CHAPTER 17

DoS Protection

Of the three categories of attacks—reconnaissance, access, and denial-of-service (DoS)—DoS attacks are the easiest to implement yet the hardest to defeat. DoS attacks are based on packet flooding, which uses up bandwidth, CPU, and memory resources on not just the victim device, but also intervening devices, such as routers, switches, and firewalls.

When you are experiencing a DoS attack, one of the first things you need to do is find out the actual kind of DoS attack that is affecting your network. As you will see in the first section of this chapter, a variety of options are available to you, including examining the CPU utilization of your routers, using ACL statements with logging parameters, and using NetFlow.

When you know the kind of DoS attack directed at your network, you can implement an appropriate solution. The remaining sections in this chapter focus on these solutions, including TCP Intercept, CBAC, and rate limiting. Of course, you always can use an ACL to block offending traffic; however, this might introduce other problems, such as the blocking of legitimate traffic. Therefore, in many cases, you need to use other tools, such as the ones discussed in the last half of this chapter, to deal with DoS issues.

Detecting DoS Attacks

A wide variety of DoS attacks can occur. The most common DoS attacks use UDP echos (Fraggle), ICMP echo and echo replies (Smurf), and TCP (TCP SYN flooding). However, you never should assume that the DoS attack that you are experiencing falls under one of these three types: Some do not fall under these categories.

Common Attacks

One of the most common DoS attacks is the Smurf attack, which I covered in Chapter 7, "Basic Access Lists." (Chapter 7 also covered how to use ACLs to deal with the Smurf attack.) In a Smurf attack, the attacker sends a flood of ICMP messages to a reflector or sets of reflectors, with the source IP address in the ICMP echo messages spoofed. The hacker changes these addresses to the address of the actual victim device or devices. The reflectors then innocently reply to the echo messages, inadvertently sending the replies to the victim. In many cases, the source address is a directed broadcast address, allowing the attack to

target a network segment instead of a specific host. Based on this information, if you see a large number of ICMP echo replies coming into your network, you probably are experiencing a Smurf attack, or at least a derivative of this kind of attack. Likewise, if you are seeing a large number of packets coming into your network with a directed broadcast address, this would indicate that you are under attack. One method of preventing this is to use the **no ip directed-broadcast** command on your router's interfaces. I discussed this command in Chapter 4, "Disabling Unnecessary Services."

The Fraggle DoS attack is similar to the Smurf attack, except that Fraggle uses UDP echo and echo reply messages instead of ICMP messages. Because UDP echos typically are filtered, Fraggle typically has a less likelihood of having the same impact that Smurf does (I discussed in Chapter 4 how to defeat a Fraggle attack by configuring the **no service tcp-small-servers** and **no service udp small-servers** commands).

The other common attack is a TCP SYN flood attack, in which an attacker tries to overwhelm one or more TCP services running on a machine. The attacker executes this attack by sending a flood of TCP SYN segments to the victim. In most cases, the hacker forges the source address to hide his tracks or possibly to create another DoS attack against a second victim. Therefore, when the server receives the TCP SYNs and tries to complete the three-way handshake, the process fails: The server sends a SYN/ACK response to a bogus or invalid address and, therefore, never gets the final ACK reply in the three-way handshake. During this period, resources are tied up on the server for each of these half-open (commonly called *embryonic*) connections. Because this kind of attack typically is targeted at a specific machine, when you experience a slowing of your server that offers TCP services, such as an e-mail, web, or FTP server, or if one of these servers crashes or halts, this might indicate that it was (and possibly still is) the victim of a TCP SYN flood attack.

Symptoms of Attacks

When your router is experiencing a DoS attack (including worm attacks), either one directed at the router or one in which the router is a transit device, you can look for the following to help you confirm your hunches:

- Your router is seeing an unusually high number of ARP requests.
- Your NAT/PAT address-translation tables have a large number of entries.
- Your router's IP Input, ARP Input, IP Cache Ager, and CEF processes are using abnormally high amounts of memory.
- Your router's ARP, IP Input, CEF, and IPC processes are running at a much higher CPU utilization rate.

TIP The following sections discuss various methods for detecting and tracing DoS attacks. However, one of the best solutions for detecting DoS attacks is to use an IDS solution. A good IDS solution should tell you exactly the kind of DoS attack that you are facing.

Examining CPU Utilization to Detect DoS Attacks

In any DoS attack situation, the network symptoms that you see typically will be common, such as high CPU utilization on your devices or a high number of certain kinds of packets. Therefore, one of the first things you want to do is to log into your perimeter routers and firewalls and examine their CPU utilization.

On a Cisco router, use the **show processes cpu** command to examine the router's CPU utilization, as shown in Example 17-1.

Example 17-1 *Using the* **show processes cpu** *Command*

```
Router# show processes cpu
CPU utilization for five seconds: 92%/34%; one minute: 90%; five minutes: 45%
 PID Runtime(ms)   Invoked      uSecs   5Sec   1Min   5Min TTY Process
   1           0         1          0  0.00%  0.00%  0.00%   0 Chunk Manager
   2           0       294          0  0.00%  0.00%  0.00%   0 Load Meter
   3        7532      2365       3184  0.00%  0.00%  0.24%   0 Exec
   4           0        13          0  0.00%  0.00%  0.00%   0 DHCPD Timer
   5         800       156       5128  0.00%  0.02%  0.03%   0 Check heaps
   6           0         1          0  0.00%  0.00%  0.00%   0 Pool Manager
<--output omitted-->
```

The shaded line is what you should focus on. Three statistics are listed here:

- **CPU utilization for five seconds**—This is the CPU utilization on the router for the last 5 seconds. The number after the slash (/) is the percentage of CPU time spent at the interrupt level. During an interrupt, the CPU must handle the functions for the process instead of an interface or a specific ASIC (application-specific integrated circuit).
- **One minute**—This is the CPU utilization of the router over the past minute.
- **Five minutes**—This is the CPU utilization of the router over the past 5 minutes.

In Example 17-1, the 5-second utilization is 92 percent and the 1-minute utilization is 90 percent. Notice that the 5-minute utilization is only 45 percent. This probably indicates a brief spike in traffic. If all three of these are at a higher number than normal and are close together in values, such as 75 or 95 percent (depending on your router's baseline, of course), this could indicate a DoS attack. In this situation, you should look at other things on your router to determine whether this is just an anomaly in traffic conditions or a DoS attack.

A derivation of the **show processes cpu** command is the addition of the **history** parameter. This command displays, in an ASCII graphical format, the total CPU usage of the router over 1 minute, 1 hour, and 72 hours. Example 17-2 shows an example of the use of this command.

Example 17-2 *Using the* **show processes cpu history** *Command*

```
Router# show processes cpu history
<-- One minute output omitted -->
     6665776865756676676666667667677676766666766767767666566667
     6378016198993513709771991443732358689932740858269643922613
```

continues

Example 17-2 *Using the* **show processes cpu history** *Command (Continued)*

```
100
 90
 80          ** *                     * *     *  * *  *
 70  * * ****# *  ** ***** ***  **** ******  *  *** ***     *
 60  #***##*##*#***#####*#*###*****#*###*#*#*##*#*#**#*##*****
 50  ############################################################*
 40  ############################################################
 30  ############################################################
 20  ############################################################
 10  ############################################################
    0....5....1....1....2....2....3....3....4....4....5....5....
              0    5    0    5    0    5    0    5    0    5

               CPU% per minute (last 60 minutes)
              * = maximum CPU%   # = average CPU%

<-- 72-hour output omitted -->
```

In Example 17-2, the Y-axis of the graph represents CPU utilization, and the X-axis represents the increment period. In this example, I omitted the output for the 1-minute and 72-hour output and listed only the hour output.

The maximum usage, represented by a *, is measured and recorded every second (this is displayed at the very top of a vertical entry). The average usage is measured over a period of 1 second and is represented by a #.

The very top of the graph lists the actual CPU utilization for the 1-minute interval, which fluctuated between 60 and 80 percent utilization.

For worm attacks, examine the following information in the output of the **show process cpu** command: ARP, IP Input, CEF, and IPC processes. Also examine the following information in the output of the **show processes memory** command: IP Input, ARP Input, IP Cache Ager, and CEF processes.

NOTE Of course, looking at the CPU utilization during a DoS attack assumes that you have captured a history of CPU utilization baselines during normal traffic conditions. If you do not have this information and your CPU utilization is running at 75 percent, this might be a normal condition for your router. Therefore, you periodically should gather statistical information about your router, including CPU utilization and the bandwidth utilization on each of its interfaces. This will help you determine whether something abnormal, such as a DoS attack, is directed at your router or is flowing through your router. Also, if your CPU utilization is running at 100 percent because of a severe DoS or worm attack, your router might reload itself. Therefore, you also should check to see if your router periodically is reloading, which could indicate that you are under attack.

Using ACLs to Detect DoS Attacks

One of the most common tools used to detect DoS attacks is ACLs. This is especially true of your perimeter routers and router firewalls because these devices already have ACLs in place to filter traffic. As I mentioned in Chapter 7, you always want to include a **deny ip any any** statement in your ACL at the end. This is not necessary to drop traffic, but it does have the router keep statistics on the number of matches on this statement. This can be useful in determining whether a DoS attack is occurring.

ACL Counters

The first thing you should do is clear the counters for your ACL with the following command:

```
Router# clear access-list counters [ACL_#_or_name]
```

After this, view the ACL in question with the **show access-lists** or **show ip access-list** commands. Example 17-3 shows sample output from the **show access-list** command.

Example 17-3 *Displaying the Counters for ACL Statement Matches*

```
Router# clear access-list counters 100
Router# show access-list 100
Extended IP access list 100
<--output omitted-->
  deny ip any any (3183 matches)
Router#
Router# show access-list 100
Extended IP access list 100
<--output omitted-->
  deny ip any any (7225 matches)
```

Notice in this example that, after I cleared the ACL counters for ACL 100 and then viewed the ACL the first time (right after clearing the ACL counters), the match on the **deny ip any any** statement is 3183, which is relatively high for just a couple of seconds between executing the two commands. Right after executing the first **show** command, I re-executed it. Notice that in the second or two between the execution of the two commands, the number of matches more than doubled. In this example, I was sending a flood of ICMP packets from three devices to one victim device.

Here are some important things to keep in mind about using ACLs to detect a DoS attack:

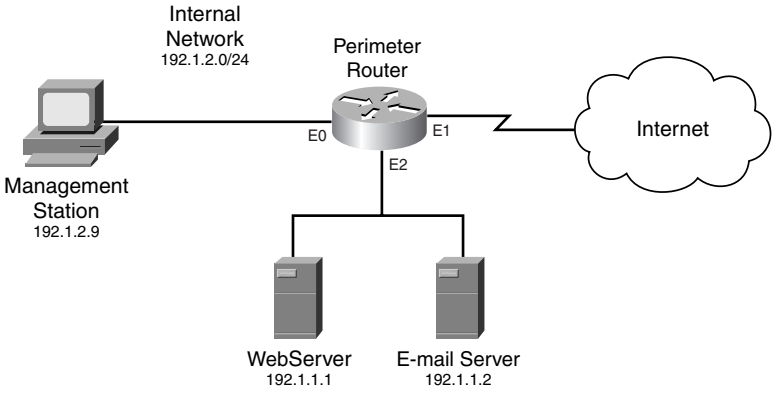
- In this example, I assumed that the traffic being dropped was the DoS attack. You also need to examine the matches on **permit** entries because your ACL might be permitting the DoS attack, such as a TCP SYN flood. Again, a good baseline of this information should assist you in determining whether a DoS attack is occurring. In other words, you should have gathered ACL statistics in normal traffic periods, and then you can use this to make a comparison.

- Remember that because you are using extended ACLs, you can specify the IP protocol, source and destination addresses, and protocol information, such as TCP and UDP port numbers as well as ICMP message types. This enables you to be as specific as possible in determining a match.
- If the ACL in question is applied to multiple interfaces, remember that the ACL counters are an aggregate of all of the matches on all interfaces on which the ACL has been activated. In this situation, you might want to have a separate named/numbered ACL for each interface.

Specific ACL Entries

Of course, the problem with the previous examination is that you can see that your router is dropping a flood of traffic, but you do not see what traffic the router is dropping. For example, if you do not want to allow any type of ICMP traffic, or even if you want to permit only certain kinds of ICMP traffic, put in the appropriate ACL entries so that, minimally, you can see ACL statistics on matches. Refer to Figure 17-1 for an example. Here, no echo messages should be allowed into the network, and only echo replies to the management station are allowed.

Figure 17-1 *ACL Example*



Example 17-4 shows a basic ACL example for this network.

Example 17-4 *A Basic ACL Example*

```
Router(config)# remark Insert other ACL statements here
Router(config)# access-list 100 deny ip any 192.1.1.0 0.0.0.0     (1)
Router(config)# access-list 100 deny ip any 192.1.1.255 0.0.0.0
Router(config)# access-list 100 deny ip any 192.1.2.0 0.0.0.0
Router(config)# access-list 100 deny ip any 192.1.2.255 0.0.0.0
Router(config)# access-list 100 permit icmp any                   (2)
  host 192.1.2.9 echo-reply
Router(config)# access-list 100 deny icmp any any echo            (3)
```

Example 17-4 *A Basic ACL Example (Continued)*

```
Router(config)# access-list 100 deny icmp any any echo-reply      (4)
Router(config)# access-list 100 deny udp any any eq echo          (5)
Router(config)# access-list 100 deny udp any eq echo any          (6)
Router(config)# access-list 100 permit tcp any                    (7)
  host 192.1.1.1 eq 80 established
Router(config)# access-list 100 permit tcp any                    (8)
  host 192.1.1.1 eq 80
Router(config)# access-list 100 permit tcp any                    (9)
  host 192.1.1.2 eq 25 established
Router(config)# access-list 100 permit tcp any                   (10)
  host 192.1.1.2 eq 25
Router(config)# remark Insert other ACL statements here
Router(config)# access-list 100 deny ip any any
Router(config)# interface ethernet1
Router(config-if)# ip access-group 100 in
```

This example not only sets up filtering, but it also has specific entries that can be used to detect four DoS attacks: directed broadcast attacks (Statement 1), which are used by many types of DoS attacks; a Smurf attack (Statements 2, 3, and 4); a Fraggle attack (Statements 5 and 6), and TCP SYN floods (Statements 7 and 8):

- For a Smurf attack, if the entries in Statements 2 or 4 are increasing rapidly, this indicates that Smurf reflectors are attacking you (you are the victim). If you are the reflector, you will see entries in Statement 3 rapidly increasing. This is especially true if you do not configure the group of statements in Statement 1. However, to determine this, you would need to enable logging for the ICMP ACL statements, which I discuss in the next section.
- For a Fraggle, attack, if the entries in Statement 5 are increasing, this indicates that you are the victim; a large number of matches against Statement 6 indicates that you are the reflector.
- For a TCP SYN flood attack, you will see the number of matches against Statements 8 and 10 increasing many times over normal baseline numbers. What is unusual about these two ACL statements is that they are normal ACL statements allowing traffic to the e-mail and web servers. However, remember that the Cisco IOS keeps track of the number of matches on all statements, so a large increase in the number of matches for either of these two servers could indicate a TCP SYN flood attack. Because there are many types of TCP flood attacks, using various TCP flags, Statements 7 and 9 look at the other TCP flags for an abnormal number of other TCP messages.

Also, the statements under Statement 1 block traffic sent to the network and directed broadcast addresses of the internal network. These addresses are commonly used in DoS attacks. By having specific entries in your ACL, you can use the **show access-lists** command to determine more easily whether a DoS attack is occurring.

ACL Logging

The main limit of the statistics from the **show access-lists** and **show ip access-list** commands is that they can show you that a high number of hits are occurring against one or a few ACL statements, but this does not tell you who the attacker (or the victim, in some cases) is. One option is to use the **log** or **log-input** parameters at the end of the ACL statements in question. As you recall from Chapter 7, the main difference between these two commands is that the **log-input** parameter displays the input interface of the received packet and the Layer 2 source address in the packet.

CAUTION In either situation, remember that using either of these two parameters disables CEF switching, which seriously impacts the performance of the router. Therefore, use the log function to pinpoint the attack, including the victim and the attacker, and then remove the **log** or **log-input** parameters from your ACL statement(s).

Smurf Example

For example, assume that the router in Figure 17-1 is under a Smurf attack. When you examine the router's ACL, you notice the output in Example 17-5.

Example 17-5 *Using the* **show access-list** *to Pinpoint a Smurf Attack*

```
Router# show access-list 100
Extended IP access list 100
  <--output omitted-->
  permit icmp any host 192.1.2.9 echo-reply (15 matches)
  deny icmp any any echo (1038 matches)
  deny icmp any any echo-reply (238482 matches)
  deny udp any any eq echo
  deny udp any eq echo any
  <--output omitted-->
```

As you can see in Example 17-5, an inordinate number of ICMP echo reply messages is being denied (remember that only the management station is allowed to ping and receive replies).

To determine the problem, you would modify the ACL statement for the dropped ICMP packets by changing ACL Statement 4 in Example 17-4 to this:

```
Router(config)# access-list 100 deny icmp any any echo reply log-input
```

In this situation, I probably would disable logging to the console but enable logging to the internal buffer (this is discussed in Chapter 18, "Logging Events." Therefore, when there is an inordinate number of log messages, the console interface will not be overwhelmed.

Given the Smurf attack in the example, here is a snippet of the output produced by log messages from a match on the ACL statement:

```
%SEC-6-IPACCESSLOGDP: list 100 denied icmp 201.1.1.1
(Ethernet1) -> 192.1.1.1 (0/0), 1 packet
```

```
%SEC-6-IPACCESSLOGDP: list 100 denied icmp 201.1.1.2
(Ethernet1) -> 192.1.1.1 (0/0), 1 packet
%SEC-6-IPACCESSLOGDP: list 100 denied icmp 201.1.1.9
(Ethernet1) -> 192.1.1.1 (0/0), 1 packet
%SEC-6-IPACCESSLOGDP: list 100 denied icmp 205.8.8.7
(Ethernet1) -> 192.1.1.1 (0/0), 1 packet
%SEC-6-IPACCESSLOGDP: list 100 denied icmp 200.1.1.37
(Ethernet1) -> 192.1.1.1 (0/0), 1 packet
%SEC-6-IPACCESSLOGDP: list 100 denied icmp 201.1.1.1
(Ethernet1) -> 192.1.1.1 (0/0), 1 packet
%SEC-6-IPACCESSLOGDP: list 100 denied icmp 205.8.8.32
(Ethernet1) -> 192.1.1.1 (0/0), 1 packet
%SEC-6-IPACCESSLOGDP: list 100 denied icmp 201.1.1.81
(Ethernet1) -> 192.1.1.1 (0/0), 1 packet
```

In this example, a single host is being attacked: 192.1.1.1. Also notice that most of the ICMP traffic is coming from two networks: 201.1.1.0/24 and 205.8.8.0/24. By using the IANA Whois database (http://www.iana.org), you should be able to see who these addresses belong to. It is important to point out that in a Smurf attack, these are probably not the actual attacker devices, but Smurf reflectors (with Smurf, it is very unlikely that the attacker would use his own address as a source, unless he is a script kiddie). However, at least you have a starting point and should be able to contact the ISPs or administrators who are responsible for these networks so that they can take the appropriate action.

Keep the following items in mind when using logging with ACLs:

- Enabling logging with ACLs can have a small- to medium-size impact on your router, depending on its model. Therefore, I recommend that you be careful using the logging function on routers that are running at 80 percent utilization or higher—especially with ACLs on high-speed interfaces.
- Depending on the version of the Cisco IOS running on your router, logging might affect the switching mode that the router must use to implement the logging function.
- If you are seeing matches against ACL entries in the **show access-list** command, but you are not seeing any entries actually being logged (you have added the **log** parameter to the ACL statement), try clearing the route cache to force packets to be process-switched. Of course, this can cause a lot of traffic to be dropped on a very busy router when it is in the process of rebuilding the cache. To get around this specific problem, use a router that supports CEF.
- ACL logging does not display every packet that matches an ACL entry. Instead, you will see at least the first packet on a match for a session and then packets at periodic intervals. The Cisco IOS uses rate limiting to prevent logging from overloading the router's CPU. It is important to remember that you will not see all packets that match an ACL statement when you have logging enabled.
- During a Smurf attack that involves multiple reflectors, the amount of logging information might be overwhelming. In this instance, set up a separate ACL statement that matches on a single reflector and logging for only this reflector. Typically, this will be multiple devices from a single network, so use an ACL that matched on all of these devices. This helps you determine all of the devices in one reflector site.

TIP Be as specific as possible in your ACL statements when looking for the DoS attack. Use the **log-input** parameter so that you can see the interface where the attack is coming from. If the router's interface is a multiaccess interface, this information also will include the MAC address of the next-hop router; use the **show ip arp** *MAC_address* command to see the router's IP address.

NOTE The purpose of this section on ACLs was to give you a basic understanding of how to track down the type of DoS attack. Obviously, you can add or modify ACL entries to determine the kind of DoS attack you are experiencing.

Damage Limitations

The attacker in a DoS attack wants to affect bandwidth and service levels, so your first concern should be preventing this traffic from traversing your ISP links. You could set up an ACL on your perimeter routers to block this traffic, but this stops it only from entering your network; it does not prevent it from traversing your ISP links.

In this situation, you need to contact your ISP and explain the situation. Hopefully they also will investigate the problem on their end and either set up a temporary ACL filter to block the offending traffic or set up rate limiting to reduce the effect that the DoS attack will have. As I mentioned, this should be a temporary filter: When the attack has been mitigated, these measures should be removed.

Finding the Attacker

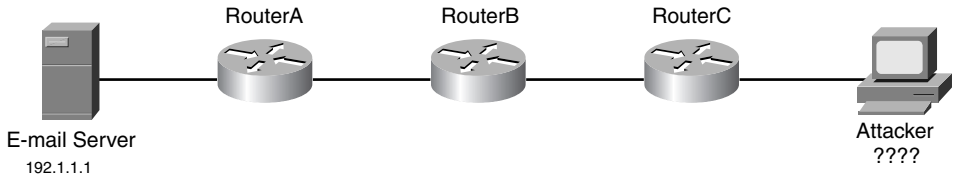
When you are the victim of a DoS attack, tracking down the actual attack, as well as the perpetrator, can be a difficult and stressful task. Remember that the source addresses in a DoS attack typically are forged and, therefore, are not very helpful in finding the actual attacker. Therefore, finding the attacker typically must be done on a hop-by-hop basis, tracing the attack from one router to the next until you finally arrive at the source of the attack. As you saw in the "Using ACLs to Detect DoS Attacks" section, this requires administrators possibly to add ACL entries and/or add logging to specific ACL entries, which can be a complex process. And because the path of the attack might take you back through multiple ISPs, you typically must involve law-enforcement agencies to get the kind of cooperation from ISPs and network administrators that might be necessary to perform this kind of trace.

The Trace Process

Tracing DoS attacks is a straightforward process. In this subsection and the following ones, I discuss the use of ACLs to perform the trace process. However, other options, such as NetFlow, can be used as well. I discuss NetFlow after discussing ACLs.

Remember that the source address in the packet probably is spoofed, so you cannot assume anything about the location of the attacker. Therefore, you have your work cut out for you: The actual trace process requires you to trace back, router by router, to the real source of the attack. Figure 17-2 illustrates this process. In this example, the e-mail server (192.1.1.1) is under a DoS attack. The first thing that you would do is to log into RouterA, which I am assuming is the router directly connected to the same segment as the e-mail server. Remember that the DoS attack might be internal, not external, to the network. From here, you would determine which interface the attack is coming from, log into the next-hop router, and perform the same process, tracing this back eventually to the hacker himself.

Figure 17-2 *Tracing Example*



Trace Problems

In a Smurf attack, in which reflectors are used to instigate the attack against a victim, it is fairly easy to determine who the reflectors are. However, when you have found the reflectors, tracing the attack back to the originator of the attack is difficult, if not impossible, because it must be done on a hop-by-hop basis, probably through many different ISPs. Given this tedious process, you face the following problems when tracking down the attacker who instigated the DoS attack:

- Because multiple administrators and networks typically are involved in the trace-back process, it becomes a tedious and slow process tracing back the attack to the real source. Even an unintelligent hacker knows that you will attempt to trace his attack; therefore, the hacker limits the duration of the DoS attack, which might not give you enough time to trace the attack.
- If the attack goes through multiple networks or ISPs, and possibly even through different countries, getting the appropriate assistance could be difficult.
- The attack might be coming from multiple sources, as in a Smurf attack. With a Smurf attack, you need to trace the attack back to each reflector and then, from each reflector, trace the attack back to the real source or sources of the attack.
- The attack might be perpetrated from a hacked computer that was compromised by the hacker. Therefore, finding the actual perpetrator might be impossible unless the hacked computer has detailed logs of unauthorized access to it.
- When you actually track down the DoS attacker, you will face legal and political issues trying to sue for damages in a civil action. You will need detailed proof of the attack to bring criminal actions against the attacker—and this can be very difficult if the attacker is in a different country.

NOTE Given these problems, you should realize that, in most cases, you will not be able to find the attacker who instigated a DoS attack. Therefore, you should pick your battles. In most cases, you will want to use tools to limit the effect of a DoS attack and attempt to trace back attacks only when they are severe enough to prevent your company from operating or reaching its business goals. Also, if the trace involves multiple companies and ISPs, you typically must get law-enforcement agencies involved to get the information necessary to find the attacker.

Using NetFlow to Detect DoS Attacks

One of the problems of using the **log** or **log-input** parameters with ACL statements is that it is very process-intensive for the router to handle the logging function. Cisco provides another solution, called NetFlow switching. NetFlow enables you to capture statistics of flows traveling through your router. It is a transparent switching method that requires you only to enable it. Unlike ACLs with logging, NetFlow is a switching technology that has much less of an impact on the router. As with ACL logging, you can export NetFlow information to a server, to be examined in more depth.

NetFlow Overview

NetFlow switching is a network-layer switching method that switches packets at high speeds and captures statistics for traffic analysis. It is supported on IP and IP-encapsulated traffic types over a variety of interfaces, including interfaces with input ACLs.

NOTE NetFlow is not supported on ISL trunk interfaces, ATM and Frame Relay interfaces if more than one input ACL is used, and ATM LANE interfaces.

Cisco developed NetFlow switching to provide the following benefits:

- Processing of ACL statements with little performance penalty (at least, compared to many other switching modes)
- Billing for bandwidth usage in corporate and ISP networks
- Capturing of traffic for network management and capacity planning

NetFlow uses flows to identify traffic. A flow is basically a unidirectional session based on the following fields in an IP packet: IP protocol, source and destination IP address, type of service, source and destination port numbers for TCP and UDP flows, and input interface. When NetFlow identifies a new flow, an entry is added to the NetFlow cache. This entry then is used to switch packets and to perform ACL checking.

NetFlow Configuration

Enabling NetFlow is a simple process. You can use many optional parameters to create a more efficient capturing process for traffic statistics. However, I focus on only the basics of enabling NetFlow, exporting NetFlow statistics, and examining the statistics.

Enabling NetFlow

Enabling NetFlow is done on your router's interface(s). When using it to track down DoS attacks, I recommend that you enable it on an interface or interfaces of your router where you think the source of the DoS attack is originating. Here is the basic configuration to enable NetFlow:

```
Router(config)# interface type [slot_#/]port_#
Router(config-if)# ip route-cache flow
```

The **ip route-cache flow** command enables NetFlow switching on a router's interface. If you have a Cisco router that supports distributed switching, such as the 7500 with RSP and VIP modules, you need to execute one more command on the router's interface:

```
Router(config-if)# ip route-cache distributed
```

This command enables distributed switching with NetFlow.

Exporting NetFlow Data

Optionally, you can export NetFlow statistical data to an external device that supports NetFlow data imports. After you set up the export function, NetFlow information is exported every time a flow expires. A few products on the market, including one from Cisco, can be used to decipher the exported information. You then can use the NetFlow statistical information to generate traffic-analysis reports and graphs. To enable export of NetFlow information, use the following command:

```
Router(config)# ip flow-export IP_address UDP_port_# [version 5]
```

With the **ip flow-export** command, you must specify the IP address of the NetFlow server, as well as the UDP port number used on the server. There are two versions of NetFlow: version 1 and 5. Version 1 is the default if no version number is specified. To export information to a NetFlow server that supports version 5, you must specify the version number in the command. One advantage that version 5 has over version 1 is that version 5 includes a sequence number in each segment. UDP is unreliable, so there is a chance that the NetFlow server will not receive some exported NetFlow packets. Using the sequence number in version 5 allows the NetFlow server to verify that it received all of the NetFlow segments or determine that some are missing.

Examining and Clearing NetFlow Statistics

After you have set up NetFlow, use the **show ip cache flow** command to examine the flows and their statistics:

```
Router# show ip cache [prefix mask] [type number] [verbose] flow
```

The *prefix mask* option displays entries in the cache that match the prefix and subnet mask, limiting your output. Likewise, you can list the specific interface, displaying only the flows for that interface. Example 17-6 displays the output of this command:

Example 17-6 *Using the* **show ip cache flow** *Command*

```
Router# show ip cache flow
IP packet size distribution (33 total packets):                    (1)
  1-32 64  96  128  160 192 224 256 288 320 352 384 416 448 480
  000  000 000 1.00 000 000 000 000 000 000 000 000 000 000 000

   512 544 576 1024 1536 2048 2560 3072 3584 4096 4608
   000 000 000 000  000  000  000  000  000  000  000

IP Flow Switching Cache, 4358208 bytes                             (2)
  1 active, 65535 inactive, 3 added
  68 ager polls, 0 flow alloc failures
  Active flows timeout in 30 minutes
  Inactive flows timeout in 15 seconds
  last clearing of statistics never

Proto Total  Flows  Packets Bytes Packets Active(Sec) Idle(Sec)   (3)
----- Flows   /Sec    /Flow  /Pkt    /Sec     /Flow     /Flow
ICMP      3    0.0        4   100     0.0       0.0      14.1
Total     3    0.0        4   100     0.0       0.0      14.1

SrcIf  SrcIPaddress  DstIf  DstIPaddress  Pr SrcP DstP Pkts       (4)
Et1    200.1.1.1     Et0    192.1.1.1     01 0000 0800    5
```

Here is an explanation of the information in Example 17-6, with reference to the numbering on the right side. Statement 1 displays the percentage of the distribution of packets based on their size. Notice that 100 percent of the packets are in the 128-byte range.

Statement 2 displays some general statistics about NetFlow switching, such as the total number of bytes that the cache table is using, as well as the number of active flows. Also notice that the NetFlow statistics have not been cleared since NetFlow was configured or the router was booted. To clear the flow statistics, use the **clear ip flow stats** command.

Statement 3 displays a section on protocol statistics. Table 17-1 describes these fields. Statement 4 displays the actual flows. Table 17-2 describes these fields.

Table 17-1 *NetFlow Protocol Fields*

Field	Description
Proto	IP protocol name or number, such as ICMP
Total Flows	Total number of flows for this protocol since the statistics last were cleared
Flows/Sec	Average number of flows for the protocol per second for this summary period
Bytes/Pkt	Average number of bytes for the packets associated with this protocol for this summary period

Table 17-1 *NetFlow Protocol Fields (Continued)*

Field	Description
Packets/Sec	Average number of packets per second for this summary period
Active(Sec)/ Flow	Total of all the seconds from the first to last packets of an expired flow, or total flows for this protocol in the summary period
Idle(Sec)/Flow	Total idle time of all seconds since the last packet for this protocol in the summary period

Table 17-2 *NetFlow Flow Fields*

Field	Description
SrcIf	The interface that the packet was received on
SrcIPaddress	The source IP address in the packet
DstIf	The interface that the packet was sent out of
DstIPAddress	The destination address in the packet
Pr	The IP protocol number (1 is ICMP)
SrcP	In hexadecimal, the source port number
DstP	In hexadecimal, the destination port number
Pkts	The number of packets for this flow

NOTE It is important to point out that the port number information in the output of the **show ip cache flow** command is in hexadecimal. Keep this in mind when looking for particular flows in the cache.

NetFlow and DoS Attacks

Now that you have a basic understanding of how to set up NetFlow and examine its statistics, this section looks at how to use this information to your advantage when trying to determine whether you are experiencing a DoS attack. The next few sections show some examples of different Worm and DoS attacks.

TIP When you are under a heavy DoS attack, I highly recommend that you use NetFlow instead of ACL logging, to help determine the kind of DoS attack as well as information about where the attack is coming from. If you use ACL logging, there is a chance that the overhead that this adds to your router will overwhelm it, possibly causing it to crash.

W32.Blaster Worm

If you are experiencing high CPU utilization on your router, as well as packet drops on your input interfaces, you might have an infestation of the W32.Blaster worm or one of its derivatives. The W32.Blaster worm appears as UDP traffic on port 69 with high volumes of traffic on ports 135 and 4444. The worm propagates using valid file-sharing ports, such as TFTP and TCP port 135, which is the Microsoft RPC protocol, and Kerberos authentication (TCP port 4444). Blocking these ports can prevent its spread.

Example 17-7 shows a simple example of searching for the W32.Blaster Worm infection/attack, based on the network shown previously in Figure 17-1.

Example 17-7 *Using NetFlow to Track Down the W32.Blaster Worms*

```
Router# show ip cache flow | include 0087
SrcIf    SrcIPaddress    DstIf  DstIPaddress   Pr SrcP DstP    Pkts
Et2      192.1.1.1       Et0    192.1.2.7      06 0B88 0087    1
Et2      192.1.1.1       Et0    192.1.2.9      06 0BF8 0087    1
Et2      192.1.1.1       Et0    192.1.2.23     06 0E80 0087    1
Et2      192.1.1.1       Et0    192.1.2.37     06 0CB0 0087    1
Et2      192.1.1.1       Et0    192.1.2.88     06 0C90 0087    1
```

Notice one interesting thing about the **show** command that I executed: You can use the vertical bar (|) and the **include** parameter to display only the results that match the string that you enter after this parameter. Port 0087, in hexadecimal, is 135 in decimal. To look for TFTP, use the hexadecimal number of 0045; for Kerberos authentication, use 115c.

In the previous output, notice that one device, 192.1.1.1, which is the DMZ web server, is scanning other internal devices using protocol 06 (TCP) with a DstP of 0087 (135—Microsoft RPC). With this information and the number of random port 135 connections it is trying to make to internal devices, you definitely should check out your web server.

TIP Before beginning, you first should clear the NetFlow statistics and then continually examine the statistics to look for incrementing patterns. This is true for all DoS attacks.

Code Red Worm

At least three variations of the Code Red worm exist. All versions exploit a security vulnerability in the Microsoft IIS web server product by sending a specially formed URI to port 80 on a vulnerable system. Upon a successful attack, the worm replaces the web page of the IIS server with graffiti. In version 1 of the worm, an infected device did one of the following:

- Scanned random IP addresses and passed the infection to other devices
- Launched a DoS attack against 198.137.240.91, which used to be www.whitehouse.gov

One problem with the version 1 worm was that it did not use truly random addresses in its scan process; version 2 does. Version 2 also does not use the hard-coded IP address of the White House's web server; it uses a DNS name and does a DNS lookup to find the address. Both worms suffer the same deficiency: They do not check to see if the devices that they are scanning and trying to infect already are subverted.

Many Cisco products were affected by the amount of traffic that this worm generated. For example, the CSS 11000 content switches, under a heavy worm infestation, experienced memory-allocation errors that required them to be rebooted. Likewise, the 600 DSL routers stopped forwarding traffic until they also were rebooted. Cisco has introduced code fixes for all of its products that experienced problems dealing with this worm.

Because the worm uses a common port, detecting it is slightly more difficult. However, using NetFlow makes your task much easier. Because Code Red spreads itself by looking for vulnerabilities in IIS, you should examine NetFlow information and look for a TCP destination port of 80 (which is 0050 in hexadecimal). The network shown previously in Figure 17-1 illustrates this in Example 17-8.

Example 17-8 *Using NetFlow to Track Down the Code Red Worms*

```
Router# show ip cache flow | include 0050
SrcIf  SrcIPaddress   DstIf  DstIPaddress   Pr SrcP  DstP  Pkts
Et2    192.1.1.1      Et0    192.1.2.3      06 0F9E  0050     1
Et2    192.1.1.1      Et0    192.1.2.9      06 0456  0050     2
Et2    192.1.1.1      Et0    192.1.2.21     06 3001  0050     2
Et2    192.1.1.1      Et0    192.1.2.37     06 B305  0050     1
Et2    192.1.1.1      Et0    192.1.2.38     06 0EEF  0050     1
Et2    192.1.1.1      Et0    192.1.2.88     06 0E75  0050     1
Et2    192.1.1.1      Et0    192.1.2.104    06 1134  0050     2
```

As you can see in this example, there are a few HTTP requests from the web server in the DMZ to internal addresses. If you see an abnormally high number of HTTP requests with the same source address, or if the source address is a web server itself, as is the case here, you have a problem. In this example, the DMZ web server has been attacked successfully and is attempting to spread itself to internal devices.

Smurf Attack

As mentioned throughout this book, the Smurf attack is one of the most common types of DoS attacks because of its simplicity in implementing it on a hacker's behalf. As with worms, you can use NetFlow to track the reflectors and possibly the source(s) of the attack. Example 17-9 shows an example, based on the network shown in Figure 17-1.

Example 17-9 *Using NetFlow to Track Down Smurf Attacks*

```
Router# show ip cache flow | include 0000
SrcIf  SrcIPaddress  DstIf  DstIPaddress   Pr SrcP DstP  Pkts
Et1    201.1.1.1     Et2    192.1.1.1      01 0000 0000    49K
Et1    201.1.1.17    Et2    192.1.1.1      01 0000 0000    50K
Et1    201.1.1.39    Et2    192.1.1.1      01 0000 0000    51K
Et1    201.1.1.104   Et2    192.1.1.1      01 0000 0000    48K
Et1    205.8.8.8     Et2    192.1.1.1      01 0000 0000    31K
Et1    205.8.8.39    Et2    192.1.1.1      01 0000 0000    30K
```

As you can see in this example, two networks seem to be set up as reflectors, attacking the web server on the DMZ (192.1.1.1). Based on this information, you can contact your ISP and the administrators of these networks to take corrective action.

TIP Besides using NetFlow for tracking down specific attacks, some people at Ohio State University have developed a quick-and-dirty set of tools that uses NetFlow records to detect network and host intrusions. More information about this nifty project can be found at http://www.usenix.org/publications/login/1999-9/osu.html.

CEF Switching

Cisco Express Forwarding (CEF) is a switching mode that was introduced in the Cisco IOS 11.1 and 11.2 software trains and that is available across all 12.0 versions. In traditional Cisco caching solutions, cache entries are built as traffic flows through the router. CEF, on the other hand, mirrors the entire system routing table, alleviating the building on any initial cache. Therefore, CEF handles large amounts of data better than traditional switching and caching methods.

With most DoS attacks, the traffic is sent to one or a handful of victim devices, which does not have any impact on either traditional or CEF switching processes. However, many kinds of SYN flood attacks use random source addresses. The victim or victims of this attack attempt to respond back to the connection attempts, creating a large number of destinations for switching paths. With traditional switching, this would create a performance problem; with CEF, the destinations already have been cached, so there is not as much of an issue dealing with this increased amount of traffic.

When a router is using fast switching on an interface, the CPU must be involved to handle the interrupt requests from the fast-switching interfaces to move traffic from one interface to another. During periods of flooding traffic, this can create a DoS condition on the router itself in which most of its CPU cycles are handling the interface interrupts. You can limit this process by using the **scheduler interval** command:

```
Router(config)# scheduler interval #_of_milliseconds
```

This command causes the router to stop handling interrupt requests at the configured interval and handle other tasks. For example, you might set the value to 250 milliseconds, which tells the Cisco IOS to handle process-level tasks for no more than 250 milliseconds at a time.

Newer Cisco router platforms use the **scheduler allocate** command instead of the **scheduler interval** command:

```
Router(config)# scheduler allocate #_of_milliseconds_of_interrupts
  #_of_milliseconds_of_no_interrupts
```

This command has two parameters: The first parameter specifies the number of milliseconds in which interrupts are handled; the second parameter specifies the number of milliseconds for which interrupts are placed on hold. A common allocation configuration is **scheduler allocate 3000 1000**. With this configuration, interrupts are handled for 3 seconds, but for the next second, the router performs other tasks.

TIP Cisco has stated that the **scheduler interval** and **scheduler allocate** commands have no negative side effects on the router and should be part of your router's standard configuration.

TCP Intercept

One option for dealing with TCP SYN flood attacks is to implement the Cisco IOS TCP Intercept feature. TCP Intercept enables you to deal with DoS attacks that attempt to take advantage of the weakness in the way that TCP connections establish a session with the three-way handshake. This was discussed in Chapter 1, "Security Threats." This section focuses on how to use TCP Intercept to limit the effectiveness of a TCP SYN food attack that is perpetrated by an attacker.

TCP SYN Flood Attacks

As you recall from Chapter 1, a TCP SYN flood attack is an easy attack to initiate. The attacker sends a flood of TCP SYN segments with no intention of completing the three-way handshake for each of these connections. Typically, the hacker combines this with an IP spoofing attack in which the source addresses in the packet are either invalid or someone else's address. Because these addresses cannot be reached (or, if they are someone else's address, are not responded to), the TCP server being attacked hangs in limbo with these half-open (commonly called *embryonic*) connections. In this situation, the server must wait until the TCP timeout expires for the connection before removing the connection from its local connection table. This creates a problem because it uses up resources on the TCP server, which might deny legitimate TCP connections.

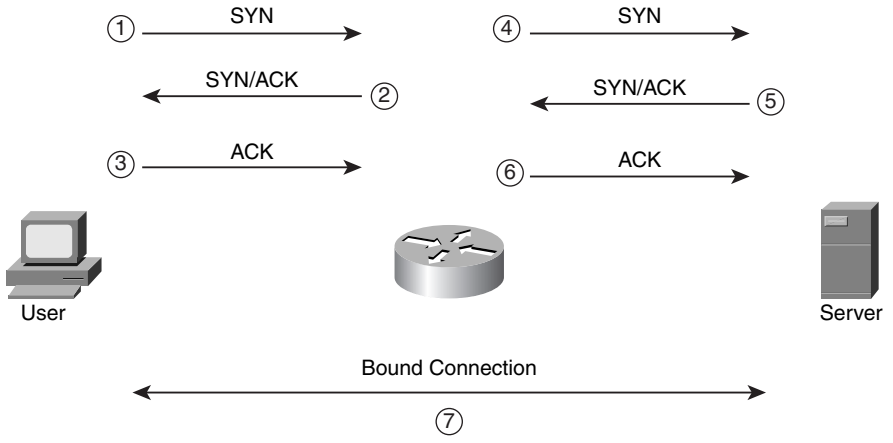
TCP Intercept Modes

TCP Intercept is a Cisco IOS feature that is used to protect TCP services from TCP SYN flood attacks. TCP supports two modes of protection: intercept and watch. The following two sections discuss how these modes operate when dealing with TCP SYN attacks.

Intercept Mode

Intercept mode takes a proactive approach to TCP SYN flood attacks. Figure 17-3 shows an example of a router using intercept mode with TCP Intercept. In intercept mode, the router intercepts all TCP connection requests, as shown in the figure. In this example, an external user is trying to access an internal server using a TCP connection. The router intercepts this request and pretends to be the internal server, completing the connection to the external user. Only upon a successful three-way handshake with the external user (Steps 1 through 3 in Figure 17-3) does the router set up a second TCP connection to the server (Steps 4 through 6). The router then binds the two connections, creating a single connection (Step 7).

Figure 17-3 *TCP Intercept Using Intercept Mode*



For the most part, this process is transparent to the two devices that make up the final TCP connection. With this approach, if a TCP SYN flood attack is occurring, the router provides a buffer to the internal servers using TCP because they are not affected by the flood: The router deals with the half-open connections, which eventually time out and are removed from the router's TCP connection table. Basically, the router sends an RST to the requesting source device. In the mean time, valid requests are permitted as long as the router successfully can complete the three-way handshake with the external user.

NOTE One limitation of intercept mode is that any TCP options that are negotiated between the external user and the router (which normally are to the end-server device), such as RFC 1323 window scaling, are not passed from the router to the internal TCP server. This is because the router does not know these options until the first three-way handshake completes with the external user and the router begins the second three-way handshake with the internal TCP server. Typically, this is not a problem because TCP allows for the dynamic negotiation of these parameters during the normal operation of the session.

Watch Mode

Whereas intercept mode takes a proactive approach to dealing with TCP SYN flood attacks, *watch* mode takes a reactive approach. One of the main advantages of intercept mode is that it removes the processing load of TCP SYN floods from the internal server. However, this a double-edged sword because, in most time periods, a TCP SYN flood is not occurring, but the router still is performing the intercept process, placing a heavy burden on it.

To alleviate this problem, you can use TCP Intercept in watch mode. Watch mode takes a passive, or reactive, approach to TCP SYN flood attacks. In watch mode, the router passively watches the setup of TCP connections from users to servers. It monitors these connections, keeping track of embryonic connections that remain in this limbo state. Then it compares this value to a preconfigured timeout value (which defaults to 30 seconds). If a TCP connection does not complete the three-way handshake in this period, the Cisco IOS sends a TCP reset to the server to remove the connection. For an attack that is directed at an internal server using TCP, this removes the half-open connections, thus reducing the load on the server and allowing legitimate connection attempts to be processed.

TCP Intercept Configuration and Verification

Enabling TCP Intercept on your router is a simple process. Only one command is required, but typically you will tune some of the parameters to ensure that TCP Intercept functions smoothly in your environment. The following sections discuss the configuration and verification of TCP Intercept.

Enabling TCP Intercept

The first step in setting up TCP Intercept is to specify which TCP sessions should be intercepted in intercept mode or monitored in watch mode. This is done by using the **ip tcp intercept list** command:

```
Router(config)# ip tcp intercept list extended_ACL_#
```

A couple of important things should be pointed out about this required command. First, this command enables TCP Intercept. Notice that the command is executed at global configuration mode, not on an interface.

Second, you must use an extended ACL to specify which traffic is to be examined by TCP Intercept. As an example, if you want to monitor all TCP setup connections, you would configure something like this:

```
Router(config)# access-list 100 tcp permit tcp any any
Router(config)# ip tcp intercept list 100
```

Typically, this configuration is a bad idea because the router basically would act like a proxy for all inbound and outbound connections.

TIP When you are setting up TCP Intercept, use it to deal with TCP SYN flood attacks from external users. This is accomplished by configuring an extended ACL for TCP Intercept that refines the traffic that should be monitored. For example, if you have only two servers in your DMZ that use TCP, such as an e-mail server and a web server, set up TCP Intercept to monitor only port 25 traffic to the e-mail server and port 80 traffic to the web server. This greatly reduces the number of TCP connections that the router has to monitor. Because you typically do not know the IP address of the source device, leave it as any.

Defining the Mode

The rest of the TCP Intercept configuration commands are optional. For example, the default mode for TCP Intercept is intercept. If this creates too much of a burden on your router's CPU, you can switch the mode to watch with the following command:

```
Router(config)# ip tcp intercept mode {intercept | watch}
```

Note that you cannot use a mixture of both intercept and watch modes for TCP Intercept, for example, based on the destination address. With TCP Intercept, the mode that it operates in is a global value.

Changing the Timers

You can tune three timers with TCP Intercept: an embryonic timer, a connection reset timer, and a connection idle timer. These can be configured with the three following commands:

```
Router(config)# ip tcp intercept watch-timeout seconds
Router(config)# ip tcp intercept finrst-timeout seconds
Router(config)# ip tcp intercept connection-timeout seconds
```

The **ip tcp intercept watch-timeout** command specifies the maximum length of time that the router will wait, in watch mode, for a TCP connection to complete the three-way handshake. This value defaults to 30 seconds. If the connection is not reached in this time period, the router sends a reset to the server (destination).

When a router with TCP Intercept enabled monitors a connection that is in the process of being torn down, it expects the connection to be torn down within 5 seconds, by default, from the receipt of a reset or FIN exchange. When this time period is reached, the router ceases to manage the connection. You can change this value with the **ip tcp intercept finrst-timeout** command.

When the TCP Intercept is managing TCP connections, it manages a connection only up to 24 hours, by default, for an idle connection. After this, the router drops the connection. You can modify this with the **ip tcp intercept connection-timeout** command.

Changing the Thresholds

In addition to the intercept and watch modes, TCP Intercept uses threshold values to deal with an excessive number of TCP connection attempts during a time of a TCP SYN flood attack. TCP Intercept supports two thresholds: one based on the total number of embryonic connections, and one based on the number of connection requests during the last 1-minute period. You can tune both of these threshold values.

During a time of attack in which your router is seeing an excessive number of connections and one of the two threshold values is reached, the router begins dropping the oldest embryonic connections first (this can be changed with the **drop-mode** parameter, discussed in the next section). For intercept mode, the router reduces the initial retransmission timeout for SYN segments by half. For watch mode, the router reduces the watch time automatically to half of the configured value (if you were using the default of 30 seconds, it becomes 15 seconds).

Aggressive operation of TCP Intercept begins when either of the two thresholds is reached. To specify the values for the maximum embryonic connection threshold, use the following two commands:

```
Router(config)# ip tcp intercept max-incomplete high number
Router(config)# ip tcp intercept max-incomplete low number
```

The **high** parameter specifies when aggressive mode should begin. When this threshold is reached, the router, by default, begins deleting the oldest half-open connections first until it drops enough of these connections to fall below the threshold value specified in the **low** parameter. The default threshold values are 1100 for the **high** parameter and 900 for the **low** parameter.

To specify the values for the expected total number of half-open connection requests in a 1-minute period, use the following two commands:

```
Router(config)# ip tcp intercept one-minute high number
Router(config)# ip tcp intercept one-minute low number
```

The **high** parameter specifies when aggressive mode should begin. When this threshold is reached in a 1-minute measurement period, the router, by default, begins deleting the oldest half-open connections first until it drops enough of these connections to fall below the threshold value specified in the **low** parameter. The default threshold values are 1100 for the **high** parameter and 900 for the **low** parameter.

CAUTION Be careful about adjusting the threshold values with the **ip tcp intercept max-incomplete** and **one-minute** commands. If you set these values too low, the router might take aggressive action to prevent legitimate connections. This is especially true of the web servers, in which lots of connections are expected when downloading the contents of a web page. Each situation and network is different; you need to establish a baseline before determining what values you should use for the high and low threshold numbers.

Changing the Drop Method

In aggressive mode, the default drop method is to drop the oldest half-open connections first until the number of remaining half-open connections falls below the configured minimum threshold (either total half-open connections or half-open connections in a 1-minute interval). You can change the drop method with the following command:

```
Router(config)# ip tcp intercept drop-mode {oldest | random}
```

The default is **oldest**; by changing it to **random**, TCP Intercept causes the router to randomly drop half-open connections until the number of remaining half-open connections falls below the configured minimum threshold.

Verifying Your Configuration

After you have configured TCP Intercept, you can use two **show** commands and one **debug** command to verify and troubleshoot your configuration. If you want to see general statistics about the operation of TCP Intercept, use the **show tcp intercept statistics** command. Example 17-10 displays sample output of the **show tcp intercept statistics** command.

Example 17-10 *Using the* **show tcp intercept statistics** *Command*

```
Router# show tcp intercept statistics
intercepting new connections using access-list 100
12 incomplete, 5 established connections (total 17)
1 minute connection request rate 2 requests/sec
```

In this example, there are currently 12 half-open connections and 5 established ones. Notice that ACL 100 is used to restrict the traffic that TCP Intercept monitors.

To see the connections that TCP Intercept is monitoring, use the **show tcp intercept connections** command. Example 17-11 displays sample output of the **show tcp intercept connections** command.

Example 17-11 *Using the* **show tcp intercept connections** *Command*

```
Router# show tcp intercept connections
Incomplete:
Client            Server           State    Create   Timeout  Mode
201.1.1.1:33772   200.1.1.30:80    SYNRCVD  00:00:09 00:00:05 I
201.1.1.1:33773   200.1.1.30:80    SYNRCVD  00:00:09 00:00:05 I

Established:
Client             Server          State    Create   Timeout  Mode
201.1.1.2:33771    200.1.1.2:23    ESTAB    00:00:08 23:59:54 I
```

The output of this command is broken into two sections: Incomplete refers to connections that are in an embryonic state; established refers to connections that have completed the three-way handshake. The Client column indicates the source of the

connection and the Server column indicates the destination. The connection can be in one of three states:

- **SYNSENT**—The client is attempting to establish a TCP session.
- **SYNRCVD**—The server is responding with a SYN/ACK to the client.
- **ESTAB**—The TCP session has been established, and both devices now can pass data.

The Create column indicates how long it has been since the connection was created. The Timeout column indicates how long the connection has before it times out. Note that the idle times for the two sets of information in this output will be different: The top half is for embryonic connections, and the bottom half is for established idle TCP sessions. The exception to the timeout for established sessions is that, when the session is in the process of being torn down, the idle timer defaults to 5 seconds. The Mode column indicates the mode for TCP Intercept: I is for intercept mode, and W is for watch mode.

For more detailed troubleshooting, you can use the **debug ip tcp intercept** command, which displays the TCP Intercept events occurring on the router. Example 17-12 shows a simple example of the output of this command, with the router in intercept mode.

Example 17-12 *Using the* **debug ip tcp intercept** *Command*

```
Router# debug ip tcp intercept
INTERCEPT: new connection (192.1.1.1:33884) => (200.1.1.2:23)     (1)
INTERCEPT: 192.1.1.1:33884 <- ACK+SYN (200.1.1.2:33884)
INTERCEPT: new connection (192.1.1.1:33885) => (200.1.1.2:23)     (2)
INTERCEPT: 192.1.1.1:33885 <- ACK+SYN (200.1.1.2:33885)
INTERCEPT: retransmit 2 (192.1.1.1:33884) <-                      (3)
          (200.1.1.2:23) SYNRCVD
INTERCEPT: retransmit 2 (192.1.1.1:33885) <-
          (200.1.1.2:23) SYNRCVD
INTERCEPT: new connection (192.1.1.2:32790) => (200.1.1.2:23)     (4)
INTERCEPT: 192.1.1.2:32790 <- ACK+SYN (200.1.1.2:32790)
INTERCEPT: retransmit 4 (192.1.1.1:33884) <-                      (5)
          (200.1.1.2:23) SYNRCVD
INTERCEPT: retransmit 4 (192.1.1.1:33885) <-
          (200.1.1.2:23) SYNRCVD
INTERCEPT: retransmit 2 (192.1.1.2:32790) <-
          (200.1.1.2:23) SYNRCVD
INTERCEPT: 1st half of connection is established                  (6)
          (192.1.1.2:32790) => (200.1.1.2:23)
INTERCEPT: (192.1.1.2:32790) SYN -> 200.1.1.2:23
INTERCEPT: retransmit 2 (192.1.1.2:32790) ->
          (200.1.1.2:23) SYNSENT
INTERCEPT: 2nd half of connection established                     (7)
          (192.1.1.2:32790) => (200.1.1.2:23)
INTERCEPT: (192.1.1.2:32790) ACK -> 200.1.1.2:23
INTERCEPT: retransmit 8 (192.1.1.1:33884) <-                      (8)
          (200.1.1.2:23) SYNRCVD
INTERCEPT: retransmit 8 (192.1.1.1:33885) <-
          (200.1.1.2:23) SYNRCVD
```

continues

Example 17-12 *Using the* **debug ip tcp intercept** *Command (Continued)*

```
INTERCEPT: retransmit 16 (192.1.1.1:33884) <-
          (200.1.1.2:23) SYNRCVD
INTERCEPT: retransmit 16 (192.1.1.1:33885) <-
          (200.1.1.2:23) SYNRCVD
INTERCEPT: retransmitting too long (192.1.1.1:33884) =>           (9)
          (200.1.1.2:23) SYNRCVD
INTERCEPT: 192.1.1.1:33884 <- RST (200.1.1.2:23)
INTERCEPT: retransmitting too long (192.1.1.1:33885) =>
          (200.1.1.2:23) SYNRCVD
INTERCEPT: 192.1.1.1:33885 <- RST (200.1.1.2:23)
```

The following is an explanation of this output, with reference to the numbering on the right side:

1. A new connection arrives from 192.1.1.1 to 200.1.1.1. The router intercepts the connection and responds with a SYN/ACK to 192.1.1.1.
2. A second connection arrives from 192.1.1.1 to the same destination. The router again intercepts this connection and responds with a SYN/ACK to the source (192.1.1.1).
3. The router did not receive an ACK from either request, so the router retransmits the SYN/ACK for the connections.
4. 192.1.1.2, a different device, requests a connection to the 200.1.1.2 server. The router intercepts the connection request and responds with a SYN/ACK.
5. The router has not received an ACK from any of the TCP connections, so it retransmits the SYN/ACK for each session request to each source device.
6. The router has received the ACK from the second device, completing the three-way handshake, so the router begins to build a separate connection to the server (200.1.1.1) on behalf of this client.
7. The server responds to the router's connection request with a SYN/ACK, and the router responds with an ACK, completing the three-way handshake.
8. The router still has not received an ACK response for the first two TCP connection requests from 192.1.1.1, so it retransmits the SYN/ACK for these connections.
9. For the two connections request from 192.1.1.1, the connection attempt has exceeded the configured timeout, so the router sends an RST to the source for both of these connection attempts.

TCP Intercept Example

To help you understand TCP Intercept's configuration, reexamine Figure 17-1. In this network, the administrator is concerned about TCP SYN flood attacks against the web server and the e-mail server located in the DMZ. To deal with TCP SYN flood attacks,

the administrator has decided to use TCP Intercept in watch mode to monitor connections. Example 17-13 shows the router's configuration for TCP Intercept.

Example 17-13 *Using TCP Intercept to Protect Internal Servers*

```
Router(config)# access-list 100 tcp permit tcp any host 192.1.1.1 eq 80
Router(config)# access-list 100 tcp permit tcp any host 192.1.1.2 eq 25
Router(config)# ip tcp intercept list 100
Router(config)# ip tcp intercept mode watch
Router(config)# ip tcp intercept watch-timeout 20
Router(config)# ip tcp intercept connection-timeout 120
Router(config)# ip tcp intercept max-incomplete high 600
Router(config)# ip tcp intercept min-incomplete low 500
Router(config)# ip tcp intercept one-minute high 800
Router(config)# ip tcp intercept one-minute low 600
```

In this example, TCP Intercept is set up for only the DMZ web and e-mail servers. The mode was changed from intercept to watch, and the watch timeout for connections was changed from 30 to 20 seconds. If a connection is not set up within the 20 seconds, the router sends a reset (RST) to the DMZ server. The idle timeout for TCP connections was changed from 1 day (86,400 seconds) to 2 minutes (120 seconds). This is a more appropriate value, based on the fact that e-mail and especially web server connections are brief and are typically not idle unless there is a connection problem.

Based on the monitoring performed on this network, the administrator chose more appropriate threshold values for the aggressive mode process. Remember, though, that setting these too low can cause half-open legitimate connections to be dropped; setting them too high can allow invalid half-open connections to use up resources on the two DMZ servers. As you can see from this example, setting up TCP Intercept is a simple process.

CBAC and DoS Attacks

Chapter 9, "Context-Based Access Control," discussed how you can use this feature from the Cisco IOS Firewall feature set to implement a stateful firewall function. One thing that I did not cover in Chapter 9 was the capability of CBAC to restrict the number of half-open sessions, which typically is used to prevent TCP SYN flood attacks. This feature is similar to TCP Intercept, but it can examine TCP as well as UDP and ICMP sessions. Of course, with UDP and ICMP, because there is no state machine that defines the setup, maintenance, and removal of a connection, CBAC uses timers instead of connection threshold values. The following sections discuss how you can use CBAC to prevent certain kinds of DoS attacks.

Timeouts and Thresholds

CBAC uses timeouts and thresholds to determine how long state information should be kept for a session and when to drop those sessions, based on them not becoming fully

established. Of course, "established" is a defined process when it comes to TCP, in which CBAC can examine the TCP flags to determine the state of the connection. With UDP and ICMP, CBAC must use idle timer limits to approximate when a connection ends. The timeouts and thresholds can be defined globally, as I will show you in this section, or can be done within the actual **ip inspect** command for the traffic that you are inspecting.

TIP Cisco recommends that you first make changes to the global timeout and threshold values before configuring your inspection rules.

Setting Connection Timeouts

To set the global CBAC timeouts for inspected connections, use the following commands:

```
Router(config)# ip inspect tcp synwait-time seconds
Router(config)# ip inspect tcp finwait-time seconds
Router(config)# ip inspect tcp idle-time seconds
Router(config)# ip inspect udp idle-time seconds
Router(config)# ip inspect dns-timeout seconds
```

These commands were discussed thoroughly in Chapter 9, so I do not go into any more depth about them here.

NOTE You can override the idle timeouts in the actual inspection commands. When a timeout is configured in an inspection command for an application or protocol, it takes precedence over the global timeout value. Also, there is no global timeout parameter for ICMP traffic: this is configured with the inspection command for ICMP traffic.

Setting Connection Thresholds

As with TCP Intercept, you can set connection thresholds for CBAC inspection. However, unlike TCP Intercept, connection thresholds for CBAC include TCP and UDP traffic. The next two sections discuss the configuration of the threshold values and how CBAC uses them to prevent DoS attacks when an attacker attempts to open an abnormally high number of connections to tie up resources on critical services.

Configuration

Setting up connection thresholds is similar to setting thresholds for TCP Intercept. For CBAC, here are the commands you use:

```
Router(config)# ip inspect max-incomplete high number
Router(config)# ip inspect max-incomplete low number
Router(config)# ip inspect one-minute high number
```

```
Router(config)# ip inspect one-minute low number
Router(config)# ip inspect tcp max-incomplete host number
  block-time minutes
```

You can specify three connection parameter threshold values: maximum half-open sessions, maximum half-open sessions in a 1-minute period, and maximum TCP half-open sessions per host.

The first two commands specify that when the maximum number of half-open sessions reaches the **high** threshold value, CBAC starts dropping the half-open sessions until it reaches the **low** threshold value. The defaults for these thresholds are 500 and 400 half-open sessions, respectively.

The third and fourth commands measure the number of new connection attempts in a 1-minute interval. If the maximum number of new connection attempts reaches the **high** threshold during the time interval, CBAC begins dropping the half-open sessions until it reaches the **low** threshold. The defaults for these thresholds are 500 and 400 half-open sessions, respectively.

Both sets of these parameters are similar to the thresholds defined with TCP Intercept. The one main difference between CBAC and TCP Intercept is the last CBAC command in the previous code listing: **ip inspect tcp max-incomplete host**. This command is a more aggressive command that is used to prevent DoS attacks against a specific host. With this command, CBAC uses two different methods, depending on the value of the **block-time** variable. If the **block-time** variable is set to 0 minutes, when the **host** half-open TCP session limit value is exceeded, CBAC deletes the oldest half-open connection request for every newly requested connection request. However, if the **block-time** value is set to a minute value greater than 0, when the number of half-open sessions exceeds the **host** parameter, CBAC blocks all new connection requests to the host (destination) until the **block-time** period expires.

CAUTION Be careful about assigning a minute value greater than 0 to the **block-time** value because this also could block legitimate traffic during a DoS attack. My recommendation is to configure a **block-time** value of 0. Also, you need to be very careful about the threshold that you configure for the **host** parameter. Assigning too low of a value for a very busy server could cause CBAC to drop legitimate connection requests.

Half-Open Sessions

One main difference between TCP Intercept and CBAC inspection is that TCP Intercept can be used only to prevent TCP SYN flood attacks. CBAC, however, can prevent DoS attacks for TCP and UDP DoS attacks. The thresholds that CBAC uses are based on the number of half-open sessions. For TCP, a half-open session is one that has not reached an established state; this includes both SYN and SYN/ACK messages (CBAC can detect both kinds of floods). For UDP, a half-open session is one in which no returning traffic is detected.

CBAC DoS Prevention Verification

After you have assigned your timing and threshold parameters, you can use various **show** and **debug** commands (discussed in Chapter 9) to verify your configuration. One option that I mentioned in this chapter is to use audit trails and alerts. Assuming that you have enabled alerts, and you are experiencing a DoS attack, you will see log output similar to the following:

```
%FW-4-ALERT_ON: getting aggressive, count (83/500) current 1-min rate: 501
%FW-4-ALERT_OFF: calming down, count (0/400) current 1-min rate: 271
```

In this example, the first alert message displays the number of half-open sessions (501) and the current limit (500). In the last minute, 271 connection attempts were made. From CBAC's perspective, this is the beginning of a DoS attack. The second message, ALERT OFF, indicates that the number of connections, through both dropping and normal setup completion, has fallen below the minimum threshold (400). In this case, the value is 0, indicating that CBAC dropped some and that the rest were initiated normally.

The combination of both an ON and a OFF message indicates a separate attack. These messages are used for both the maximum number of half-open sessions and the maximum number of new connection attempts in a 1-minute interval.

If a DoS attack is geared at a specific host, you would see the following alert messages:

```
000022: Jan 02 15:42:11.048: %FW-4-HOST_TCP_ALERT_ON:
                         Max tcp half open connections (50)
                         exceeded for host 192.1.1.2
000023: Jan 02 15:42:11.361: %FW-4-BLOCK_HOST:
                         Blocking new TCP connections to host
                         192.1.1.2 for 2 minutes
                         (half-open count 50 exceeded)
000024: Jan 02 15:44:11.372:%FW-4-UNBLOCK_HOST:
                         New TCP connections to host
                         192.1.1.2 no longer blocked
```

In this example, the first message indicates that the maximum number of half-open TCP connections destined to 192.1.1.2 was exceeded (the limit is 50). The second message indicates that the blocking interval was defined at 2 minutes, so subsequent TCP connection requests are denied. The third message indicates that the 2-minute blocking interval has expired and that new connection requests are allowed to 192.1.1.2.

CBAC Example Configuration

You must be careful about changing the half-open session thresholds and timeout values for CBAC. Setting these values too low might cause CBAC to trigger dropping connections during a period of high activity with legitimate traffic. Likewise, setting them too high might put more of a burden on a device during an attack because more half-open sessions are allowed. Example 17-14 shows the configuration for the network in Figure 17-1 to illustrate CBAC's configuration to prevent DoS attacks.

Example 17-14 *Using CBAC to Prevent DoS Attacks*

```
Router(config)# ip inspect tcp synwait-time 20
Router(config)# ip inspect tcp idle-time 60
Router(config)# ip inspect udp idle-time 20
Router(config)# ip inspect max-incomplete high 400
Router(config)# ip inspect max-incomplete low 300
Router(config)# ip inspect one-minute high 600
Router(config)# ip inspect one-minute low 500
Router(config)# ip inspect tcp max-incomplete host 300
  block-time 0
```

In this example, I have modified the timeout for three parameters. The TCP half-open session timeout was modified from 30 to 20 seconds; this is a small network, so connections should complete fairly quickly. I also have modified the idle timeout for TCP connections, changing it from 3600 seconds (1 hour) to 60 seconds; again, most of the connections are either HTTP or SMTP, so they should not be idle for a long period of time. Because UDP is not used very often, I have modified the UDP idle timer from 30 to 20 seconds.

I also have modified the maximum number of half-open connection threshold values to 400 and 300, as well as the 1-minute interval threshold values to 600 and 500. These were configured after monitoring normal activity in this network.

One of the most difficult parameters to figure out is the maximum number of half-open sessions to a host (server). Normally, determining what this parameter should be is very difficult because on several servers, such as e-mail and www, the connection requests can vary wildly from one server to the next; coming up with a valid threshold value that will work with all devices is very difficult. In this instance, there are only two servers—the e-mail server and web server. I set the maximum number of half-open sessions to 300 per destination. This is kind of high for the small network e-mail server, but it is a more appropriate number for the web server (remember that downloading a web page from a web server might involve a dozen or more connection requests).

NOTE When you are modifying the timeout and threshold values, carefully monitor CBAC to ensure that you are not making the problem worse instead of better. Too often I have seen this situation in real networks.

Also, remember my warning from Chapter 16, "Intrusion Detection System": when you enable IDS on your router, some components of CBAC are enabled to examine application layer information to detect certain kinds of attacks. One of the enabled components include the timeout and thresholds for CBAC; therefore, you will definitely need to tune these to ensure that your router doesn't use the default values, which might be artificially low for your environment, causing legitimate connections to be dropped.

Rate Limiting

With some types of DoS attacks, there's not much you can do to stop the flow of the attack, especially in a distributed DoS (DDos) attack in which the hacker is spoofing the source addresses and using an unsuspecting company or ISP as the reflector in the attack. Tracing this kind of attack to the hacker can be difficult.

In this situation, your first concern should be limiting the impact of the attack on your network, which can be done with rate limiting. Rate limiting enables you to assign a bandwidth restriction to a category of traffic, such as ICMP, UDP, or specific connection types. This section focuses on three solutions for rate limiting: ICMP rate limiting, committed access rate (CAR), and class-based policing (CBP) using Network-Based Application Recognition (NBAR).

TIP Rate limiting is best used on the ISP's router that connects to your network. In other words, if you are experiencing a flood attack that is saturating your Internet link, implementing rate limiting on your perimeter router will not do much good. Instead, you work with your ISP to put this in place on the ISP's router. Rate limiting is something you configure to restrict the amounts of various outbound traffic. As an example, if you were a reflector in a Smurf attack, you could use rate limiting as a temporary solution to limit the flood of traffic that you are sending to a victim's network. Of course, you will want to try to track down the culprit and stop the attack.

ICMP Rate Limiting

Many attacks create a DoS attack by sending a flood of traffic to a device or devices that do not exist, causing an intervening router to reply back with an ICMP unreachable message for each unknown destination. A good example of this is a worm attack, such as an attack by the SQL Slammer worm. Through this process, the worm, purposefully or inadvertently, tries to find other machines with the same weakness. It typically does this through a ping sweep. Most worms are not intelligent about how they do this: They continually scan the same networks, trying to find the same security hole or holes that the worm used initially to subvert one of your devices.

Using Other Solutions

To prevent the overwhelming number of ICMP unreachables that your router would generate in a worm attack, you can filter the specific kind of traffic, with an ACL that looks for ICMP unreachable messages, like this:

```
Router(config)# <--permit and deny other types of traffic-->
Router(config)# access-list 100 deny icmp any any unreachable
Router(config)# <--permit and deny other types of traffic-->
```

With the SQL Slammer worm attack, you can set up an ACL to block UDP traffic destined to port 1434, like this:

```
Router(config)# <--permit and deny other types of traffic-->
Router(config)# access-list 100 deny udp any any eq 1434
Router(config)# <--permit and deny other types of traffic-->
```

This example blocks all Microsoft SQL traffic and is configured on the external interface of your perimeter router.

NOTE Of course, if the worm is inside your network, and if either of these two ACLs is applied on your perimeter router inbound on its external interface, these ACLs will not help you much. For these to be effective, you must set up similar ACLs on all or most of your routers throughout your network. Plus, with the first ACL, you might need to allow certain ICMP unreachable messages to specific devices, making this configuration more difficult to implement.

Another possible solution was discussed in Chapter 4, which covered many services that you should disable. One of these is the generation of ICMP unreachable messages by the router. For example, on a perimeter router, you could configure the following on its interfaces (at least, the external interface):

```
Router(config)# interface type [slot_#/]port_#
Router(if-config)# no ip unreachables
```

Again, the problem with this approach is that you need to enable this on all of the routers (and their interfaces) in your network to prevent the generation of ICMP unreachable messages. In many cases, this presents additional problems: For example, it becomes much more difficult to use tools such as ping and traceroute to track down problems because your routers are not generating ICMP unreachable messages.

Using the ICMP Rate-Limiting Feature

Starting in 12.0 of the Cisco IOS, Cisco implemented a default rate limit of one ICMP unreachable packet that a router would generate in a 500-millisecond (ms) interval. This prevents a router from responding to thousands of packets with unreachable destinations with a separate ICMP message for each of these access requests.

In Cisco IOS 12.1, you can tune this operation manually with the following command:

```
Router(config)# ip icmp rate-limit unreachable [df] milliseconds
```

First, notice that this is a global configuration mode command: it applies to any ICMP unreachable message responses on any interface. Second, the **df** parameter is used to restrict the number of ICMP unreachable messages generated by the router when the fragmentation of the packet is needed and the DF bit in the IP packet header is set. Third, you can specify only the time interval for ICMP unreachable messages (in milliseconds).

This can range from 1 to 4,294,967,295. During the specified interval, the Cisco IOS generates only one ICMP unreachable message for the first packet that requires one; for other unreachable events, the router ignores them until the configured time period expires.

Here is a simple example:

```
Router(config)# ip icmp rate-limit unreachable 1000
```

In this example, only one ICMP unreachable message is generated each second.

CAR

The main problem with the **ip icmp rate-limit** command is that it works only for ICMP unreachable messages. To deal with other kinds of traffic, including ICMP unreachables, you can use committed access rate (CAR). CAR enables you to limit traffic entering or leaving an interface, and it can match on any of the following criteria: all IP traffic, IP precedence value, MAC address, or information that matches a **permit** statement in either a standard or an extended ACL.

CAR typically is implemented at the perimeter of your network for egress traffic, and it enables you to have different rate-limiting policies for different types of traffic. For example, you might have different rate limits for ICMP traffic compared to HTTP traffic to a web server.

CAUTION Using an ACL to specify matching traffic can be process intensive. In addition, if you specify more than one policy, your router will have an additional CPU utilization hit; try to limit the number of policies that you define. Because of these issues, you should use CAR with caution. Typically, I use CAR when I know that I am under attack and want a quick-and-dirty solution to prevent the unwanted traffic while giving valid users and connections the bandwidth that they need. When the attack has been mitigated, I remove the CAR configuration.

CAR Configuration

This section focuses on using CAR to deal with DoS traffic. Here are the commands to set up CAR:

```
Router(config-if)# interface type [slot_#]port_#
Router(config-if)# rate-limit {input | output}
  [access-group [rate-limit] acl-index] bps burst_normal burst_max
  conform-action action exceed-action action
```

Use either a rate-limit, a standard, or an extended ACL to match on the traffic for rate limiting by using one of these three respective commands:

```
Router(config-if)# access-list rate-limit ACL_# {precedence_value |
  MAC_address | mask precedence_mask}
```

or

```
Router(config-if)# access-list ACL_# {deny | permit}
  source_IP_address [source-wildcard_mask]
```

or

```
Router(config-if)# access-list ACL_# {deny | permit} protocol
  source source_wildcard destination destination-wildcard
  [precedence precedence][tos tos] [log]
```

The standard and extended ACLs can be numbered or named.

NOTE CAR requires that CEF be enabled on your router first.

rate-limit Command

Configuration of CAR is done on a router's interface with the **rate-limit** command. This command specifies the rate policy to be used for the matching traffic. The **input** and **output** parameters specify the direction in which CAR should be performed.

ACL Parameter

Optionally, you can specify an ACL number or name with the **access-group** parameter, which is used to examine specific traffic. You can use three different kinds of ACLs with CAR: rate-limit, standard, and extended (these are listed as the last three separate statements in the previous code examples). A rate-limit ACL enables you to match on a specific precedence value, a range of precedence values, or a specific MAC address. A standard ACL enables you to match on only source IP addresses. An extended ACL enables you to match on many types of traffic, such as the IP protocol, source and destination addresses, source and/or destination protocol information, precedence values, ToS values, and other fields supported by an ACL. The use of the **log** parameter, when combined with CAR, displays the actual matching process, but this is not recommended because of performance issues.

Rate Parameters

Three rate functions are defined in your CAR configuration:

- The average rate, specified in bits per second (bps), for the matching traffic. This is measured by a long-term average of the transmitted rate of traffic on the interface. Traffic under this rate is considered to be conforming.
- The normal burst size, specified in bits per second (bps). This determines how long traffic can burst above the average rate before it is considered nonconforming.
- The excessive burst rate, specified in bits per second (bps). Traffic that exceeds the excessive burst rate is considered nonconforming.

As you can see from these rate parameters, this is similar to Frame Relay's traffic-shaping parameters. However, one important point about the Frame Relay's parameters is that they are used for traffic shaping: CAR's parameters are used for rate limiting.

Rate Actions

When there is a match in an ACL or traffic type, the Cisco IOS can take the actions described in Table 17-3 for conforming (**conform-action** parameter) or nonconforming (**exceed-action** parameter) traffic.

Table 17-3 *CAR Actions*

Action Parameter	Explanation
continue	Evaluates the next **rate-limit** command on the interface
drop	Drops the packet
transmit	Transmits the packet
set prec-continue *precedence_value*	Sets the precedence value in the IP header to the specified value, and then evaluates the next **rate-limit** command on the interface
set prec-transmit *precedence_value*	Sets the precedence value in the IP header to the specified value and transmits the packet

Normally, you use the **continue**, **transmit**, and **set** parameters for conforming traffic; you use the **drop** command for nonconforming traffic. Remember that you are using CAR, in this instance, to limit the effect of DoS attacks. The options of setting a precedence value can be used in further CAR **rate-limit** commands on the interface or by other interfaces on this router or other routers.

Verifying CAR

After you have set up CAR, you can use one of the following three commands to verify its operation:

```
Router# show access-lists
Router# show access-lists rate-limit [ACL_#]
Router# show interfaces [interface_type_and_#] rate-limit
```

Example 17-15 displays sample output using the last command.

Example 17-15 *Using the* **show interfaces rate-limit** *Command for a Specified Interface*

```
Router# show interfaces ethernet2/0 rate-limit
Ethernet2/0
 Input
  matches: access-group rate-limit 100
    params: 64000 bps, 8000 limit, 8000 extended limit
    conformed 0 packets, 0 bytes; action: transmit
    exceeded 0 packets, 0 bytes; action: drop
    last packet: 9383444ms ago, current burst: 0 bytes
    last cleared 01:32:00 ago, conformed 0 bps, exceeded 0 bps
```

In this example, ACL 100 is used to define rate limiting. The average rate is 64,000 bps, and the two burst sizes are set to 8,000 bps, limiting the burst for traffic specified in **permit** statements in ACL 100.

Rate Limiting for ICMP and Smurf Attacks

Now that you have a basic understanding of the configuration of CAR, take a look at a couple of examples of its use for dealing with DoS attacks. In this section, I show you how to use CAR to limit the impact of the Smurf attack. In this example, the customer has a T1 connection and is experiencing a Smurf attack. The customer has talked to the ISP about the problem, and it has decided to implement rate limiting on its interfaces, to limit the scope of the attack as it tries to track down the originator of the attack. Example 17-16 shows the code used on the ISP router.

Example 17-16 *Using CAR on the ISP Router to Rate-Limit Smurf Traffic*

```
ISP(config)# access-list 100 permit icmp any any echo
ISP(config)# access-list 100 permit icmp any any echo-reply
ISP(config)# interface serial0
ISP(config-if)# rate-limit output access-group 100 64000 4000 4000
  conform-action transmit exceed-action drop
```

In Example 17-16, ICMP echos and echo replies are limited to 64 kbps of bandwidth, with a burst size of 4 kbps of bandwidth. Example 17-17 shows the code used on the customer's router.

Example 17-17 *Using CAR on the Customer Router to Rate-Limit Smurf Traffic*

```
Customer(config)# access-list 100 permit icmp any any echo
Customer(config)# access-list 100 permit icmp any any echo-reply
Customer(config)# interface serial0
Customer(config-if)# rate-limit output access-group 100 64000 4000 4000
  conform-action transmit exceed-action drop
```

Notice something interesting about the customer's router configuration: It is the same as the ISP's, applied in the same direction (out). If the attack is being started from the Internet to the company, the customer's configuration is not necessary. However, if the attack is occurring in both directions, possibly because an internal customer device was compromised and used as a reflector, setting up rate limiting in both directions is typically necessary.

TIP Remember that in DoS flood situations, you need to contact your ISP promptly and have it implement rate limiting in the outbound direction of its router's interface that connects to you. Using the configuration in Example 17-17 on your router still will cause you bandwidth problems on your link to the ISP if the attack is occurring from the outside; however, if your ISP is not cooperative, you can use this configuration on your perimeter router in the ingress direction, restricting the Smurf traffic to your internal devices.

Rate Limiting for TCP SYN and Other TCP Floods

You also can use rate limiting to limit the effect of TCP SYN flood attacks. Example 17-18 shows a configuration for a T1 link, which assumes that the hacker's source IP address is 201.1.1.1.

Example 17-18 *Using CAR to Rate-Limit TCP SYN Floods*

```
Router(config)# access-list 100 deny tcp any host 192.1.1.1 established
Router(config)# access-list 100 permit tcp any host 192.1.1.1
Router(config)# interface serial0
Router(config-if)# rate-limit input access-group 100 8000 4000 4000
  conform-action transmit exceed-action drop
```

Example 17-18 assumes that the configuration is applied on the customer's router. Notice something interesting about the ACL in this example. It uses the **established** keyword to block out all TCP packets except TCP SYN, which is used to establish connections, from being rate limited. Also, it restricts the TCP SYNs from any attacking host to the 192.1.1.1 server; other traffic sent to other internal servers is not rate limited by this configuration.

How to Choose a Rate Limit

If you suspect that you are under a TCP SYN flood or ICMP flood attack, you will want to set up rate limiting for your traffic. If you are not sure of the source of the attack, you can set up rate limiting to the full bandwidth of the link and then use the **show interfaces rate-limit** command to determine how you actually should set the rates in the **rate-limit** command.

Here is a simple example. Assume that you have a T1 connection to the Internet and that you suspect that your web servers in your DMZ are under some kind of TCP SYN flood attack; however, you are not absolutely sure. In this situation, configure CAR (assuming a T1 link) as displayed in Example 17-19.

Example 17-19 *Configuring CAR for a T1 Link*

```
Router(config)# access-list 100 permit tcp any host eq www established
Router(config)# access-list 101 permit tcp any host eq www
Router(config)# interface serial0
Router(config-if)# rate-limit output access-group 100 1544000 64000 64000
  conform-action transmit exceed-action drop
Router(config-if)# rate-limit output access-group 101 64000 16000 16000
  conform-action transmit exceed-action drop
```

In Example 17-19, there are two ACL statements. The first ACL statement looks for any web traffic that is established. The second one looks only for any web traffic. Notice that, with the rate limiting, the first **rate-limit** command allows full bandwidth for already established connections (ACL #101); the second command limits the initial setup of the HTTP connection (the TCP SYNs).

A couple things should be pointed out about this configuration. First, in the first **rate-limit** command, you need to replace the 1,544,000 bps value to match the actual speed of your interface. Second, do not choose arbitrary rate-limiting values for the second **rate-limit** command for the TCP SYN setup of the web connections. Before you define any limits, you should understand your traffic patterns: Putting in a value that is too small or too large might create additional problems. Use the **show interfaces rate-limit** command to tune this process.

TIP Before implementing CAR, make sure that you have a baseline of traffic flowing through the router under normal circumstances, categorized by traffic type and destination addresses.

Rate Limiting for W32.Blaster Worm

As you might have experienced firsthand, dealing with a worm attack can be a hair-raising experience, especially because most worm attacks either purposefully or inadvertently cause a DoS attack, sucking up bandwidth and causing problems for other applications and services in your network. Assuming that you know the protocols and ports that these worms are using, you can set up rate limiting, using CAR, to limit the bandwidth that they can use.

Consider the W32 Blaster worm as an example. Example 17-20 shows a sample CAR configuration to rate limit the bandwidth to traffic on the ports used by the worm.

Example 17-20 *Using CAR to Rate Limit the W32 Blaster Worm*

```
Router(config)# access-list 199 permit udp any any eq 135
Router(config)# access-list 199 permit udp any any eq 139
Router(config)# access-list 199 permit tcp any any eq 135
Router(config)# access-list 199 permit tcp any any eq 135
Router(config)# access-list 199 permit udp any any eq 445
Router(config)# access-list 199 permit tcp any any eq 445
Router(config)# access-list 199 deny ip any any
Router(config)# interface vlan10
Router(config-if)# rate-limit input access-group rate-limit 199
  256000 64000 64000
  conform-action transmit exceed-action drop
```

NOTE It is important to point out that you typically are not allowing the previous ports into your network—at least, you should not be. Therefore, this kind of rate limiting typically is done internal to your network on which you have a worm infection that is running rampant. In the previous example, this was done on a 3550 Catalyst switch, to limit the bandwidth for these connections and to limit the impact of the worm. Also make sure that whatever bandwidth limits you have imposed are high enough to allow at least some legitimate traffic of the same type, but low enough to allow other types of traffic. When you have eradicated the worm from your network by updating your antivirus software and applying the appropriate patches, remove the CAR configuration for this traffic because it also can affect legitimate traffic.

NBAR

I discussed NBAR in Chapter 10, "Filtering Web and Application Traffic," and Chapter 15, "Routing Protocol Protection." In Chapter 10, I used NBAR to drop P2P traffic. In Chapter 15, I used NBAR to route unwanted or undesirable traffic to a null interface. You also can use NBAR to limit packet rates of either undesirable traffic or flooding traffic. The configuration of NBAR for rate limiting is actually very similar to using it for policing, with a few minor differences.

NOTE Cisco recommends using NBAR instead of other rate-limiting features when trying to deal with DoS attacks.

File-Sharing and P2P Programs

I have walked into many environments where P2P programs are using up a lot of a company's bandwidth, but the company's security policy does not explicitly prohibit their use. In one particular situation, many managers and officers of the corporation used these as well as Instant Messenger products and wanted to continue their use, even though they did not really add to the company's bottom line. In this situation, I recommended that the director of networking implement rate limiting for these kinds of products. In this situation, employee file sharing was much slower in their downloads, but employees still were able to access P2P sites, as well as legitimate applications that did not suffer from the burden of these programs.

Smurf Example

Instead of rehashing the syntax of the NBAR configuration commands, which were covered in Chapter 10, I display in this section an example configuration and focus on one new command, **police**, which specifies the policing action to take. This section uses the Smurf attack as an example. Example 17-21 shows the code.

Example 17-21 *Using NBAR to Rate Limit Smurf Traffic*

```
Router(config)# access-list 100 permit icmp any any echo          (1)
Router(config)# access-list 100 permit icmp any any echo-reply
Router(config)# class-map match-any smurf-attacks                 (2)
Router(config-cmap)# match access-group 100
Router(config-cmap)# exit
Router(config)# policy-map mark-smurf                             (3)
Router(config-pmap)# class smurf-attacks
Router(config-pmap-c)# set ip dscp 1
Router(config-pmac-c)# exit
Router(config)# interface ethernet1
```

Example 17-21 *Using NBAR to Rate Limit Smurf Traffic (Continued)*

```
Router(config-if)# description ***To the ISP***
Router(config-if)# service-policy input mark-smurf                (4)
Router(config-if)# exit
Router(config)# class-map match-any smurf-marked                  (5)
Router(config-cmap)# match dscp 1
Router(config-cmap)# exit
Router(config)# police-map limit-smurfs                           (6)
Router(config-pmap)# class smurf-marked
Router(config-pmap-c)# police 64000 4000 4000
  conform-action transmit exceed-action drop violation drop
Router(config)# interface ethernet0
Router(config-if)# description ***To the Internal Network***
Router(config-if)# service-policy output limit-smurfs             (7)
```

The following is an explanation of the configuration in Example 17-21, with reference to the numbering on the right side:

1. ACL 100 defines the traffic that I am limiting. Smurf uses ICMP echos and echo replies.
2. The **class-map** command looks for this traffic with the **match access-group** command.
3. The policy map marks all packets matched in the **class-map** command with a DSCP value of 1.
4. The policy map is activated on the perimeter router's external interface, marking traffic as it comes from the Internet and enters this network.
5. Remember that NBAR cannot perform two policies on one interface, such as mark and drop, or mark and limit. Therefore, a separate policy must be created. In Step 5, a second class map is created, looking for the packets (ICMP) that had their DSCP value set to 1.
6. The **police-map** command specifies the policy to be applied to this traffic. In this instance, I first reference the class map, which matches on packets with the DSCP value set to 1. Second, the **police** command is used to enforce rate limiting. This is very similar to CAR's **rate-limit** command. The only difference between this command and the **police** command is that the latter command supports an additional parameter, **violation**, which is optional and specifies the action to take if both the normal and excessive burst sizes are violated. In the previous example, ICMP traffic is limited to an average throughput of 64 kbps, with a burst of 4 kbps. If either of these limits is exceeded, the policy is to drop these excessive ICMP messages.
7. The last step is to activate the second policy map on the router's internal interface (or interfaces, if there is more than one internal interface). After it is activated on the internal interface, the second policy map enforces rate limiting for packets that have their DSCP value set to 1.

Use the **show policy-map interface** command to examine the matching process on Ethernet1 and the policing process on Ethernet0.

As I mentioned in Chapters 10 and 15, NBAR is a multifunctional tool that can be very useful in implementing security measures, which is not what Cisco initially had in mind when offering this feature. Given the last example, you easily can modify it to suite other types of attacks, such as an attack by the SQL Slammer worm. You only need to modify the ACL that looks for the offending traffic.

You even can use this process to limit the bandwidth for other types of traffic that are not necessary for business purposes, such as instant messenger, P2P, and other programs. This approach still allows your users to use these programs, but it limits the bandwidth that they can use. In this situation, however, your first class map looks for the traffic and is applied to the inside interface of your perimeter router, and the policing policy is applied to the external interface.

W32.Blaster Worm Example

In the "CAR" section, I discussed how to use rate limiting to limit the effectiveness of the W32.Blaster worm. I use this worm as an example to set up NBAR to implement rate limiting. Example 17-22 shows the basic configuration for NBAR to rate limit the W32.Blaster worm.

Example 17-22 *Using NBAR to Rate Limit the W32.Blaster Worm*

```
Router(config)# ip nbar port-map netbios1 tcp 135 139 445          (1)
Router(config)# ip nbar port-map netbios2 udp 135 139 445
Router(config)# class-map match-any w32blaster-attack              (2)
Router(config-cmap)# match protocol netbios1
Router(config-cmap)# match protocol netbios2
Router(config-cmap)# exit
Router(config)# policy-map mark-w32blaster                         (3)
Router(config-pmap)# class w32blaster-attack
Router(config-pmap-c)# set ip dscp 1
Router(config-pmac-c)# exit
Router(config)# interface ethernet1
Router(config-if)# description ***To the ISP***
Router(config-if)# service-policy input mark-w32blaster            (4)
Router(config-if)# exit
Router(config)# class-map match-any mark-w32blaster                (5)
Router(config-cmap)# match dscp 1
Router(config-cmap)# exit
Router(config)# police-map limit-w32blaster                        (6)
Router(config-pmap)# class w32blaster-marked
Router(config-pmap-c)# police 64000 4000 4000
  conform-action transmit exceed-action drop violation drop
Router(config)# interface ethernet0
Router(config-if)# description ***To the Internal Network***
Router(config-if)# service-policy output limit-w32blaster          (7)
```

In Statement 1, the first two commands create a custom protocol for NBAR, called netbios, which includes both TCP and UDP ports 135, 139, and 445. Statement 2 points to the custom protocol that you defined for NBAR. All of the other Statements (3 through 7) refer to the same steps that you performed in setting up NBAR for Smurf attacks.

NOTE Setting up NBAR to rate limit other types of attacks, including other worms, necessitates that you set up rate limiting similar to the Smurf or W32.Blaster configuration.

Summary

This chapter showed you the basics of dealing with DoS attacks. If you suspect that you are under a DoS attack, examine your router's CPU and memory utilization, and look for abnormalities. You also can examine your ACL counters to see if a specific kind of traffic is increasing in an unusual manner. You can use ACL logging to gather more information about the attack, but this is process intensive for the router. In this situation, I recommend using NetFlow to gather information about the attack.

For TCP SYN flood attacks, you can use the router's TCP Intercept feature. However, if you already have the Cisco IOS Firewall feature set installed on your router, use CBAC's timeouts and thresholds to limit the effectiveness of a DoS attack.

In many cases, you need to limit the amount of traffic generated by the DoS attack, to allow legitimate traffic while you track down the culprit of the attack. For ICMP attacks, you can use ICMP rate limiting. You also can use CAR or NBAR.

Next up is Chapter 18, which shows you how to configure your router to produce logging information, as well as how to examine this information.