list acl_dmz extended permit tcp object-group dmz_hosts object-
group inside_hosts eq www
Firewall(config)# access-list acl_dmz extended permit tcp object-group dmz_hosts object-
group inside_hosts eq sqlnet
Firewall(config)# access-list acl_dmz extended permit tcp object-group dmz_hosts object-
group inside_hosts eq 5003
```

Now when you need to add a new DMZ host, you simply make one addition to the *dmz_hosts* object group:

```
Firewall(config)# object-group network dmz_hosts
Firewall(config-network)# network-object host 10.1.1.3
Firewall(config-network)# exit
```

The firewall automatically expands the object group within the ACL, and the necessary rules are replicated with the new DMZ host address! You can easily add new addresses to the list of inside hosts as well.

You could also carry this idea one step further by creating a service object group that contains all of the TCP ports that are common to the groups of hosts. To do this, you could enter the following configuration commands:

```
Firewall(config)# object-group service dmz_inside tcp
Firewall(config-service)# port-object eq https
Firewall(config-service)# port-object eq www
Firewall(config-service)# port-object eq sqlnet
Firewall(config-service)# port-object eq 5003
Firewall(config-network)# exit
Firewall(config)# access-list acl_dmz extended permit tcp object-group dmz_hosts object-
group inside_hosts object-group dmz_inside
```

The basic service object group must be referenced in the place of source or destination ports in the ACE.

Beginning with ASA 8.0, you have the advantage of defining an enhanced service object group that can contain any combination of protocol, source port, and destination port parameters. After you configure an enhanced service object group, you need reference it only once in the **access-list** command. The firewall takes care of expanding the object group into all of the appropriate parameter locations in the ACE.

This makes the **access-list** command syntax a little simpler:

```
Firewall(config)# access-list acl_id [line line-num] [extended] {permit | deny}
   object-group enh_service_obj_group
   {source_addr  source_mask | object-group network_obj_group}
   {destination_addr destination_mask | object-group network_obj_group}
   [log [[disable | default] | [level]]] [interval secs]]
   [time-range name] [inactive]
```

Notice that the enhanced service object group completely takes the place of the protocol, source port, and destination port parameters.

The following example shows the configuration of an enhanced service object group to identify the ESP protocol; ISAKMP (UDP destination port 500); UDP destination ports 10000 through 10001; TCP destination port 10000; ICMP echo, echo-reply, and time-exceeded types; and connections from TCP source port 5000 to destination port 6970. Then the object group is applied to access list **acl_outside** from any source address to destination host 10.10.10.10.

```
Firewall(config)# object-group service group1
Firewall(config-service)# service-object esp
Firewall(config-service)# service-object udp eq isakmp
Firewall(config-service)# service-object udp range 10000 10001
Firewall(config-service)# service-object tcp eq 10000
Firewall(config-service)# service-object icmp echo
Firewall(config-service)# service-object icmp echo-reply
Firewall(config-service)# service-object icmp time-exceeded
Firewall(config-service)# service-object tcp source 5000 6970
Firewall(config-service)# exit
Firewall(config)# access-list acl_outside extended permit object-group group1 any host
10.10.10.10
```

Because an enhanced service object group is referenced only once in an ACE, it might be confusing how it actually expands within the access list. You can see the final results with the **show access-list** command:

```
asa-a/admin# show access-list acl_outside
access-list acl_outside; 8 elements
access-list acl_outside line 1 extended permit object-group group1 any host 10.10.10.10
0x1a469c46
access-list acl_outside line 1 extended permit esp any host 10.10.10.10 (hitcnt=0)
0x6b6ae91e
access-list acl_outside line 1 extended permit udp any host 10.10.10.10 eq isakmp
(hitcnt=0) 0x2b45ae69
access-list acl_outside line 1 extended permit udp any host 10.10.10.10 range 10000 10001
(hitcnt=0) 0x25eb4d7a
access-list acl_outside line 1 extended permit tcp any host 10.10.10.10 eq 10000
(hitcnt=0) 0x1dda7d99
access-list acl_outside line 1 extended permit icmp any host 10.10.10.10 echo (hitcnt=0)
0xa2294c03
access-list acl_outside line 1 extended permit icmp any host 10.10.10.10 echo-reply
(hitcnt=0) 0xb1c482a3
access-list acl_outside line 1 extended permit icmp any host 10.10.10.10 time-exceeded
(hitcnt=0) 0xd8647b13
```

```
access-list acl_outside line 1 extended permit tcp any eq 5000 host 10.10.10.10 eq 6970
(hitcnt=0) 0xcc3269c
asa-a/admin#
```

Logging ACE Activity

By default, no syslog messages are generated for **permit** ACEs, but each traffic flow that matches a **deny** ACE triggers a syslog 106023 message (default severity level 4, warnings). If you want to disable syslog completely for an individual ACE, add the **log disable** keywords. For example, by adding **log disable** to the following command, the firewall will not generate a syslog message:

```
Firewall(config)# access-list acl_outside deny icmp any any log disable
```

TIP You can also change the severity level of the 106023 message, if the default level 4 is not appropriate. Use the following command:

```
Firewall(config)# logging message 106023 level level
```

The new *level* is one of the following keywords or numbers: **emergencies** (**0**), **alerts** (**1**), **critical** (**2**), **errors** (**3**), **warnings** (**4**), **notifications** (**5**), **informational** (**6**), or **debugging** (**7**).

You can also enable syslog activity for both permit and deny conditions. When the **log** keyword is added to an **access-list** command, it will enable syslog 106100 messages (default severity 6, informational) for each connection that is permitted or denied by the ACE. Subsequent flows or connections cause the ACE hit count to be incremented.

After a time interval, another syslog 106100 message is generated for the connection, showing the updated hit count. You can use the **interval** keyword to alter this syslog hold interval to *secs* (1 to 600 seconds, default 300).

You can also change the syslog severity level for message 106100 by adding the *level* value (0 to 7, default 6 or "informational").

TIP Syslog usage on a Cisco firewall is covered in more detail in Chapter 10.

If the ACE with the **log** keyword is denying packets, the firewall caches the denied flow so that it can be tracked. Because of this, there must be a limit to the number of cached denied flows to keep the firewall's memory from becoming exhausted.

After the limit is reached, syslog message 106101 is generated. You can limit the number of cached denied flows with the following command:

```
Firewall(config)# access-list deny-flow-max n
```

This sets the maximum number of flows denied by an ACE to *n* flows. The maximum limit is dependent upon the amount of firewall RAM: 4096 flows (more than 64 MB RAM), 1024 flows (16 MB to 64 MB RAM), or 256 flows (less than 16 MB RAM). On a firewall running PIX 7.0 or later, you can use the **access-list deny-flow-max ?** command to display the maximum limit value for that platform.

You can also set the cache limit syslog interval with the following command:

```
Firewall(config)# access-list alert-interval secs
```

Syslog message 106101 is generated each time the cache limit is reached, but only at an interval of *secs* (1 to 3600 seconds, default 300).

Monitoring Access Lists

You can review an access list definition by displaying the firewall configuration with this EXEC command:

```
Firewall# show running-config
```

or

```
Firewall# write term
```

To jump right to the access-list in the configuration, you can use this variation:

```
Firewall# show running-config | begin access-list [acl_id]
```

Or to display only the lines of the access-list configuration and nothing else, you can use a further variation:

```
Firewall# show running-config | include access-list [acl_id]
```

Beginning with ASA 7.0, you can display an access-list configuration with this command:

```
Firewall# show running-config access-list [acl_id]
```

Object groups and access list contents are shown exactly as they were configured. In fact, only the object group references are shown in the ACL configuration; the actual object group definitions are shown in a different point in the configuration. This makes it difficult to review a large access list because you have to refer back and forth between the ACL and any object groups.

After an access list has been configured and applied to an interface, you can monitor its use. Use this EXEC command to see a breakdown of ACL contents and activity counters:

```
Firewall# show access-list [acl_id]
```

Each line of the ACL is shown, along with a hit counter indicating how many connections or flows (or packets for ICMP) have been matched by that line. This is shown as "(hitcnt=n)" at the end of each ACE. For example, an access list configured to permit inbound HTTP connections to several web servers is shown to have the following contents and hit counters:

```
Firewall# show access-list acl_outside
access-list acl_outside line 1 permit tcp any host 192.168.3.16 eq www (hitcnt=97)
access-list acl_outside line 2 permit tcp any host 192.168.3.19 eq www (hitcnt=69513)
access-list acl_outside line 3 permit tcp any host 192.168.3.23 eq www (hitcnt=12)
access-list acl_outside line 4 permit tcp any host 192.168.3.231 eq www (hitcnt=82)
access-list acl_outside line 5 permit tcp any host 192.168.3.242 eq www (hitcnt=27)
```

From this information, it is clear that host 192.168.3.19 (line 2) is receiving the greatest volume of inbound HTTP connections.

TIP Beginning with ASA 7.3(1) and FWSM 3.1, each line of the **show access-list** command output ends with a unique string of hex numbers. For example, see how the following line ends with *0xa2294c03*:

```
access-list acl_outside line 5 extended permit tcp any host 192.168.3.242 eq
www (hitcnt=27) 0xa2294c03
```

The hex string listed is the same string that is generated in syslog messages 106023 (deny packet by ACL, default warnings level) and 106100 (deny/permit packet by ACL, default informational level). This does not mean much for human readers, but it gives Adaptive Security Device Manager (ASDM) an easy way to reference syslog messages it collects with the actual ACL lines that generated the messages.

Now suppose that an object group has been configured to list the web servers with the following commands:

```
Firewall(config)# object-group network web-servers
Firewall(config-network)# network-object host 192.168.3.16
Firewall(config-network)# network-object host 192.168.3.19
Firewall(config-network)# network-object host 192.168.3.23
Firewall(config-network)# network-object host 192.168.3.231
Firewall(config-network)# network-object host 192.168.3.242
Firewall(config-network)# exit
Firewall(config)# access-list acl_outside permit tcp any object-group web-servers eq www
```

Using the **show access-list** command also expands any object groups that are referenced in an ACL. This allows you to see the actual ACEs that the firewall is evaluating. In this example, the ACL would be expanded as follows:

```
Firewall# show access-list acl_outside
access-list acl_outside line 1 permit tcp any object-group web-servers eq www
access-list acl_outside line 1 permit tcp any host 192.168.3.16 eq www (hitcnt=97)
access-list acl_outside line 1 permit tcp any host 192.168.3.19 eq www (hitcnt=69513)
access-list acl_outside line 1 permit tcp any host 192.168.3.23 eq www (hitcnt=12)
access-list acl_outside line 1 permit tcp any host 192.168.3.231 eq www (hitcnt=82)
access-list acl_outside line 1 permit tcp any host 192.168.3.242 eq www (hitcnt=27)
```

Notice that each line of the output is shown as line 1 of the ACL. Although the object group is expanded and evaluated as sequential ACEs, it appears as the one ACE that referenced it.

TIP You can reset the hit counters of an ACL by using this command:

```
Firewall# clear access-list acl_id counters
```

In releases prior to ASA 7.0, be careful when you use this command—if you omit the **counters** keyword, the entire ACL named *acl_id* is removed from the firewall configuration! ASA 7.0 and later, as well as all releases of FWSM, do not allow this command to be executed unless the **counters** keyword is included, so you are in no danger of deleting any configuration.

6-4: Shunning Traffic

Sometimes it might be possible for malicious hosts to open connections into the protected network. This could occur if the inbound access list policies are not configured correctly or tightly. As soon as these connections are noticed (after they are built), you might want to react by blocking connections coming from the malicious source address.

To do this, you could edit the access list each time the source of an attack is discovered. This would deny any future connections; xlate entries would also need to be cleared to drop existing connections. This would also quickly become an administrative burden.

A more efficient alternative is the **shun** command. When a shun is activated, all current connections from a malicious host can be dropped and all future connections blocked.

Connections are shunned regardless of the firewall interface being traversed. The firewall examines the connection table and the connection building process to identify and shun the specified connections.

Shuns can be configured through the firewall command-line interface (CLI) or through an automatic action from a Cisco Intrusion Protection System or an integrated feature such as Threat Detection. After shuns are configured, they remain in place until they are removed.

Shuns are dynamic in nature, and are *not* stored as a part of the firewall configuration. If the firewall loses power or reloads, any active shuns are lost. As well, shuns are *not* maintained in a failover firewall pair. If the units failover, any active shuns are lost.

You can use the following steps to configure a shun:

1. Manually shun connections:

ASA, FWSM	`Firewall# shun src_ip [dst_ip sport dport [protocol]] [vlan_id]`
PIX 6.3	`Firewall# shun src_ip [dst_ip sport dport [protocol]]`

You can shun any new connections (any IP protocol) passing through the firewall originating from source address *src_ip*. This is most useful to stop an attack that is in progress from the source address to many destinations. Any existing connections stay up, however. Those must idle out of the xlate and conn tables normally, or you can clear any related xlate entries to manually kill the connections.

For more granular shunning, you can also identify the destination address *dst_ip*, the source and destination ports *sport* and *dport*, and the *protocol*. You can only define one shun entry per source and destination address pair. When a shun is defined, all existing and future connections are blocked until the shun is later removed.

2. Display active shuns:

```
Firewall# show shun [src_ip]
```

All active shuns are listed. If a specific source address *src_ip* is given, only shuns involving that address are shown.

As an example, the following output displays the four shuns that are currently active. The source interface is automatically determined and shown in parentheses.

```
Firewall# show shun
shun (outside) 172.21.104.93 0.0.0.0 0 0
shun (outside) 172.21.196.50 0.0.0.0 0 0
shun (inside) 192.168.198.24 0.0.0.0 0 0 0
shun (outside) 10.10.1.1 172.21.4.19 0 80 6
Firewall#
```

Notice in the shaded output line that an inside host has been the target of a shun. Shuns can be used on any host located on any interface. In this case, the inside host was playing the role of the malicious user, attacking hosts on the outside of the firewall.

You can monitor the activity of each active shun with the **show shun statistics** command. Each of the firewall interfaces are shown, along with the current shun activity. The firewall looks at its routing information to determine the interfaces where shun source addresses can be found. These interfaces are shown as "ON". A cumulative count of shunned connections is also shown.

Each configured shun is listed with its source address, a cumulative count of shunned connections, and the total elapsed time since the shun was enabled.

For example, a firewall is configured with a long list of shun commands. Notice that the outside interface, where malicious hosts on the public Internet were discovered, has had 17,184,951 shunned connections. The inside interface has had even more! In this case, a number of inside hosts have been discovered to be compromised and participating in malicious activity toward the outside network. Until these hosts can be cleaned, they have been "quarantined" through the use of firewall shuns.

```
Firewall# show shun statistics
stateful=OFF, cnt=0
dmz2=OFF, cnt=0
outside=ON, cnt=17184951
```

```
inside=ON, cnt=255823449

Shun 172.21.96.89 cnt=32502918, time=(112:04:34)
Shun 172.21.61.83 cnt=0, time=(112:04:32)
Shun 172.21.24.79 cnt=0, time=(112:04:35)
Shun 172.21.108.68 cnt=0, time=(112:04:35)
Shun 192.168.93.16 cnt=0, time=(112:04:34)
Shun 172.21.184.106 cnt=21277328, time=(112:04:33)
Shun 192.168.97.9 cnt=0, time=(112:04:34)
Shun 172.21.184.107 cnt=21264263, time=(112:04:33)
Shun 192.168.228.11 cnt=0, time=(243:35:21)
Shun 192.168.228.12 cnt=0, time=(243:35:18)
Shun 192.168.228.13 cnt=0, time=(243:35:16)
Shun 172.21.184.108 cnt=21311395, time=(112:04:33)
Shun 192.168.228.14 cnt=0, time=(243:35:12)
Shun 192.168.228.15 cnt=0, time=(243:35:10)
Shun 172.21.72.99 cnt=334699, time=(112:04:34)
[output omitted]
```

TIP To avoid sifting through long lists of shun statistics to find a single source address, you can filter the output through the **include** or **grep** commands. In this example, only the shun for source address 172.21.72.99 is needed. It is shown to have blocked 334,699 packets, and has been active for 112 hours, 13 minutes, and 19 seconds:

```
Firewall# show shun statistics | include 172.21.72.99
Shun 172.21.72.99 cnt=334699, time=(112:13:19)
Firewall#
```

You can remove an existing shun for a specific source address with the following global configuration command:

```
Firewall(config)# no shun src_ip
```

Shun Example

A host at 172.21.4.8 is discovered to be involved in malicious activity. (In this example, only a Telnet connection is shown for simplicity.) A shun will be configured on the firewall to stop any current or future connections involving that host.

First, look at an active connection involving 172.21.4.8:

```
Firewall# show conn
1 in use, 3 most used
TCP out 172.21.4.8:4334 in 192.168.199.100:23 idle 0:00:04 Bytes 138 flags UIOB
Firewall#
```

It does have at least one active connection, so a shun is put into place:

```
Firewall# shun 172.21.4.8
```

```
Shun 172.21.4.8 successful
Firewall# show conn
1 in use, 3 most used
TCP out 172.21.4.8:4334 in 192.168.199.100:23 idle 0:04:25 Bytes 138 flags UIOB
Firewall#
```

Indeed, the current connection has become unresponsive from 172.21.4.8 (as shown by the increasing idle time), and it is also unable to start new sessions through the firewall. But why is the connection still shown in the firewall's connection table?

First, look at the buffered syslog information on the firewall. Hopefully it will show something interesting:

```
Firewall# show logging
Syslog logging: enabled
    Facility: 20
    Timestamp logging: enabled
    Standby logging: disabled
    Console logging: disabled
    Monitor logging: disabled
    Buffer logging: level informational, 3523423 messages logged
    Trap logging: level informational, 3523423 messages logged
        Logging to outside 192.168.199.10 (EMBLEM format)
    History logging: disabled
    Device ID: hostname "Firewall"
401002: Shun added: 172.21.4.8 0.0.0.0 0 0
111008: User 'enable_15' executed the 'shun 172.21.4.8' command.
401004: Shunned packet: 172.21.4.8 ==> 169.54.89.249 on interface outside
401004: Shunned packet: 172.21.4.8 ==> 169.54.89.249 on interface outside
401004: Shunned packet: 172.21.4.8 ==> 169.54.89.249 on interface outside
401004: Shunned packet: 172.21.4.8 ==> 169.54.89.249 on interface outside
```

The syslog record shows that the shun was put into place and that it is actively shunning new packets. Why didn't the active connection drop immediately? The answer is this: After the shun became active, the firewall began dropping packets involving 172.21.4.8. The TCP connection shown in the connection table was already established, after a three-way handshake. As soon as packets started being dropped, the TCP connection became isolated; each host in the connection was no longer able to hear from the other end.

As a result, the TCP connection becomes a "half-open" or "half-closed" connection until it either times out or both endpoints handshake with a FIN flag. The firewall will keep the shunned connection in its table until its own half-closed timer expires. This can be seen by verifying the timer value and the shun statistics:

```
Firewall# show conn
1 in use, 3 most used
TCP out 172.21.4.8:4334 in 192.168.199.100:23 idle 0:06:48 Bytes 138 flags UIOB
pix-f# sh timeout
timeout xlate 3:00:00
timeout conn 1:00:00 half-closed 0:10:00 udp 0:02:00 rpc 0:10:00 h225 1:00:00
timeout h323 0:05:00 mgcp 0:05:00 sip 0:30:00 sip_media 0:02:00
timeout uauth 0:05:00 absolute
```

```
Firewall# show shun statistics
outside=ON, cnt=13
inside=ON, cnt=0
dmz=OFF, cnt=0

Shun 10.1.1.1 cnt=0, time=(121:03:07)
Shun 192.168.200.199 cnt=0, time=(119:52:22)
Shun 172.21.4.8 cnt=13, time=(0:03:58)
Firewall#
```

Indeed, the half-closed timer is 10 minutes and the active connection has only been shunned for 3 minutes and 58 seconds. As soon as 10 minutes have elapsed, the TCP connection is deleted from the firewall's table.