

Inspection of Basic Internet Protocols

The following topics explain application inspection for basic Internet protocols. For information on why you need to use inspection for certain protocols, and the overall methods for applying inspection, see Getting Started with Application Layer Protocol Inspection.

- DCERPC Inspection, on page 1
- DNS Inspection, on page 4
- FTP Inspection, on page 9
- HTTP Inspection, on page 14
- ICMP Inspection, on page 18
- ICMP Error Inspection, on page 18
- ILS Inspection, on page 19
- Instant Messaging Inspection, on page 19
- IP Options Inspection, on page 22
- IPsec Pass Through Inspection, on page 24
- IPv6 Inspection, on page 26
- NetBIOS Inspection, on page 28
- PPTP Inspection, on page 29
- RSH Inspection, on page 30
- SMTP and Extended SMTP Inspection, on page 30
- SNMP Inspection, on page 34
- SQL*Net Inspection, on page 35
- Sun RPC Inspection, on page 36
- TFTP Inspection, on page 37
- XDMCP Inspection, on page 37
- VXLAN Inspection, on page 38
- History for Basic Internet Protocol Inspection, on page 38

DCERPC Inspection

DCERPC inspection is not enabled in the default inspection policy, so you must enable it if you need this inspection. You can simply edit the default global inspection policy to add DCERPC inspection. You can alternatively create a new service policy as desired, for example, an interface-specific policy.

The following sections describe the DCERPC inspection engine.

DCERPC Overview

Microsoft Remote Procedure Call (MSRPC), based on DCERPC, is a protocol widely used by Microsoft distributed client and server applications that allows software clients to execute programs on a server remotely.

This typically involves a client querying a server called the Endpoint Mapper listening on a well known port number for the dynamically allocated network information of a required service. The client then sets up a secondary connection to the server instance providing the service. The security appliance allows the appropriate port number and network address and also applies NAT, if needed, for the secondary connection.

The DCERPC inspection engine inspects for native TCP communication between the EPM and client on well known TCP port 135. Map and lookup operations of the EPM are supported for clients. Client and server can be located in any security zone. The embedded server IP address and Port number are received from the applicable EPM response messages. Since a client may attempt multiple connections to the server port returned by EPM, multiple use of pinholes are allowed, which have configurable timeouts.

DCE inspection supports the following universally unique identifiers (UUIDs) and messages:

- End point mapper (EPM) UUID. All EPM messages are supported.
- ISystemMapper UUID (non-EPM). Supported messages are:
  - RemoteCreateInstance opnum4
  - RemoteGetClassObject opnum3
- OxidResolver UUID (non-EPM). Supported message is:
  - ServerAlive2 opnum5
- Any message that does not contain an IP address or port information because these messages do not require inspection.

Configure a DCERPC Inspection Policy Map

To specify additional DCERPC inspection parameters, create a DCERPC inspection policy map. You can then apply the inspection policy map when you enable DCERPC inspection.

When defining traffic matching criteria, you can either create a class map or include the match statements 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 you can reuse class maps.

Procedure

Step 1 (Optional) Create a DCERPC inspection class map.

For the traffic that you identify in this class map, you specify actions to take on the traffic 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: **class-map type inspect dcerpc** [**match-all** | **match-any**] *class_map_name*

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 **match** statement. The CLI enters class-map configuration mode.

b) Specify the traffic on which you want to perform actions using the following **match** command. 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.

- **match** [**not**] **uuid** *type*—Matches the universally unique identifier (UUID) of the DCERPC message. The *type* can be one of the following:
  - **ms-rpc-epm**—Matches Microsoft RPC EPM messages.
  - **ms-rpc-isystemactivator**—Matches ISystemMapper messages.
  - **ms-rpc-oxidresolver**—Matches OxidResolver messages.

c) Enter **exit** to leave class map configuration mode.

Step 2 Create a DCERPC inspection policy map: **policy-map type inspect dcerpc** *policy_map_name*

Where the *policy_map_name* is the name of the policy map. The CLI enters policy-map configuration mode.

Step 3 (Optional) Add a description to the policy map: **description** *string*

Step 4 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:

- If you created a DCERPC class map, specify it by entering the following command: **class** *class_map_name*
- Specify traffic directly in the policy map using one of the **match** commands described for DCERPC class maps. 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 one of the following commands:

- **reset** [**log**]—Drop the packet, close the connection, and send a TCP reset to the server or client.
- **log**—Send a system log message. You can use this option alone or with one of the other actions.

You can specify multiple **class** or **match** commands in the policy map.

Example:

```
hostname(config)# policy-map type inspect dcerpc dcerpc-map
hostname(config-pmap)# match uuid ms-rpc-epm
hostname(config-pmap-c)# log
```

Step 5 To configure parameters that affect the inspection engine, perform the following steps:

a) Enter parameters configuration mode:

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

b) Set one or more parameters. You can set the following options; use the **no** form of the command to disable the option:

- **timeout pinhole** *hh:mm:ss*—Configures the timeout for DCERPC pinholes and override the global system pinhole timeout of two minutes. The timeout can be from 00:00:01 to 119:00:00.
- **endpoint-mapper** [**epm-service-only**] [**lookup-operation** [**timeout** *hh:mm:ss*]]—Configures options for the endpoint mapper traffic. The **epm-service-only** keyword enforces endpoint mapper service during binding so that only its service traffic is processed. The **lookup-operation** keyword enables the lookup operation of the endpoint mapper service. You can configure the timeout for pinholes generated from the lookup operation. If no timeout is configured for the lookup operation, the timeout pinhole command or the default is used.

Examples

The following example shows how to define a DCERPC inspection policy map with the timeout configured for DCERPC pinholes.

```
hostname(config)# policy-map type inspect dcerpc dcerpc_map
hostname(config-pmap)# timeout pinhole 0:10:00

hostname(config)# class-map dcerpc
hostname(config-cmap)# match port tcp eq 135

hostname(config)# policy-map global-policy
hostname(config-pmap)# class dcerpc
hostname(config-pmap-c)# inspect dcerpc dcerpc-map

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

What to do next

You can now configure an inspection policy to use the map. See Configure Application Layer Protocol Inspection.

DNS Inspection

DNS inspection is enabled by default. You need to configure it only if you want non-default processing. The following sections describe DNS application inspection.

Defaults for DNS Inspection

DNS inspection is enabled by default, using the preset_dns_map inspection class map:

- The maximum DNS message length is 512 bytes.
- DNS over TCP inspection is disabled.
- The maximum client DNS message length is automatically set to match the Resource Record.

- DNS Guard is enabled, so 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.
- Translation of the DNS record based on the NAT configuration is enabled.
- Protocol enforcement is enabled, which enables DNS message format check, including domain name length of no more than 255 characters, label length of 63 characters, compression, and looped pointer check.

See the following default DNS inspection commands:

```
class-map inspection_default
 match default-inspection-traffic
policy-map type inspect dns preset_dns_map
 parameters
    message-length maximum client auto
    message-length maximum 512
    dns-guard
    protocol-enforcement
    nat-rewrite
policy-map global_policy
 class inspection_default
  inspect dns preset_dns_map
! ...
service-policy global_policy global
```



Note Elevated CPU utilization is likely to occur in Cisco ASA when a large number of packets is inspected per second. As the number of packets being processed increases—especially when the device nears its maximum throughput or connection capacity—the CPU usage too rises. This occurs because ASA must perform more operations per second, which increases CPU resource consumption.

Configure DNS Inspection Policy Map

You can create a DNS inspection policy map to customize DNS inspection actions if the default inspection behavior is not sufficient for your network.

Before you begin

Some traffic matching options use regular expressions for matching purposes. If you intend to use one of those techniques, first create the regular expression or regular expression class map.

Procedure

Step 1 (Optional) Create a DNS inspection class map by performing the following steps.

A class map groups multiple traffic matches. 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 specify actions to take on the traffic 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: **class-map type inspect dns** [**match-all** | **match-any**] *class_map_name*

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 **match** statement. The CLI enters class-map configuration mode, where you can enter one or more **match** commands.

b) (Optional) Add a description to the class map: **description** *string*

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

c) Specify the traffic on which you want to perform actions using one of the following **match** commands. 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.

- **match** [**not**] **header-flag** [**eq**] {*f_name* [*f_name*...] | *f_value*}—Matches the DNS flag. The *f_name* argument is the DNS flag name, one of the following: **AA** (Authoritative Answer), **QR** (Query), **RA** (Recursion Available), **RD** (Recursion Desired), **TC** (Truncation). The *f_value* argument is the 16-bit value in hex starting with 0x, from 0x0 to 0xffff. The eq keyword specifies an exact match (match all); without the **eq** keyword, the packet only needs to match one of the specified headers (match any). For example, **match header-flag AA QR**.
- **match** [**not**] **dns-type {eq** {*t_name* | *t_value*} | **range** *t_value1 t_value2*}—Matches the DNS type. The *t_name* argument is the DNS type name, one of the following: **A** (IPv4 address), **AXFR** (full zone transfer), **CNAME** (canonical name), **IXFR** (incremental zone transfer), **NS** (authoritative name server), **SOA** (start of a zone of authority) or **TSIG** (transaction signature). The t_value 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. For example: **match dns-type eq A**.
- **match** [**not**] **dns-class {eq** {**in** | *c_value*} | **range** *c_value1 c_value2*}—Matches the DNS class. The class is either **in** (for Internet) or c_value, an arbitrary value from 0 to 65535 in the DNS class field. The range keyword specifies a range, and the eq keyword specifies an exact match. For example: **match dns-class eq in**.
- **match** [**not**] {**question** | **resource-record** {**answer** | **authority** | **additional**}}—Matches a DNS question or resource record. The **question** keyword specifies the question portion of a DNS message. The resource-record keyword specifies one of these sections of the resource record: **answer**, authority, or **additional**. For example: **match resource-record answer**.
- **match** [**not**] **domain-name regex** {*regex_name* | **class** *class_name*}—Matches the DNS message domain name list against the specified regular expression or regular expression class.

d) Enter **exit** to leave class map configuration mode.

Step 2 Create a DNS inspection policy map: **policy-map type inspect dns** *policy_map_name*

Where the *policy_map_name* is the name of the policy map. The CLI enters policy-map configuration mode.

Step 3 (Optional) Add a description to the policy map: **description** *string*

Step 4 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:

- If you created a DNS class map, specify it by entering the following command: **class** *class_map_name*
- Specify traffic directly in the policy map using one of the **match** commands described for DNS class maps. 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 one of the following commands:

- **drop** [**log**]—Drop all packets that match.
- **drop-connection** [**log**]—Drop the packet and close the connection.
- **mask** [**log**]—Mask out the matching portion of the packet. This action is available for header flag matches only.
- **log**—Send a system log message. You can use this option alone or with one of the other actions.
- **enforce-tsig** [**drop**] [**log**]—Enforce the presence of the TSIG resource record in a message. You can drop a packet without the TSIG resource record, log it, or drop and log it. You can use this option in conjunction with the mask action for header flag matches; otherwise, this action is exclusive with the other actions.

You can specify multiple **class** or **match** commands in the policy map. For information about the order of **class** and **match** commands, see How Multiple Traffic Classes are Handled.

Example:

```
hostname(config)# policy-map type inspect dns dns-map
hostname(config-pmap)# class dns-class-map
hostname(config-pmap-c)# drop
hostname(config-pmap-c)# match header-flag eq aa
hostname(config-pmap-c)# drop log
```

Step 5 To configure parameters that affect the inspection engine, perform the following steps:

a) Enter parameters configuration mode.

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

b) Set one or more parameters. You can set the following options; use the **no** form of the command to disable the option.

- **dnscrypt**—Enables DNScrypt to encrypt connections between the device and Cisco Umbrella. Enabling DNScrypt starts the key-exchange thread with the Umbrella resolver. The key-exchange thread performs the handshake with the resolver every hour and updates the device with a new secret key. Because DNScrypt uses UDP/443, you must ensure that the class map used for DNS inspection includes that port. Note that the default inspection class already includes UDP/443 for DNS inspection.

- **dns-guard**—Enables DNS Guard. 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.
- **id-mismatch count** *number* **duration** *seconds* **action log**—Enables logging for excessive DNS ID mismatches, where the **count** *number* **duration** *seconds* arguments specify the maximum number of mismatch instances per second before a system message log is sent.
- **id-randomization**—Randomizes the DNS identifier for a DNS query.
- **message-length maximum** {*length* | **client** {*length* | **auto**} | **server** {*length* | **auto**}}—Sets the maximum DNS message length, from 512 to 65535 bytes. You can also set the maximum length for client or server messages. The **auto** keyword sets the maximum length to the value in the Resource Record.
- **nat-rewrite**—Translates the DNS record based on the NAT configuration.
- **protocol-enforcement**—Enables DNS message format check, including domain name length of no more than 255 characters, label length of 63 characters, compression, and looped pointer check.
- **tcp-inspection**—Enables inspection of DNS over TCP traffic. Ensure that DNS/TCP port 53 traffic is part of the class to which you apply DNS inspection. The inspection default class includes TCP/53.
- **tsig enforced action** {[**drop**] [**log**]}—Requires a TSIG resource record to be present. You can **drop** a non-conforming packet, **log** the packet, or both.
- **umbrella** [**tag** *umbrella_policy*] [**fail-open**]—Enables Cisco Umbrella and optionally specifies the name of the Cisco Umbrella policy (**tag**) to apply to the device. If you do not specify a policy, the default policy is applied. For more information, see Cisco Umbrella.

  Include the **fail-open** keyword if you want DNS resolution to work if the Umbrella DNS server is unavailable. When failing open, if the Cisco Umbrella DNS server is unavailable, Umbrella disables itself on this policy map and allows DNS requests to go to the other DNS servers configured on the system, if any. When the Umbrella DNS servers are available again, the policy map resumes using them. If you do not include this option, DNS requests continue to go to the unreachable Umbrella resolver, so they will not get a response.

Example:

```
hostname(config-pmap)# parameters
hostname(config-pmap-p)# dns-guard
hostname(config-pmap-p)# message-length maximum 1024
hostname(config-pmap-p)# nat-rewrite
hostname(config-pmap-p)# protocol-enforcement
```

Example

The following example shows a how to use a new inspection policy map in the global default configuration:

```
regex domain_example "example\.com"
regex domain_foo "foo\.com"
```

```
! define the domain names that the server serves
class-map type regex match-any my_domains
   match regex domain_example
   match regex domain_foo

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

policy-map type inspect dns new_dns_map
    class pub_server_map
        drop log
    match header-flag RD
        mask log
    parameters
        message-length maximum client auto
        message-length maximum 512
        dns-guard
        protocol-enforcement
        nat-rewrite

policy-map global_policy
 class inspection_default
  no inspect dns preset_dns_map
  inspect dns new_dns_map
service-policy global_policy global
```

What to do next

You can now configure an inspection policy to use the map. See Configure Application Layer Protocol Inspection.

FTP Inspection

FTP inspection is enabled by default. You need to configure it only if you want non-default processing. The following sections describe the FTP inspection engine.

FTP Inspection Overview

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

- Prepares dynamic secondary data connection 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.
- Tracks the FTP command-response sequence.
- Generates an audit trail.
  - Audit record 303002 is generated for each file that is retrieved or uploaded.
  - Audit record 201005 is generated if the secondary dynamic channel preparation failed due to memory shortage.

- Translates the embedded IP address.

Note If you disable FTP inspection, outbound users can start connections only in passive mode, and all inbound FTP is disabled.

Strict FTP

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

When you use strict FTP, you can optionally specify an FTP inspection policy map to specify FTP commands that are not permitted to pass through the ASA.

Strict 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 strict FTP may cause the failure of FTP clients that are not strictly compliant with FTP RFCs. Additionally, you must ensure you apply the inspection to your FTP ports only (TCP/21 is the normal FTP port). Strict FTP inspection applied to non-FTP traffic can result in unexpected traffic loss, especially HTTP traffic.

With strict FTP inspection, 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.

Configure an FTP Inspection Policy Map

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 inspection policy map. You can then apply the map when you enable FTP inspection.

Before you begin

Some traffic matching options use regular expressions for matching purposes. If you intend to use one of those techniques, first create the regular expression or regular expression class map.

Procedure

Step 1 (Optional) Create an FTP inspection class map by performing the following steps.

A class map groups multiple traffic matches. 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 specify actions to take on the traffic 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: **class-map type inspect ftp** [**match-all** | **match-any**] *class_map_name*

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 **match** statement. The CLI enters class-map configuration mode, where you can enter one or more **match** commands.

b) (Optional) Add a description to the class map: **description** *string*

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

c) Specify the traffic on which you want to perform actions using one of the following **match** commands. 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.

- **match** [**not**] **filename regex** {*regex_name* | **class** *class_name*}—Matches the filename in the FTP transfer against the specified regular expression or regular expression class.
- **match** [**not**] **filetype regex** {*regex_name* | **class** *class_name*}—Matches the file type in the FTP transfer against the specified regular expression or regular expression class.
- **match** [**not**] **request-command** *ftp_command* [*ftp_command*...]—Matches the FTP command, one or more of the following:
  - **APPE**—Append to a file.
  - **CDUP**—Changes to the parent directory of the current working directory.
  - **DELE**—Delete a file on the server.
  - **GET**—Gets a file from the server.
  - **HELP**—Provides help information.
  - **MKD**—Makes a directory on the server.
  - **PUT**—Sends a file to the server.
  - **RMD**—Deletes a directory on the server.
  - **RNFR**—Specifies the “rename-from” filename.
  - **RNTO**—Specifies the “rename-to” filename.
  - **SITE**—Used to specify a server-specific command. This is usually used for remote administration.
  - **STOU**—Stores a file using a unique file name.
- **match** [**not**] **server regex** {*regex_name* | **class** *class_name*}—Matches the FTP server name against the specified regular expression or regular expression class.
- **match** [**not**] **username regex** {*regex_name* | **class** *class_name*}—Matches the FTP username against the specified regular expression or regular expression class.

d) Enter **exit** to leave class map configuration mode.

Step 2 Create an FTP inspection policy map: **policy-map type inspect ftp** *policy_map_name*

Where the *policy_map_name* is the name of the policy map. The CLI enters policy-map configuration mode.

Step 3 (Optional) Add a description to the policy map: **description** *string*

Step 4 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:

- If you created an FTP class map, specify it by entering the following command: **class** *class_map_name*
- Specify traffic directly in the policy map using one of the **match** commands described for FTP class maps. 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:

- **reset** [**log**]—Drop the packet, close the connection, and send a TCP reset to the server or client. Add the **log** keyword to send a system log message.

You can specify multiple **class** or **match** commands in the policy map. For information about the order of **class** and **match** commands, see How Multiple Traffic Classes are Handled.

Step 5 To configure parameters that affect the inspection engine, perform the following steps:

a) Enter parameters configuration mode.

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

b) Set one or more parameters. You can set the following options; use the **no** form of the command to disable the option:

- **mask-banner**—Masks the greeting banner from the FTP server.
- **mask-syst-reply**—Masks the reply to **syst** command.

Example

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
```

What to do next

You can now configure an inspection policy to use the map. See Configure Application Layer Protocol Inspection.

HTTP Inspection

HTTP inspection is not enabled in the default inspection policy, so you must enable it if you need this inspection. However, the default inspect class does include the default HTTP ports, so you can simply edit the default global inspection policy to add HTTP inspection. You can alternatively create a new service policy as desired, for example, an interface-specific policy.

The following sections describe the HTTP inspection engine.

HTTP Inspection Overview

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

HTTP application inspection scans HTTP headers and body, and performs various checks on the data. These checks prevent various HTTP constructs, content types, and tunneling and messaging protocols from traversing the security appliance.

The enhanced HTTP inspection feature, which is also known as an application firewall and is available when you configure an HTTP inspection policy map, can help prevent attackers from using HTTP messages for circumventing network security policy.

HTTP application inspection can block tunneled applications and non-ASCII characters in HTTP requests and responses, preventing malicious content from reaching the web server. Size limiting of various elements in HTTP request and response headers, URL blocking, and HTTP server header type spoofing are also supported.

Enhanced HTTP inspection verifies the following for all HTTP messages:

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

Configure an HTTP Inspection Policy Map

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.

Before you begin

Some traffic matching options use regular expressions for matching purposes. If you intend to use one of those techniques, first create the regular expression or regular expression class map.

Procedure

Step 1 (Optional) Create an HTTP inspection class map by performing the following steps.

A class map groups multiple traffic matches. 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 specify actions to take on the traffic 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: **class-map type inspect http** [**match-all** | **match-any**] *class_map_name*

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 **match** statement. The CLI enters class-map configuration mode, where you can enter one or more **match** commands.

b) (Optional) Add a description to the class map: **description** *string*

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

c) Specify the traffic on which you want to perform actions using one of the following **match** commands. 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.

- **match** [**not**] **req-resp content-type mismatch**—Matches traffic with a content-type field in the HTTP response that does not match the accept field in the corresponding HTTP request message.
- **match** [**not**] **request args regex** {*regex_name* | **class** *class_name*}—Matches text found in the HTTP request message arguments against the specified regular expression or regular expression class.
- **match** [**not**] **request body** {**regex** {*regex_name* | **class** *class_name*} | **length gt** *bytes*}—Matches text found in the HTTP request message body against the specified regular expression or regular expression class, or messages where the request body is greater than the specified length.
- **match** [**not**] **request header** {*field* | **regex** *regex_name*} **regex** {*regex_name* | **class** *class_name*}—Matches the content of a field in the HTTP request message header against the specified regular expression or regular expression class. You can specify the field name explicitly or match the field name to a regular expression. Field names are: accept, accept-charset, accept-encoding, accept-language, allow, authorization, cache-control, connection, content-encoding, content-language, content-length, content-location, content-md5, content-range, content-type, cookie, date, expect, expires, from, host, if-match, if-modified-since, if-none-match, if-range, if-unmodified-since, last-modified, max-forwards, pragma, proxy-authorization, range, referer, te, trailer, transfer-encoding, upgrade, user-agent, via, warning.
- **match** [**not**] **request header** {*field* | **regex** {*regex_name* | **class** *class_name*}} {**length gt** *bytes* | **count gt** *number*}—Matches the length of the specified fields in the HTTP request message header,

or the overall number of fields (count) in the header. You can specify the field name explicitly or match the field name to a regular expression or regular expression class. Field names are listed in the previous bullet.

- **match** [**not**] **request header** {**length gt** *bytes* | **count gt** *number* | **non-ascii**}—Matches the overall length of the HTTP request message header, or the overall number of fields (count) in the header, or headers that have non-ASCII characters.
- **match** [**not**] **request method** {*method* | **regex** {*regex_name* | **class** *class_name*}}—Matches the HTTP request method. You can specify the method explicitly or match the method to a regular expression or regular expression class. Methods are: bcopy, bdelete, bmove, bpropfind, bproppatch, connect, copy, delete, edit, get, getattribute, getattributenames, getproperties, head, index, lock, mkcol, mkdir, move, notify, options, poll, post, propfind, proppatch, put, revadd, revlabel, revlog, revnum, save, search, setattribute, startrev, stoprev, subscribe, trace, unedit, unlock, unsubscribe.
- **match** [**not**] **request uri** {**regex** {*regex_name* | **class** *class_name*} | **length gt** *bytes*}—Matches text found in the HTTP request message URI against the specified regular expression or regular expression class, or messages where the request URI is greater than the specified length.
- **match** [**not**] **response body** {**active-x** | **java-applet** | **regex** {*regex_name* | **class** *class_name*}}—Matches text found in the HTTP response message body against the specified regular expression or regular expression class, or comments out Java applet and Active X object tags in order to filter them.
- **match** [**not**] **response body length gt** *bytes*—Matches HTTP response messages where the body is greater than the specified length.
- **match** [**not**] **response header** {*field* | **regex** *regex_name*} **regex** {*regex_name* | **class** *class_name*}—Matches the content of a field in the HTTP response message header against the specified regular expression or regular expression class. You can specify the field name explicitly or match the field name to a regular expression. Field names are: accept-ranges, age, allow, cache-control, connection, content-encoding, content-language, content-length, content-location, content-md5, content-range, content-type, date, etag, expires, last-modified, location, pragma, proxy-authenticate, retry-after, server, set-cookie, trailer, transfer-encoding, upgrade, vary, via, warning, www-authenticate.
- **match** [**not**] **response header** {*field* | **regex** {*regex_name* | **class** *class_name*}} {**length gt** *bytes* | **count gt** *number*}—Matches the length of the specified fields in the HTTP response message header, or the overall number of fields (count) in the header. You can specify the field name explicitly or match the field name to a regular expression or regular expression class. Field names are listed in the previous bullet.
- **match** [**not**] **response header** {**length gt** *bytes* | **count gt** *number* | **non-ascii**}—Matches the overall length of the HTTP response message header, or the overall number of fields (count) in the header, or headers that have non-ASCII characters.
- **match** [**not**] **response status-line regex** {*regex_name* | **class** *class_name*}—Matches text found in the HTTP response message status line against the specified regular expression or regular expression class.

d) Enter **exit** to leave class map configuration mode.

Step 2 Create an HTTP inspection policy map: **policy-map type inspect http** *policy_map_name*

Where the *policy_map_name* is the name of the policy map. The CLI enters policy-map configuration mode.

Step 3 (Optional) Add a description to the policy map: **description** *string*

Step 4 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:

- If you created an HTTP class map, specify it by entering the following command: **class** *class_map_name*
- Specify traffic directly in the policy map using one of the **match** commands described for HTTP class maps. 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 one of the following commands:

- **drop-connection** [**log**]—Drop the packet and close the connection.
- **reset** [**log**]—Drop the packet, close the connection, and send a TCP reset to the server or client.
- **log**—Send a system log message. You can use this option alone or with one of the other actions.

You can specify multiple **class** or **match** commands in the policy map. For information about the order of **class** and **match** commands, see How Multiple Traffic Classes are Handled.

Step 5 To configure parameters that affect the inspection engine, perform the following steps:

a) Enter parameters configuration mode:

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

b) Set one or more parameters. You can set the following options; use the **no** form of the command to disable the option:

- **body-match-maximum** *number*—Sets the maximum number of characters in the body of an HTTP message that should be searched in a body match. The default is 200 bytes. A large number will have a significant impact on performance.
- **protocol-violation action** {**drop-connection** [**log**] | **reset** [**log**] | **log**}—Checks for HTTP protocol violations. You must also choose the action to take for violations (drop connection, reset, or log) and whether to enable or disable logging.
- **spoof-server** *string*—Substitutes a string for the server header field. WebVPN streams are not subject to the spoof-server command.

Example

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
```

What to do next

You can now configure an inspection policy to use the map. See Configure Application Layer Protocol Inspection.

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 ACL. 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.

However, ICMP traffic directed to an ASA interface is never inspected, even if you enable ICMP inspection. Thus, a ping (echo request) to an interface can fail under specific circumstances, such as when the echo request comes from a source that the ASA can reach through a backup default route.

Note NAT uses ICMP inspection when translating packets even if you disable ICMP inspection.

For information on enabling ICMP inspection, see Configure Application Layer Protocol Inspection.

ICMP Error Inspection

When ICMP Error inspection 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.

Note You should always enable ICMP Error inspection if there is a possibility that NAT will be used on ICMP packets. Because NAT automatically uses ICMP inspection for ICMP packets, even if you have ICMP inspection disabled, the use of the mapped destination address as the source address can make it look like a scanner is examining your network. For example, without ICMP Error inspection also enabled, if the echo request packet has its destination translated, when it is embedded in a ICMP time exceeded response, the outer header of the time exceeded request uses the translated destination as the source address. If you enable ICMP Error inspection, the time exceeded source address will be set to the correct value.

For information on enabling ICMP Error inspection, see Configure Application Layer Protocol Inspection.

ILS Inspection

The Internet Locator Service (ILS) inspection engine provides NAT support for Microsoft NetMeeting, SiteServer, and Active Directory products that use LDAP to exchange directory information with an ILS server. You cannot use PAT with ILS inspection because only IP addresses are stored by an LDAP database.

For search responses, when the LDAP server is located outside, consider using NAT to allow internal peers to communicate locally while registered to external LDAP servers. If you do not need to use NAT, we recommend that you turn off the inspection engine to provide better performance.

Additional configuration may be necessary when the ILS server is located inside the ASA border. This would require a hole for outside clients to access the LDAP server on the specified port, typically TCP 389.

Note Because ILS traffic (H225 call signaling) only occurs on the secondary UDP channel, the TCP connection is disconnected after the TCP inactivity interval. By default, this interval is 60 minutes and can be adjusted using the TCP **timeout** command. In ASDM, this is on the **Configuration > Firewall > Advanced > Global Timeouts** pane.

ILS inspection has the following limitations:

- Referral requests and responses are not supported.
- Users in multiple directories are not unified.
- Single users having multiple identities in multiple directories cannot be recognized by NAT.

For information on enabling ILS inspection, see Configure Application Layer Protocol Inspection.

Instant Messaging Inspection

The Instant Messaging (IM) inspect engine lets you control the network usage of IM and stop leakage of confidential data, propagation of worms, and other threats to the corporate network.

IM inspection is not enabled in the default inspection policy, so you must enable it if you need this inspection. However, the default inspect class does include the default IM ports, so you can simply edit the default global inspection policy to add IM inspection. You can alternatively create a new service policy as desired, for example, an interface-specific policy.

If you decide to implement IM inspection, you can also configure an IM inspection policy map to specify actions when a message violates a parameter. The following procedure explains IM inspection policy maps.

Before you begin

Some traffic matching options use regular expressions for matching purposes. If you intend to use one of those techniques, first create the regular expression or regular expression class map.

Procedure

Step 1 (Optional) Create an IM inspection class map by performing the following steps.

A class map groups multiple traffic matches. 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 specify actions to take on the traffic 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: **class-map type inspect im** [**match-all** | **match-any**] *class_map_name*

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 **match** statement. The CLI enters class-map configuration mode, where you can enter one or more **match** commands.

b) (Optional) Add a description to the class map: **description** *string*

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

c) Specify the traffic on which you want to perform actions using one of the following **match** commands. 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.

- **match** [**not**] **protocol {im-yahoo | im-msn}**—Matches a specific IM protocol, either Yahoo or MSN.
- **match** [**not**] **service {chat | file-transfer** | **webcam** | **voice-chat** | **conference** | **games}**—Matches the specific IM service.
- **match** [**not**] **login-name regex** {*regex_name* | **class** *class_name*}—Matches the source client login name of the IM message against the specified regular expression or regular expression class.

- **match** [**not**] **peer-login-name regex** {*regex_name* | **class** *class_name*}—Matches the destination peer login name of the IM message against the specified regular expression or regular expression class.
- **match** [**not**] **ip-address** *ip_address mask*}—Matches the source IP address and mask of the IM message.
- **match** [**not**] **peer-ip-address** *ip_address mask*}—Matches the destination IP address and mask of the IM message.
- **match** [**not**] **version regex** {*regex_name* | **class** *class_name*}—Matches the version of the IM message against the specified regular expression or regular expression class.
- **match** [**not**] **filename regex** {*regex_name* | **class** *class_name*}—Matches the filename of the IM message against the specified regular expression or regular expression class. This match is not supported for the MSN IM protocol.

d) Enter **exit** to leave class map configuration mode.

Step 2 Create an IM inspection policy map: **policy-map type inspect im** *policy_map_name*

Where the *policy_map_name* is the name of the policy map. The CLI enters policy-map configuration mode.

Step 3 (Optional) Add a description to the policy map: **description** *string*

Step 4 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:

- If you created an IM class map, specify it by entering the following command: **class** *class_map_name*
- Specify traffic directly in the policy map using one of the **match** commands described for IM class maps. 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:

- **drop-connection** [**log**]—Drop the packet and close the connection.
- **reset** [**log**]—Drop the packet, close the connection, and send a TCP reset to the server or client.
- **log**—Send a system log message. You can use this option alone or with one of the other actions.

You can specify multiple **class** or **match** commands in the policy map. For information about the order of **class** and **match** commands, see How Multiple Traffic Classes are Handled.

Example

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
```

What to do next

You can now configure an inspection policy to use the map. See Configure Application Layer Protocol Inspection.

IP Options Inspection

You can configure IP Options inspection to control which IP packets are allowed based on the contents of the IP Options field in the packet header. You can drop packets that have unwanted options, clear the options (and allow the packet), or allow the packet without change.

IP options provide 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.

For a list of IP options, with references to the relevant RFCs, see the IANA page, http://www.iana.org/assignments/ip-parameters/ip-parameters.xhtml.

IP options inspection is enabled by default, but for RSVP traffic only. You need to configure it only if you want to allow additional options than the default map allows, or if you want to apply it to other types of traffic by using a non-default inspection traffic class map.

Note IP options inspection does not work on fragmented packets. For example, options are not cleared from fragments.

The following sections describe IP Options inspection.

Defaults for IP Options Inspection

IP Options inspection is enabled by default for RSVP traffic only, using the _default_ip_options_map inspection policy map.

- The Router Alert option is allowed.

  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 that require relatively complex processing from the routers along the packet's delivery path. Dropping RSVP packets containing the Router Alert option can cause problems in VoIP implementations.

- Packets that contain any other options are dropped.

  Each time a packet is dropped due to inspection, syslog 106012 is issued. The message shows which option caused the drop. Use the **show service-policy inspect ip-options** command to view statistics for each option.

Following is the policy map configuration:

```
policy-map type inspect ip-options _default_ip_options_map
 description Default IP-OPTIONS policy-map
 parameters
  router-alert action allow
```

Configure an IP Options Inspection Policy Map

If you want to perform non-default IP options inspection, create an IP options inspection policy map to specify how you want to handle each option type.

Procedure

Step 1 Create an IP options inspection policy map: **policy-map type inspect ip-options** *policy_map_name*

Where the *policy_map_name* is the name of the policy map. The CLI enters policy-map configuration mode.

Step 2 (Optional) Add a description to the policy map: **description** *string*

Step 3 Enter parameters configuration mode:

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

Step 4 Identify the options you want to allow.

You can inspect the following options. In all cases, the **allow** action allows packets that contain the option without modification; the **clear** action allows the packets but removes the option from the header.

Use the **no** form of the command to remove the option from the map. Any packet that contains an option that you do not include in the map is dropped, even if the packet contains otherwise allowed or cleared options.

For a list of IP options, with references to the relevant RFCs, see the IANA page, http://www.iana.org/assignments/ip-parameters/ip-parameters.xhtml.

- **default action** {**allow** | **clear**}—Sets the default action for any option not explicitly included in the map. If you do not set a default action of allow or clear, packets that contain non-allowed options are dropped
- **basic-security action** {**allow** | **clear**}—Allows or clears the Security (SEC) option.
- **commercial-security action** {**allow** | **clear**}—Allows or clears the Commercial Security (CIPSO) option.
- **eool action** {**allow** | **clear**}—Allows or clears the End of Options List option.
- **exp-flow-control action** {**allow** | **clear**}—Allows or clears the Experimental Flow Control (FINN) option.
- **exp-measurement action** {**allow** | **clear**}—Allows or clears the Experimental Measurement (ZSU) option.
- **extended-security action** {**allow** | **clear**}—Allows or clears the Extended Security (E-SEC) option.
- **imi-traffic-descriptor action** {**allow** | **clear**}—Allows or clears the IMI Traffic Descriptor (IMITD) option.
- **nop action** {**allow** | **clear**}—Allows or clears the No Operation option.
- **quick-start action** {**allow** | **clear**}—Allows or clears the Quick-Start (QS) option.
- **record-route action** {**allow** | **clear**}—Allows or clears the Record Route (RR) option.
- **router-alert action** {**allow** | **clear**}—Allows or clears the Router Alert (RTRALT) option.
- **timestamp action** {**allow** | **clear**}—Allows or clears the Time Stamp (TS) option.
- {**0-255**} **action** {**allow** | **clear**}—Allows or clears the option identified by the option type number. The number is the whole option type octet (copy, class, and option number), not just the option number portion of the octet. These option types might not represent real options. Non-standard options must be in the expected type-length-value format defined in the Internet Protocol RFC 791, http://tools.ietf.org/html/rfc791.

What to do next

You can now configure an inspection policy to use the map. See Configure Application Layer Protocol Inspection.

IPsec Pass Through Inspection

IPsec Pass Through inspection is not enabled in the default inspection policy, so you must enable it if you need this inspection. However, the default inspect class does include the default IPsec ports, so you can simply edit the default global inspection policy to add IPsec inspection. You can alternatively create a new service policy as desired, for example, an interface-specific policy.

The following sections describe the IPsec Pass Through inspection engine.

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 ACL configuration to permit ESP and AH traffic and also provides security using timeout and max connections.

Configure a policy map for IPsec Pass Through to specify the restrictions for ESP or AH traffic. You can set the per client max connections and the idle timeout.

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

Configure an IPsec Pass Through Inspection Policy Map

An IPsec Pass Through map lets you change the default configuration values used for IPsec Pass Through application inspection. You can use an IPsec Pass Through map to permit certain flows without using an ACL.

The configuration includes a default map, _default_ipsec_passthru_map, that sets no maximum limit on ESP connections per client, and sets the ESP idle timeout at 10 minutes. You need to configure an inspection policy map only if you want different values, or if you want to set AH values.

Procedure

Step 1 Create an IPsec Pass Through inspection policy map: **policy-map type inspect ipsec-pass-thru** *policy_map_name*

Where the *policy_map_name* is the name of the policy map. The CLI enters policy-map configuration mode.

Step 2 (Optional) Add a description to the policy map: **description** *string*

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

a) Enter parameters configuration mode:

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

b) Set one or more parameters. You can set the following options; use the **no** form of the command to disable the option:

- **esp per-client-max** *number* **timeout** *time*—Allows ESP tunnels and sets the maximum connections allowed per client and the idle timeout (in hh:mm:ss format). To allow an unlimited number of connections, specify 0 for the number.

- **ah per-client-max** *number* **timeout** *time*—Allows AH tunnels. The parameters have the same meaning as for the esp command.

Example

The following example shows how to use ACLs to identify IKE traffic, define an IPsec Pass Thru 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

IPv6 inspection lets you selectively log or drop IPv6 traffic based on the extension header. In addition, IPv6 inspection can check conformance to RFC 2460 for type and order of extension headers in IPv6 packets.

IPv6 inspection is not enabled in the default inspection policy, so you must enable it if you need this inspection. You can simply edit the default global inspection policy to add IPv6 inspection. You can alternatively create a new service policy as desired, for example, an interface-specific policy.

Defaults for IPv6 Inspection

If you enable IPv6 inspection and do not specify an inspection policy map, then the default IPv6 inspection policy map is used, and the following actions are taken:

- Allows only known IPv6 extension headers. Non-conforming packets are dropped and logged.
- Enforces the order of IPv6 extension headers as defined in the RFC 2460 specification. Non-conforming packets are dropped and logged.
- Drops any packet with a routing type header.

Following is the policy map configuration:

```
policy-map type inspect ipv6 _default_ipv6_map
 description Default IPV6 policy-map
 parameters
  verify-header type
  verify-header order
 match header routing-type range 0 255
  drop log
```

Configure an IPv6 Inspection Policy Map

To identify extension headers to drop or log, or to disable packet verification, create an IPv6 inspection policy map to be used by the service policy.

Procedure

Step 1 Create an IPv6 inspection policy map: **policy-map type inspect ipv6** *policy_map_name*

Where the *policy_map_name* is the name of the policy map. The CLI enters policy-map configuration mode.

Step 2 (Optional) Add a description to the policy map: **description** *string*

Step 3 (Optional) Drop or log traffic based on the headers in IPv6 messages.

a) Identify the traffic based on the IPv6 header: **match header** *type*

Where *type* is one of the following:

- **ah**—Matches the IPv6 Authentication extension header.
- **count gt** *number*—Specifies the maximum number of IPv6 extension headers, from 0 to 255.
- **destination-option**—Matches the IPv6 destination-option extension header.
- **esp**—Matches the IPv6 Encapsulation Security Payload (ESP) extension header.
- **fragment**—Matches the IPv6 fragment extension header.
- **hop-by-hop**—Matches the IPv6 hop-by-hop extension header.
- **routing-address count gt** *number*—Sets the maximum number of IPv6 routing header type 0 addresses, greater than a number between 0 and 255.
- **routing-type** {**eq** | **range**} *number*—Matches the IPv6 routing header type, from 0 to 255. For a range, separate values by a space, for example, **30 40**.

b) Specify the action to perform on matching packets. You can drop the packet and optionally log it, or just log it. If you do not enter an action, the packet is logged.

- **drop** [**log**]—Drop all packets that match.
- **log**—Send a system log message. You can use this option alone or with one of the other actions.

c) Repeat the process until you identify all headers that you want to drop or log.

Step 4 Configure parameters that affect the inspection engine.

a) Enter parameters configuration mode.

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

b) Set one or more parameters. You can set the following options; use the **no** form of the command to disable the option:

- **verify-header type**—Allows only known IPv6 extension headers.

- **verify-header order**—Enforces the order of IPv6 extension headers as defined in RFC 2460.

Example

The following example creates an inspection policy map that will drop and log all IPv6 packets with the hop-by-hop, destination-option, routing-address, and routing type 0 headers. It also enforces header order and type.

```
policy-map type inspect ipv6 ipv6-pm
 parameters
  verify-header type
  verify-header order
 match header hop-by-hop
  drop log
 match header destination-option
  drop log
 match header routing-address count gt 0
  drop log
 match header routing-type eq 0
  drop log

policy-map global_policy
 class class-default
  inspect ipv6 ipv6-pm
!
service-policy global_policy global
```

What to do next

You can now configure an inspection policy to use the map. See Configure Application Layer Protocol Inspection.

NetBIOS Inspection

NetBIOS application inspection performs NAT for the embedded IP address in the NetBIOS name service (NBNS) packets and NetBIOS datagram services packets. It also enforces protocol conformance, checking the various count and length fields for consistency.

NetBIOS inspection is enabled by default. You can optionally create a policy map to drop or log NetBIOS protocol violations. The following procedure explains how to configure a NetBIOS inspection policy map.

Procedure

Step 1 Create a NetBIOS inspection policy map: **policy-map type inspect netbios** *policy_map_name*

Where the *policy_map_name* is the name of the policy map. The CLI enters policy-map configuration mode.

Step 2 (Optional) Add a description to the policy map: **description** *string*

Step 3 Enter parameters configuration mode.

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

Step 4 Specify the action to take for NETBIOS protocol violations: **protocol-violation action** {**drop** [**log**] | **log**}

Where the **drop** action drops the packet. The **log** action sends a system log message when this policy map matches traffic.

Example

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

hostname(config)# policy-map netbios_policy
hostname(config-pmap)# class inspection_default
hostname(config-pmap-c)# no inspect netbios
hostname(config-pmap-c)# inspect netbios netbios_map
```

What to do next

You can now configure an inspection policy to use the map. See Configure Application Layer Protocol Inspection.

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 carry 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.

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 dynamically 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).

For information on enabling PPTP inspection, see Configure Application Layer Protocol Inspection.

RSH Inspection

RSH inspection is enabled by default. The RSH protocol uses a TCP connection from the RSH client to the RSH server on TCP port 514. The client and server negotiate the TCP port number where the client listens for the STDERR output stream. RSH inspection supports NAT of the negotiated port number if necessary.

For information on enabling RSH inspection, see Configure Application Layer Protocol Inspection.

SMTP and Extended SMTP Inspection

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

ESMTP inspection is enabled by default. You need to configure it only if you want different processing than that provided by the default inspection map.

The following sections describe the ESMTP inspection engine.

SMTP and ESMTP Inspection Overview

Extended SMTP (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 is most respects to SMTP.

ESMTP application inspection controls and reduces the commands that the user can use as well as the messages that the server returns. ESMTP inspection performs three primary tasks:

- Restricts SMTP requests to seven basic SMTP commands and eight extended commands. Supported commands are the following:
  - Extended SMTP—AUTH, EHLO, ETRN, HELP, SAML, SEND, SOML, STARTTLS, and VRFY.
  - SMTP (RFC 821)—DATA, HELO, MAIL, NOOP, QUIT, RCPT, RSET.
- Monitors the SMTP command-response sequence.
- Generates an audit trail—Audit record 108002 is generated when an invalid character embedded in the mail address is replaced. For more information, see RFC 821.

ESMTP 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 “<”).
- Unexpected transition by the SMTP server.

- For unknown or unsupported commands, the inspection engine changes all the characters in the packet to X, which are rejected by the internal server. This results in a message such as “500 Command unknown: 'XXX'.” Incomplete commands are discarded

  Unsupported ESMTP commands are ATRN, ONEX, VERB, CHUNKING, and private extensions..
- TCP stream editing.
- Command pipelining.



Note With ESMTP 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; they must be terminated with carriage return and line feed; and you must wait for a response before issuing the next reply.

Defaults for ESMTP Inspection

ESMTP inspection is enabled by default, using the _default_esmtp_map inspection policy map.

- The server banner is masked. 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.
- Encrypted connections are allowed but not inspected.
- Special characters in sender and receiver address are not noticed, no action is taken.
- Connections with command line length greater than 512 are dropped and logged.
- Connections with more than 100 recipients are dropped and logged.
- Messages with body length greater than 998 bytes are logged.
- Connections with header line length greater than 998 are dropped and logged.
- Messages with MIME filenames greater than 255 characters are dropped and logged.
- EHLO reply parameters matching “others” are masked.

Following is the policy map configuration:

```
policy-map type inspect esmtp _default_esmtp_map
 description Default ESMTP policy-map
 parameters
  mask-banner
  no mail-relay
  no special-character
  allow-tls
 match cmd line length gt 512
  drop-connection log
 match cmd RCPT count gt 100
  drop-connection log
 match body line length gt 998
  log
 match header line length gt 998
  drop-connection log
```

```
match sender-address length gt 320
 drop-connection log
match MIME filename length gt 255
 drop-connection log
match ehlo-reply-parameter others
 mask
```

Configure an ESMTP Inspection Policy Map

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.

Before you begin

Some traffic matching options use regular expressions for matching purposes. If you intend to use one of those techniques, first create the regular expression or regular expression class map.

Procedure

Step 1 Create an ESMTP inspection policy map: **policy-map type inspect esmtp** *policy_map_name*

Where the *policy_map_name* is the name of the policy map. The CLI enters policy-map configuration mode.

Step 2 (Optional) Add a description to the policy map: **description** *string*

Step 3 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 **match** commands. 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.

- **match** [**not**] **body** {**length** | **line length**} **gt** *bytes*—Matches messages where the length or length of a line in an ESMTP body message is greater than the specified number of bytes.
- **match** [**not**] **cmd verb** *verb1* [*verb2...*]—Matches the command verb in the message. You can specify one or more of the following commands: auth, data, ehlo, etrn, helo, help, mail, noop, quit, rcpt, rset, saml, soml, vrfy.
- **match** [**not**] **cmd line length gt** *bytes*—Matches messages where the length of a line in the command verb is greater than the specified number of bytes.
- **match** [**not**] **cmd rcpt count gt** *count*—Matches messages where the number of recipients is greater than the specified count.
- **match** [**not**] **ehlo-reply-parameter** *parameter* [*parameter2...*]—Matches ESMTP EHLO reply parameters. You can specify one or more of the following parameters: 8bitmime, auth, binaryname, checkpoint, dsn, etrn, others, pipelining, size, vrfy.
- **match** [**not**] **header** {**length** | **line length**} **gt** *bytes*—Matches messages where the length or length of a line in an ESMTP header is greater than the specified number of bytes.
- **match** [**not**] **header to-fields count gt** *count*—Matches messages where the number of To fields in the header is greater than the specified number.

- **match** [**not**] **invalid-recipients count gt** *number*—Matches messages where the number of invalid recipients is greater than the specified count.
- **match** [**not**] **mime filetype regex** {*regex_name* | **class** *class_name*}—Matches the MIME or media file type against the specified regular expression or regular expression class.
- **match** [**not**] **mime filename length gt** *bytes*—Matches messages where a file name is longer than the specified number of bytes.
- **match** [**not**] **mime encoding** *type* [*type2...*]—Matches the MIME encoding type. You can specify one or more of the following types: 7bit, 8bit, base64, binary, others, quoted-printable.
- **match** [**not**] **sender-address regex** {*regex_name* | **class** *class_name*}—Matches the sender email address against the specified regular expression or regular expression class.
- **match** [**not**] **sender-address length gt** *bytes*—Matches messages where the sender address is greater than the specified number of bytes.

b) Specify the action you want to perform on the matching traffic by entering one of the following commands:

- **drop-connection** [**log**]—Drop the packet and close the connection.
- **mask** [**log**]—Mask out the matching portion of the packet. This action is available for **ehlo-reply-parameter** and **cmd verb** only.
- **reset** [**log**]—Drop the packet, close the connection, and send a TCP reset to the server or client.
- **log**—Send a system log message. You can use this option alone or with one of the other actions.
- **rate-limit** *message_rate*—Limit the rate of messages in packets per second. This option is available with **cmd verb** only, where you can use it as the only action, or you can use it in conjunction with the **mask** action.

You can specify multiple **match** commands in the policy map. For information about the order of **match** commands, see How Multiple Traffic Classes are Handled.

Step 4 To configure parameters that affect the inspection engine, perform the following steps:

a) Enter parameters configuration mode:

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

b) Set one or more parameters. You can set the following options; use the **no** form of the command to disable the option:

- **mail-relay** *domain-name* **action** {**drop-connection** [**log**] | **log**}—Identifies a domain name for mail relay. You can either drop the connection and optionally log it, or log it.
- **mask-banner**—Masks the banner from the ESMTP server.
- **special-character action** {**drop-connection** [**log**] | **log**}—Identifies the action to take for messages that include the special characters pipe (|), back quote, and NUL in the sender or receiver email addresses. You can either drop the connection and optionally log it, or log it.
- **allow-tls** [**action log**]—Whether to allow ESMTP over TLS (encrypted connections) without inspection. You can optionally log encrypted connections. The default is to allow TLS sessions

without inspection. If you specify **no allow-tls**, the system strips the STARTTLS indication from the session connection and forces a plain-text connection.

Example

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
```

What to do next

You can now configure an inspection policy to use the map. See Configure Application Layer Protocol Inspection.

SNMP Inspection

SNMP application inspection is applied to both to-the-device and through-the-device traffic. This inspection is necessary if you configure SNMP v3 where users are limited to specific SNMP hosts. Without the inspection, a defined v3 user can poll the device from any allowed host. SNMP inspection is enabled by default for the default ports, so you need to configure it only if you use non-default ports. The default ports are UDP/161, 162 (for all device types) and UDP/4161 for devices that also run FXOS, as FXOS listens on UDP/161.

By default, the SNMP inspection limits the polling to the configured version.

Note If you configure SNMP on a device that also runs FXOS, SNMP inspection is mandatory, and is re-enabled if you disable it. SNMP inspection is enabled on a traffic class map that includes port UDP/4161.

Optionally, you can further restrict SNMP traffic to a specific version of SNMP. Earlier versions of SNMP are less secure; therefore, denying certain SNMP versions may be required by your security policy. The system

can deny SNMP versions 1, 2, 2c, or 3. You control the versions permitted by creating an SNMP map, as explained below. If you do not need to control the versions, simply enable SNMP inspection without a map.

Procedure

Create an SNMP map.

Use the **snmp-map** *map_name* command to create the map and enter SNMP map configuration mode, then the **deny version** *version* command to identify the versions to disallow. The version can be 1, 2, 2c, or 3.

Example:

The following example denies SNMP Versions 1 and 2:

```
hostname(config)# snmp-map sample_map
hostname(config-snmp-map)# deny version 1
hostname(config-snmp-map)# deny version 2
```

What to do next

You can now configure an inspection policy to use the map. See Configure Application Layer Protocol Inspection.

SQL*Net Inspection

SQL*Net inspection is enabled by default. The inspection engine supports SQL*Net versions 1 and 2, but only the Transparent Network Substrate (TNS) format. Inspection does not support the Tabular Data Stream (TDS) format. SQL*Net messages are scanned for embedded addresses and ports, and NAT rewrite is applied when necessary.

The default port assignment for SQL*Net is 1521. This is the value used by Oracle for SQL*Net, but this value does not agree with IANA port assignments for Structured Query Language (SQL). If your application uses a different port, apply the SQL*Net inspection to a traffic class that includes that port.

Disable SQL*Net inspection when:

- SQL data transfer occurs on the same port as the SQL control TCP port 1521. The security appliance acts as a proxy when SQL*Net inspection is enabled and reduces the client window size from 65000 to about 16000 causing data transfer issues.
- The inspection of high rates of SQL traffic causes unacceptable spikes in CPU usage.

After disabling SQL*Net inspection, use the **clear conn port 1521** command so that connections can be rebuilt without inspection.

For information on enabling SQL*Net inspection, see Configure Application Layer Protocol Inspection.

Sun RPC Inspection

This section describes Sun RPC application inspection.

Sun RPC Inspection Overview

Sun RPC protocol inspection is enabled by default. You simply need to manage the Sun RPC server table to identify which services are allowed to traverse the firewall. However, pinholing for NFS is done for any server even without the server table configuration.

Sun RPC is used by NFS and NIS. Sun RPC services can run on any port. When a client attempts to access a Sun RPC service on a server, it must learn the port that service is running on. It does this by querying the port mapper process, usually rpcbind, on the well-known port of 111.

The client sends the Sun RPC program number of the service and the port mapper process responds with the port number of the service. The client sends its Sun RPC queries to the server, specifying the port identified by the port mapper process. When the server replies, the ASA intercepts this packet and opens both embryonic TCP and UDP connections on that port.

NAT or PAT of Sun RPC payload information is not supported.

Manage Sun RPC Services

Use the Sun RPC services table to control Sun RPC traffic based on established Sun RPC sessions.

Procedure

Step 1 Configure the Sun RPC service properties.

sunrpc-server *interface_name ip_address mask* **service** *service_type* **protocol** {**tcp** | **udp**} *port*[*-port*] **timeout** *hh:mm:ss*

Where:

- *interface_name*—The interface through which traffic to the server flows.
- *ip_address mask*—The address of the Sun RPC server.
- **service** *service_type* —The service type on the server, which is the mapping between a specific service type and the port number used for the service. To determine the service type (for example, 100003), use the **sunrpcinfo** command at the UNIX or Linux command line on the Sun RPC server machine.
- **protocol** {**tcp** | **udp**}—Whether the service uses TCP or UDP.
- *port*[*-port*]—The port or range of ports used by the service. To specify a range of ports, separate the starting and ending port numbers in the range with a hyphen (for example, 111-113).
- **timeout** *hh:mm:ss*—The idle timeout for the pinhole opened for the connection by Sun RPC inspection.

Example:

For example, to create a timeout of 30 minutes to the Sun RPC server with the IP address 192.168.100.2, enter the following command. In this example, the Sun RPC server is on the inside interface using TCP port 111.

```
hostname(config)# sunrpc-server inside 192.168.100.2 255.255.255.255
service 100003 protocol tcp 111 timeout 00:30:00
```

Step 2 (Optional.) Monitor the pinholes created for these services.

To display the pinholes open for Sun RPC services, enter the **show sunrpc-server active** command. For example:

```
hostname# show sunrpc-server active
LOCAL FOREIGN SERVICE TIMEOUT
-----------------------------------------------
1 209.165.200.5/0 192.168.100.2/2049 100003 0:30:00
2 209.165.200.5/0 192.168.100.2/2049 100003 0:30:00
3 209.165.200.5/0 192.168.100.2/647 100005 0:30:00
4 209.165.200.5/0 192.168.100.2/650 100005 0:30:00
```

The entry in the LOCAL column shows the IP address of the client or server on the inside interface, while the value in the FOREIGN column shows the IP address of the client or server on the outside interface.

If necessary, you can clear these services using the **clear sunrpc-server active**

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

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.

For information on enabling TFTP inspection, see Configure Application Layer Protocol Inspection.

XDMCP Inspection

XDMCP is a protocol that uses UDP port 177 to negotiate X sessions, which use TCP when established.

For successful negotiation and start of an XWindows session, the ASA must allow the TCP back connection from the Xhosted computer. To permit the back connection, you can use access rules to allow the TCP ports. Alternatively, you can use the **established** command on the ASA. Once XDMCP negotiates the port to send the display, the **established** command is consulted to verify if this back connection should be permitted.

During the XWindows session, the manager talks to the display Xserver on the well-known port 6000 | n. Each display has a separate connection to the Xserver, as a result of the following terminal setting.

```
setenv DISPLAY Xserver:n
```

where *n* is the display number.

When XDMCP is used, the display is negotiated using IP addresses, which the ASA can NAT if needed. XDCMP inspection does not support PAT.

For information on enabling XDMCP inspection, see Configure Application Layer Protocol Inspection.

VXLAN Inspection

Virtual Extensible Local Area Network (VXLAN) inspection works on VXLAN encapsulated traffic that passes through the ASA. It ensures that the VXLAN header format conforms to standards, dropping any malformed packets. VXLAN inspection is not done on traffic for which the ASA acts as a VXLAN Tunnel End Point (VTEP) or a VXLAN gateway, as those checks are done as a normal part of decapsulating VXLAN packets.

VXLAN packets are UDP, normally on port 4789. This port is part of the default-inspection-traffic class, so you can simply add VXLAN inspection to the inspection_default service policy rule. Alternatively, you can create a class for it using port or ACL matching.

History for Basic Internet Protocol Inspection

Feature Name	Releases	Feature Information
DCERPC inspection support for ISystemMapper UUID message RemoteGetClassObject opnum3.	9.4(1)	The ASA started supporting non-EPM DCERPC messages in release 8.3, supporting the ISystemMapper UUID message RemoteCreateInstance opnum4. This change extends support to the RemoteGetClassObject opnum3 message. We did not modify any commands.
VXLAN packet inspection	9.4(1)	The ASA can inspect the VXLAN header to enforce compliance with the standard format. We introduced the following command: **inspect vxlan**.

Feature Name	Releases	Feature Information
ESMTP inspection change in default behavior for TLS sessions.	9.4(1)	The default for ESMTP inspection was changed to allow TLS sessions, which are not inspected. However, this default applies to new or reimaged systems. If you upgrade a system that includes **no allow-tls**, the command is not changed. The change in default behavior was also made in these older versions: 8.4(7.25), 8.5(1.23), 8.6(1.16), 8.7(1.15), 9.0(4.28), 9.1(6.1), 9.2(3.2) 9.3(1.2), 9.3(2.2).
IP Options inspection improvements.	9.5(1)	IP Options inspection now supports all possible IP options. You can tune the inspection to allow, clear, or drop any standard or experimental options, including those not yet defined. You can also set a default behavior for options not explicitly defined in an IP options inspection map. We added the following commands: **basic-security**, **commercial-security**, **default**, **exp-flow-control**, **exp-measure**, **extended-security**, **imi-traffic-description**, **quick-start**, **record-route**, **timestamp**, and {**0-255**} (which indicates an IP option type number).
DCERPC inspection improvements and UUID filtering	9.5(2)	DCERPC inspection now supports NAT for OxidResolver ServerAlive2 opnum5 messages. You can also now filter on DCERPC message universally unique identifiers (UUIDs) to reset or log particular message types. There is a new DCERPC inspection class map for UUID filtering. We introduced the following command: **match** [**not**] **uuid**. We modified the following command: **class-map type inspect**.
DNS over TCP inspection.	9.6(2)	You can now inspect DNS over TCP traffic (TCP/53). We added the following command: **tcp-inspection**.
Cisco Umbrella support.	9.10(1)	You can configure the device to redirect DNS requests to Cisco Umbrella, so that your Enterprise Security policy defined in Cisco Umbrella can be applied to user connections. You can allow or block connections based on FQDN, or for suspicious FQDNs, you can redirect the user to the Cisco Umbrella intelligent proxy, which can perform URL filtering. The Umbrella configuration is part of the DNS inspection policy. We added or modified the following commands: **umbrella** (global and policy-map parameters configuration modes), **token**, **public-key**, **timeout edns**, **dnscrypt**, **show service-policy inspect dns detail**.

Feature Name	Releases	Feature Information
Cisco Umbrella Enhancements.	9.12(1)	You can now identify local domain names that should bypass Cisco Umbrella. DNS requests for these domains go directly to the DNS servers without Umbrella processing. You can also identify which Umbrella servers to use for resolving DNS requests. Finally, you can define the Umbrella inspection policy to fail open, so that DNS requests are not blocked if the Umbrella server is unavailable. We added or changed the following commands: **local-domain-bypass**, **resolver**, **umbrella fail-open**.
XDMCP inspection disabled by default in new installations.	9.15(1)	Previously, XDMCP inspection was enabled by default for all traffic. Now, on new installations, which includes new systems and reimaged systems, XDMCP is off by default. If you need this inspection, please enable it. Note that on upgrades, your current settings for XDMCP inspection are retained, even if you simply had it enabled by way of the default inspection settings.