Connection Settings

This chapter describes how to configure connection settings for connections that go through the ASA, or for management connections that go to the ASA.

What Are Connection Settings?

Connection settings comprise a variety of features related to managing traffic connections, such as a TCP flow through the ASA. Some features are named components that you would configure to supply specific services.

Connection settings include the following:

- **Global timeouts for various protocols**—All global timeouts have default values, so you need to change them only if you are experiencing premature connection loss.
- **Connection timeouts per traffic class**—You can override the global timeouts for specific types of traffic using service policies. All traffic class timeouts have default values, so you do not have to set them.
- **Connection limits and TCP Intercept**—By default, there are no limits on how many connections can go through (or to) the ASA. You can set limits on particular traffic classes using service policy rules to protect servers from denial of service (DoS) attacks. Particularly, you can set limits on embryonic connections (those that have not finished the TCP handshake), which protects against SYN flooding attacks. When embryonic limits are exceeded, the TCP Intercept component gets involved to proxy connections and ensure that attacks are throttled.
- **Dead Connection Detection (DCD)**—If you have persistent connections that are valid but often idle, so that they get closed because they exceed idle timeout settings, you can enable Dead Connection Detection to identify idle but valid connections and keep them alive (by resetting their idle timers). Whenever idle times are exceeded, DCD probes both sides of the connection to see if both sides agree the connection is valid. The **show service-policy** command includes counters to show the amount of activity from DCD.

- **TCP sequence randomization**—Each TCP connection has two ISNs: one generated by the client and one generated by the server. By default, the ASA randomizes the ISN of the TCP SYN passing in both the inbound and outbound directions. Randomization prevents an attacker from predicting the next ISN for a new connection and potentially hijacking the new session. You can disable randomization per traffic class if desired.
- **TCP Normalization**—The TCP Normalizer protects against abnormal packets. You can configure how some types of packet abnormalities are handled by traffic class.
- **TCP State Bypass**—You can bypass TCP state checking if you use asymmetrical routing in your network.

Configure Connection Settings

Connection limits, timeouts, TCP Normalization, TCP sequence randomization, and decrementing time-to-live (TTL) have default values that are appropriate for most networks. You need to configure these connection settings only if you have unusual requirements, your network has specific types of configuration, or if you are experiencing unusual connection loss due to premature idle timeouts.

TCP Intercept, TCP State Bypass, and Dead Connection Detection (DCD) are not enabled. You would configure these services on specific traffic classes only, and not as a general service.

The following general procedure covers the gamut of possible connection setting configurations. Pick and choose which to implement based on your needs.

Procedure

Step 1 Configure Global Timeouts, page 11-3. These settings change the default idle timeouts for various protocols for all traffic that passes through the device. If you are having problems with connections being reset due to premature timeouts, first try changing the global timeouts.

Step 2 Protect Servers from a SYN Flood DoS Attack (TCP Intercept), page 11-4. Use this procedure to configure TCP Intercept.

Step 3 Customize Abnormal TCP Packet Handling (TCP Maps, TCP Normalizer), page 11-7, if you want to alter the default TCP Normalization behavior for specific traffic classes.

Step 4 Bypass TCP State Checks for Asynchronous Routing (TCP State Bypass), page 11-10, if you have this type of routing environment.

Step 5 Disable TCP Sequence Randomization, page 11-13, if the default randomization is scrambling data for certain connections.

Step 6 Configure Connection Settings for Specific Traffic Classes (All Services), page 11-14. This is a catch-all procedure for connection settings. These settings can override the global defaults for specific traffic classes using service policy rules. You also use these rules to customize TCP Normalizer, change TCP sequence randomization, decrement time-to-live on packets, and implement TCP Intercept, Dead Connection Detection, or TCP State Bypass.

Configure Global Timeouts

You can set the global idle timeout durations for the connection and translation slots of various protocols. If the slot has not been used for the idle time specified, the resource is returned to the free pool. TCP connection slots are freed approximately 60 seconds after a normal connection close sequence.

Changing the global timeout sets a new default timeout, which in some cases can be overridden for particular traffic flows through service policies.

Procedure

Step 1 Use the **timeout** command to set global timeouts.

```
hostname(config)# timeout feature time
```

All timeout values are in the format *hh*:*mm*:*ss*, with a maximum duration of 1193:0:0. Use the **no timeout** command to reset all timeouts to their default values. If you want to simply reset one timer to the default, enter the **timeout** command for that setting with the default value.

Use **0** for the value to disable a timer.

You can configure the following global timeouts.

- **timeout conn** *hh*:*mm*:*ss*—The idle time after which a connection closes, between 0:5:0 and 1193:0:0. The default is 1 hour (1:0:0).
- **timeout half-closed** *hh*:*mm*:*ss*—The idle time until a TCP half-closed connection closes. The minimum is 5 minutes. The default is 10 minutes.
- **timeout udp** *hh*:*mm*:*ss*—The idle time until a UDP connection closes. This duration must be at least 1 minute. The default is 2 minutes.
- **timeout icmp** *hh*:*mm*:*ss*—The idle time for ICMP, between 0:0:2 and 1193:0:0. The default is 2 seconds (0:0:2).
- **timeout sunrpc** *hh*:*mm*:*ss*—The idle time until a SunRPC slot is freed. This duration must be at least 1 minute. The default is 10 minutes.
- **timeout H323** *hh*:*mm*:*ss*—The idle time after which H.245 (TCP) and H.323 (UDP) media connections close, between 0:0:0 and 1193:0:0. The default is 5 minutes (0:5:0). Because the same connection flag is set on both H.245 and H.323 media connections, the H.245 (TCP) connection shares the idle timeout with the H.323 (RTP and RTCP) media connection.
- **timeout h225** *hh*:*mm*:*ss*—The idle time until an H.225 signaling connection closes. The H.225 default timeout is 1 hour (1:0:0). To close a connection immediately after all calls are cleared, a value of 1 second (0:0:1) is recommended.
- **timeout mgcp** *hh*:*mm*:*ss*—The idle time after which an MGCP media connection is removed, between 0:0:0 and 1193:0:0. The default is 5 minutes (0:5:0)
- **timeout mgcp-pat** *hh*:*mm*:*ss*—The absolute interval after which an MGCP PAT translation is removed, between 0:0:0 and 1193:0:0. The default is 5 minutes (0:5:0). The minimum time is 30 seconds.
- **timeout sip** *hh*:*mm*:*ss*—The idle time until a SIP signaling port connection closes, between 0:5:0 and 1193:0:0. The default is 30 minutes (0:30:0).
- **timeout sip_media** *hh*:*mm*:*ss*—The idle time until an SIP media port connection closes. This duration must be at least 1 minute. The default is 2 minutes. The SIP media timer is used used for SIP RTP/RTCP with SIP UDP media packets, instead of the UDP inactivity timeout.

- **timeout sip-provisional-media** *hh*:*mm*:*ss*—The timeout value for SIP provisional media connections, between 0:1:0 and 1193:0:0. The default is 2 minutes.
- **timeout sip-invite** *hh*:*mm*:*ss*—The idle time after which pinholes for PROVISIONAL responses and media xlates will be closed, between 0:1:0 and 00:30:0. The default is 3 minutes (0:3:0).
- **timeout sip-disconnect** *hh*:*mm*:*ss*—The idle time after which a SIP session is deleted if the 200 OK is not received for a CANCEL or a BYE message, between 0:0:1 and 00:10:0. The default is 2 minutes (0:2:0).
- **timeout uauth** *hh*:*mm*:*ss* {**absolute** | **inactivity**}—The duration before the authentication and authorization cache times out and the user has to reauthenticate the next connection, between 0:0:0 and 1193:0:0. The default is 5 minutes (0:5:0). The default timer is **absolute**; you can set the timeout to occur after a period of inactivity by entering the **inactivity** keyword. The uauth duration must be shorter than the xlate duration. Set to 0 to disable caching. Do not use 0 if passive FTP is used for the connection or if the virtual http command is used for web authentication.
- **timeout xlate** *hh*:*mm*:*ss*—The idle time until a translation slot is freed. This duration must be at least 1 minute. The default is 3 hours.
- **timeout tcp-proxy-reassembly** *hh*:*mm*:*ss*—The idle timeout after which buffered packets waiting for reassembly are dropped, between 0:0:10 and 1193:0:0. The default is 1 minute (0:1:0).
- **timeout floating-conn** *hh*:*mm*:*ss*—When multiple static routes exist to a network with different metrics, the ASA uses the one with the best metric at the time of connection creation. If a better route becomes available, then this timeout lets connections be closed so a connection can be reestablished to use the better route. The default is 0 (the connection never times out). To take advantage of this feature, change the timeout to a new value between 0:1:0 and 1193:0:0.
- **timeout pat-xlate** *hh*:*mm*:*ss*—The idle time until a PAT translation slot is freed, between 0:0:30 and 0:5:0. The default is 30 seconds. You may want to increase the timeout if upstream routers reject new connections using a freed PAT port because the previous connection might still be open on the upstream device.

Protect Servers from a SYN Flood DoS Attack (TCP Intercept)

A SYN-flooding denial of service (DoS) attack occurs when an attacker sends a series of SYN packets to a host. These packets usually originate from spoofed IP addresses. The constant flood of SYN packets keeps the server SYN queue full, which prevents it from servicing connection requests from legitimate users.

You can limit the number of embryonic connections to help prevent SYN flooding attacks. An embryonic connection is a connection request that has not finished the necessary handshake between source and destination.

When the embryonic connection threshold of a connection is crossed, the ASA acts as a proxy for the server and generates a SYN-ACK response to the client SYN request using the SYN cookie method (see Wikipedia for details on SYN cookies). When the ASA receives an ACK back from the client, it can then authenticate that the client is real and allow the connection to the server. The component that performs the proxy is called TCP Intercept.

Note Ensure that you set the embryonic connection limit lower than the TCP SYN backlog queue on the server that you want to protect. Otherwise, valid clients can no longer access the server during a SYN attack. To determine reasonable values for embryonic limits, carefully analyze the capacity of the server, the network, and server usage.

The end-to-end process for protecting a server from a SYN flood attack involves setting connection limits, enabling TCP Intercept statistics, and then monitoring the results.

Before You Begin

- Ensure that you set the embryonic connection limit lower than the TCP SYN backlog queue on the server that you want to protect. Otherwise, valid clients can no longer access the server during a SYN attack. To determine reasonable values for embryonic limits, carefully analyze the capacity of the server, the network, and server usage.
- Depending on the number of CPU cores on your ASA model, the maximum concurrent and embryonic connections can exceed the configured numbers due to the way each core manages connections. In the worst case scenario, the ASA allows up to *n*-1 extra connections and embryonic connections, where *n* is the number of cores. For example, if your model has 4 cores, if you configure 6 concurrent connections and 4 embryonic connections, you could have an additional 3 of each type. To determine the number of cores for your model, enter the **show cpu core** command.

Procedure

Step 1 Create an L3/L4 class map to identify the servers you are protecting. Use an access-list match.

```
class-map name
match parameter
```

Example:

```
hostname(config)# access-list servers extended permit tcp any host 10.1.1.5 eq http
hostname(config)# access-list servers extended permit tcp any host 10.1.1.6 eq http
hostname(config)# class-map protected-servers
hostname(config-cmap)# match access-list servers
```

Step 2 Add or edit a policy map that sets the actions to take with the class map traffic, and identify the class map.

```
policy-map name
class name
```

Example:

```
hostname(config)# policy-map global_policy
hostname(config-pmap)# class protected-servers
```

In the default configuration, the global_policy policy map is assigned globally to all interfaces. If you want to edit the global_policy, enter global_policy as the policy name. For the class map, specify the class you created earlier in this procedure.

Step 3 Set the embryonic connection limits.

- **set connection embryonic-conn-max** n—The maximum number of simultaneous embryonic connections allowed, between 0 and 2000000. The default is 0, which allows unlimited connections.

- **set connection per-client-embryonic-max** n—The maximum number of simultaneous embryonic connections allowed per client, between 0 and 2000000. The default is 0, which allows unlimited connections.

Example:

```
hostname(config-pmap-c)# set connection embryonic-conn-max 1000
hostname(config-pmap-c)# set connection per-client-embryonic-max 50
```

Step 4 If you are editing an existing service policy (such as the default global policy called global_policy), you can skip this step. Otherwise, activate the policy map on one or more interfaces.

service-policy *policymap_name* {**global** | **interface** *interface_name*}

Example:

```
hostname(config)# service-policy global_policy global
```

The **global** keyword applies the policy map to all interfaces, and **interface** applies the policy to one interface. Only one global policy is allowed. You can override the global policy on an interface by applying a service policy to that interface. You can only apply one policy map to each interface.

Step 5 Configure threat detection statistics for attacks intercepted by TCP Intercept.

threat-detection statistics tcp-intercept [**rate-interval** *minutes*] [**burst-rate** *attacks_per_sec*] [**average-rate** *attacks_per_sec*]

Example:

```
hostname(config)# threat-detection statistics tcp-intercept
```

The **rate-interval** keyword sets the size of the history monitoring window, between 1 and 1440 minutes. The default is 30 minutes. During this interval, the ASA samples the number of attacks 30 times.

The **burst-rate** keyword sets the threshold for syslog message generation, between 25 and 2147483647. The default is 400 per second. When the burst rate is exceeded, syslog message 733104 is generated.

The **average-rate** keyword sets the average rate threshold for syslog message generation, between 25 and 2147483647. The default is 200 per second. When the average rate is exceeded, syslog message 733105 is generated.

Step 6 Monitor the results with the following commands:

- **show threat-detection statistics top tcp-intercept** [**all** | **detail**]—View the top 10 protected servers under attack. The **all** keyword shows the history data of all the traced servers. The **detail** keyword shows history sampling data. The ASA samples the number of attacks 30 times during the rate interval, so for the default 30 minute period, statistics are collected every 60 seconds.
- **clear threat-detection statistics tcp-intercept**—Erases TCP Intercept statistics.

Example:

```
hostname(config)# show threat-detection statistics top tcp-intercept
Top 10 protected servers under attack (sorted by average rate)
Monitoring window size: 30 mins    Sampling interval: 30 secs
<Rank> <Server IP:Port> <Interface> <Ave Rate> <Cur Rate> <Total> <Source IP (Last Attack
Time)>
------------------------------------------------------------------------------
1    10.1.1.5:80 inside 1249 9503 2249245 <various> Last: 10.0.0.3 (0 secs ago)
2    10.1.1.6:80 inside 10 10 6080 10.0.0.200 (0 secs ago)
```

Customize Abnormal TCP Packet Handling (TCP Maps, TCP Normalizer)

The TCP Normalizer identifies abnormal packets that the ASA can act on when they are detected; for example, the ASA can allow, drop, or clear the packets. TCP normalization helps protect the ASA from attacks. TCP normalization is always enabled, but you can customize how some features behave.

The default configuration includes the following settings:

```
no check-retransmission
no checksum-verification
exceed-mss allow
queue-limit 0 timeout 4
reserved-bits allow
syn-data allow
synack-data drop
invalid-ack drop
seq-past-window drop
tcp-options range 6 7 clear
tcp-options range 9 255 clear
tcp-options selective-ack allow
tcp-options timestamp allow
tcp-options window-scale allow
ttl-evasion-protection
urgent-flag clear
window-variation allow-connection
```

To customize the TCP normalizer, first define the settings using a TCP map. Then, you can apply the map to selected traffic classes using service policies.

Procedure

Step 1 Create a TCP map to specify the TCP normalization criteria that you want to look for.

```
hostname(config)# tcp-map tcp-map-name
```

Step 2 Configure the TCP map criteria by entering one or more of the following commands. The defaults are used for any commands you do not enter. Use the **no** form of a command to disable the setting.

- **check-retransmission**—Prevent inconsistent TCP retransmissions. This command is disabled by default.
- **checksum-verification**—Verify the TCP checksum, dropping packets that fail verification. This command is disabled by default.
- **exceed-mss** {**allow** | **drop**}—Allow or drop packets whose data length exceeds the TCP maximum segment size. The default is to allow the packets.
- **invalid-ack** {**allow** | **drop**}—Allow or drop packets with an invalid ACK. The default is to drop the packet, with the exception of WAAS connections, where they are allowed. You might see invalid ACKs in the following instances:
  - In the TCP connection SYN-ACK-received status, if the ACK number of a received TCP packet is not exactly the same as the sequence number of the next TCP packet sending out, it is an invalid ACK.
  - Whenever the ACK number of a received TCP packet is greater than the sequence number of the next TCP packet sending out, it is an invalid ACK.

- **queue-limit** *pkt_num* [**timeout** *seconds*]—Set the maximum number of out-of-order packets that can be buffered and put in order for a TCP connection, between 1 and 250 packets. The default is 0, which means this setting is disabled and the default system queue limit is used depending on the type of traffic:
  - Connections for application inspection (the **inspect** command), IPS (the **ips** command), and TCP check-retransmission (the TCP map **check-retransmission** command) have a queue limit of 3 packets. If the ASA receives a TCP packet with a different window size, then the queue limit is dynamically changed to match the advertised setting.
  - For other TCP connections, out-of-order packets are passed through untouched.

  If you set the **queue-limit** command to be 1 or above, then the number of out-of-order packets allowed for all TCP traffic matches this setting. For example, for application inspection, IPS, and TCP check-retransmission traffic, any advertised settings from TCP packets are ignored in favor of the **queue-limit** setting. For other TCP traffic, out-of-order packets are now buffered and put in order instead of passed through untouched.

  The **timeout** *seconds* argument sets the maximum amount of time that out-of-order packets can remain in the buffer, between 1 and 20 seconds; if they are not put in order and passed on within the timeout period, then they are dropped. The default is 4 seconds. You cannot change the timeout for any traffic if the *pkt_num* argument is set to 0; you need to set the limit to be 1 or above for the **timeout** keyword to take effect.
- **reserved-bits** {**allow** | **clear** | **drop**}—Set the action for reserved bits in the TCP header. You can **allow** the packet (without changing the bits), **clear** the bits and allow the packet, or **drop** the packet.
- **seq-past-window** {**allow** | **drop**}—Set the action for packets that have past-window sequence numbers, namely the sequence number of a received TCP packet is greater than the right edge of the TCP receiving window. You can **allow** the packets only if the **queue-limit** command is set to 0 (disabled). The default is to drop the packets.
- **synack-data** {**allow** | **drop**}—Allow or drop TCP SYNACK packets that contain data. The default is to drop the packet.
- **syn-data** {**allow** | **drop**}—Allow or drop SYN packets with data. The default is to allow the packet.
- **tcp-options** {**selective-ack** | **timestamp** | **window-scale** | **range** *lower upper*} {**allow** | **clear**}—Set the action for packets with TCP options. Three options are named: **selective-ack** (selective acknowledgment mechanism), **timestamp**, and **window-scale** (window scale mechanism). For other options, you specify them by number on the **range** keyword, where the range limits are 6-7, 9-255. You can enter the command multiple times in a map to define your complete policy.

  You can **allow** the packet (without changing the options), **clear** the options and allow the packet, or **drop** the packet. The default for the three named options is to allow them; the default for all other options is to clear them. Note that clearing the timestamp option disables PAWS and RTT.
- **ttl-evasion-protection**—Protect against TTL evasion attacks. TTL evasion protection is enabled by default, so you would only need to enter the **no** form of this command.

  For example, an attacker can send a packet that passes policy with a very short TTL. When the TTL goes to zero, a router between the ASA and the endpoint drops the packet. It is at this point that the attacker can send a malicious packet with a long TTL that appears to the ASA to be a retransmission and is passed. To the endpoint host, however, it is the first packet that has been received by the attacker. In this case, an attacker is able to succeed without security preventing the attack.
- **urgent-flag** {**allow** | **clear**}—Set the action for packets with the URG flag. You can **allow** the packet, or **clear** the flag and allow the packet. The default is to clear the flag.

The URG flag is used to indicate that the packet contains information that is of higher priority than other data within the stream. The TCP RFC is vague about the exact interpretation of the URG flag, therefore end systems handle urgent offsets in different ways, which may make the end system vulnerable to attacks.

- **window-variation** {**allow** | **drop**}—Allow or drop a connection that has changed its window size unexpectedly. The default is to allow the connection.

  The window size mechanism allows TCP to advertise a large window and to subsequently advertise a much smaller window without having accepted too much data. From the TCP specification, "shrinking the window" is strongly discouraged. When this condition is detected, the connection can be dropped.

Step 3 Apply the TCP map to a traffic class using a service policy.

a. Define the traffic class with an L3/L4 class map and add the map to a policy map.

```
class-map name
match parameter
policy-map name
class name
```

Example:

```
hostname(config)# class-map normalization
hostname(config-cmap)# match any
hostname(config)# policy-map global_policy
hostname(config-pmap)# class normalization
```

In the default configuration, the global_policy policy map is assigned globally to all interfaces. If you want to edit the global_policy, enter global_policy as the policy name. For information on matching statements for class maps, see Identify Traffic (Layer 3/4 Class Maps), page 1-13.

b. Apply the TCP map.

set connection advanced-options *tcp-map-name*

Example:

```
hostname(config-pmap-c)# set connection advanced-options tcp_map1
```

c. If you are editing an existing service policy (such as the default global policy called global_policy), you are done. Otherwise, activate the policy map on one or more interfaces.

service-policy *policymap_name* {**global** | **interface** *interface_name*}

Example:

```
hostname(config)# service-policy global_policy global
```

The **global** keyword applies the policy map to all interfaces, and **interface** applies the policy to one interface. Only one global policy is allowed. You can override the global policy on an interface by applying a service policy to that interface. You can only apply one policy map to each interface.

Examples

For example, to allow urgent flag and urgent offset packets for all traffic sent to the range of TCP ports between the well known FTP data port and the Telnet port, enter the following commands:

```
hostname(config)# tcp-map tmap
hostname(config-tcp-map)# urgent-flag allow
hostname(config-tcp-map)# class-map urg-class
hostname(config-cmap)# match port tcp range ftp-data telnet
```

```
hostname(config-cmap)# policy-map pmap
hostname(config-pmap)# class urg-class
hostname(config-pmap-c)# set connection advanced-options tmap
hostname(config-pmap-c)# service-policy pmap global
```

Bypass TCP State Checks for Asynchronous Routing (TCP State Bypass)

If you have an asynchronous routing environment in your network, where the outbound and inbound flow for a given connection can go through two different ASA devices, you need to implement TCP State Bypass on the affected traffic.

However, TCP State Bypass weakens the security of your network, so you should apply bypass on very specific, limited traffic classes.

The following topics explain the problem and solution in more detail.

The Asynchronous Routing Problem

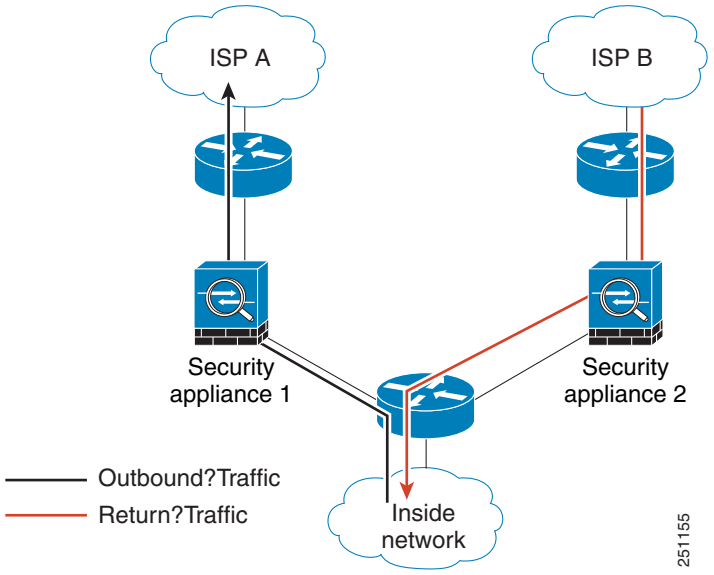
By default, all traffic that goes through the ASA is inspected using the Adaptive Security Algorithm and is either allowed through or dropped based on the security policy. The ASA maximizes the firewall performance by checking the state of each packet (is this a new connection or an established connection?) and assigning it to either the session management path (a new connection SYN packet), the fast path (an established connection), or the control plane path (advanced inspection). See the general operations configuration guide for more detailed information about the stateful firewall.

TCP packets that match existing connections in the fast path can pass through the ASA without rechecking every aspect of the security policy. This feature maximizes performance. However, the method of establishing the session in the fast path using the SYN packet, and the checks that occur in the fast path (such as TCP sequence number), can stand in the way of asymmetrical routing solutions: both the outbound and inbound flow of a connection must pass through the same ASA.

For example, a new connection goes to ASA 1. The SYN packet goes through the session management path, and an entry for the connection is added to the fast path table. If subsequent packets of this connection go through ASA 1, then the packets will match the entry in the fast path, and are passed through. But if subsequent packets go to ASA 2, where there was not a SYN packet that went through

the session management path, then there is no entry in the fast path for the connection, and the packets are dropped. The following figure shows an asymmetric routing example where the outbound traffic goes through a different ASA than the inbound traffic:

Figure 11-1 Asymmetric Routing



If you have asymmetric routing configured on upstream routers, and traffic alternates between two ASAs, then you can configure TCP state bypass for specific traffic. TCP state bypass alters the way sessions are established in the fast path and disables the fast path checks. This feature treats TCP traffic much as it treats a UDP connection: when a non-SYN packet matching the specified networks enters the ASA, and there is not an fast path entry, then the packet goes through the session management path to establish the connection in the fast path. Once in the fast path, the traffic bypasses the fast path checks.

Guidelines for TCP State Bypass

TCP State Bypass Unsupported Features

The following features are not supported when you use TCP state bypass:

- Application inspection—Application inspection requires both inbound and outbound traffic to go through the same ASA, so application inspection is applied TCP state bypass traffic.
- AAA authenticated sessions—When a user authenticates with one ASA, traffic returning via the other ASA will be denied because the user did not authenticate with that ASA.
- TCP Intercept, maximum embryonic connection limit, TCP sequence number randomization—The ASA does not keep track of the state of the connection, so these features are not applied.
- TCP normalization—The TCP normalizer is disabled.
- Service module functionality—You cannot use TCP state bypass and any application running on an any type of service module, such as IPS or CX.
- Stateful failover

TCP State Bypass NAT Guidelines

Because the translation session is established separately for each ASA, be sure to configure static NAT on both ASAs for TCP state bypass traffic. If you use dynamic NAT, the address chosen for the session on ASA 1 will differ from the address chosen for the session on ASA 2.

Configure TCP State Bypass

To bypass TCP state checking in asynchronous routing environments, carefully define a traffic class that applies to the affected hosts or networks only, then enable TCP State Bypass on the traffic class using a service policy. Because bypass reduces the security of the network, limit its application as much as possible.

Procedure

Step 1 Create an L3/L4 class map to identify the hosts that require TCP State Bypass. Use an access-list match to identify the source and destination hosts.

class-map *name*
match *parameter*

Example:

```
hostname(config)# access-list bypass extended permit tcp host 10.1.1.1 host 10.2.2.2
hostname(config)# class-map bypass-class
hostname(config-cmap)# match access-list bypass
```

Step 2 Add or edit a policy map that sets the actions to take with the class map traffic, and identify the class map.

policy-map *name*
class *name*

Example:

```
hostname(config)# policy-map global_policy
hostname(config-pmap)# class bypass-class
```

In the default configuration, the global_policy policy map is assigned globally to all interfaces. If you want to edit the global_policy, enter global_policy as the policy name. For the class map, specify the class you created earlier in this procedure.

Step 3 Enable TCP State Bypass on the class.

set connection advanced-options tcp-state-bypass

Step 4 If you are editing an existing service policy (such as the default global policy called global_policy), you are done. Otherwise, activate the policy map on one or more interfaces.

service-policy *policymap_name* {**global** | **interface** *interface_name*}

Example:

```
hostname(config)# service-policy global_policy global
```

The **global** keyword applies the policy map to all interfaces, and **interface** applies the policy to one interface. Only one global policy is allowed. You can override the global policy on an interface by applying a service policy to that interface. You can only apply one policy map to each interface.

Examples

The following is a sample configuration for TCP state bypass:

```
hostname(config)# access-list tcp_bypass extended permit tcp 10.1.1.0 255.255.255.224 any

hostname(config)# class-map tcp_bypass
hostname(config-cmap)# description "TCP traffic that bypasses stateful firewall"
hostname(config-cmap)# match access-list tcp_bypass

hostname(config-cmap)# policy-map tcp_bypass_policy
hostname(config-pmap)# class tcp_bypass
hostname(config-pmap-c)# set connection advanced-options tcp-state-bypass

hostname(config-pmap-c)# service-policy tcp_bypass_policy outside
```

Disable TCP Sequence Randomization

Each TCP connection has two ISNs: one generated by the client and one generated by the server. The ASA randomizes the ISN of the TCP SYN passing in both the inbound and outbound directions.

Randomizing the ISN of the protected host prevents an attacker from predicting the next ISN for a new connection and potentially hijacking the new session.

You can disable TCP initial sequence number randomization if necessary, for example, because data is getting scrambled. For example:

- If another in-line firewall is also randomizing the initial sequence numbers, there is no need for both firewalls to be performing this action, even though this action does not affect the traffic.
- If you use eBGP multi-hop through the ASA, and the eBGP peers are using MD5. Randomization breaks the MD5 checksum.
- You use a WAAS device that requires the ASA not to randomize the sequence numbers of connections.

Procedure

Step 1 Create an L3/L4 class map to identify the traffic whose TCP sequence numbers should not be randomized. The class match should be for TCP traffic; you can identify specific hosts (with an ACL) do a TCP port match, or simply match any traffic.

class-map *name*
match *parameter*

Example:

```
hostname(config)# access-list preserve-sq-no extended permit tcp any host 10.2.2.2
hostname(config)# class-map no-tcp-random
hostname(config-cmap)# match access-list preserve-sq-no
```

Step 2 Add or edit a policy map that sets the actions to take with the class map traffic, and identify the class map.

policy-map *name*
class *name*

Example:

```
hostname(config)# policy-map global_policy
hostname(config-pmap)# class preserve-sq-no
```

In the default configuration, the global_policy policy map is assigned globally to all interfaces. If you want to edit the global_policy, enter global_policy as the policy name. For the class map, specify the class you created earlier in this procedure.

Step 3 Disable TCP sequence number randomization on the class.

```
set connection random-sequence-number disable
```

If you later decide to turn it back on, replace “disable “with **enable**.

Step 4 If you are editing an existing service policy (such as the default global policy called global_policy), you are done. Otherwise, activate the policy map on one or more interfaces.

service-policy *policymap_name* {**global** | **interface** *interface_name*}

Example:

```
hostname(config)# service-policy global_policy global
```

The **global** keyword applies the policy map to all interfaces, and **interface** applies the policy to one interface. Only one global policy is allowed. You can override the global policy on an interface by applying a service policy to that interface. You can only apply one policy map to each interface.

Configure Connection Settings for Specific Traffic Classes (All Services)

You can configure different connection settings for specific traffic classes using service policies. Use service policies to:

- Customize connection limits and timeouts used to protect against DoS and SYN-flooding attacks.
- Implement Dead Connection Detection so that valid but idle connections remain alive.
- Disable TCP sequence number randomization in cases where you do not need it.
- Customize how the TCP Normalizer protects against abnormal TCP packets.
- Implement TCP State Bypass for traffic subject to asynchronous routing. Bypass traffic is not subject to inspection.
- Decrement time-to-live (TTL) on packets so that the ASA will show up on trace route output.

You can configure any combination of these settings for a given traffic class, except for TCP State Bypass and TCP Normalizer customization, which are mutually exclusive.

Tip This procedure shows a service policy for traffic that goes through the ASA. You can also configure the connection maximum and embryonic connection maximum for management (to the box) traffic.

Before You Begin

If you want to customize the TCP Normalizer, create the required TCP Map before proceeding.

The **set connection** command (for connection limits and sequence randomization) and **set connection timeout** commands are described here separately for each parameter. However, you can enter the commands on one line, and if you enter them separately, they are shown in the configuration as one command.

Procedure

Step 1 Create an L3/L4 class map to identify the traffic for which you want to customize connection settings.

```
class-map name
match parameter
```

Example:

```
hostname(config)# class-map CONNS
hostname(config-cmap)# match any
```

For information on matching statements, see Identify Traffic (Layer 3/4 Class Maps), page 1-13.

Step 2 Add or edit a policy map that sets the actions to take with the class map traffic, and identify the class map.

```
policy-map name
class name
```

Example:

```
hostname(config)# policy-map global_policy
hostname(config-pmap)# class CONNS
```

In the default configuration, the global_policy policy map is assigned globally to all interfaces. If you want to edit the global_policy, enter global_policy as the policy name. For the class map, specify the class you created earlier in this procedure.

Step 3 Set connection limits and TCP sequence number randomization. (TCP Intercept.)

- **set connection conn-max** n—The maximum number of simultaneous TCP or UDP connections that are allowed, between 0 and 2000000, for the entire class. The default is 0, which allows unlimited connections.
  - If two servers are configured to allow simultaneous TCP or UDP connections, the connection limit is applied to each configured server separately.
  - Because the limit is applied to a class, one attack host can consume all the connections and leave none for the rest of the hosts that are matched to the class.
- **set connection embryonic-conn-max** n—The maximum number of simultaneous embryonic connections allowed, between 0 and 2000000. The default is 0, which allows unlimited connections. By setting a non-zero limit, you enable TCP Intercept, which protects inside systems from a DoS attack perpetrated by flooding an interface with TCP SYN packets. Also set the per-client options to protect against SYN flooding.
- **set connection per-client-embryonic-max** n—The maximum number of simultaneous embryonic connections allowed per client, between 0 and 2000000. The default is 0, which allows unlimited connections.
- **set connection per-client-max** n—The maximum number of simultaneous connections allowed per client, between 0 and 2000000. The default is 0, which allows unlimited connections. This argument restricts the maximum number of simultaneous connections that are allowed for each host that is matched to the class.
- **set connection random-sequence-number** {**enable** | **disable**}—Whether to enable or disable TCP sequence number randomization. Randomization is enabled by default.

Example:

```
hostname(config-pmap-c)# set connection conn-max 256 random-sequence-number disable
```

Step 4 Set connection timeouts and Dead Connection Detection (DCD).

The defaults described below assume you have not changed the global defaults for these behaviors using the **timeout** command; the global defaults override the ones described here. Enter **0** to disable the timer, so that a connection never times out.

- **set connection timeout embryonic** *hh*:*mm*:*ss*—The timeout period until a TCP embryonic (half-open) connection is closed, between 0:0:5 and 1193:00:00. The default is 0:0:30.
- **set connection idle** *hh*:*mm*:*ss* [**reset**]—The idle timeout period after which an established connection of any protocol closes, between 0:0:1 and 1193:0:0. The default is 1:0:0. For TCP traffic, the **reset** keyword sends a reset to TCP endpoints when the connection times out.

  The default **udp** idle timeout is 2 minutes. The default **icmp** idle timeout is 2 seconds. The default **esp** and **ha** idle timeout is 30 seconds. For all other protocols, the default idle timeout is 2 minutes.
- **set connection half-closed** *hh*:*mm*:*ss*—The idle timeout period until a half-closed connection is closed, between 0:5:0 (for 9.1(1) and earlier) or 0:0:30 (for 9.1(2) and later) and 1193:0:0. The default is 0:10:0. Half-closed connections are not affected by DCD. Also, the ASA does not send a reset when taking down half-closed connections.
- **set connection dcd** [*retry-interval* [*max_retries*]]—Enable Dead Connection Detection (DCD). Before expiring an idle connection, the ASA probes the end hosts to determine if the connection is valid. If both hosts respond, the connection is preserved, otherwise the connection is freed.

  The *retry-interval* sets the time duration in *hh***:***mm***:***ss* format to wait after each unresponsive DCD probe before sending another probe, between 0:0:1 and 24:0:0. The default is 0:0:15. The *max-retries* sets the number of consecutive failed retries for DCD before declaring the connection as dead. The minimum value is 1 and the maximum value is 255. The default is 5.

Example:

```
hostname(config-pmap-c)# set connection timeout idle 2:0:0 embryonic 0:40:0
half-closed 0:20:0 dcd
```

Step 5 Decrement time-to-live (TTL) on packets that match the class.

set connection decrement-ttl

This command, along with the **icmp unreachable** command, is required to allow a traceroute through the ASA that shows the ASA as one of the hops.

Example:

```
hostname(config)# class-map global-policy
hostname(config-cmap)# match any
hostname(config-cmap)# exit
hostname(config)# policy-map global_policy
hostname(config-pmap)# class global-policy
hostname(config-pmap-c)# set connection decrement-ttl
hostname(config-pmap-c)# exit
hostname(config)# icmp unreachable rate-limit 50 burst-size 6
```

Step 6 Customize TCP Normalizer behavior by applying a TCP map.

set connection advanced-options *tcp-map-name*

Example:

```
hostname(config-pmap-c)# set connection advanced-options tcp_map1
```

Step 7 Implement TCP State Bypass.

set connection advanced-options tcp-state-bypass

Step 8 If you are editing an existing service policy (such as the default global policy called global_policy), you are done. Otherwise, activate the policy map on one or more interfaces.

service-policy *policymap_name* {**global** | **interface** *interface_name*}

Example:

```
hostname(config)# service-policy global_policy global
```

The **global** keyword applies the policy map to all interfaces, and **interface** applies the policy to one interface. Only one global policy is allowed. You can override the global policy on an interface by applying a service policy to that interface. You can only apply one policy map to each interface.

Examples

The following example sets the connection limits and timeouts for all traffic:

```
hostname(config)# class-map CONNS
hostname(config-cmap)# match any
hostname(config-cmap)# policy-map CONNS
hostname(config-pmap)# class CONNS
hostname(config-pmap-c)# set connection conn-max 1000 embryonic-conn-max 3000
hostname(config-pmap-c)# set connection timeout idle 2:0:0 embryonic 0:40:0
half-closed 0:20:0 dcd
hostname(config-pmap-c)# service-policy CONNS interface outside
```

You can enter **set connection** commands with multiple parameters or you can enter each parameter as a separate command. The ASA combines the commands into one line in the running configuration. For example, if you entered the following two commands in class configuration mode:

```
hostname(config-pmap-c)# set connection conn-max 600
hostname(config-pmap-c)# set connection embryonic-conn-max 50
```

The output of the **show running-config policy-map** command would display the result of the two commands in a single, combined command:

```
set connection conn-max 600 embryonic-conn-max 50
```

Monitoring Connections

You can use the following commands to monitor connections:

- **show conn**

  Shows connection information. The “b” flag indicates traffic subject to TCP State Bypass.

- **show service-policy**

  Shows service policy statistics, including Dead Connection Detection (DCD) statistics.

- **show threat-detection statistics top tcp-intercept** [**all** | **detail**]

  View the top 10 protected servers under attack. The **all** keyword shows the history data of all the traced servers. The **detail** keyword shows history sampling data. The ASA samples the number of attacks 30 times during the rate interval, so for the default 30 minute period, statistics are collected every 60 seconds.

History for Connection Settings

Feature Name	Platform Releases	Description
TCP state bypass	8.2(1)	This feature was introduced. The following command was introduced: **set connection advanced-options tcp-state-bypass**.
Connection timeout for all protocols	8.2(2)	The idle timeout was changed to apply to all protocols, not just TCP. The following command was modified: **set connection timeout**
Timeout for connections using a backup static route	8.2(5)/8.4(2)	When multiple static routes exist to a network with different metrics, the ASA uses the one with the best metric at the time of connection creation. If a better route becomes available, then this timeout lets connections be closed so a connection can be reestablished to use the better route. The default is 0 (the connection never times out). To take advantage of this feature, change the timeout to a new value. We modified the following command: **timeout floating-conn**.
Configurable timeout for PAT xlate	8.4(3)	When a PAT xlate times out (by default after 30 seconds), and the ASA reuses the port for a new translation, some upstream routers might reject the new connection because the previous connection might still be open on the upstream device. The PAT xlate timeout is now configurable, to a value between 30 seconds and 5 minutes. We introduced the following command: **timeout pat-xlate**. *This feature is not available in 8.5(1) or 8.6(1).*
Increased maximum connection limits for service policy rules	9.0(1)	The maximum number of connections for service policy rules was increased from 65535 to 2000000. We modified the following commands: **set connection conn-max**, **set connection embryonic-conn-max**, **set connection per-client-embryonic-max**, **set connection per-client-max**.
Decreased the half-closed timeout minimum value to 30 seconds	9.1(2)	The half-closed timeout minimum value for both the global timeout and connection timeout was lowered from 5 minutes to 30 seconds to provide better DoS protection. We modified the following commands: **set connection timeout half-closed**, **timeout half-closed**.