



CHAPTER 20

Configuring Advanced Connection Features

This chapter describes how to customize connection features, and includes the following sections:

- [Configuring Connection Limits and Timeouts, page 20-1](#)
- [Permitting or Denying Application Types with PISA Integration, page 20-4](#)
- [Configuring TCP State Bypass, page 20-10](#)
- [Disabling TCP Normalization, page 20-14](#)
- [Preventing IP Spoofing, page 20-14](#)
- [Configuring the Fragment Size, page 20-15](#)
- [Blocking Unwanted Connections, page 20-15](#)

Configuring Connection Limits and Timeouts

This section describes how to set maximum TCP and UDP connections, the maximum connection rate, connection timeouts, and how to disable TCP sequence randomization.

Each TCP connection has two ISNs: one generated by the client and one generated by the server. The FWSM 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.

TCP initial sequence number randomization can be disabled if required. 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 FWSM, and the eBGP peers are using MD5. Randomization breaks the MD5 checksum.
- You use a WAAS device that requires the FWSM not to randomize the sequence numbers of connections.



Note

Because of the way TCP sequence randomization is implemented, if you enable Xlate Bypass (see the [“Configuring Xlate Bypass” section on page 15-18](#)), then disabling TCP sequence randomization only works for control connections, and not data connections; for data connections, the TCP sequence continues to be randomized.

You can also configure maximum connections and TCP sequence randomization in the NAT

configuration. If you configure these settings for the same traffic using both methods, then the FWSM uses the lower limit. For TCP sequence randomization, if it is disabled using either method, then the FWSM disables TCP sequence randomization.

NAT also lets you configure embryonic connection limits, which triggers TCP Intercept to prevent a DoS attack. To configure connection limits, TCP randomization, and embryonic limits, see [Chapter 15, “Configuring NAT.”](#)

To set connection limits and timeouts, perform the following steps:

- Step 1** To identify the traffic, add a class map using the **class-map** command. See the “[Identifying Traffic \(Layer 3/4 Class Map\)](#)” section on page 19-4 for more information.

For example, you can match all traffic using the following commands:

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

To match specific traffic, you can match an access list:

```
hostname(config)# access list CONNS extended permit ip any 10.1.1.1 255.255.255.255
hostname(config)# class-map CONNS
hostname(config-cmap)# match access-list CONNS
```



Note In 3.x, when you used the **set connection** command for an access list (**match access-list**), then connection settings were applied to each individual ACE; in 4.0, connection settings are applied to the access list as a whole.

- Step 2** To add or edit a policy map that sets the actions to take with the class map traffic, enter the following commands:

```
hostname(config)# policy-map name
hostname(config-pmap)# class class_map_name
hostname(config-pmap-c)#
```

where the *class_map_name* is the class map from [Step 1](#).

For example:

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

- Step 3** To set maximum connection limits, connection rate limit, or whether TCP sequence randomization is enabled, enter the following command:

```
hostname(config-pmap-c)# set connection {[conn-max n] [conn-rate-limit n]
[random-sequence-number {enable | disable}]}
```

where the **conn-max** *n* argument sets the maximum number of simultaneous TCP and/or UDP connections that are allowed, between 0 and 65535. The default is 0, which means no limit on connections.

The **conn-rate-limit** *n* argument sets the maximum TCP and/or UDP connections per second between 0 and 65535. The default is 0, which means no limit on the connection rate.

The **random-sequence-number** {**enable** | **disable**} keyword enables or disables TCP sequence number randomization.

You can enter this command all on one line (in any order), or you can enter each attribute as a separate command. The FWSM combines the command into one line in the running configuration.

- Step 4** To set the timeout for TCP embryonic connections (half-opened) or TCP half-closed connections, enter the following command:

```
hostname(config-pmap-c)# set connection timeout {[embryonic hh:mm:ss] [half-closed hh:mm:ss]}
```

where the **embryonic** *hh:mm:ss* keyword sets the timeout period until a TCP embryonic (half-open) connection is closed, between 0:0:1 and 0:4:15. The default is 0:0:20. You can also set this value to 0, which means the connection never times out.

The **half-closed** *hh:mm:ss* keyword sets the idle timeout between 0:0:1 and 0:4:15. The default is 0:0:20. You can also set this value to 0, which means the connection never times out. The FWSM does not send a reset when taking down half-closed connections.

You can enter this command all on one line (in any order), or you can enter each attribute as a separate command. The command is combined onto one line in the running configuration.



Note This command does not affect secondary connections created by an inspection engine. For example, you cannot change the connection settings for secondary flows like SQL*Net, FTP data flows, and so on using the **set connection timeout** command. For these connections, use the global **timeout conn** command to change the idle time. Note that the **timeout conn** command affects *all* traffic flows unless you otherwise use the **set connection timeout** command for eligible traffic.

- Step 5** To set the timeout for idle connections for all protocols, enter the following command:

```
hostname(config-pmap-c)# set connection timeout idle hh:mm:0
```

where the **idle** *hh:mm:0* argument defines the idle time after which an established connection of any protocol closes, between 0:5:0 and 1092:15:0. The default is 0:60:0. You can also set the value to 0, which means the connection never times out.



Note This command ignores the value you set for seconds; you can only specify the hours and minutes. Therefore, you should set the seconds to be 0.

The **idle** keyword has replaced the **tcp** keyword in the **set connection timeout** command, but if your configuration includes the **tcp** command (for TCP connections only), it is still accepted. If your policy includes both the **idle** and **tcp** commands, then the **tcp** command takes precedence for TCP traffic only if the class map matches an access list that specifies TCP traffic explicitly. See the **set connection timeout** command in the *Catalyst 6500 Series Switch and Cisco 7600 Series Router Firewall Services Module Command Reference* for more information.

This command does not affect secondary connections created by an inspection engine. For example, you cannot change the connection settings for secondary flows like SQL*Net, FTP data flows, and so on using the **set connection timeout** command. For these connections, use the global **timeout conn** command to change the idle time. Note that the **timeout conn** command affects *all* traffic flows unless you otherwise use the **set connection timeout** command for eligible traffic.

- Step 6** To activate the policy map on one or more interfaces, enter the following command:

```
hostname(config)# service-policy policymap_name {global | interface interface_name}
```

where *policy_map_name* is the policy map you configured in [Step 2](#). To apply the policy map to traffic on all the interfaces, use the **global** keyword. To apply the policy map to traffic on a specific interface, use the **interface** *interface_name* option, where *interface_name* is the name assigned to the interface with the **nameif** command.

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.

The following example sets the maximum TCP and UDP connections to 5000, the maximum connections per second to 500, and sets the maximum embryonic timeout to 40 seconds, the half-closed timeout to 20 minutes, and the idle timeout to 2 hours for traffic going to 10.1.1.1:

```
hostname(config)# access-list CONNS permit ip any host 10.1.1.1

hostname(config)# class-map conns
hostname(config-cmap)# match access-list CONNS

hostname(config-cmap)# policy-map conns
hostname(config-pmap)# class conns
hostname(config-pmap-c)# set connection conn-max 5000 conn-rate-limit 500
hostname(config-pmap-c)# set connection timeout embryonic 0:0:40 half-closed 0:20:0
hostname(config-pmap-c)# set connection timeout idle 2:0:0

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

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 timeout embryonic 0:0:40 half-closed 0:20:0
```

Permitting or Denying Application Types with PISA Integration



Note

This feature depends on Cisco IOS Release 12.2(18)ZYA or later, and is only available on the Catalyst 6500 switch.

The Programmable Intelligent Services Accelerator (PISA) on the switch supervisor can quickly determine the application type of a given flow by performing deep packet inspection. This determination can be made even if the traffic is not using standard ports. The FWSM can leverage the high-performance deep packet inspection of the PISA card so that it can permit or deny traffic based on the application type. Unlike the FWSM inspection feature, which passes through the control plane path, traffic that the PISA tags can pass through the FWSM accelerated path. Another benefit of FWSM and PISA integration is to consolidate your security configuration on a single FWSM instead of having to configure multiple upstream switches with PISAs installed.

You might want to deny certain types of application traffic when you want to preserve bandwidth for critical application types. For example, you might deny the use of peer-to-peer (P2P) applications if they are affecting your other critical applications.

This section includes the following topics:

- [PISA Integration Overview, page 20-5](#)
- [Configuring the FWSM to Deny PISA Traffic, page 20-6](#)
- [Configuring the Switch for PISA/FWSM Integration, page 20-7](#)
- [Monitoring PISA Connections, page 20-10](#)

PISA Integration Overview

This section describes how the PISA works with the FWSM, and includes the following topics:

- [PISA Integration Guidelines and Limitations, page 20-5](#)
- [Using GRE for Tagging, page 20-5](#)
- [Failover Support, page 20-6](#)

PISA Integration Guidelines and Limitations

The following guidelines and limitations apply to PISA integration:

- The PISA and the FWSM cannot be in the same switch chassis. You can, however, use multiple PISAs upstream and downstream of the FWSM if desired.
- There is a slight performance impact on the PISA for traffic sent to the FWSM, due to the need to tag the packets for the FWSM (see the [“Using GRE for Tagging”](#) section.)
- When a UDP packet is denied due to the FWSM service policy, the corresponding session is not immediately deleted. Instead, it is allowed to time out, and the packets that hit this session in the meantime are dropped.
- It is possible for an end-user application to use the special GRE key that is used between the FWSM and the PISA. In such instances, the PISA generates a syslog message and drops these packets.
- The PISA takes several packets to determine the application type; therefore a session starts to be established on the FWSM before the PISA tagging commences. When the PISA tagging commences, the FWSM security policy is then applied, and if the policy is to deny the flow, the session is prevented from completing.
- For fragmented packets, the PISA tags the first fragment, and the FWSM reassembles the packet and acts upon it based on the encapsulation included in the first fragment.

See also the [“PISA Limitations and Restrictions”](#) section on page 20-7.

Using GRE for Tagging

After the PISA identifies the application used by a given traffic flow, it encapsulates all packets using GRE and includes a tag informing the FWSM of the application type. In addition, an outer IP header almost identical (except for the Layer 4 protocol, which now indicates GRE) to the inner/original IP header is added. The original Layer 2 header is maintained. This preserves the original routing/switching paths for the modified packet. The GRE encapsulation adds 32 bytes (20 bytes for the outer IP header and 12 bytes for the GRE header).

After the FWSM receives the packet and acts on the information, it strips the GRE encapsulation from the packet.

When you configure the FWSM to deny traffic based on the PISA encapsulation, for the VLAN on which that traffic resides, the PISA encapsulates all traffic (including traffic that you did not specify for denial).

The GRE encapsulation increases the packet size slightly, so you should increase the MTU between the PISA and the FWSM according to the [“Changing the MTU on the Switch to Support Longer Packet Length” section on page 20-8](#).

The GRE encapsulation causes a slight performance impact for PISA traffic sent to the FWSM.

Failover Support

Failover of the PISA is independent of failover of the FWSM. If you have Stateful Failover on the FWSM, then the session information is maintained across the failover.

Configuring the FWSM to Deny PISA Traffic

To identify traffic that you want to deny using PISA tagging, perform the following steps:

- Step 1** To identify the traffic that you want to deny based on the application type, add a class map using the **class-map** command. See the [“Identifying Traffic \(Layer 3/4 Class Map\)” section on page 19-4](#) for more information.

For example, you can match an access list:

```
hostname(config)# access list BAD_APPS extended permit any 10.1.1.1 255.255.255.255
hostname(config)# class-map denied_apps
hostname(config-cmap)# match access-list BAD_APPS
```

- Step 2** To add or edit a policy map that sets the actions to take with the class map traffic, enter the following commands:

```
hostname(config)# policy-map name
hostname(config-pmap)# class class_map_name
hostname(config-pmap-c)#
```

where the *class_map_name* is the class map from [Step 1](#).

For example:

```
hostname(config)# policy-map denied_apps_policy
hostname(config-pmap)# class denied_apps
hostname(config-pmap-c)#
```

- Step 3** Determine which applications are permitted or denied by entering the following commands:

```
hostname(config-pmap-c)# deny {all | protocol}
hostname(config-pmap-c)# permit protocol
```

Where the *protocol* argument is the protocol name or number. To see the supported protocol names, use the **permit ?** or **deny ?** command.

You can combine **permit** and **deny** statements to narrow the traffic that you want denied. You must enter at least one **deny** statement. Unlike access lists, which have an implicit deny at the end, PISA actions have an implicit permit at the end.

For example, to permit all traffic except for Skype, eDonkey, and Yahoo, enter the following commands:

```
hostname(config-pmap-c) # deny skype
hostname(config-pmap-c) # deny yahoo
hostname(config-pmap-c) # deny eDonkey
```

The following example denies all traffic except for Kazaa and eDonkey:

```
hostname(config-pmap-c) # deny all
hostname(config-pmap-c) # permit kazaa
hostname(config-pmap-c) # permit eDonkey
```



Note For a class map with the **permit** and **deny** commands, you cannot also include any **inspect** commands.

Step 4 Activate the policy map on one or more interfaces by entering the following command:

```
hostname(config) # service-policy polycymap_name {global | interface interface_name}
```

Where **global** 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.

The following is an example configuration for PISA integration:

```
hostname(config) # access-list BAD_APPS extended permit 10.1.1.0 255.255.255.0 10.2.1.0
255.255.255.0

hostname(config) # class-map denied_apps
hostname(config-cmap) # description "Apps to be blocked"
hostname(config-cmap) # match access-list BAD_APPS

hostname(config-cmap) # policy-map denied_apps_policy
hostname(config-pmap) # class denied_apps
hostname(config-pmap-c) # deny skype
hostname(config-pmap-c) # deny yahoo
hostname(config-pmap-c) # deny eDonkey

hostname(config-pmap-c) # service-policy denied_apps_policy inside
```

Configuring the Switch for PISA/FWSM Integration

This section describes how to configure the switch for PISA/FWSM integration and includes the following topics:

- [PISA Limitations and Restrictions, page 20-7](#)
- [Changing the MTU on the Switch to Support Longer Packet Length, page 20-8](#)
- [Configuring Classification on the PISA, page 20-8](#)
- [Configuring Tagging on the PISA, page 20-8](#)
- [Sample Switch Configurations for PISA Integration, page 20-9](#)

PISA Limitations and Restrictions

The following limitations and restrictions apply to the PISA:

- Network Based Application Recognition (NBAR) does not work on Layer 3 EtherChannels. Layer 2 EtherChannels are supported.
- The RP on the PISA does not support protocol tagging. So any packets going to the FWSM from the RP will not be tagged.
- NBAR implementation does not support IPv6. So protocol discovery and tagging are only applicable to IPv4. In addition to this restriction imposed by NBAR, the underlying PISA infrastructure also does not support acceleration of IPv6 packets.
- Currently there is a caveat in the L2 PISA implementation for VLANs that have been PISA-accelerated on an Layer 2 port (for example, a trunk); the SVI interfaces for VLANs passing through the accelerated Layer 2 port cannot be in an up state (they will become admin down).
- Multi-VLAN access ports are not supported.

See also the “[PISA Integration Guidelines and Limitations](#)” section on page 20-5.

Changing the MTU on the Switch to Support Longer Packet Length

Because of the GRE encapsulation, you should increase the MTU size on VLANs used between the PISA and the FWSM. The GRE encapsulation adds 32 bytes (20 bytes for the outer IP header and 12 bytes for the GRE header).

- To change the MTU on a routed switch port or a Layer 3 interface (SVI), enter the following command:

```
Router(config-if)# mtu mtu_size
```

For an SVI, the *mtu_size* is between 64 and 9216 bytes. For a routed switch port, the *mtu_size* is between 1500 and 9216 bytes. The default MTU size is 1500 bytes.

- To configure the global LAN port MTU size for Layer 2 ports, enter the following command:

```
Router(config)# system jumbo mtu mtu_size
```

The *mtu_size* can be between 1500 and 9216 bytes. The default size is 9216 bytes.

Configuring Classification on the PISA

- To enable classification on a Layer 2 switch port (access, trunk or EtherChannel configured on a physical port) or a Layer 3 interface (SVI, routed port, or subinterface), enter the following command in interface configuration mode.

```
Router(config-if)# ip nbar protocol-discovery
```

- To show protocol discovery statistics on a Layer 2 or Layer 3 interface, enter the following command:

```
Router# show ip nbar protocol-discovery interface ifname
```

Configuring Tagging on the PISA

After protocol discovery is enabled, enable egress packet tagging by entering the following commands.



Note

Classification and tagging need to be enabled on the same port; for example, you cannot enable classification on access ports and tagging on a trunk port.

- To enable tagging on a switch port (access port) or a Layer 3 interface (SVI, routed port, or subinterface), enter the following command in interface configuration mode:

```
Router(config-if)# ip nbar protocol-tagging
```

- To enable tagging on a trunk port, enter the following command in interface configuration mode:

```
Router(config-if)# ip nbar protocol-tagging [vlan-list vlan-list]
```

Where the **vlan-list** *vlan-list* argument specifies a list of VLANs to be tagged. If not specified, all active VLANs are tagged.

The following commands help you monitor the tagging by the PISA:

- The following command displays tagging configuration information:

```
Router# show ip nbar protocol-tagging {key | interface ifname | summary}
```

Where the **key** keyword shows the GRE key used for the tagging.

The **interface** *ifname* argument shows if tagging is enabled on an interface.

The **summary** keyword shows all interfaces with tagging enabled.

- The following command shows the mapping of protocol name to ID:

```
Router# show ip nbar protocol-id [protocol_name]
```

If you enter the *protocol_name*, the mapped ID is shown. When omitted, the complete list of protocol names and IDs is shown.

- To show the number of packets tagged on the PISA, enter the following command:

```
Router# show platform pisa np tx counters
```

For example:

```
Router# show platform pisa np tx counters
```

```
TX Statistics(ME1)
-----
Errors: 0
.....
TX NBAR Protocol tagged pkt: 9869
```

Sample Switch Configurations for PISA Integration

Example 20-1 Layer 3 Mode (Interface-based, Routed port/SVI)

```
Router(config)# interface vlan 100
Router(config-if)# ip nbar protocol-discovery
! enables discovery
Router(config-if)# ip nbar protocol-tagging
! enables tagging
Router(config-if)# mtu 9216
! Allows packet sizes up to 9216 bytes without fragmenting
```

Example 20-2 Layer 2 Mode (Interface-based, Protocol Discovery on Uplink Ports)

```
Router(config)# interface gigabitethernet 6/1
Router(config-if)# ip nbar protocol-discovery
! Classification
```

```

Router(config-if)# ip nbar protocol-tagging vlan-list 100
! Tagging
Router(config-if)# mtu 9216
! Allow packet size up to 9216 bytes without fragmenting
Router(config)# system jumbomtu 9216
! Set global LAN port MTU to 9216 bytes

```

Monitoring PISA Connections

This section includes the following topics:

- [Syslog Message for Dropped Connections, page 20-10](#)
- [Viewing PISA Connections on the FWSM, page 20-10](#)

Syslog Message for Dropped Connections

Syslog message 302014 (for TCP) and 302016 (for UDP) display when a PISA connection is denied. For example:

```
%FWSM-6-302014: Teardown TCP connection 144547133155839947 for inside:10.1.1.12/33407 to
outside:209.165.201.10/21 duration 0:00:00 bytes 160 PISA denied protocol
```

Viewing PISA Connections on the FWSM

To monitor connections from the PISA, use the **show conn** command. Connections that are tagged by the PISA are listed in the output with the “p” flag. The following is sample output from the **show conn** command:

```

hostname# show conn
2 in use, 3 most used
  Network Processor 1 connections
TCP out 10.1.1.10:21 in 209.165.201.12:33406 idle 0:00:04 Bytes 1668 FLAGS - UOIp
  Network Processor 2 connections
UDP out 10.1.1.255:137 in 10.1.1.11:137 idle 0:00:48 Bytes 288 FLAGS -
Multicast sessions:
  Network Processor 1 connections
  Network Processor 2 connections
IPv6 connections:
...

```

Configuring TCP State Bypass

This section describes how to configure TCP state bypass, and includes the following topics:

- [TCP State Bypass Overview, page 20-11](#)
- [Enabling TCP State Bypass, page 20-13](#)

TCP State Bypass Overview

This section describes how to use TCP state bypass, and includes the following topics:

- [Allowing Outbound and Inbound Flows through Separate FWSMs, page 20-11](#)
- [Unsupported Features, page 20-12](#)
- [Compatibility with NAT, page 20-12](#)
- [Connection Timeout, page 20-13](#)

Allowing Outbound and Inbound Flows through Separate FWSMs

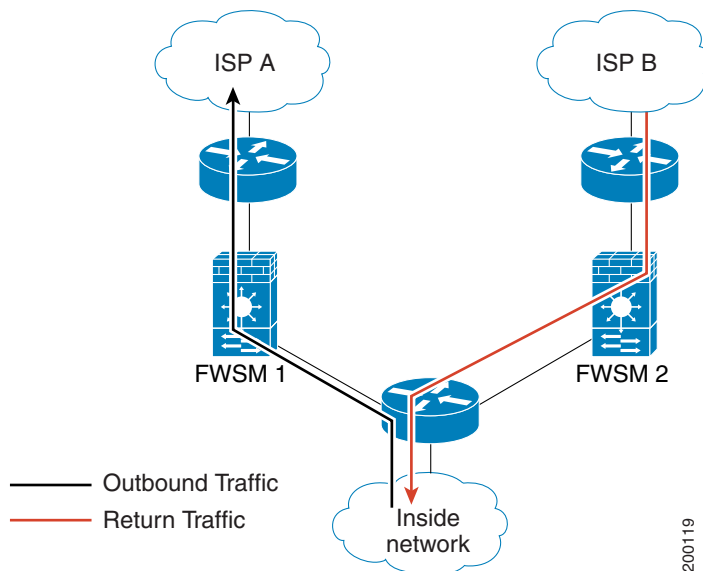
By default, all traffic that goes through the FWSM is inspected using the Adaptive Security Algorithm and is either allowed through or dropped based on the security policy. The FWSM maximizes the firewall performance by checking the state of each packet (is this a new connection or an established connection?) and assigning it to either the session management path (a new connection SYN packet), the accelerated path (an established connection), or the control plane path (advanced inspection). See the [“Stateful Inspection Overview” section on page 1-9](#) for more detailed information about the stateful firewall.

TCP packets that match existing connections in the accelerated path can pass through the FWSM without rechecking every aspect of the security policy. This feature maximizes performance. However, the method of establishing the session in the accelerated path using the SYN packet, and the checks that occur in the accelerated 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 FWSM.

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

through the session management path, then there is no entry in the accelerated path for the connection, and the packets are dropped. [Figure 20-1](#) shows an asymmetric routing example where the outbound traffic goes through a different FWSM than the inbound traffic:

Figure 20-1 Asymmetric Routing



If you have asymmetric routing configured on upstream routers, and traffic alternates between two FWSMs, then you can configure TCP state bypass for specific traffic. TCP state bypass alters the way sessions are established in the accelerated path and disables the accelerated 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 FWSM, and there is not an accelerated path entry, then the packet goes through the session management path to establish the connection in the accelerated path. Once in the accelerated path, the traffic bypasses the accelerated path checks.

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 FWSM, so application inspection is not supported with TCP state bypass.
- AAA authenticated sessions—When a user authenticates with one FWSM, traffic returning via the other FWSM will be denied because the user did not authenticate with that FWSM.

Compatibility with NAT

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

Connection Timeout

If there is no traffic on a given connection for 2 minutes, the connection times out. You can override this default using the **set connection timeout tcp** command. Normal TCP connections timeout by default after 60 minutes.

Enabling TCP State Bypass

To enable TCP state bypass, perform the following steps:

- Step 1** To identify the traffic for which you want to disable stateful firewall inspection, add a class map using the **class-map** command. See the “[Identifying Traffic \(Layer 3/4 Class Map\)](#)” section on page 19-4 for more information.

For example, you can match an access list:

```
hostname(config)# access list bypass extended permit tcp any 10.1.1.1 255.255.255.255
hostname(config)# class-map bypass_traffic
hostname(config-cmap)# match access-list bypass
```

- Step 2** To add or edit a policy map that sets the actions to take with the class map traffic, enter the following commands:

```
hostname(config)# policy-map name
hostname(config-pmap)# class class_map_name
hostname(config-pmap-c)#
```

where the *class_map_name* is the class map from [Step 1](#).

For example:

```
hostname(config)# policy-map tcp_bypass_policy
hostname(config-pmap)# class bypass_traffic
hostname(config-pmap-c)#
```

- Step 3** Enable TCP state bypass by entering the following command:

```
hostname(config-pmap-c)# set connection advanced-options tcp-state-bypass
```

- Step 4** Activate the policy map on one or more interfaces by entering the following command:

```
hostname(config)# service-policy policymap_name {global | interface interface_name}
```

Where **global** 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.



Note

If you use the **show conn** command, the display for connections that use TCP state bypass includes the flag “b.”

The following is an example configuration for TCP state bypass:

```
hostname(config)# access-list tcp_bypass extended permit tcp 10.1.1.0 255.255.255.0
10.2.1.0 255.255.255.0

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

```

Disabling TCP Normalization

For traffic that passes through the control-plane path, such as packets that require Layer 7 inspection or management traffic, the FWSM sets the maximum number of out-of-order packets that can be queued for a TCP connection to 2 packets, which is not user-configurable. Other TCP normalization features that are supported on the PIX and ASA platforms are not enabled for FWSM. You can disable the limited TCP normalization support for the FWSM using the **no control-point tcp-normalizer** command.

Preventing IP Spoofing

This section lets you enable Unicast Reverse Path Forwarding on an interface. Unicast RPF guards against IP spoofing (a packet uses an incorrect source IP address to obscure its true source) by ensuring that all packets have a source IP address that matches the correct source interface according to the routing table.

Normally, the FWSM only looks at the destination address when determining where to forward the packet. Unicast RPF instructs the FWSM to also look at the source address; this is why it is called Reverse Path Forwarding. For any traffic that you want to allow through the FWSM, the FWSM routing table must include a route back to the source address. See RFC 2267 for more information.

For outside traffic, for example, the FWSM can use the default route to satisfy the Unicast RPF protection. If traffic enters from an outside interface, and the source address is not known to the routing table, the FWSM uses the default route to correctly identify the outside interface as the source interface.

If traffic enters the outside interface from an address that is known to the routing table, but is associated with the inside interface, then the FWSM drops the packet. Similarly, if traffic enters the inside interface from an unknown source address, the FWSM drops the packet because the matching route (the default route) indicates the outside interface.

Unicast RPF is implemented as follows:

- ICMP packets have no session, so each packet is checked.
- UDP and TCP have sessions, so the initial packet requires a reverse route lookup. Subsequent packets arriving during the session are checked using an existing state maintained as part of the session. Non-initial packets are checked to ensure they arrived on the same interface used by the initial packet.

To enable Unicast RPF, enter the following command:

```
hostname(config)# ip verify reverse-path interface interface_name
```

Configuring the Fragment Size

By default, the FWSM allows up to 24 fragments per IP packet, and up to 200 fragments awaiting reassembly. You might need to let fragments on your network if you have an application that routinely fragments packets, such as NFS over UDP. However, if you do not have an application that fragments traffic, we recommend that you do not allow fragments through the FWSM. Fragmented packets are often used as DoS attacks. To set disallow fragments, enter the following command:

```
hostname(config)# fragment chain 1 [interface_name]
```

Enter an interface name if you want to prevent fragmentation on a specific interface. By default, this command applies to all interfaces.

Blocking Unwanted Connections

If you know that a host is attempting to attack your network (for example, system log messages show an attack), then you can block (or shun) connections based on the source IP address and other identifying parameters. No new connections can be made until you remove the shun.



Note

If you have an IPS that monitors traffic, then the IPS can shun connections automatically.

To shun a connection manually, perform the following steps:

Step 1 If necessary, view information about the connection by entering the following command:

```
hostname# show conn
```

The FWSM shows information about each connection, such as the following:

```
TCP out 64.101.68.161:4300 in 10.86.194.60:23 idle 0:00:00 bytes 1297 flags UIO
```

Step 2 To shun connections from the source IP address, enter the following command:

```
hostname(config)# shun src_ip [dst_ip src_port dest_port [protocol]] [vlan vlan_id]
```

This command drops an existing connection, as well as blocking future connections. By default, the protocol is 0 for IP.

For multiple context mode, you can enter this command in the admin context, and by specifying a VLAN ID that is assigned to an interface in other contexts, you can shun the connection in other contexts.

Step 3 To remove the shun, enter the following command:

```
hostname(config)# no shun src_ip [vlan vlan_id]
```

