



CHAPTER 43

Configuring Inspection of Basic Internet Protocols

This chapter describes how to configure application layer protocol inspection. Inspection engines are required for services that embed IP addressing information in the user data packet or that open secondary channels on dynamically assigned ports. These protocols require the ASA to do a deep packet inspection instead of passing the packet through the fast path. As a result, inspection engines can affect overall throughput.

Several common inspection engines are enabled on the ASA by default, but you might need to enable others depending on your network.

This chapter includes the following sections:

- [DNS Inspection, page 43-1](#)
- [FTP Inspection, page 43-11](#)
- [HTTP Inspection, page 43-16](#)
- [ICMP Inspection, page 43-20](#)
- [ICMP Error Inspection, page 43-21](#)
- [Instant Messaging Inspection, page 43-21](#)
- [IP Options Inspection, page 43-24](#)
- [IPsec Pass Through Inspection, page 43-26](#)
- [IPv6 Inspection, page 43-27](#)
- [NetBIOS Inspection, page 43-28](#)
- [PPTP Inspection, page 43-30](#)
- [SMTP and Extended SMTP Inspection, page 43-31](#)
- [TFTP Inspection, page 43-34](#)

DNS Inspection

This section describes DNS application inspection. This section includes the following topics:

- [How DNS Application Inspection Works, page 43-2](#)
- [How DNS Rewrite Works, page 43-2](#)
- [Configuring DNS Rewrite, page 43-3](#)

- [Configuring a DNS Inspection Policy Map for Additional Inspection Control, page 43-7](#)
- [Verifying and Monitoring DNS Inspection, page 43-10](#)

How DNS Application Inspection Works

The ASA tears down the DNS session associated with a DNS query as soon as the DNS reply is forwarded by the ASA. The ASA also monitors the message exchange to ensure that the ID of the DNS reply matches the ID of the DNS query.

When DNS inspection is enabled, which is the default, the ASA performs the following additional tasks:

- Translates the DNS record based on the configuration completed using the **alias**, **static** and **nat** commands (DNS Rewrite). Translation only applies to the A-record in the DNS reply; therefore, DNS Rewrite does not affect reverse lookups, which request the PTR record.



Note DNS Rewrite is not applicable for PAT because multiple PAT rules are applicable for each A-record and the PAT rule to use is ambiguous.

- Enforces the maximum DNS message length (the default is 512 bytes and the maximum length is 65535 bytes). The ASA performs reassembly as needed to verify that the packet length is less than the maximum length configured. The ASA drops the packet if it exceeds the maximum length.



Note If you enter the **inspect dns** command without the **maximum-length** option, DNS packet size is not checked

- Enforces a domain-name length of 255 bytes and a label length of 63 bytes.
- Verifies the integrity of the domain-name referred to by the pointer if compression pointers are encountered in the DNS message.
- Checks to see if a compression pointer loop exists.

A single connection is created for multiple DNS sessions, as long as they are between the same two hosts, and the sessions have the same 5-tuple (source/destination IP address, source/destination port, and protocol). DNS identification is tracked by *app_id*, and the idle timer for each *app_id* runs independently.

Because the *app_id* expires independently, a legitimate DNS response can only pass through the ASA within a limited period of time and there is no resource build-up. However, if you enter the **show conn** command, you will see the idle timer of a DNS connection being reset by a new DNS session. This is due to the nature of the shared DNS connection and is by design.

How DNS Rewrite Works

When DNS inspection is enabled, DNS rewrite provides full support for NAT of DNS messages originating from any interface.

If a client on an inside network requests DNS resolution of an inside address from a DNS server on an outside interface, the DNS A-record is translated correctly. If the DNS inspection engine is disabled, the A-record is not translated.

As long as DNS inspection remains enabled, you can configure DNS rewrite using the **alias**, **static**, or **nat** commands.

For details about the configuration required see the “[Configuring DNS Rewrite](#)” section on page 43-3.

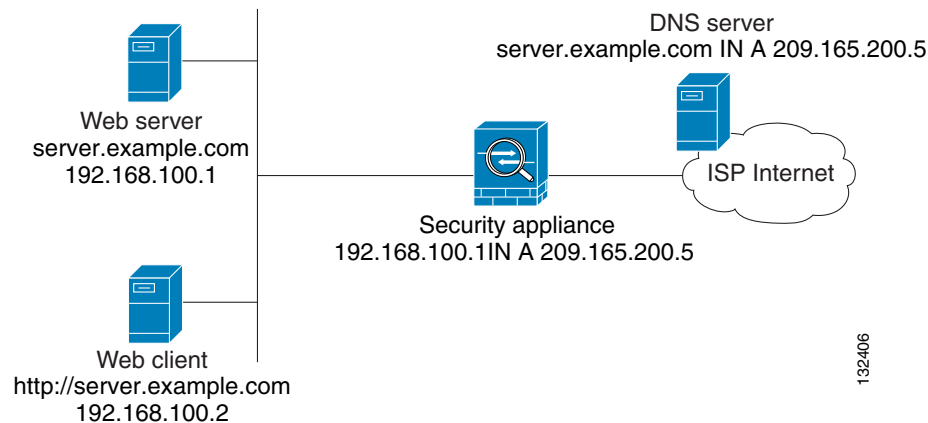
DNS Rewrite performs two functions:

- Translating a public address (the routable or “mapped” address) in a DNS reply to a private address (the “real” address) when the DNS client is on a private interface.
- Translating a private address to a public address when the DNS client is on the public interface.

In [Figure 43-1](#), the DNS server resides on the external (ISP) network. The real address of the server (192.168.100.1) has been mapped using the **static** command to the ISP-assigned address (209.165.200.5). When a web client on the inside interface attempts to access the web server with the URL `http://server.example.com`, the host running the web client sends a DNS request to the DNS server to resolve the IP address of the web server. The ASA translates the non-routable source address in the IP header and forwards the request to the ISP network on its outside interface. When the DNS reply is returned, the ASA applies address translation not only to the destination address, but also to the embedded IP address of the web server, which is contained in the A-record in the DNS reply. As a result, the web client on the inside network gets the correct address for connecting to the web server on the inside network.

For configuration instructions for scenarios similar to this one, see the “[Configuring DNS Rewrite with Two NAT Zones](#)” section on page 43-4.

Figure 43-1 Translating the Address in a DNS Reply (DNS Rewrite)



DNS rewrite also works if the client making the DNS request is on a DMZ network and the DNS server is on an inside interface. For an illustration and configuration instructions for this scenario, see the “[Overview of DNS Rewrite with Three NAT Zones](#)” section on page 43-4.

Configuring DNS Rewrite

You configure DNS rewrite using the NAT configuration.

This section includes the following topics:

- [Configuring DNS Rewrite with Two NAT Zones](#), page 43-4
- [Overview of DNS Rewrite with Three NAT Zones](#), page 43-4
- [Configuring DNS Rewrite with Three NAT Zones](#), page 43-6

Configuring DNS Rewrite with Two NAT Zones

To implement a DNS Rewrite scenario similar to the one shown in [Figure 43-1](#), perform the following steps:

-
- Step 1** Create a static translation for the web server using the `dns` option. See [Chapter 30, “Configuring Network Object NAT.”](#)
- Step 2** Create an access list that permits traffic to the port that the web server listens to for HTTP requests.
- ```
hostname(config)# access-list acl-name extended permit tcp any host mapped-address eq port
```
- where the arguments are as follows:
- acl-name*—The name you give the access list.
- mapped-address*—The translated IP address of the web server.
- port*—The TCP port that the web server listens to for HTTP requests.
- Step 3** Apply the access list created in [Step 2](#) to the mapped interface. To do so, use the `access-group` command, as follows:
- ```
hostname(config)# access-group acl-name in interface mapped_ifc
```
- Step 4** If DNS inspection is disabled or if you want to change the maximum DNS packet length, configure DNS inspection. DNS application inspection is enabled by default with a maximum DNS packet length of 512 bytes. For configuration instructions, see the [“Configuring a DNS Inspection Policy Map for Additional Inspection Control”](#) section on page 43-7.
- Step 5** On the public DNS server, add an A-record for the web server, such as:
- ```
domain-qualified-hostname. IN A mapped-address
```

where *domain-qualified-hostname* is the hostname with a domain suffix, as in `server.example.com`. The period after the hostname is important. *mapped-address* is the translated IP address of the web server.

---

The following example configures the ASA for the scenario shown in [Figure 43-1](#). It assumes DNS inspection is already enabled.

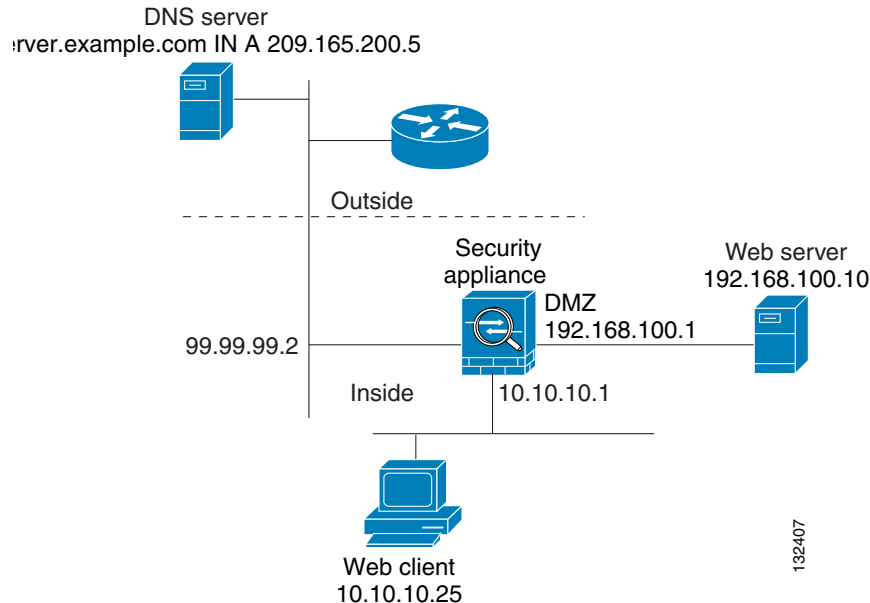
```
hostname(config)# object network obj-192.168.100.1-01
hostname(config-network-object)# host 192.168.100.1
hostname(config-network-object)# nat (inside,outside) static 209.165.200.225 dns
hostname(config)# access-list 101 permit tcp any host 209.165.200.225 eq www
hostname(config)# access-group 101 in interface outside
```

This configuration requires the following A-record on the DNS server:

```
server.example.com. IN A 209.165.200.225
```

## Overview of DNS Rewrite with Three NAT Zones

[Figure 43-2](#) provides a more complex scenario to illustrate how DNS inspection allows NAT to operate transparently with a DNS server with minimal configuration. For configuration instructions for scenarios like this one, see the [“Configuring DNS Rewrite with Three NAT Zones”](#) section on page 43-6.

**Figure 43-2 DNS Rewrite with Three NAT Zones**

In [Figure 43-2](#), a web server, server.example.com, has the real address 192.168.100.10 on the DMZ interface of the ASA. A web client with the IP address 10.10.10.25 is on the inside interface and a public DNS server is on the outside interface. The site NAT policies are as follows:

- The outside DNS server holds the authoritative address record for server.example.com.
- Hosts on the outside network can contact the web server with the domain name server.example.com through the outside DNS server or with the IP address 209.165.200.5.
- Clients on the inside network can access the web server with the domain name server.example.com through the outside DNS server or with the IP address 192.168.100.10.

When a host or client on any interface accesses the DMZ web server, it queries the public DNS server for the A-record of server.example.com. The DNS server returns the A-record showing that server.example.com binds to address 209.165.200.5.

When a web client on the *outside* network attempts to access `http://server.example.com`, the sequence of events is as follows:

1. The host running the web client sends the DNS server a request for the IP address of server.example.com.
2. The DNS server responds with the IP address 209.165.200.225 in the reply.
3. The web client sends its HTTP request to 209.165.200.225.
4. The packet from the outside host reaches the ASA at the outside interface.
5. The static rule translates the address 209.165.200.225 to 192.168.100.10 and the ASA directs the packet to the web server on the DMZ.

When a web client on the *inside* network attempts to access `http://server.example.com`, the sequence of events is as follows:

1. The host running the web client sends the DNS server a request for the IP address of server.example.com.
2. The DNS server responds with the IP address 209.165.200.225 in the reply.

3. The ASA receives the DNS reply and submits it to the DNS application inspection engine.
4. The DNS application inspection engine does the following:

- a. Searches for any NAT rule to undo the translation of the embedded A-record address “[outside]:209.165.200.5”. In this example, it finds the following static configuration:

```
object network obj-192.168.100.10-01
 host 192.168.100.10
 nat (dmz,outside) static 209.165.200.5 dns
```

- b. Uses the static rule to rewrite the A-record as follows because the **dns** option is included:

```
[outside]:209.165.200.225 --> [dmz]:192.168.100.10
```




---

**Note** If the **dns** option were not included with the **nat** command, DNS Rewrite would not be performed and other processing for the packet continues.

---

- c. Searches for any NAT to translate the web server address, [dmz]:192.168.100.10, when communicating with the inside web client.

No NAT rule is applicable, so application inspection completes.

If a NAT rule (nat or static) were applicable, the **dns** option must also be specified. If the **dns** option were not specified, the A-record rewrite in step **b** would be reverted and other processing for the packet continues.

5. The ASA sends the HTTP request to server.example.com on the DMZ interface.

## Configuring DNS Rewrite with Three NAT Zones

To enable the NAT policies for the scenario in [Figure 43-2](#), perform the following steps:

- Step 1** Create a static translation for the web server on the DMZ network using the **dns** option. See [Chapter 30, “Configuring Network Object NAT.”](#)
- Step 2** Create an access list that permits traffic to the port that the web server listens to for HTTP requests.
 

```
hostname(config)# access-list acl-name extended permit tcp any host mapped-address eq port
```

 where the arguments are as follows:
  - acl-name*—The name you give the access list.
  - mapped-address*—The translated IP address of the web server.
  - port*—The TCP port that the web server listens to for HTTP requests.
- Step 3** Apply the access list created in [Step 2](#) to the outside interface. To do so, use the **access-group** command, as follows:
 

```
hostname(config)# access-group acl-name in interface outside
```
- Step 4** If DNS inspection is disabled or if you want to change the maximum DNS packet length, configure DNS inspection. DNS application inspection is enabled by default with a maximum DNS packet length of 512 bytes. For configuration instructions, see the [“Configuring a DNS Inspection Policy Map for Additional Inspection Control”](#) section on page 43-7.
- Step 5** On the public DNS server, add an A-record for the web server, such as:
 

```
domain-qualified-hostname. IN A mapped-address
```

where *domain-qualified-hostname* is the hostname with a domain suffix, as in server.example.com. The period after the hostname is important. *mapped-address* is the translated IP address of the web server.

The following example configures the ASA for the scenario shown in [Figure 43-2](#). It assumes DNS inspection is already enabled.

```
hostname(config)# object network obj-192.168.100.10-01
hostname(config-network-object)# host 192.168.100.10
hostname(config-network-object)# nat (dmz,outside) static 209.165.200.225 dns
hostname(config)# access-list 101 permit tcp any host 209.165.200.225 eq www
hostname(config)# access-group 101 in interface outside
```

This configuration requires the following A-record on the DNS server:

```
server.example.com. IN A 209.165.200.225
```

## Configuring a DNS Inspection Policy Map for Additional Inspection Control

DNS application inspection supports DNS message controls that provide protection against DNS spoofing and cache poisoning. User configurable rules allow filtering based on DNS header, domain name, resource record type and class. Zone transfer can be restricted between servers with this function, for example.

The Recursion Desired and Recursion Available flags in the DNS header can be masked to protect a public server from attack if that server only supports a particular internal zone. In addition, DNS randomization can be enabled to avoid spoofing and cache poisoning of servers that either do not support randomization, or utilize a weak pseudo random number generator. Limiting the domain names that can be queried also restricts the domain names which can be queried, which protects the public server further.

A configurable DNS mismatch alert can be used as notification if an excessive number of mismatching DNS responses are received, which could indicate a cache poisoning attack. In addition, a configurable check to enforce a Transaction Signature be attached to all DNS messages is also supported.

To specify actions when a message violates a parameter, create a DNS inspection policy map. You can then apply the inspection policy map when you enable DNS inspection.

To create a DNS inspection policy map, perform the following steps:

- Step 1** (Optional) Add one or more regular expressions for use in traffic matching commands according to the [“Creating a Regular Expression”](#) section on page 13-12. See the types of text you can match in the **match** commands described in [Step 3](#).
- Step 2** (Optional) Create one or more regular expression class maps to group regular expressions according to the [“Creating a Regular Expression Class Map”](#) section on page 13-15.
- Step 3** (Optional) Create a DNS inspection class map by performing the following steps.

A class map groups multiple traffic matches. Traffic must match *all* of the **match** commands to match the class map. You can alternatively identify **match** commands directly in the policy map. The difference between creating a class map and defining the traffic match directly in the inspection policy map is that the class map lets you create more complex match criteria, and you can reuse class maps.

To specify traffic that should not match the class map, use the **match not** command. For example, if the **match not** command specifies the string “example.com,” then any traffic that includes “example.com” does not match the class map.

For the traffic that you identify in this class map, you can specify actions such as drop, drop-connection, reset, mask, set the rate limit, and/or log the connection in the inspection policy map.

If you want to perform different actions for each **match** command, you should identify the traffic directly in the policy map.

- a. Create the class map by entering the following command:

```
hostname(config)# class-map type inspect dns [match-all | match-any] class_map_name
hostname(config-cmap)#
```

Where *class\_map\_name* is the name of the class map. The **match-all** keyword is the default, and specifies that traffic must match all criteria to match the class map. The **match-any** keyword specifies that the traffic matches the class map if it matches at least one of the criteria. The CLI enters class-map configuration mode, where you can enter one or more **match** commands.

- b. (Optional) To add a description to the class map, enter the following command:

```
hostname(config-cmap)# description string
```

- c. (Optional) To match a specific flag that is set in the DNS header, enter the following command:

```
hostname(config-cmap)# match [not] header-flag [eq] {f_well_known | f_value}
```

Where the *f\_well\_known* argument is the DNS flag bit. The *f\_value* argument is the 16-bit value in hex. The **eq** keyword specifies an exact match.

- d. (Optional) To match a DNS type, including Query type and RR type, enter the following command:

```
hostname(config-cmap)# match [not] dns-type {eq t_well_known | t_val} {range t_val1
t_val2}
```

Where the *t\_well\_known* argument is the DNS flag bit. The *t\_val* arguments are arbitrary values in the DNS type field (0-65535). The **range** keyword specifies a range and the **eq** keyword specifies an exact match.

- e. (Optional) To match a DNS class, enter the following command:

```
hostname(config-cmap)# match [not] dns-class {eq c_well_known | c_val} {range c_val1
c_val2}
```

Where the *c\_well\_known* argument is the DNS class. The *c\_val* arguments are arbitrary values in the DNS class field. The **range** keyword specifies a range and the **eq** keyword specifies an exact match.

- f. (Optional) To match a DNS question or resource record, enter the following command:

```
hostname(config-cmap)# match {question | {resource-record answer | authority | any}}
```

Where the **question** keyword specifies the question portion of a DNS message. The **resource-record** keyword specifies the resource record portion of a DNS message. The **answer** keyword specifies the Answer RR section. The **authority** keyword specifies the Authority RR section. The **additional** keyword specifies the Additional RR section.

- g. (Optional) To match a DNS message domain name list, enter the following command:

```
hostname(config-cmap)# match [not] domain-name {regex regex_id | regex class class_id}
```

The **regex** *regex\_name* argument is the regular expression you created in [Step 1](#). The **class** *regex\_class\_name* is the regular expression class map you created in [Step 2](#).

- Step 4** Create a DNS inspection policy map, enter the following command:

```
hostname(config)# policy-map type inspect dns policy_map_name
hostname(config-pmap)#
```



Where the *policy\_map\_name* is the name of the policy map. The CLI enters policy-map configuration mode.

**Step 5** (Optional) To add a description to the policy map, enter the following command:

```
hostname(config-pmap)# description string
```

**Step 6** To apply actions to matching traffic, perform the following steps.

a. Specify the traffic on which you want to perform actions using one of the following methods:

- Specify the DNS class map that you created in [Step 3](#) by entering the following command:

```
hostname(config-pmap)# class class_map_name
hostname(config-pmap-c)#
```

- Specify traffic directly in the policy map using one of the **match** commands described in [Step 3](#). If you use a **match not** command, then any traffic that does not match the criterion in the **match not** command has the action applied.

b. Specify the action you want to perform on the matching traffic by entering the following command:

```
hostname(config-pmap-c)# {[drop [send-protocol-error] |
drop-connection [send-protocol-error] | mask | reset] [log] | rate-limit message_rate}
```

Not all options are available for each **match** or **class** command. See the CLI help or the command reference for the exact options available.

The **drop** keyword drops all packets that match.

The **send-protocol-error** keyword sends a protocol error message.

The **drop-connection** keyword drops the packet and closes the connection.

The **mask** keyword masks out the matching portion of the packet.

The **reset** keyword drops the packet, closes the connection, and sends a TCP reset to the server and/or client.

The **log** keyword, which you can use alone or with one of the other keywords, sends a system log message.

The **rate-limit** *message\_rate* argument limits the rate of messages.

You can specify multiple **class** or **match** commands in the policy map. For information about the order of **class** and **match** commands, see the “[Defining Actions in an Inspection Policy Map](#)” section on [page 31-2](#).

**Step 7** To configure parameters that affect the inspection engine, perform the following steps:

a. To enter parameters configuration mode, enter the following command:

```
hostname(config-pmap)# parameters
hostname(config-pmap-p)#
```

b. To randomize the DNS identifier for a DNS query, enter the following command:

```
hostname(config-pmap-p)# id-randomization
```

c. To enable logging for excessive DNS ID mismatches, enter the following command:

```
hostname(config-pmap-p)# id-mismatch [count number duration seconds] action log
```

Where the **count** *string* argument specifies the maximum number of mismatch instances before a system message log is sent. The **duration** *seconds* specifies the period, in seconds, to monitor.

d. To require a TSIG resource record to be present, enter the following command:

```
hostname(config-pmap-p)# tsig enforced action {drop [log] | [log]}
```

Where the **count string** argument specifies the maximum number of mismatch instances before a system message log is sent. The **duration seconds** specifies the period, in seconds, to monitor.

The following example shows a how to define a DNS inspection policy map.

```
hostname(config)# regex domain_example "example\.com"
hostname(config)# regex domain_foo "foo\.com"

hostname(config)# ! define the domain names that the server serves
hostname(config)# class-map type inspect regex match-any my_domains
hostname(config-cmap)# match regex domain_example
hostname(config-cmap)# match regex domain_foo

hostname(config)# ! Define a DNS map for query only
hostname(config)# class-map type inspect dns match-all pub_server_map
hostname(config-cmap)# match not header-flag QR
hostname(config-cmap)# match question
hostname(config-cmap)# match not domain-name regex class my_domains

hostname(config)# policy-map type inspect dns serv_prot
hostname(config-pmap)# class pub_server_map
hostname(config-pmap-c)# drop log
hostname(config-pmap-c)# match header-flag RD
hostname(config-pmap-c)# mask log

hostname(config)# class-map dns_serv_map
hostname(config-cmap)# match default-inspection-traffic

hostname(config)# policy-map pub_policy
hostname(config-pmap)# class dns_serv_map
hostname(config-pmap-c)# inspect dns serv_prot

hostname(config)# service-policy pub_policy interface dmz
```

## Verifying and Monitoring DNS Inspection

To view information about the current DNS connections, enter the following command:

```
hostname# show conn
```

For connections using a DNS server, the source port of the connection may be replaced by the IP address of DNS server in the show conn command output.

A single connection is created for multiple DNS sessions, as long as they are between the same two hosts, and the sessions have the same 5-tuple (source/destination IP address, source/destination port, and protocol). DNS identification is tracked by app\_id, and the idle timer for each app\_id runs independently.

Because the app\_id expires independently, a legitimate DNS response can only pass through the security appliance within a limited period of time and there is no resource build-up. However, when you enter the **show conn** command, you see the idle timer of a DNS connection being reset by a new DNS session. This is due to the nature of the shared DNS connection and is by design.

To display the statistics for DNS application inspection, enter the **show service-policy** command. The following is sample output from the **show service-policy** command:

```
hostname# show service-policy
Interface outside:
```

```
Service-policy: sample_policy
Class-map: dns_port
Inspect: dns maximum-length 1500, packet 0, drop 0, reset-drop 0
```

## FTP Inspection

This section describes the FTP inspection engine. This section includes the following topics:

- [FTP Inspection Overview, page 43-11](#)
- [Using the strict Option, page 43-11](#)
- [Configuring an FTP Inspection Policy Map for Additional Inspection Control, page 43-12](#)
- [Verifying and Monitoring FTP Inspection, page 43-16](#)

## FTP Inspection Overview

The FTP application inspection inspects the FTP sessions and performs four tasks:

- Prepares dynamic secondary data connection
- Tracks the FTP command-response sequence
- Generates an audit trail
- Translates the embedded IP address

FTP application inspection prepares secondary channels for FTP data transfer. Ports for these channels are negotiated through PORT or PASV commands. The channels are allocated in response to a file upload, a file download, or a directory listing event.

**Note**

If you disable FTP inspection engines with the **no inspect ftp** command, outbound users can start connections only in passive mode, and all inbound FTP is disabled.

## Using the strict Option

Using the **strict** option with the **inspect ftp** command increases the security of protected networks by preventing web browsers from sending embedded commands in FTP requests.

**Note**

To specify FTP commands that are not permitted to pass through the ASA, create an FTP map according to the [“Configuring an FTP Inspection Policy Map for Additional Inspection Control”](#) section on page 43-12.

After you enable the **strict** option on an interface, FTP inspection enforces the following behavior:

- An FTP command must be acknowledged before the ASA allows a new command.
- The ASA drops connections that send embedded commands.
- The 227 and PORT commands are checked to ensure they do not appear in an error string.

**Caution**

Using the **strict** option may cause the failure of FTP clients that are not strictly compliant with FTP RFCs.

If the **strict** option is enabled, each FTP command and response sequence is tracked for the following anomalous activity:

- Truncated command—Number of commas in the PORT and PASV reply command is checked to see if it is five. If it is not five, then the PORT command is assumed to be truncated and the TCP connection is closed.
- Incorrect command—Checks the FTP command to see if it ends with <CR><LF> characters, as required by the RFC. If it does not, the connection is closed.
- Size of RETR and STOR commands—These are checked against a fixed constant. If the size is greater, then an error message is logged and the connection is closed.
- Command spoofing—The PORT command should always be sent from the client. The TCP connection is denied if a PORT command is sent from the server.
- Reply spoofing—PASV reply command (227) should always be sent from the server. The TCP connection is denied if a PASV reply command is sent from the client. This prevents the security hole when the user executes “227 xxxxx a1, a2, a3, a4, p1, p2.”
- TCP stream editing—The ASA closes the connection if it detects TCP stream editing.
- Invalid port negotiation—The negotiated dynamic port value is checked to see if it is less than 1024. As port numbers in the range from 1 to 1024 are reserved for well-known connections, if the negotiated port falls in this range, then the TCP connection is freed.
- Command pipelining—The number of characters present after the port numbers in the PORT and PASV reply command is cross checked with a constant value of 8. If it is more than 8, then the TCP connection is closed.
- The ASA replaces the FTP server response to the SYST command with a series of Xs. to prevent the server from revealing its system type to FTP clients. To override this default behavior, use the **no mask-syst-reply** command in the FTP map.

## Configuring an FTP Inspection Policy Map for Additional Inspection Control

FTP command filtering and security checks are provided using strict FTP inspection for improved security and control. Protocol conformance includes packet length checks, delimiters and packet format checks, command terminator checks, and command validation.

Blocking FTP based on user values is also supported so that it is possible for FTP sites to post files for download, but restrict access to certain users. You can block FTP connections based on file type, server name, and other attributes. System message logs are generated if an FTP connection is denied after inspection.

If you want FTP inspection to allow FTP servers to reveal their system type to FTP clients, and limit the allowed FTP commands, then create and configure an FTP map. You can then apply the FTP map when you enable FTP inspection.

To create an FTP map, perform the following steps:

- Step 1** (Optional) Add one or more regular expressions for use in traffic matching commands according to the “[Creating a Regular Expression](#)” section on page 13-12. See the types of text you can match in the **match** commands described in [Step 3](#).

**Step 2** (Optional) Create one or more regular expression class maps to group regular expressions according to the “[Creating a Regular Expression Class Map](#)” section on page 13-15.

**Step 3** (Optional) Create an FTP inspection class map by performing the following steps.

A class map groups multiple traffic matches. Traffic must match *all* of the **match** commands to match the class map. You can alternatively identify **match** commands directly in the policy map. The difference between creating a class map and defining the traffic match directly in the inspection policy map is that the class map lets you create more complex match criteria, and you can reuse class maps.

To specify traffic that should not match the class map, use the **match not** command. For example, if the **match not** command specifies the string “example.com,” then any traffic that includes “example.com” does not match the class map.

For the traffic that you identify in this class map, you can specify actions such as drop, drop-connection, reset, mask, set the rate limit, and/or log the connection in the inspection policy map.

If you want to perform different actions for each **match** command, you should identify the traffic directly in the policy map.

- a. Create the class map by entering the following command:

```
hostname(config)# class-map type inspect ftp [match-all | match-any] class_map_name
hostname(config-cmap)#
```

Where *class\_map\_name* is the name of the class map. The **match-all** keyword is the default, and specifies that traffic must match all criteria to match the class map. The **match-any** keyword specifies that the traffic matches the class map if it matches at least one of the criteria. The CLI enters class-map configuration mode, where you can enter one or more **match** commands.

- b. (Optional) To add a description to the class map, enter the following command:

```
hostname(config-cmap)# description string
```

- c. (Optional) To match a filename for FTP transfer, enter the following command:

```
hostname(config-cmap)# match [not] filename regex [regex_name |
class regex_class_name]
```

Where the *regex\_name* is the regular expression you created in [Step 1](#). The **class** *regex\_class\_name* is the regular expression class map you created in [Step 2](#).

- d. (Optional) To match a file type for FTP transfer, enter the following command:

```
hostname(config-cmap)# match [not] filetype regex [regex_name |
class regex_class_name]
```

Where the *regex\_name* is the regular expression you created in [Step 1](#). The **class** *regex\_class\_name* is the regular expression class map you created in [Step 2](#).

- e. (Optional) To disallow specific FTP commands, use the following command:

```
hostname(config-cmap)# match [not] request-command ftp_command [ftp_command...]
```

Where *ftp\_command* with one or more FTP commands that you want to restrict. See [Table 43-1](#) for a list of the FTP commands that you can restrict.

Table 43-1 FTP Map request-command deny Options

| request-command deny Option | Purpose                                                                                               |
|-----------------------------|-------------------------------------------------------------------------------------------------------|
| <b>appe</b>                 | Disallows the command that appends to a file.                                                         |
| <b>cdup</b>                 | Disallows the command that changes to the parent directory of the current working directory.          |
| <b>dele</b>                 | Disallows the command that deletes a file on the server.                                              |
| <b>get</b>                  | Disallows the client command for retrieving a file from the server.                                   |
| <b>help</b>                 | Disallows the command that provides help information.                                                 |
| <b>mkd</b>                  | Disallows the command that makes a directory on the server.                                           |
| <b>put</b>                  | Disallows the client command for sending a file to the server.                                        |
| <b>rmd</b>                  | Disallows the command that deletes a directory on the server.                                         |
| <b>rnfr</b>                 | Disallows the command that specifies rename-from filename.                                            |
| <b>rnto</b>                 | Disallows the command that specifies rename-to filename.                                              |
| <b>site</b>                 | Disallows the command that are specific to the server system. Usually used for remote administration. |
| <b>stou</b>                 | Disallows the command that stores a file using a unique file name.                                    |

- f. (Optional) To match an FTP server, enter the following command:

```
hostname(config-cmap)# match [not] server regex [regex_name | class regex_class_name]
```

Where the *regex\_name* is the regular expression you created in [Step 1](#). The **class** *regex\_class\_name* is the regular expression class map you created in [Step 2](#).

- g. (Optional) To match an FTP username, enter the following command:

```
hostname(config-cmap)# match [not] username regex [regex_name | class regex_class_name]
```

Where the *regex\_name* is the regular expression you created in [Step 1](#). The **class** *regex\_class\_name* is the regular expression class map you created in [Step 2](#).

- Step 4** Create an FTP inspection policy map, enter the following command:

```
hostname(config)# policy-map type inspect ftp policy_map_name
hostname(config-pmap)#
```

Where the *policy\_map\_name* is the name of the policy map. The CLI enters policy-map configuration mode.

- Step 5** (Optional) To add a description to the policy map, enter the following command:

```
hostname(config-pmap)# description string
```

- Step 6** To apply actions to matching traffic, perform the following steps.

- a. Specify the traffic on which you want to perform actions using one of the following methods:

- Specify the FTP class map that you created in [Step 3](#) by entering the following command:

```
hostname(config-pmap)# class class_map_name
hostname(config-pmap-c)#
```

- Specify traffic directly in the policy map using one of the **match** commands described in [Step 3](#). If you use a **match not** command, then any traffic that does not match the criterion in the **match not** command has the action applied.
- b. Specify the action you want to perform on the matching traffic by entering the following command:

```
hostname(config-pmap-c)# {[drop [send-protocol-error] |
drop-connection [send-protocol-error] | mask | reset] [log] | rate-limit message_rate}
```

Not all options are available for each **match** or **class** command. See the CLI help or the command reference for the exact options available.

The **drop** keyword drops all packets that match.

The **send-protocol-error** keyword sends a protocol error message.

The **drop-connection** keyword drops the packet and closes the connection.

The **mask** keyword masks out the matching portion of the packet.

The **reset** keyword drops the packet, closes the connection, and sends a TCP reset to the server and/or client.

The **log** keyword, which you can use alone or with one of the other keywords, sends a system log message.

The **rate-limit message\_rate** argument limits the rate of messages.

You can specify multiple **class** or **match** commands in the policy map. For information about the order of **class** and **match** commands, see the “[Defining Actions in an Inspection Policy Map](#)” section on [page 31-2](#).

**Step 7** To configure parameters that affect the inspection engine, perform the following steps:

- a. To enter parameters configuration mode, enter the following command:

```
hostname(config-pmap)# parameters
hostname(config-pmap-p)#
```

- b. To mask the greeting banner from the FTP server, enter the following command:

```
hostname(config-pmap-p)# mask-banner
```

- c. To mask the reply to **syst** command, enter the following command:

```
hostname(config-pmap-p)# mask-syst-reply
```

Before submitting a username and password, all FTP users are presented with a greeting banner. By default, this banner includes version information useful to hackers trying to identify weaknesses in a system. The following example shows how to mask this banner:

```
hostname(config)# policy-map type inspect ftp mymap
hostname(config-pmap)# parameters
hostname(config-pmap-p)# mask-banner

hostname(config)# class-map match-all ftp-traffic
hostname(config-cmap)# match port tcp eq ftp

hostname(config)# policy-map ftp-policy
hostname(config-pmap)# class ftp-traffic
hostname(config-pmap-c)# inspect ftp strict mymap

hostname(config)# service-policy ftp-policy interface inside
```

## Verifying and Monitoring FTP Inspection

FTP application inspection generates the following log messages:

- An Audit record 303002 is generated for each file that is retrieved or uploaded.
- The FTP command is checked to see if it is RETR or STOR and the retrieve and store commands are logged.
- The username is obtained by looking up a table providing the IP address.
- The username, source IP address, destination IP address, NAT address, and the file operation are logged.
- Audit record 201005 is generated if the secondary dynamic channel preparation failed due to memory shortage.

In conjunction with NAT, the FTP application inspection translates the IP address within the application payload. This is described in detail in RFC 959.

## HTTP Inspection

This section describes the HTTP inspection engine. This section includes the following topics:

- [HTTP Inspection Overview, page 43-16](#)
- [Configuring an HTTP Inspection Policy Map for Additional Inspection Control, page 43-17](#)

## HTTP Inspection Overview

Use the HTTP inspection engine to protect against specific attacks and other threats that are associated with HTTP traffic. HTTP inspection performs several functions:

- Enhanced HTTP inspection
- URL screening through N2H2 or Websense  
See [Information About URL Filtering, page 39-6](#) for information.
- Java and ActiveX filtering

The latter two features are configured in conjunction with the **filter** command. For more information about filtering, see [Chapter 39, “Configuring Filtering Services.”](#)

The enhanced HTTP inspection feature, which is also known as an application firewall and is available when you configure an HTTP map (see [“Configuring an HTTP Inspection Policy Map for Additional Inspection Control”](#)), can help prevent attackers from using HTTP messages for circumventing network security policy. It verifies the following for all HTTP messages:

- Conformance to RFC 2616
- Use of RFC-defined methods only.
- Compliance with the additional criteria.



## Configuring an HTTP Inspection Policy Map for Additional Inspection Control

To specify actions when a message violates a parameter, create an HTTP inspection policy map. You can then apply the inspection policy map when you enable HTTP inspection.



### Note

When you enable HTTP inspection with an inspection policy map, strict HTTP inspection with the action reset and log is enabled by default. You can change the actions performed in response to inspection failure, but you cannot disable strict inspection as long as the inspection policy map remains enabled.

To create an HTTP inspection policy map, perform the following steps:

**Step 1** (Optional) Add one or more regular expressions for use in traffic matching commands according to the “[Creating a Regular Expression](#)” section on page 13-12. See the types of text you can match in the **match** commands described in [Step 3](#).

**Step 2** (Optional) Create one or more regular expression class maps to group regular expressions according to the “[Creating a Regular Expression Class Map](#)” section on page 13-15.

**Step 3** (Optional) Create an HTTP inspection class map by performing the following steps.

A class map groups multiple traffic matches. Traffic must match *all* of the **match** commands to match the class map. You can alternatively identify **match** commands directly in the policy map. The difference between creating a class map and defining the traffic match directly in the inspection policy map is that the class map lets you create more complex match criteria, and you can reuse class maps.

To specify traffic that should not match the class map, use the **match not** command. For example, if the **match not** command specifies the string “example.com,” then any traffic that includes “example.com” does not match the class map.

For the traffic that you identify in this class map, you can specify actions such as drop, drop-connection, reset, mask, set the rate limit, and/or log the connection in the inspection policy map.

If you want to perform different actions for each **match** command, you should identify the traffic directly in the policy map.

- a. Create the class map by entering the following command:

```
hostname(config)# class-map type inspect http [match-all | match-any] class_map_name
hostname(config-cmap)#
```

Where *class\_map\_name* is the name of the class map. The **match-all** keyword is the default, and specifies that traffic must match all criteria to match the class map. The **match-any** keyword specifies that the traffic matches the class map if it matches at least one of the criteria. The CLI enters class-map configuration mode, where you can enter one or more **match** commands.

- b. (Optional) To add a description to the class map, enter the following command:

```
hostname(config-cmap)# description string
```

- c. (Optional) To match traffic with a content-type field in the HTTP response that does not match the accept field in the corresponding HTTP request message, enter the following command:

```
hostname(config-cmap)# match [not] req-resp content-type mismatch
```

- d. (Optional) To match text found in the HTTP request message arguments, enter the following command:

```
hostname(config-cmap)# match [not] request args regex [regex_name | class
regex_class_name]
```

Where the *regex\_name* is the regular expression you created in [Step 1](#). The **class** *regex\_class\_name* is the regular expression class map you created in [Step 2](#).

- e. (Optional) To match text found in the HTTP request message body or to match traffic that exceeds the maximum HTTP request message body length, enter the following command:

```
hostname(config-cmap)# match [not] request body {regex [regex_name | class
regex_class_name] | length gt max_bytes}
```

Where the **regex** *regex\_name* argument is the regular expression you created in [Step 1](#). The **class** *regex\_class\_name* is the regular expression class map you created in [Step 2](#). The **length gt** *max\_bytes* is the maximum message body length in bytes.

- f. (Optional) To match text found in the HTTP request message header, or to restrict the count or length of the header, enter the following command:

```
hostname(config-cmap)# match [not] request header {[field]
[regex [regex_name | class regex_class_name]] |
[length gt max_length_bytes | count gt max_count_bytes]}
```

Where the *field* is the predefined message header keyword. The **regex** *regex\_name* argument is the regular expression you created in [Step 1](#). The **class** *regex\_class\_name* is the regular expression class map you created in [Step 2](#). The **length gt** *max\_bytes* is the maximum message body length in bytes. The **count gt** *max\_count* is the maximum number of header fields.

- g. (Optional) To match text found in the HTTP request message method, enter the following command:

```
hostname(config-cmap)# match [not] request method {[method] |
[regex [regex_name | class regex_class_name]]}
```

Where the *method* is the predefined message method keyword. The **regex** *regex\_name* argument is the regular expression you created in [Step 1](#). The **class** *regex\_class\_name* is the regular expression class map you created in [Step 2](#).

- h. (Optional) To match text found in the HTTP request message URI, enter the following command:

```
hostname(config-cmap)# match [not] request uri {regex [regex_name | class
regex_class_name] | length gt max_bytes}
```

Where the **regex** *regex\_name* argument is the regular expression you created in [Step 1](#). The **class** *regex\_class\_name* is the regular expression class map you created in [Step 2](#). The **length gt** *max\_bytes* is the maximum message body length in bytes.

- i. (Optional) To match text found in the HTTP response message body, or to comment out Java applet and Active X object tags in order to filter them, enter the following command:

```
hostname(config-cmap)# match [not] response body {[active-x] | [java-applet] |
[regex [regex_name | class regex_class_name]] | length gt max_bytes}
```

Where the **regex** *regex\_name* argument is the regular expression you created in [Step 1](#). The **class** *regex\_class\_name* is the regular expression class map you created in [Step 2](#). The **length gt** *max\_bytes* is the maximum message body length in bytes.

- j. (Optional) To match text found in the HTTP response message header, or to restrict the count or length of the header, enter the following command:

```
hostname(config-cmap)# match [not] response header {[field]
[regex [regex_name | class regex_class_name]] |
[length gt max_length_bytes | count gt max_count]}
```

Where the *field* is the predefined message header keyword. The **regex** *regex\_name* argument is the regular expression you created in [Step 1](#). The **class** *regex\_class\_name* is the regular expression class map you created in [Step 2](#). The **length gt** *max\_bytes* is the maximum message body length in bytes. The **count gt** *max\_count* is the maximum number of header fields.

- k. (Optional) To match text found in the HTTP response message status line, enter the following command:

```
hostname(config-cmap)# match [not] response status-line {regex [regex_name | class
regex_class_name] }
```

Where the **regex** *regex\_name* argument is the regular expression you created in [Step 1](#). The **class** *regex\_class\_name* is the regular expression class map you created in [Step 2](#).

- Step 4** Create an HTTP inspection policy map, enter the following command:

```
hostname(config)# policy-map type inspect http policy_map_name
hostname(config-pmap)#
```

Where the *policy\_map\_name* is the name of the policy map. The CLI enters policy-map configuration mode.

- Step 5** (Optional) To add a description to the policy map, enter the following command:

```
hostname(config-pmap)# description string
```

- Step 6** To apply actions to matching traffic, perform the following steps.

- a. Specify the traffic on which you want to perform actions using one of the following methods:

- Specify the HTTP class map that you created in [Step 3](#) by entering the following command:

```
hostname(config-pmap)# class class_map_name
hostname(config-pmap-c)#
```

- Specify traffic directly in the policy map using one of the **match** commands described in [Step 3](#). If you use a **match not** command, then any traffic that does not match the criterion in the **match not** command has the action applied.

- b. Specify the action you want to perform on the matching traffic by entering the following command:

```
hostname(config-pmap-c)# {[drop [send-protocol-error] |
drop-connection [send-protocol-error] | mask | reset] [log] | rate-limit message_rate}
```

Not all options are available for each **match** or **class** command. See the CLI help or the command reference for the exact options available.

The **drop** keyword drops all packets that match.

The **send-protocol-error** keyword sends a protocol error message.

The **drop-connection** keyword drops the packet and closes the connection.

The **mask** keyword masks out the matching portion of the packet.

The **reset** keyword drops the packet, closes the connection, and sends a TCP reset to the server and/or client.

The **log** keyword, which you can use alone or with one of the other keywords, sends a system log message.

The **rate-limit** *message\_rate* argument limits the rate of messages.

You can specify multiple **class** or **match** commands in the policy map. For information about the order of **class** and **match** commands, see the “[Defining Actions in an Inspection Policy Map](#)” section on [page 31-2](#).

**Step 7** To configure parameters that affect the inspection engine, perform the following steps:

- a. To enter parameters configuration mode, enter the following command:

```
hostname(config-pmap)# parameters
hostname(config-pmap-p)#
```

- b. To check for HTTP protocol violations, enter the following command:

```
hostname(config-pmap-p)# protocol-violation [action [drop-connection / reset / log]]
```

Where the **drop-connection** action closes the connection. The **reset** action closes the connection and sends a TCP reset to the client. The **log** action sends a system log message when this policy map matches traffic.

- c. To substitute a string for the server header field, enter the following command:

```
hostname(config-pmap-p)# spoofer-server string
```

Where the *string* argument is the string to substitute for the server header field. Note: WebVPN streams are not subject to the **spoofer-server** command.

---

The following example shows how to define an HTTP inspection policy map that will allow and log any HTTP connection that attempts to access "www.xyz.com/\*.asp" or "www.xyz[0-9][0-9].com" with methods "GET" or "PUT." All other URL/Method combinations will be silently allowed.

```
hostname(config)# regex url1 "www\.xyz\.com/.*\.asp"
hostname(config)# regex url2 "www\.xyz[0-9][0-9]\.com"
hostname(config)# regex get "GET"
hostname(config)# regex put "PUT"

hostname(config)# class-map type regex match-any url_to_log
hostname(config-cmap)# match regex url1
hostname(config-cmap)# match regex url2
hostname(config-cmap)# exit

hostname(config)# class-map type regex match-any methods_to_log
hostname(config-cmap)# match regex get
hostname(config-cmap)# match regex put
hostname(config-cmap)# exit

hostname(config)# class-map type inspect http http_url_policy
hostname(config-cmap)# match request uri regex class url_to_log
hostname(config-cmap)# match request method regex class methods_to_log
hostname(config-cmap)# exit

hostname(config)# policy-map type inspect http http_policy
hostname(config-pmap)# class http_url_policy
hostname(config-pmap-c)# log
```

## ICMP Inspection

The ICMP inspection engine allows ICMP traffic to have a "session" so it can be inspected like TCP and UDP traffic. Without the ICMP inspection engine, we recommend that you do not allow ICMP through the ASA in an access list. Without stateful inspection, ICMP can be used to attack your network. The ICMP inspection engine ensures that there is only one response for each request, and that the sequence number is correct.

## ICMP Error Inspection

When this feature is enabled, the ASA creates translation sessions for intermediate hops that send ICMP error messages, based on the NAT configuration. The ASA overwrites the packet with the translated IP addresses.

When disabled, the ASA does not create translation sessions for intermediate nodes that generate ICMP error messages. ICMP error messages generated by the intermediate nodes between the inside host and the ASA reach the outside host without consuming any additional NAT resource. This is undesirable when an outside host uses the traceroute command to trace the hops to the destination on the inside of the ASA. When the ASA does not translate the intermediate hops, all the intermediate hops appear with the mapped destination IP address.

The ICMP payload is scanned to retrieve the five-tuple from the original packet. Using the retrieved five-tuple, a lookup is performed to determine the original address of the client. The ICMP error inspection engine makes the following changes to the ICMP packet:

- In the IP Header, the mapped IP is changed to the real IP (Destination Address) and the IP checksum is modified.
- In the ICMP Header, the ICMP checksum is modified due to the changes in the ICMP packet.
- In the Payload, the following changes are made:
  - Original packet mapped IP is changed to the real IP
  - Original packet mapped port is changed to the real Port
  - Original packet IP checksum is recalculated

## Instant Messaging Inspection

This section describes the IM inspection engine. This section includes the following topics:

- [IM Inspection Overview, page 43-21](#)
- [Configuring an Instant Messaging Inspection Policy Map for Additional Inspection Control, page 43-21](#)

### IM Inspection Overview

The IM inspect engine lets you apply fine grained controls on the IM application to control the network usage and stop leakage of confidential data, propagation of worms, and other threats to the corporate network.

### Configuring an Instant Messaging Inspection Policy Map for Additional Inspection Control

To specify actions when a message violates a parameter, create an IM inspection policy map. You can then apply the inspection policy map when you enable IM inspection.

To create an IM inspection policy map, perform the following steps:

- Step 1** (Optional) Add one or more regular expressions for use in traffic matching commands according to the “[Creating a Regular Expression](#)” section on page 13-12. See the types of text you can match in the **match** commands described in [Step 3](#).
- Step 2** (Optional) Create one or more regular expression class maps to group regular expressions according to the “[Creating a Regular Expression Class Map](#)” section on page 13-15.s
- Step 3** (Optional) Create an IM inspection class map by performing the following steps.

A class map groups multiple traffic matches. Traffic must match *all* of the **match** commands to match the class map. You can alternatively identify **match** commands directly in the policy map. The difference between creating a class map and defining the traffic match directly in the inspection policy map is that the class map lets you create more complex match criteria, and you can reuse class maps.

To specify traffic that should not match the class map, use the **match not** command. For example, if the **match not** command specifies the string “example.com,” then any traffic that includes “example.com” does not match the class map.

For the traffic that you identify in this class map, you can specify actions such as drop-connection, reset, and/or log the connection in the inspection policy map.

If you want to perform different actions for each **match** command, you should identify the traffic directly in the policy map.

- a. Create the class map by entering the following command:

```
hostname(config)# class-map type inspect im [match-all | match-any] class_map_name
hostname(config-cmap)#
```

Where *the class\_map\_name* is the name of the class map. The **match-all** keyword is the default, and specifies that traffic must match all criteria to match the class map. The **match-any** keyword specifies that the traffic matches the class map if it matches at least one of the criteria. The CLI enters class-map configuration mode, where you can enter one or more **match** commands.

- b. (Optional) To add a description to the class map, enter the following command:

```
hostname(config-cmap)# description string
```

Where *the string* is the description of the class map (up to 200 characters).

- c. (Optional) To match traffic of a specific IM protocol, such as Yahoo or MSN, enter the following command:

```
hostname(config-cmap)# match [not] protocol {im-yahoo | im-msn}
```

- d. (Optional) To match a specific IM service, such as chat, file-transfer, webcam, voice-chat, conference, or games, enter the following command:

```
hostname(config-cmap)# match [not] service {chat | file-transfer | webcam | voice-chat
| conference | games}
```

- e. (Optional) To match the source login name of the IM message, enter the following command:

```
hostname(config-cmap)# match [not] login-name regex {class class_name | regex_name}
```

Where the **regex regex\_name** argument is the regular expression you created in [Step 1](#). The **class regex\_class\_name** is the regular expression class map you created in [Step 2](#).

- f. (Optional) To match the destination login name of the IM message, enter the following command:

```
hostname(config-cmap)# match [not] peer-login-name regex {class class_name |
regex_name}
```

Where the **regex** *regex\_name* argument is the regular expression you created in [Step 1](#). The **class** *regex\_class\_name* is the regular expression class map you created in [Step 2](#).

- g. (Optional) To match the source IP address of the IM message, enter the following command:

```
hostname(config-cmap)# match [not] ip-address ip_address ip_address_mask
```

Where the *ip\_address* and the *ip\_address\_mask* is the IP address and netmask of the message source.

- h. (Optional) To match the destination IP address of the IM message, enter the following command:

```
hostname(config-cmap)# match [not] peer-ip-address ip_address ip_address_mask
```

Where the *ip\_address* and the *ip\_address\_mask* is the IP address and netmask of the message destination.

- i. (Optional) To match the version of the IM message, enter the following command:

```
hostname(config-cmap)# match [not] version regex {class class_name | regex_name}
```

Where the **regex** *regex\_name* argument is the regular expression you created in [Step 1](#). The **class** *regex\_class\_name* is the regular expression class map you created in [Step 2](#).

- j. (Optional) To match the filename of the IM message, enter the following command:

```
hostname(config-cmap)# match [not] filename regex {class class_name | regex_name}
```

Where the **regex** *regex\_name* argument is the regular expression you created in [Step 1](#). The **class** *regex\_class\_name* is the regular expression class map you created in [Step 2](#).




---

**Note** Not supported using MSN IM protocol.

---

- Step 4** Create an IM inspection policy map, enter the following command:

```
hostname(config)# policy-map type inspect im policy_map_name
hostname(config-pmap)#
```

Where the *policy\_map\_name* is the name of the policy map. The CLI enters policy-map configuration mode.

- Step 5** (Optional) To add a description to the policy map, enter the following command:

```
hostname(config-pmap)# description string
```

- Step 6** Specify the traffic on which you want to perform actions using one of the following methods:

- Specify the IM class map that you created in [Step 3](#) by entering the following command:

```
hostname(config-pmap)# class class_map_name
hostname(config-pmap-c)#
```

- Specify traffic directly in the policy map using one of the **match** commands described in [Step 3](#). If you use a **match not** command, then any traffic that does not match the criterion in the **match not** command has the action applied.

You can specify multiple **class** or **match** commands in the policy map. For information about the order of **class** and **match** commands, see the “[Defining Actions in an Inspection Policy Map](#)” section on [page 31-2](#).

- Step 7** Specify the action you want to perform on the matching traffic by entering the following command:

```
hostname(config-pmap-c)# {drop-connection | reset | log}
```

Where the **drop-connection** action closes the connection. The **reset** action closes the connection and sends a TCP reset to the client. The **log** action sends a system log message when this policy map matches traffic.

The following example shows how to define an IM inspection policy map.

```
hostname(config)# regex loginname1 "ying@yahoo.com"
hostname(config)# regex loginname2 "Kevin@yahoo.com"
hostname(config)# regex loginname3 "rahul@yahoo.com"
hostname(config)# regex loginname4 "darshant@yahoo.com"
hostname(config)# regex yahoo_version_regex "1\\.0"
hostname(config)# regex gif_files "\.gif"
hostname(config)# regex exe_files "\.exe"

hostname(config)# class-map type regex match-any yahoo_src_login_name_regex
hostname(config-cmap)# match regex loginname1
hostname(config-cmap)# match regex loginname2

hostname(config)# class-map type regex match-any yahoo_dst_login_name_regex
hostname(config-cmap)# match regex loginname3
hostname(config-cmap)# match regex loginname4

hostname(config)# class-map type inspect im match-any yahoo_file_block_list
hostname(config-cmap)# match filename regex gif_files
hostname(config-cmap)# match filename regex exe_files

hostname(config)# class-map type inspect im match-all yahoo_im_policy
hostname(config-cmap)# match login-name regex class yahoo_src_login_name_regex
hostname(config-cmap)# match peer-login-name regex class yahoo_dst_login_name_regex

hostname(config)# class-map type inspect im match-all yahoo_im_policy2
hostname(config-cmap)# match version regex yahoo_version_regex

hostname(config)# class-map im_inspect_class_map
hostname(config-cmap)# match default-inspection-traffic

hostname(config)# policy-map type inspect im im_policy_all
hostname(config-pmap)# class yahoo_file_block_list
hostname(config-pmap-c)# match service file-transfer
hostname(config-pmap)# class yahoo_im_policy
hostname(config-pmap-c)# drop-connection
hostname(config-pmap)# class yahoo_im_policy2
hostname(config-pmap-c)# reset
hostname(config)# policy-map global_policy_name
hostname(config-pmap)# class im_inspect_class_map
hostname(config-pmap-c)# inspect im im_policy_all
```

## IP Options Inspection

This section describes the IP Options inspection engine. This section includes the following topics:

- [IP Options Inspection Overview, page 43-25](#)
- [Configuring an IP Options Inspection Policy Map for Additional Inspection Control, page 43-25](#)



## IP Options Inspection Overview

Each IP packet contains an IP header with the Options field. The Options field, commonly referred to as IP Options, provide for control functions that are required in some situations but unnecessary for most common communications. In particular, IP Options include provisions for time stamps, security, and special routing. Use of IP Options is optional, and the field can contain zero, one, or more options.

You can configure IP Options inspection to control which IP packets with specific IP options are allowed through the ASA. Configuring this inspection instructs the ASA to allow a packet to pass or to clear the specified IP options and then allow the packet to pass.

IP Options inspection can check for the following three IP options in a packet:

- End of Options List (EOOL) or IP Option 0—This option, which contains just a single zero byte, appears at the end of all options to mark the end of a list of options. This might not coincide with the end of the header according to the header length.
- No Operation (NOP) or IP Option 1—The Options field in the IP header can contain zero, one, or more options, which makes the total length of the field variable. However, the IP header must be a multiple of 32 bits. If the number of bits of all options is not a multiple of 32 bits, the NOP option is used as “internal padding” to align the options on a 32-bit boundary.
- Router Alert (RTRALT) or IP Option 20—This option notifies transit routers to inspect the contents of the packet even when the packet is not destined for that router. This inspection is valuable when implementing RSVP and similar protocols require relatively complex processing from the routers along the packets delivery path.

**Note**

IP Options inspection is included by default in the global inspection policy. Therefore, the ASA allows RSVP traffic that contains packets with the Router Alert option (option 20) when the ASA is in routed mode.

Dropping RSVP packets containing the Router Alert option can cause problems in VoIP implementations.

When you configure the ASA to clear the Router Alert option from IP headers, the IP header changes in the following ways:

- The Options field is padded so that the field ends on a 32 bit boundary.
- Internet header length (IHL) changes.
- The total length of the packet changes.
- The checksum is recomputed.

If an IP header contains additional options other than EOOL, NOP, or RTRALT, regardless of whether the ASA is configured to allow these options, the ASA will drop the packet.

## Configuring an IP Options Inspection Policy Map for Additional Inspection Control

**Step 1** To create an IP Options inspection policy map, enter the following command:

```
hostname(config)# policy-map type inspect ip-options policy_map_name
hostname(config-pmap)#
```

Where the *policy\_map\_name* is the name of the policy map. The CLI enters policy-map configuration mode.

**Step 2** (Optional) To add a description to the policy map, enter the following command:

```
hostname(config-pmap)# description string
```

**Step 3** To configure parameters that affect the inspection engine, perform the following steps:

**a.** To enter parameters configuration mode, enter the following command:

```
hostname(config-pmap)# parameters
hostname(config-pmap-p)#
```

**b.** To allow or clear packets with the End of Options List (EOOL) option, enter the following command:

```
hostname(config-pmap-p)# eoool action {allow | clear}
```

This option, which contains just a single zero byte, appears at the end of all options to mark the end of a list of options. This might not coincide with the end of the header according to the header length.

**c.** To allow or clear packets with the No Operation (NOP) option, enter the following command:

```
hostname(config-pmap-p)# nop action {allow | clear}
```

The Options field in the IP header can contain zero, one, or more options, which makes the total length of the field variable. However, the IP header must be a multiple of 32 bits. If the number of bits of all options is not a multiple of 32 bits, the NOP option is used as “internal padding” to align the options on a 32-bit boundary.

**d.** To allow or clear packets with the Router Alert (RTRALT) option, enter the following command:

```
hostname(config-pmap-p)# router-alert action {allow | clear}
```

This option notifies transit routers to inspect the contents of the packet even when the packet is not destined for that router. This inspection is valuable when implementing RSVP and similar protocols require relatively complex processing from the routers along the packets delivery path.




---

**Note** Enter the **clear** command to clear the IP option from the packet before allowing the packet through the ASA.

---

## IPsec Pass Through Inspection

This section describes the IPsec Pass Through inspection engine. This section includes the following topics:

- [IPsec Pass Through Inspection Overview, page 43-27](#)
- [“Example for Defining an IPsec Pass Through Parameter Map” section on page 43-27](#)

## IPsec Pass Through Inspection Overview

Internet Protocol Security (IPsec) is a protocol suite for securing IP communications by authenticating and encrypting each IP packet of a data stream. IPsec also includes protocols for establishing mutual authentication between agents at the beginning of the session and negotiation of cryptographic keys to be used during the session. IPsec can be used to protect data flows between a pair of hosts (for example, computer users or servers), between a pair of security gateways (such as routers or firewalls), or between a security gateway and a host.

IPsec Pass Through application inspection provides convenient traversal of ESP (IP protocol 50) and AH (IP protocol 51) traffic associated with an IKE UDP port 500 connection. It avoids lengthy access list configuration to permit ESP and AH traffic and also provides security using timeout and max connections.

Specify IPsec Pass Through inspection parameters to identify a specific map to use for defining the parameters for the inspection. Configure a policy map for Specify IPsec Pass Through inspection to access the parameters configuration, which lets you specify the restrictions for ESP or AH traffic. You can set the per client max connections and the idle timeout in parameters configuration.

NAT and non-NAT traffic is permitted. However, PAT is not supported.

## Example for Defining an IPsec Pass Through Parameter Map

The following example shows how to use access lists to identify IKE traffic, define an IPsec Pass Through parameter map, define a policy, and apply the policy to the outside interface:

```
hostname(config)# access-list ipsecpassthruacl permit udp any any eq 500
hostname(config)# class-map ipsecpassthru-traffic
hostname(config-cmap)# match access-list ipsecpassthruacl
hostname(config)# policy-map type inspect ipsec-pass-thru iptmap
hostname(config-pmap)# parameters
hostname(config-pmap-p)# esp per-client-max 10 timeout 0:11:00
hostname(config-pmap-p)# ah per-client-max 5 timeout 0:06:00
hostname(config)# policy-map inspection_policy
hostname(config-pmap)# class ipsecpassthru-traffic
hostname(config-pmap-c)# inspect ipsec-pass-thru iptmap
hostname(config)# service-policy inspection_policy interface outside
```

## IPv6 Inspection

You can configure IPv6 Inspection by using MPF rules to selectively block IPv6 traffic based on the extension header. IPv6 packets are subjected to an early security check. The ASA always passes hop-by-hop and destination option types of extension headers while blocking router header and no next header.

You can enable default IPv6 inspection or define IPv6 inspection. By defining an MPF policy map for IPv6 inspection you can configure the ASA to selectively drop IPv6 packets based on following types of extension headers found anywhere in the IPv6 packet:

- Hop-by-Hop Options
- Routing (Type 0)
- Fragment
- Destination Options

- Authentication
- Encapsulating Security Payload

In addition, default IPv6 inspection checks conformance to RFC 2460 for type and order of extension headers in IPv6 packets:

- IPv6 header
- Hop-by-Hop Options header (0)
- Destination Options header (60)
- Routing header (43)
- Fragment header (44)
- Authentication (51)
- Encapsulating Security Payload header(50)
- Destination Options header (60)
- No Next Header (59)

When a policy map is not configured for IPv6 inspection or a configured policy map is not associated with an interface, the ASA drops packets with any mobility type and a routing-type IPv6 extension header that arrive at the interface.

When an IPv6 inspection policy map is created, the ASA automatically generates a configuration to drop packets that match header routing-type in the range 0-255.

## Configuring an IPv6 Inspection Policy Map

You can configure a policy map for IPv6 inspection to handle IPv6 extension headers. The IPv6 policy map is applied to each classified IPv6 packet on the specified direction. Currently, only incoming IPv6 traffic is inspected.

---

## NetBIOS Inspection

This section describes the IM inspection engine. This section includes the following topics:

- [NetBIOS Inspection Overview, page 43-28](#)
- [Configuring a NetBIOS Inspection Policy Map for Additional Inspection Control, page 43-29](#)

## NetBIOS Inspection Overview

NetBIOS inspection is enabled by default. The NetBios inspection engine translates IP addresses in the NetBios name service (NBNS) packets according to the ASA NAT configuration.

## Configuring a NetBIOS Inspection Policy Map for Additional Inspection Control

To specify actions when a message violates a parameter, create a NETBIOS inspection policy map. You can then apply the inspection policy map when you enable NETBIOS inspection.

To create a NETBIOS inspection policy map, perform the following steps:

**Step 1** (Optional) Add one or more regular expressions for use in traffic matching commands according to the “[Creating a Regular Expression](#)” section on page 13-12. See the types of text you can match in the **match** commands described in [Step 3](#).

**Step 2** (Optional) Create one or more regular expression class maps to group regular expressions according to the “[Creating a Regular Expression Class Map](#)” section on page 13-15.

**Step 3** Create a NetBIOS inspection policy map, enter the following command:

```
hostname(config)# policy-map type inspect netbios policy_map_name
hostname(config-pmap)#
```

Where the *policy\_map\_name* is the name of the policy map. The CLI enters policy-map configuration mode.

**Step 4** (Optional) To add a description to the policy map, enter the following command:

```
hostname(config-pmap)# description string
```

**Step 5** To apply actions to matching traffic, perform the following steps.

a. Specify the traffic on which you want to perform actions using one of the following methods:

- Specify the NetBIOS class map that you created in [Step 3](#) by entering the following command:

```
hostname(config-pmap)# class class_map_name
hostname(config-pmap-c)#
```

- Specify traffic directly in the policy map using one of the **match** commands described in [Step 3](#). If you use a **match not** command, then any traffic that does not match the criterion in the **match not** command has the action applied.

b. Specify the action you want to perform on the matching traffic by entering the following command:

```
hostname(config-pmap-c)# {[drop [send-protocol-error] |
drop-connection [send-protocol-error] | mask | reset] [log] | rate-limit message_rate}
```

Not all options are available for each **match** or **class** command. See the CLI help or the command reference for the exact options available.

The **drop** keyword drops all packets that match.

The **send-protocol-error** keyword sends a protocol error message.

The **drop-connection** keyword drops the packet and closes the connection.

The **mask** keyword masks out the matching portion of the packet.

The **reset** keyword drops the packet, closes the connection, and sends a TCP reset to the server and/or client.

The **log** keyword, which you can use alone or with one of the other keywords, sends a system log message.

The **rate-limit** *message\_rate* argument limits the rate of messages.

You can specify multiple **class** or **match** commands in the policy map. For information about the order of **class** and **match** commands, see the “[Defining Actions in an Inspection Policy Map](#)” section on page 31-2.

**Step 6** To configure parameters that affect the inspection engine, perform the following steps:

- a. To enter parameters configuration mode, enter the following command:

```
hostname(config-pmap)# parameters
hostname(config-pmap-p)#
```

- b. To check for NETBIOS protocol violations, enter the following command:

```
hostname(config-pmap-p)# protocol-violation [action [drop-connection / reset / log]]
```

Where the **drop-connection** action closes the connection. The **reset** action closes the connection and sends a TCP reset to the client. The **log** action sends a system log message when this policy map matches traffic.

The following example shows how to define a NETBIOS inspection policy map.

```
hostname(config)# policy-map type inspect netbios netbios_map
hostname(config-pmap)# protocol-violation drop log

hostname(config)# policy-map netbios_policy
hostname(config-pmap)# class inspection_default
hostname(config-pmap-c)# inspect netbios netbios_map
```

## PPTP Inspection

PPTP is a protocol for tunneling PPP traffic. A PPTP session is composed of one TCP channel and usually two PPTP GRE tunnels. The TCP channel is the control channel used for negotiating and managing the PPTP GRE tunnels. The GRE tunnels carries PPP sessions between the two hosts.

When enabled, PPTP application inspection inspects PPTP protocol packets and dynamically creates the GRE connections and xlates necessary to permit PPTP traffic. Only Version 1, as defined in RFC 2637, is supported.

PAT is only performed for the modified version of GRE [RFC 2637] when negotiated over the PPTP TCP control channel. Port Address Translation is *not* performed for the unmodified version of GRE [RFC 1701, RFC 1702].

Specifically, the ASA inspects the PPTP version announcements and the outgoing call request/response sequence. Only PPTP Version 1, as defined in RFC 2637, is inspected. Further inspection on the TCP control channel is disabled if the version announced by either side is not Version 1. In addition, the outgoing-call request and reply sequence are tracked. Connections and xlates are dynamic allocated as necessary to permit subsequent secondary GRE data traffic.

The PPTP inspection engine must be enabled for PPTP traffic to be translated by PAT. Additionally, PAT is only performed for a modified version of GRE (RFC2637) and only if it is negotiated over the PPTP TCP control channel. PAT is not performed for the unmodified version of GRE (RFC 1701 and RFC 1702).

As described in RFC 2637, the PPTP protocol is mainly used for the tunneling of PPP sessions initiated from a modem bank PAC (PPTP Access Concentrator) to the headend PNS (PPTP Network Server).

When used this way, the PAC is the remote client and the PNS is the server.

However, when used for VPN by Windows, the interaction is inverted. The PNS is a remote single-user PC that initiates connection to the head-end PAC to gain access to a central network.

## SMTP and Extended SMTP Inspection

This section describes the IM inspection engine. This section includes the following topics:

- [SMTP and ESMTP Inspection Overview, page 43-31](#)
- [Configuring an ESMTP Inspection Policy Map for Additional Inspection Control, page 43-32](#)

### SMTP and ESMTP Inspection Overview

ESMTP application inspection provides improved protection against SMTP-based attacks by restricting the types of SMTP commands that can pass through the ASA and by adding monitoring capabilities.

ESMTP is an enhancement to the SMTP protocol and is similar in most respects to SMTP. For convenience, the term SMTP is used in this document to refer to both SMTP and ESMTP. The application inspection process for extended SMTP is similar to SMTP application inspection and includes support for SMTP sessions. Most commands used in an extended SMTP session are the same as those used in an SMTP session but an ESMTP session is considerably faster and offers more options related to reliability and security, such as delivery status notification.

Extended SMTP application inspection adds support for these extended SMTP commands, including AUTH, EHLO, ETRN, HELP, SAML, SEND, SOML, STARTTLS, and VRFY. Along with the support for seven RFC 821 commands (DATA, HELO, MAIL, NOOP, QUIT, RCPT, RSET), the ASA supports a total of fifteen SMTP commands.

Other extended SMTP commands, such as ATRN, ONEX, VERB, CHUNKING, and private extensions are not supported. Unsupported commands are translated into Xs, which are rejected by the internal server. This results in a message such as “500 Command unknown: 'XXX'.” Incomplete commands are discarded.

The ESMTP inspection engine changes the characters in the server SMTP banner to asterisks except for the “2”, “0”, “0” characters. Carriage return (CR) and linefeed (LF) characters are ignored.

With SMTP inspection enabled, a Telnet session used for interactive SMTP may hang if the following rules are not observed: SMTP commands must be at least four characters in length; must be terminated with carriage return and line feed; and must wait for a response before issuing the next reply.

An SMTP server responds to client requests with numeric reply codes and optional human-readable strings. SMTP application inspection controls and reduces the commands that the user can use as well as the messages that the server returns. SMTP inspection performs three primary tasks:

- Restricts SMTP requests to seven basic SMTP commands and eight extended commands.
- Monitors the SMTP command-response sequence.
- Generates an audit trail—Audit record 108002 is generated when invalid character embedded in the mail address is replaced. For more information, see RFC 821.

SMTP inspection monitors the command and response sequence for the following anomalous signatures:

- Truncated commands.
- Incorrect command termination (not terminated with <CR><LR>).

- The MAIL and RCPT commands specify who are the sender and the receiver of the mail. Mail addresses are scanned for strange characters. The pipeline character (|) is deleted (changed to a blank space) and “<” ,”>” are only allowed if they are used to define a mail address (“>” must be preceded by “<”). To close the session when the PIPE character is found as a parameter to a MAIL from or RCPT to command, include the **special-character** command in the configuration as part of the inspection parameters (**parameters** command).
- Unexpected transition by the SMTP server.
- For unknown commands, the ASA changes all the characters in the packet to X. In this case, the server generates an error code to the client. Because of the change in the packed, the TCP checksum has to be recalculated or adjusted.
- TCP stream editing.
- Command pipelining.

## Configuring an ESMTP Inspection Policy Map for Additional Inspection Control

ESMTP inspection detects attacks, including spam, phishing, malformed message attacks, buffer overflow/underflow attacks. It also provides support for application security and protocol conformance, which enforce the sanity of the ESMTP messages as well as detect several attacks, block senders/receivers, and block mail relay.

To specify actions when a message violates a parameter, create an ESMTP inspection policy map. You can then apply the inspection policy map when you enable ESMTP inspection.

To create an ESMTP inspection policy map, perform the following steps:

---

**Step 1** (Optional) Add one or more regular expressions for use in traffic matching commands according to the “[Creating a Regular Expression](#)” section on page 13-12. See the types of text you can match in the **match** commands described in [Step 3](#).

**Step 2** (Optional) Create one or more regular expression class maps to group regular expressions according to the “[Creating a Regular Expression Class Map](#)” section on page 13-15.

**Step 3** Create an ESMTP inspection policy map, enter the following command:

```
hostname(config)# policy-map type inspect esmtp policy_map_name
hostname(config-pmap)#
```

Where the *policy\_map\_name* is the name of the policy map. The CLI enters policy-map configuration mode.

**Step 4** (Optional) To add a description to the policy map, enter the following command:

```
hostname(config-pmap)# description string
```

**Step 5** To apply actions to matching traffic, perform the following steps.

a. Specify the traffic on which you want to perform actions using one of the following methods:

- Specify the ESMTP class map that you created in [Step 3](#) by entering the following command:

```
hostname(config-pmap)# class class_map_name
hostname(config-pmap-c)#
```

- Specify traffic directly in the policy map using one of the **match** commands described in [Step 3](#). If you use a **match not** command, then any traffic that does not match the criterion in the **match not** command has the action applied.



- b. Specify the action you want to perform on the matching traffic by entering the following command:

```
hostname(config-pmap-c)# {[drop [send-protocol-error] |
drop-connection [send-protocol-error] | mask | reset] [log] | rate-limit message_rate}
```

Not all options are available for each **match** or **class** command. See the CLI help or the command reference for the exact options available.

The **drop** keyword drops all packets that match.

The **send-protocol-error** keyword sends a protocol error message.

The **drop-connection** keyword drops the packet and closes the connection.

The **mask** keyword masks out the matching portion of the packet.

The **reset** keyword drops the packet, closes the connection, and sends a TCP reset to the server and/or client.

The **log** keyword, which you can use alone or with one of the other keywords, sends a system log message.

The **rate-limit message\_rate** argument limits the rate of messages.

You can specify multiple **class** or **match** commands in the policy map. For information about the order of **class** and **match** commands, see the “[Defining Actions in an Inspection Policy Map](#)” section on page 31-2.

- Step 6** To configure parameters that affect the inspection engine, perform the following steps:

- a. To enter parameters configuration mode, enter the following command:

```
hostname(config-pmap) # parameters
hostname(config-pmap-p) #
```

- b. To configure a local domain name, enter the following command:

```
hostname(config-pmap-p) # mail-relay domain-name action [drop-connection / log]]
```

Where the **drop-connection** action closes the connection. The **log** action sends a system log message when this policy map matches traffic.

- c. To enforce banner obfuscation, enter the following command:

```
hostname(config-pmap-p) # mask-banner
```

The following example shows how to define an ESMTP inspection policy map.

```
hostname(config)# regex user1 "user1@cisco.com"
hostname(config)# regex user2 "user2@cisco.com"
hostname(config)# regex user3 "user3@cisco.com"
hostname(config)# class-map type regex senders_black_list
hostname(config-cmap)# description "Regular expressions to filter out undesired senders"
hostname(config-cmap)# match regex user1
hostname(config-cmap)# match regex user2
hostname(config-cmap)# match regex user3

hostname(config)# policy-map type inspect esmtp advanced_esmtp_map
hostname(config-pmap)# match sender-address regex class senders_black_list
hostname(config-pmap-c)# drop-connection log

hostname(config)# policy-map outside_policy
hostname(config-pmap)# class inspection_default
hostname(config-pmap-c)# inspect esmtp advanced_esmtp_map
```

```
hostname(config)# service-policy outside_policy interface outside
```

## TFTP Inspection

TFTP inspection is enabled by default.

TFTP, described in RFC 1350, is a simple protocol to read and write files between a TFTP server and client.

The ASA inspects TFTP traffic and dynamically creates connections and translations, if necessary, to permit file transfer between a TFTP client and server. Specifically, the inspection engine inspects TFTP read request (RRQ), write request (WRQ), and error notification (ERROR).

A dynamic secondary channel and a PAT translation, if necessary, are allocated on a reception of a valid read (RRQ) or write (WRQ) request. This secondary channel is subsequently used by TFTP for file transfer or error notification.

Only the TFTP server can initiate traffic over the secondary channel, and at most one incomplete secondary channel can exist between the TFTP client and server. An error notification from the server closes the secondary channel.

TFTP inspection must be enabled if static PAT is used to redirect TFTP traffic.