Advanced Configurations for the SSL Services Module

This chapter describes the following advanced configurations:

Configuring Policies

See the "Configuring SSL Proxy Services" section on page 3-45 for procedures for applying policies to a proxy service.

This section describes how to configure SSL and TCP policies:

Configuring SSL Policy

Note The SSL commands for the SSL Services Module apply either globally or to a particular proxy server.

The SSL policy template allows you to define parameters associated with the SSL stack.

One of the parameters you can configure is the SSL close-protocol behavior. The SSL close-protocol specifies that each of the SSL peers (client and server) should send a close-notify alert and receive a close-notify alert before closing the connection properly. If the SSL connection is not closed properly, the session is removed so that the peers cannot use same SSL session ID in future SSL connections.

However, many SSL implementations do not follow the SSL close-protocol strictly (for example, an SSL peer sends a close-notify alert but does not wait for the close-notify alert from the remote SSL peer before closing the connection).

When an SSL peer initiates the close connection sequence, the SSL Services Module strictly expects a close-notify alert message. If an SSL peer does not send a close-notify alert, SSL Services Module removes the session from the session cache so that the same session-id cannot be used for future ssl connections.

When the SSL Services Module initiates the close connection sequence, you can configure the following close-protocol options:

- **strict—**The SSL Services Module sends a close-notify alert message to the SSL peer, and the SSL Services Module expects a close-notify alert message from the SSL peer. If the SSL Services Module does not receive a close-notify alert, SSL resumption is not allowed for that session.
- **none**—The SSL Services Module does not send a close-notify alert message to the SSL peer, nor does the SSL Services Module expect a close-notify alert message from the SSL peer. The SSL Services Module preserves the session information so that SSL resumption can be used for future SSL connections.
- disabled (default)—The SSL Services Module sends a close-notify alert message to the SSL peer; however, the SSL peer does not expect a close-notify alert before removing the session. Whether SSL peer sends close-notify or not, the session information is preserved allowing session resumption for future SSL connections.

If you do not associate an SSL policy with a particular proxy server, the proxy server enables all the supported cipher suites and protocol versions by default.

To define an SSL policy template and associate an SSL policy with a particular proxy server, perform this task:

	Command	Purpose
Step 1	`ssl-proxy(config)# ssl-proxy context` *name*	Enters the SSL context subcommand mode. The optional SSL context name name is used to specify an SSL virtualization instance.
Step 2	`ssl-proxy(config-context)# policy ssl` *ssl_policy_name*	Defines SSL policy templates.

	Command	Purpose
Step 3	`ssl-proxy(config-ctx-ssl-policy)#` **`cipher`** *ciphersuite*	Configures a list of cipher-suite names acceptable to the proxy server. The cipher-suite names follow the same convention as that of existing SSL stacks. The default setting is **all-strong**. Valid values are as follows: • **all** • **all-export** • **all-strong** (default) • **rsa-exp-with-des40-cbc-sha** • **rsa-exp-with-rc4-40-md5** • **rsa-exp1024-with-des-cbc-sha** • **rsa-exp1024-with-rc4-56-md5** • **rsa-exp1024-with-rc4-56-sha** • **rsa-with-3des-ede-cbc-sha** • **rsa-with-des-cbc-sha** • **rsa-with-null-md5** • **rsa-with-rc4-128-md5** • **rsa-with-rc4-128-sha**
Step 4	`ssl-proxy(config-ctx-ssl-policy)#` **`version`** {**`ssl3`** | **`tls1`** | **`all`**}	Defines the various protocol versions supported by the proxy server.
Step 5	`ssl-proxy(config-ctx-ssl-policy)#` **`close-protocol`** {**`strict`** | **`none`**}	Configures the SSL close-protocol behavior. Close-protocol is disabled by default.
Step 6	`ssl-proxy(config-ctx-ssl-policy)#` **`session-cache`**	Enables the session-caching feature. Session caching is enabled by default.
Step 7	`ssl-proxy(config-ctx-ssl-policy)#` **`timeout handshake`** *time*	Configures how long the module keeps the connection in handshake phase. The valid range is 0 to 65535 seconds.
Step 8	`ssl-proxy(config-ctx-ssl-policy)#` **`timeout session`** *timeout* [**`absolute`**[1]]	Configures the amount of time that an entry is kept in the session cache. The valid range is 1 to 72000 seconds. **Note** The **absolute** keyword is required in order to configure session-cache size. **Note** The **absolute** keyword specifies that the session entry is kept in the session cache for the specified *timeout*. When the **absolute** keyword is specified, new incoming connections are rejected if there are no free entries available in the session cache.
Step 9	`ssl-proxy(config-ctx-ssl-policy)#` **`session-cache size`** *size*	(Optional) Specifies the size of the session cache[1]. The valid range is 1 to 262143 entries. **Note** Specify the session cache size when you enter the **absolute** keyword with the **timeout session** command. If this command is not entered or if no *size* is specified, the session cache size is the maximum size (262,144).

	Command	Purpose
Step 10	`ssl-proxy(config-ctx-ssl-policy)#` **`cert-req empty`**	Specifies that the SSL Services Module does not look for a CA-name match before returning a certificate. Enter this command if the backend SSL servers do not include CA name list in the certificate request during server authentication. **Note** By default, the SSL Services Module always looks for the CA name match before returning the certificate. If the SSL server does not include a CA-name list in the certificate request during client authentication, handshake will fail.
Step 11	`ssl-proxy(config-ctx-ssl-policy)#` **`tls-rollback`** [**`current`** | **`any`**]	Specifies the version of SSL protocol (SSL2.0, SSL3.0, TLS1.0) in the ClientHello message. TLS-rollback is disabled by default. When you configure the **current** keyword, the SSL protocol version can be either the maximum supported version or the negotiated version. When you configure the **any** keyword, the SSL protocol version is not checked at all. **Note** By default, the SSL Services Module uses the maximum supported version. Enter this command if the SSL client uses the negotiated version instead of the maximum supported version (as specified in the ClientHello message).
Step 12	`ssl-proxy(config-ctx-ssl-policy)#` **`renegotiation volume`** *`size`*	Enables autorenegotiation and specifies the data volume size (in kilobytes). When the encrypted or decrypted data amount exceeds this size, the SSL Services Module sends a renegotiation request. This setting is disabled by default. The valid range is from 1024 to 1073741824 kilobytes.
Step 13	`ssl-proxy(config-ctx-ssl-policy)#` **`renegotiation interval`** *`time`*	Enables autorenegotiation and specifies the interval (in seconds). After the set interval, the SSL Services Module sends an renegotiation request. This setting is disabled by default. The valid range is from 60 to 86400 seconds.
Step 14	`ssl-proxy(config-ctx-ssl-policy)#` **`renegotiation wait-time`** *`time`*	(Optional) When autorenegotiation is enabled, this command specifies the amount of time (in seconds) that the SSL Services Module waits for the peer to respond to the renegotiation request. The default is 100 seconds. The valid range is from 10 to 300 seconds.
Step 15	`ssl-proxy(config-ctx-ssl-policy)#` **`renegotiation optional`**	(Optional) When autorenegotiation is enabled, the SSL Services Module allows the session to continue if the peer does not respond to the renegotiation request after timeout. This setting is disabled by default and the session is disconnected after timeout.

	Command	Purpose
Step 16	`ssl-proxy(config-ctx-ssl-policy)#` **`exit`**	Returns to config mode.
Step 17	`ssl-proxy(config-context)#` **`service`** *service_name*	Defines the name of the SSL proxy service. See the "Configuring SSL Proxy Services" section on page 3-45 for more information about configuring SSL proxy services. **Note** The *service_name* value is case-sensitive.
Step 18	`ssl-proxy(config-ctx-ssl-proxy)#` **`virtual policy ssl`** *ssl_policy_name*	Applies the SSL policy to the client side of the proxy server.

1. When the **absolute** keyword is configured, the session entry is not reused until the configured session timeout expires. When **absolute** is configured, the number of session entries required is equal to (new_connection_rate * absolute_timeout). Depending on the timeout configuration and the new connection rate, the number of session entries might be very large. In this case, you can limit the number of session entries used by configuring the session-cache size.

Configuring TCP Policy

Note The TCP commands for the SSL Services Module apply either globally or to a particular proxy server.

The TCP policy template allows you to define parameters associated with the TCP stack.

To define an TCP policy template and associate an TCP policy with a particular proxy server, perform this task:

	Command	Purpose
Step 1	`ssl-proxy(config)#` **`ssl-proxy context`** *name*	Enters the SSL context subcommand mode. The optional SSL context name name is used to specify an SSL virtualization instance.
Step 2	`ssl-proxy(config-context)#` **`policy tcp`** *tcp_policy_name*	Defines TCP policy templates. All defaults are assumed unless otherwise specified.
Step 3	`ssl-proxy(config-ctx-tcp-policy)#` **`mss`** *max_segment_size*	Configures the maximum segment size (MSS), in bytes, that the connection will identify in the SYN packet that it generates. **Note** This command allows you to configure a different MSS for the client side and server side of the proxy server. The default is 1460 bytes. The valid range is from 256 to 2460 bytes[1].
Step 4	`ssl-proxy(config-ctx-tcp-policy)#` **`timeout syn`** *time*	Configures the connection establishment timeout. The default is 75 seconds. The valid range is from 5 to 75 seconds.
Step 5	`ssl-proxy(config-ctx-tcp-policy)#` **`timeout reassembly`** *time*	Configures the amount of time, in seconds, before the reassembly queue is cleared. If the transaction is not complete within the specified time, the reassembly queue is cleared and the connection is dropped. The default is 60 seconds. The valid range is 0 (disabled) to 960 seconds.

	Command	Purpose
Step 6	`ssl-proxy(config-ctx-tcp-policy)#` **`timeout inactivity`** *time*	Configures the amount of time, in seconds, that an established connection can be inactive. The default is 600 seconds. The valid range is 0 (disabled) to 7200 seconds (2 hours).
Step 7	`ssl-proxy(config-ctx-tcp-policy)#` **`timeout fin-wait`** *time*	Configures the FIN wait timeout in seconds. The default value is 600 seconds. The valid range is from 75 to 600 seconds.
Step 8	`ssl-proxy(config-ctx-tcp-policy)#` **`timeout persist`** *time*	Configures the persist wait timeout in seconds. The default value is 0 seconds. The valid range is from 5 to 7200 seconds.
Step 9	`ssl-proxy(config-ctx-tcp-policy)#` **`buffer-share rx`** *buffer_limit*	Configures the maximum receive buffer share per connection in bytes. The default value is 32768 bytes. The valid range is from 8192 to 262144 bytes.
Step 10	`ssl-proxy(config-ctx-tcp-policy)#` **`buffer-share tx`** *buffer_limit*	Configures the maximum transmit buffer share per connection in bytes. The default value is 32768 bytes. The valid range is from 8192 to 262144 bytes.
Step 11	`ssl-proxy(config-ctx-tcp-policy)#` **`forced-ack`**	Enable the forced-ACK algorithm.
Step 12	`ssl-proxy(config-ctx-tcp-policy)#` **`delayed-ack-threshold`** *delay*	Configures the delayed ACK threshold. The default is 2. The valid range is from 1 to 10.
Step 13	`ssl-proxy(config-ctx-tcp-policy)#` **`delayed-ack-timeout`** *timer*	Configures the delayed ACK timeout. The default is 200 seconds. The valid range is from 50 to 500 seconds.
Step 14	`ssl-proxy(config-ctx-tcp-policy)#` **`tos carryover`**	Forwards the type of service (ToS) value to all packets within a flow. **Note** If the policy is configured as a server TCP policy, the ToS value is sent from the server to the client. If the policy is configured as a virtual policy, the ToS value is sent from the client to the server. **Note** The ToS value needs to be learned before it can be propagated. For example, when a ToS value is configured to be propagated from the server to client connection, the server connection must be established before the value is learned and propagated. Therefore, some of the initial packets will not carry the ToS value.
Step 15	`ssl-proxy(config-ctx-tcp-policy)#` **`[no] nagle`**	Enables or disables the Nagle algorithm. Nagle is enabled by default.
Step 16	`ssl-proxy(config-ctx-tcp-policy)#` **`exit`**	Returns to config mode.
Step 17	`ssl-proxy(config-context)#` **`service`** *service_name*	Defines the name of the SSL proxy service. See the "Configuring SSL Proxy Services" section on page 3-45 for more information about configuring SSL proxy services. **Note** The *service_name* value is case-sensitive.
Step 18	`ssl-proxy(config-ctx-ssl-proxy)#` {**`virtual`**|**`server`**} **`policy tcp`** *tcp_policy_name*	Applies the TCP policy to the client (virtual) side or the server side of the proxy server.

1. If fragmentation occurs, decrease the MSS value until there is no fragmentation.

Note When large encrypted files are transferred by the module, the transmit and receive buffer sizes must be at least the maximum SSL record size of 16384 bytes for reassembly of the SSL record. For the **buffer-share rx** and **buffer-share tx** commands, we recommend transmit and receive share sizes of at least 20000 bytes for optimal performance.

Examples of situations requiring this setting are:
- When a service is configured for backend server encryption and the server sends large files.
- When the client-side is encrypted and a client posts large files to the server.

HTTP Header Insertion

In a typical SSL offloading environment, an SSL offloader terminates secure client HTTP (HTTPS) connections, decrypts the SSL traffic into cleartext, and forwards the cleartext to a Web server through an HTTP connection. The HTTPS connections become non-secure HTTP connections at the backend server. The backend server does not know that the client connection came in as a secure connection.

Here are a few reasons to configure HTTP header insertion:

- HTTP header insertion allows the SSL Services Module to embed information into an HTTP header during a client connection. When the backend server recognizes this header, the server returns all the URLs as HTTPS.
- You can have a backend application that logs information per connection by configuring an SSL offloader to insert the client certificate information into the HTTP header received from the client.
- When you use the SSL Services Module in a site-to-site configuration to send traffic over a secured channel, the server end of the connection may need to know the client IP address and port information, which gets removed during NAT.

HTTP header insertion is performed for the following methods: GET, HEAD, PUT, TRACE, POST, DELETE. HTTP header insertion is not performed for the CONNECT method.

Custom headers and client IP and port headers are inserted in every HTTP request packet. Full session headers and decoded client certificate fields and the full certificate in PEM format are inserted in the first HTTP request packets; only the session ID is inserted in subsequent HTTP requests that use the same session ID. Servers are expected to cache the session or client certificate headers and the full certificate in PEM format based on the session ID and use the session ID in subsequent requests to get the session and client certificate headers.

You can configure up to 100 HTTP header insertion policies, each policy consisting of 1 prefix and 16 custom headers. Prefix and custom headers can include up to 239 characters.

The information that can be inserted in the HTTP header is described in the following sections:

- Prefix, page 4-8
- Client Certificate Headers, page 4-8
- Client IP and Port Address Headers, page 4-9
- Custom Headers, page 4-9
- Header Alias, page 4-9
- SSL Session Headers, page 4-9

Prefix

When you specify **prefix** *prefix_string*, the SSL Services Module adds the specified prefix to every inserted HTTP header. Adding a prefix enables the server to identify connections as coming from the SSL Services Module, and not from other appliances. A prefix is not added to standard HTTP headers from the client. The *prefix_string* can be up to 239 characters.

Client Certificate Headers

Client certificate header insertion allows the backend server to see the attributes of the client certificate that the SSL Services Module has authenticated and approved. Client certificate headers are sent only once per session. The server is expected to cache these values using the session ID, which is also inserted with the headers. In subsequent requests, the server uses the session ID to look up the cached client certificate headers on the server itself.

Note If the client does not send a certificate, the SSL handshake fails. There is no data phase or header insertion.

Note You can insert the headers listed below by entering the **client-cert** command, or you can send the entire client certificate in PEM format by entering the **client-cert pem** command.

Note The client certificate headers, or the client certificate in PEM format, are inserted only if the policy's service is configured for client authentication. The root CA and intermediate CA certificates will not be inserted when client certificate is inserted in the HTTP header.

When you specify **client-cert**, the SSL Services Module passes the following headers to the backend server.

Field To Insert	Description
ClientCert-Valid	Certificate validity state
ClientCert-Error	Error conditions
ClientCert-Fingerprint	Hash output
ClientCert-Subject-CN	X.509 subject's common name
ClientCert-Issuer-CN	X.509 certificate issuer's common name
ClientCert-Certificate-Version	X.509 certificate version
ClientCert-Serial-Number	Certificate serial number
ClientCert-Data-Signature-Algorithm	X.509 hashing and encryption method
ClientCert-Subject	X.509 subject's distinguished name
ClientCert-Issuer	X.509 certificate issuer's distinguished name
ClientCert-Not-Before	Certificate is not valid before this date
ClientCert-Not-After	Certificate is not valid after this date
ClientCert-Public-Key-Algorithm	The algorithm used for the public key

Field To Insert	Description
ClientCert-RSA-Public-Key-Size	Size of the RSA public key
ClientCert-RSA-Modulus-Size	Size of the RSA private key
ClientCert-RSA-Modulus	RSA modulus
ClientCert-RSA-Exponent	The public RSA exponent
ClientCert-X509v3-Authority-Key-Identifier	X.509 authority key identifier
ClientCert-X509v3-Basic-Constraints	X.509 basic constraints
ClientCert-X509v3-Key-Usage	X.509 key usage
ClientCert-X509v3-Subject-Alternative-Name	X.509 subject alternative name
ClientCert-X509v3-CRL-Distribution-Points	X.509 CRL distribution points
ClientCert-X509v3-Authority-Information-Access	X.509 authority information access
ClientCert-Signature-Algorithm	Certificate signature algorithm
ClientCert-Signature	Certificate signature

Client IP and Port Address Headers

Network address translation (NAT) changes the client IP address and destination TCP port number information. When you specify **client-ip-port,** the SSL Services Module inserts the client IP address and TCP destination port information in the HTTP header, allowing the server to see the client IP address and destination port number.

Custom Headers

When you specify **custom** *custom_string*, the SSL Services Module inserts the user-defined header verbatim in the HTTP header. You can configure up to 16 custom headers per HTTP header policy. The *custom_string* can include up to 239 characters. If the string includes spaces, you must enclose it in quotes (“”).

Note The syntax for *custom_string* is in the form *name*:*value*. For example:

```
ssl-proxy(config-ctx-http-header-policy)# custom "SOFTWARE VERSION:3.1(1)"
```

Header Alias

Some applications use different names for the standard certificate header. You can create an alias for the standard name of the header so that the same value is passed using the aliased name instead of the standard name that the SSL Services Module sends. If you have specified a prefix for header insertion, the prefix is also applied to the aliased name.

SSL Session Headers

Session headers, including the session ID, are used to cache client certificates based on the session ID. Session headers are also cached based on the session ID if the server wants to track connections based on a particular cipher suite. The SSL Services Module inserts the full session headers in the HTTP request during full SSL handshake, but inserts only the session ID when the session resumes.

When you configure the SSL Services Module as a client, the module inserts the session ID of the connection between the module and the backend SSL server.

When you specify **session**, the SSL Services Module passes information specific to an SSL connection to the backend server in the form of the following session headers.

Field to insert	Description
Session-Id	The SSL session ID
Session-Cipher-Name	The symmetric cipher suite
Session-Cipher-Key-Size	The symmetric cipher key size
Session-Cipher-Use-Size	The symmetric cipher use size
Session-Step-Up	TRUE if the server presented a stepup certificate and the client renegotiated the cipher; otherwise FALSE
Session-Initial-Cipher-Name	If Session-Step-Up is TRUE, the initially negotiated cipher name
Session-Initial-Cipher-Key-Size	If Session-Step-Up is TRUE, the initially negotiated cipher's key size
Session-Initial-Cipher-Use-Size	If Session-Step-Up is TRUE, the initially negotiated cipher’s use size

Configuring HTTP Header Insertion

To configure HTTP header insertion, perform this task:

	Command	Purpose
Step 1	`ssl-proxy(config)# ssl-proxy context` *name*	Enters the SSL context subcommand mode. The optional SSL context name name is used to specify an SSL virtualization instance
Step 2	`ssl-proxy(config-context)# policy http-header` *policy_name*	Configures HTTP header insertion.
Step 3	`ssl-proxy(config-ctx-http-header-policy)# {prefix` *prefix_string* `| client-cert [pem]| client-ip-port | custom` *custom_string*`} | session | alias` *user-defined-name standard-name*`]`	Specifies the prefix, type of header, or alias name of header. **Note** You can configure only one alias per standard name. You cannot configure the same alias name for multiple standard names.
Step 4	`ssl-proxy(config-ctx-http-header-policy)# exit`	Returns to context subcommand mode.
Step 5	`ssl-proxy(config-context)# service` *service_name*	Defines the name of the SSL proxy service. **Note** The *service_name* value is case-sensitive.
Step 6	`ssl-proxy(config-ctx-ssl-proxy)# policy http-header` *http_header_policy_name*	Applies the HTTP header policy to the proxy server, for request.

The following example shows how to configure the SSL Services Module to insert a prefix, session headers, custom headers, and a header alias:

```
ssl-proxy(config)# ssl-proxy context s1
ssl-proxy(config-context)# policy http-header ssl-offload
ssl-proxy(config-ctx-http-header-policy)# prefix SSL-OFFLOAD
ssl-proxy(config-ctx-http-header-policy)# session
ssl-proxy(config-ctx-http-header-policy)# custom "SOFTWARE VERSION:3.1(1)"
ssl-proxy(config-ctx-http-header-policy)# custom "module:SSL MODULE - CATALYST 6500"
ssl-proxy(config-ctx-http-header-policy)# custom
type-of-proxy:server_proxy_1024_bit_key_size
ssl-proxy(config-ctx-http-header-policy)# alias My-Session-Cipher session-cipher-name
ssl-proxy(config-ctx-http-header-policy)# exit
ssl-proxy(config-context)# ssl-proxy service ssl-offload
ssl-proxy(config-ctx-ssl-proxy)# policy http-header ssl-offload
```

In addition to the standard HTTP headers, the following header information is inserted:

Note The alias name (My-Session-Cipher) is used instead of the standard name (session-cipher-name).

```
SSL-OFFLOAD-SOFTWARE VERSION:3.1(1)
SSL-OFFLOAD-module:SSL MODULE - CATALYST 6500
SSL-OFFLOAD-type-of-proxy:server_proxy_1024_bit_key_size
SSL-OFFLOAD-Session-Id:33:FF:2C:2D:25:15:3C:50:56:AB:FA:5A:81:0A:EC:E9:00:00:0A:03:00:60:
 2F:30:9C:2F:CD:56:2B:91:F2:FF
SSL-OFFLOAD-My-Session-Cipher:RC4-SHA
SSL-OFFLOAD-Session-Cipher-Key-Size:128
SSL-OFFLOAD-Session-Cipher-Use-Size:128
SSL-OFFLOAD-Session-Step-Up:FALSE
SSL-OFFLOAD-Session-Initial-Cipher-Key-Size:
SSL-OFFLOAD-Session-Initial-Cipher-Name:
SSL-OFFLOAD-Session-Initial-Cipher-Use-Size:
```

Configuring URL Rewrite

In a typical SSL offloading environment, an SSL offloader terminates secure client HTTP (HTTPS) connections, decrypts the SSL traffic into cleartext, and forwards the cleartext to a Web server through an HTTP connection. The HTTPS connections become non-secure HTTP connections at the backend server. The backend server doesn't know that the client connection came in as a secure connection.

If data returned to the client contains an HTTP redirection link, and the client follows this link, the client leaves the secure domain and no longer has a secure connection. The redirected link may not be available from the server using a clear text connection.

You can avoid problems with nonsecure HTTP redirects from the backend server by configuring one or more URL rewrite rules. Each rewrite rule is associated with a service in the SSL proxy list. URL rewrite rules resolve the problem of a web site redirecting you to a nonsecure HTTP URL by rewriting the domain from http:// to https://. By configuring URL rewrite, all client connections to the Web server are SSL connections, ensuring the secure delivery of HTTPS content back to the client.

Note The URL rewrite feature supports the rewriting of redirection links. The system scans only the "Location:" HTTP header field in the response from the server and rewrites the rules accordingly. The URL rewrite feature does not support embedded links.

The URL rewrite feature rewrites the protocol and the nondefault port (default ports are port 80 for cleartext and port 443 for SSL).

You can configure up to 100 URL rewrite policies, each policy consisting of up to 32 rewrite rules per SSL proxy service, up to 200 characters per rule.

Follow these guidelines for URL rewrite:

- An exact URL match takes precedence over a wildcard rule. A suffix wildcard rule takes precedence over a prefix wildcard rule.

  For example, **www.cisco.com** takes precedence, then **www.cisco.***, then ***.cisco.com**.
- Enter only one suffix or prefix wildcard rule at one time. For example, do not enter **www.cisco.*** and **www.cisco.c*** in the same policy. Similarly, do not enter ***w.cisco.com** and ***.cisco.com** in the same policy.
- Do not enter two exact URL match rules in the same policy. For example, do not enter **www.cisco.com clearport 80 sslport 443** and **www.cisco.com clearport 81 sslport 444** in the same policy. In this case, the second rule entered overwrites the first rule.
- URL rewrite is performed for both offload and backend (HTTP-to-HTTPS, and HTTPS-to-HTTP). This includes port rewrites.

To configure URL rewrite, perform this task:

	Command	Purpose
Step 1	`ssl-proxy(config)# ssl-proxy context` *name*	Enters the SSL context subcommand mode. The optional SSL context name name is used to specify an SSL virtualization instance.
Step 2	`ssl-proxy(config-context)# policy url-rewrite` *policy_name*	Configures the URL rewrite policy.
Step 3	`ssl-proxy(config-ctx-url-rewrite-policy)# url` *url* `[clearport` *port_number*[1]`] [sslport` *port_number*[2]`]`	Specifies the URL rewrite rules. You can configure up to 32 rewrite rules per SSL proxy service, up to 240 characters per rule. **Note** You should enter only one suffix or prefix wildcard character (*) only once per rewrite rule.
Step 4	`ssl-proxy(config-ctx-url-rewrite-policy)# exit`	Returns to config mode.
Step 5	`ssl-proxy(config-context)# service` *service_name*	Defines the name of the SSL proxy service. **Note** The *service_name* value is case-sensitive.
Step 6	`ssl-proxy(config-ctx-ssl-proxy)# policy url-rewrite` *policy_name*	Applies the URL rewrite policy.

1. The **clearport** *port_number* specifies the port portion of the URL to be rewritten. Specify the **cleartext** *port_number* if it is not the default cleartext port 80.
2. The **sslport** *port_number* specifies the port portion of the URL that should be rewritten. Specify the **ssltext** *port_number* if it is not the default SSL port 443.

Note When a server includes the default HTTP port number 80 in a URL redirect (for example, www.example.com:80), then the **url** command must be configured in the same manner (for example, **url www.example.com:80)**. Non-standard port numbers need not be configured as part of the URL, but may instead by configured using the **clearport** keyword.

This example shows how to configure URL rewrite policy and apply the policy to a proxy service:

```
ssl-proxy(config)# ssl-proxy context s1
ssl-proxy(config-context)# policy url-rewrite cisco_url
ssl-proxy(config-ctx-url-rewrite-policy)# url www.cisco.*
ssl-proxy(config-ctx-url-rewrite-policy)# url www.cisco.com clearport 81 sslport 444
ssl-proxy(config-ctx-url-rewrite-policy)# url wwwin.cisco.com clearport 81 sslport 440
ssl-proxy(config-ctx-url-rewrite-policy)# url 10.1.1.10 clearport 81 sslport 444
ssl-proxy(config-ctx-url-rewrite-policy)# exit
ssl-proxy(config-context)# service cisco_service
ssl-proxy(config-ctx-ssl-proxy)# policy url-rewrite cisco_url
```

See Table 4-1 for examples that show URL rewrite.

Table 4-1 Rules and Outcome for Server Proxy

URL Rewrite Rule	URLs that Match	URL Rewrite:
url www.cisco.com	http://www.cisco.com/	https://www.cisco.com/
url www.cisco.com clearport 81	http://www.cisco.com:81/	https://www.cisco.com/
url www.cisco.com sslport 444	http://www.cisco.com/	https://www.cisco.com:444/
url www.cisco.com clearport 81 sslport 444	http://www.cisco.com:81/	https://www.cisco.com:444/

Health Probe

You can configure the SSL Services Module to probe a server to detect a server failure. When a server failure is detected, the SSL service relays the failure condition to the SLB device.

To configure the TCP health probe, perform this task:

	Command	Purpose
Step 1	`ssl-proxy(config-context)# policy health-probe tcp` *name*	Configures the health probe policy.
Step 2	`ssl-proxy(config-ctx-tcp-probe)# interval` *seconds*	(Optional) Sets the interval between probes in seconds (from the end of the previous probe to the beginning of the next probe) when the server is healthy. The default is 30 seconds. The valid range is from 30 to 300 seconds.
Step 3	`ssl-proxy(config-ctx-tcp-probe)# failed-interval` *seconds*	(Optional) Sets the time between health checks after the service has been marked as failed. The default is 60 seconds. The valid range is from 30 to 3600 seconds.
Step 4	`ssl-proxy(config-ctx-tcp-probe)# maximum-retry` *retries*	(Optional) Sets the number of failed probes that are allowed before marking the service as failed. The default is 0 retries. The valid range is from 1 to 5 retries.

	Command	Purpose
Step 5	`ssl-proxy(config-ctx-tcp-probe)#` **`open-timeout`** *seconds*	(Optional) Sets the maximum time to wait to establish a TCP connection. The default is 80 seconds. The valid range is from 70 to 120 seconds.
Step 6	`ssl-proxy(config-ctx-tcp-probe)#` **`port`** *port_number*	(Optional) Configures an optional port for the health probe. Valid values are from 1 to 65535. By default, the TCP health probe uses the server IP address and port for the SSL server proxy service. Enter the **port** command to specify a different port for the health probe. If you configured the SSL server proxy service with **no nat server**, the TCP health probe uses the virtual IP address that you configured on the SSL server proxy service instead of the server IP address. **Note** TCP health probe is not supported when you configure a wildcard proxy and **no nat server** on the SSL server proxy service. See the "SSL Server Proxy Services" section on page 3-45 for information about configuring the SSL server proxy service.

The following example shows how to configure TCP health probe to check whether service at port 81 (server port) is up and running on server IP address 19.0.0.1

```
ssl-proxy(config)# ssl-proxy context ssl
ssl-proxy(config-context)# service ssl-1
ssl-proxy(config-ctx-ssl-proxy)# virtual ipddr 7.100.100.180 protocol tcp port 443
ssl-proxy(config-ctx-ssl-proxy)# server ipaddr 19.0.0.1 protocol tcp port 81
ssl-proxy(config-ctx-ssl-proxy)# certificate rsa general-purpose trustpoint cert1024
ssl-proxy(config-ctx-ssl-proxy)# policy health-probe tcp probe1
ssl-proxy(config-ctx-ssl-proxy)# inservice
ssl-proxy(config-ctx-ssl-proxy)# exit
ssl-proxy(config-context)# policy health-probe tcp probe1
ssl-proxy(config-ctx-tcp-probe)# end
ssl-proxy#
```

The following example shows the state of the SSL proxy service when the health probe has failed:

Note The proxy service is down until service at port 81 is up and running again.

```
ssl-proxy# show ssl-proxy service ssl-1 context ssl
Service id: 0, bound_service_id: 256
Virtual IP: 7.100.100.180, port: 443
Server IP: 19.0.0.1, port: 81
TCP Health Probe Policy: probe1
rsa-general-purpose certificate trustpoint: cert1024
  Certificate chain for new connections:
    Certificate:
       Key Label: cert1024.key, 1024-bit, exportable
       Key Timestamp: 05:18:23 UTC Dec 30 2005
       Serial Number: 12F332E200000000000D
    Root CA Certificate:
       Serial Number: 6522F512C30E078447D8AFC35567B101
```

```
  Certificate chain complete
Context name: ssl
Context Id  : 1
Admin Status: up
Operation Status: down
Proxy status: Health Probe Failed
```

Configuring NAT

Client connections originate from the client and are terminated on the SSL Services Module. Server connections originate from the SSL Services Module.

You can configure client NAT, server NAT, or both, on the server connection.

Server NAT

The server IP address configured with the **ssl-proxy service** command specifies the IP address and port for the destination device, either the CSM or the real server for which the SSL Services Module acts as a proxy. If you configure server NAT, the server IP address is used as the destination IP address for the server connection. If the server NAT is not configured, the destination IP address for the server connection is the same as the **virtual ipaddress** for which SSL Services Module is a proxy. The server IP address points to a next hop router IP address or a virtual IP address on the CSM. The SSL Services Module always performs the port translation by using the port number entered in the **server ipaddress** subcommand.

To configure server NAT, enter the **nat server** subcommand under the **ssl-proxy service** command:

	Command	Purpose
Step 1	`ssl-proxy(config)# ssl-proxy context` *name*	Enters the SSL context subcommand mode. The optional SSL context name name is used to specify an SSL virtualization instance.
Step 2	`ssl-proxy(config-context)# service` *service_name*	Defines the name of the SSL proxy service. **Note** The *service_name* value is case-sensitive.
Step 3	`ssl-proxy(config-ctx-ssl-proxy)# nat server` *natpool_name*	Enables a NAT server address for the server connection of the specified service SSL offload.

Client NAT

If you configure client NAT, the server connection source IP address and port are derived from a NAT pool. If client NAT is not configured, the server connection source IP address and port are derived from the source IP address and source port of the client connection.

Allocate enough IP addresses to satisfy the total number of connections supported by the SSL Services Module (256,000 connections). Assuming you have 32,000 ports per IP address, configure 8 IP addresses in the NAT pool. If you try to configure fewer IP addresses than required by the total connections supported by the SSL Services Module, the command is rejected.

Note The addresses specified in the **natpool** command must match the address of one of the connected networks configured on the SSL Services Module subinterfaces.

To configure a NAT pool and assign the NAT pool to the proxy service, perform this task:

	Command	Purpose
Step 1	`ssl-proxy(config)# ssl-proxy context` *name*	Enters the SSL context subcommand mode. The optional SSL context name name is used to specify an SSL virtualization instance.
Step 2	`ssl-proxy(config-context)# natpool` *natpool_name start_ip_addr end_ip_addr* `netmask` *netmask*	Defines a pool of IP addresses that the SSL Services Module uses for implementing the client NAT.
Step 3	`ssl-proxy(config-context)# service` *service_name*	Defines the name of the SSL proxy service. **Note** The *service_name* value is case-sensitive.
Step 4	`ssl-proxy(config-ctx-ssl-proxy)# nat client` *natpool_name*	Configures a NAT pool for the client address used in the server connection of the specified service SSL offload.

Configuring Redundancy

In systems with an SSL Services Module and a Content Switching Module (CSM), the failover functionality on the CSM provides stateless redundancy on the SSL Services Module. When the SSL Services Module is used in a standalone configuration (using policy-based routing), you can configure HSRP to provide redundancy.

When configuring HSRP, note the following restrictions:

- You can configure up to 16 HSRP groups per SSL Services Module.
- You can configure HSRP for an SSL Services Module or for individual proxy services.
- Configuration changes in one module do not automatically propagate to the redundant module. You must manually apply configuration changes to all SSL Services Modules in the redundant system.
- The SSL Services Module software provides only stateless redundancy. When a standby module takes over the functionality of the active module, the existing connections are lost. New connections are established on the standby (now active) module using the same configuration available on the active module.

Before you configure HSRP on the SSL Services Module, do the following:

1. Configure proxy services. Enter the **secondary** keyword when you specify the virtual IP address.

   See the "Configuring SSL Proxy Services" section on page 3-45 for information on configuring proxy services.

2. Configure client NAT on the proxy service.

   See the "Client NAT" section on page 4-15 for information on configuring client NAT.

3. Configure policy-based routing. The next hop IP address is the standby IP address specified below.

   See the "Configuring Policy-Based Routing" section on page 5-2 or information on configuring policy-based routing.

To configure HSRP on the SSL Services Module, perform this task:

	Command	Purpose
Step 1	`ssl-proxy(config)# interface ssl-proxy 0.`*subinterface-number*	Selects a subinterface to configure.
Step 1	`ssl-proxy(config-subif)# standby` [*group_number*] `ip` *ip_address*	Enable HSRP and specify the HSRP IP address. If you do not specify a *group_number*, group 0 is used.
Step 2	`ssl-proxy(config-subif)# standby` [*group_number*] `priority` *priority*	(Optional) Specify the priority for the HSRP group on this module. The default *group_number* is 100. The module with the highest priority becomes active for that HSRP group.
Step 3	`ssl-proxy(config-subif)# standby` [*group_number*] [`preempt`] [`delay` *delay*]	(Optional) Configure the group to preempt the current active HSRP group and become active if the group priority is higher than the priority of the current active interface.
Step 4	`ssl-proxy(config-subif)# standby` [*group_number*] `timers` *hellotime holdtime*	(Optional) Set the HSRP hello timer and holdtime timer for the group. The default values are 3 (hello) and 10 (holdtime). All modules in the HSRP group should use the same timer values.
Step 5	`ssl-proxy(config-subif)# standby` [*group_number*] `authentication` *string*	(Optional) Specify a clear-text HSRP authentication string for the group. All modules in the HSRP group should use the same authentication string.

This example shows how to configure the HSRP on the SSL Services Module:

```
ssl-proxy(config)# interface ssl-proxy 0.100
ssl-proxy(config-subif)# encapsulation dot1q 100
ssl-proxy(config-subif)# ip address 10.1.0.20 255.255.255.0
ssl-proxy(config-subif)# standby 1 ip 10.1.0.21
ssl-proxy(config-subif)# standby 1 priority 110
ssl-proxy(config-subif)# standby 1 preempt
ssl-proxy(config-subif)# standby 2 ip 10.1.0.22
ssl-proxy(config-subif)# standby 2 priority 100
ssl-proxy(config-subif)# standby 2 preempt
ssl-proxy(config-subif)# exit
ssl-proxy(config)# ip route 223.255.254.0 255.255.255.0 10.1.0.254
```

Configuring TACACS, TACACS+, and RADIUS

For information on configuring TACACS, TACACS+, and RADIUS, refer to following URLs:

- "Configuring RADIUS" chapter in the *Cisco IOS Security Configuration Guide, Release 12.2*:
  http://www.cisco.com/univercd/cc/td/doc/product/software/ios122/122cgcr/fsecur_c/fsecsp/scfrad.htm
- "Configuring TACACS+" chapter in the *Cisco IOS Security Configuration Guide, Release 12.2*:
  http://www.cisco.com/univercd/cc/td/doc/product/software/ios122/122cgcr/fsecur_c/fsecsp/scftplus.htm

Configuring SNMP Traps

Note For more information on MIBs, refer to this URL:
http://www.cisco.com/public/sw-center/netmgmt/cmtk/mibs.shtml

Supported MIBs are as follows:

- CISCO-SSL-PROXY-MIB (all objects are read-only)
  - cspGlobalConfigGroup

    Version string

    Supported cipher suites

    Trap configuration setting
  - cspProxyServiceConfigGroup

    Type of proxy service

    IP addresses and TCP ports

    Policy names

    Keys and certificates
  - cspProxyServiceNotificationGroup

    Proxy service operational status change

    Proxy service certificate expiration warning
  - cspSslGroup

    Protocol counters

    Error counters

    Cumulative total values are reported in get responses, even if counters are cleared using CLI commands.
  - cspSsl3Group
  - cspTls1Group
  - cspSslErrorGroup
  - cspCpuStatusGroup

    Utilization of each CPU

    If counters have been cleared using CLI commands, the current values are reported in get responses, and the time of the last clear command are reported.
- CISCO-SSL-PROXY-CAPABILITY

To enable SNMP traps, perform this task:

	Command	Purpose
Step 1	`ssl-proxy(config)# snmp-server host addr traps version version ssl-proxy`	Specifies the IP address of an external network management device to which traps are sent.
Step 2	`ssl-proxy(config)# snmp-server enable traps ssl-proxy cert-expiring`	(Optional) Enable the SSL proxy certificate expiration notification trap. **Note** If you have set the certificate check-expiring interval to **0**, expiration notification traps are not sent. See the "Configuring Certificate Expiration Warning" section on page 3-39 for information on enabling certificate expiration warnings. **Note** Expiration notification traps are sent only for proxy service certificates that are currently configured.
Step 3	`ssl-proxy(config)# snmp-server enable traps ssl-proxy oper-status`	(Optional) Enable the SSL proxy operation status notification trap.
Step 4	`ssl-proxy(config)# snmp-server queue-length length`	(Optional) Specifies the number of trap events that are held before the queue must be emptied. The default *length* is 10; valid values are 1 through 1000.
Step 5	`ssl-proxy# show snmp`	Displays the SNMP information.

This example shows how to enable SNMP traps:

```
ssl-proxy# configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
ssl-proxy(config)# snmp-server host 10.1.1.1 traps version 2c ssl-proxy
ssl-proxy(config)# snmp-server enable traps ssl-proxy cert-expiring
*Nov 27 03:47:10.739:%STE-6-PROXY_CERT_EXPIRING_TRAP_ENABLED:SNMP trap for proxy service
certificate expiration warning has been enabled.
ssl-proxy(config)# snmp-server enable traps ssl-proxy oper-status
*Nov 27 03:46:59.607:%STE-6-PROXY_OPER_STATUS_TRAP_ENABLED:SNMP trap for proxy service
operational status change has been enabled.
ssl-proxy(config)# snmp-server queue-length 256
ssl-proxy(config)# end

ssl-proxy# show snmp
0 SNMP packets input
    0 Bad SNMP version errors
    0 Unknown community name
    0 Illegal operation for community name supplied
    0 Encoding errors
    0 Number of requested variables
    0 Number of altered variables
    0 Get-request PDUs
    0 Get-next PDUs
    0 Set-request PDUs
```

```
8 SNMP packets output
    0 Too big errors (Maximum packet size 1500)
    0 No such name errors
    0 Bad values errors
    0 General errors
    0 Response PDUs
    8 Trap PDUs

SNMP logging:enabled
    Logging to 10.1.1.1.162, 0/256, 0 sent, 0 dropped.
ssl-proxy#
```

Enabling the Cryptographic Self-Test

Note The power-on crypto chip self-test and key test are run only once at bootup.

Note Use the self-test for troubleshooting only. Running this test will impact run-time performance.

To run the self-test, perform this task:

Command	Purpose
`ssl-proxy(config)# ssl-proxy crypto self-test time-interval` *time*	Enables the crytographic self-test. The default value for *time* is 3 seconds; valid values are 1 though 8.

This example shows how to enable the cryptographic self-test and display cryptographic information:

```
ssl-proxy(config)# ssl-proxy crypto self-test time-interval 1
ssl-proxy(config)# end
```

Displaying Statistics Information

To display statistics information, perform this task:

Command	Purpose
`ssl-proxy(config)# show ssl-proxy stats {crypto | hdr | ipc | pki [auth | cache | cert-header | database | expiring | history | ipc | memory] | service | ssl | tcp | url}`	Displays specified statistics information.

This example shows how to display header insertion information:

```
ssl-proxy# show ssl-proxy stats hdr
Header Insert Statistics:
    Session Headers Inserted : 1          Custom Headers Inserted : 2
    Session Id's Inserted    : 1          Client Cert. Inserted   : 0
    Client IP/Port Inserted  : 0          PEM Cert. Inserted      : 0
    Aliased Hdrs Inserted    : 1
    No End of Hdr Detected   : 0          Payload no HTTP header  : 0
    Desc Alloc Failed        : 0          Buffer Alloc Failed     : 0
    Client Cert Errors       : 0          Malloc failed           : 0
    Service Errors           : 0          Conn Entry Invalid      : 0
    Buffers allocated        : 0          Buffers Scanned         : 1
    Insertion Points Found   : 1          Hdrs Spanning Records   : 0
    End of Header Found      : 1          Buffers Accumulated     : 1
    Multi-buffer IP Port     : 0          Multi-buffer Session Id : 0
    Multi-buffer Session Hdr : 0          Multi-buffer Custom Hdr : 0
    Scan Internal Error      : 0          Database Not Initialized: 0
```

See Table 4-2 for descriptions of the header insertion statistics:

Note The remaining statistics are for internal debugging.

Table 4-2 Header Insertion Statistics

Statistic	Description
Session Headers Inserted	The number of times session headers were inserted.
Custom Headers Inserted	The number of custom headers inserted.
Session Id's Inserted	The number of times session id was inserted.
Client Cert. Inserted	The number of times headers from a client certificate were inserted.
Client IP/Port Inserted	The number of times the client's IP address and port number were inserted.
PEM Cert. Inserted	The number of times a client certificate was inserted in PEM format.
Aliased Hdrs Inserted	The number of aliased headers inserted.

This example shows how to display crypto information:

```
ssl-proxy# show ssl-proxy stats crypto
Crypto Statistics from SSL Module:1
  Self-test is running
  Current device index is 1
  Time interval between tests is 1 seconds
  Device 0 statistics:
    Total Number of runs:50
    Runs all passed:50
    Number of timer error:0
    ------------------------------------------------------------
    Test Name                     Passed  Failed  Did-not-run
    ------------------------------------------------------------
      0  Power-on Crypto chip sel    1       0       0
      1  Power-on Crypto chip key    1       0       0
      2  Hash Test Case 1           50       0       0
      3  Hash Test Case 2           50       0       0
      4  Hash Test Case 3           50       0       0
      5  Hash Test Case 4           50       0       0
      6  SSL3 MAC Test Case 1       50       0       0
      7  SSL3 MAC Test Case 2       50       0       0
      8  TLS1 MAC Test Case 1       50       0       0
      9  TLS1 MAC Test Case 2       50       0       0
     10  DES Server Test            50       0       0
     11  DES Encrypt Test 1         50       0       0
     12  DES Decrypt Test 1         50       0       0
     13  DES Encrypt Test 2         50       0       0
     14  DES Decrypt Test 2         50       0       0
     15  ARC4 Test Case 1           50       0       0
     16  ARC4 Test Case 2           50       0       0
     17  ARC4 Test Case 3           50       0       0
     18  ARC4 State Test Case 1     50       0       0
     19  ARC4 State Test Case 2     50       0       0
     20  ARC4 State Test Case 3     50       0       0
     21  ARC4 State Test Case 4     50       0       0
     22  HMAC Test Case 1           50       0       0
     23  HMAC Test Case 2           50       0       0
     24  Random Bytes Generation    50       0       0
     25  RSA Encrypt/Decrypt Test   50       0       0
     26  Master Secret Generation   50       0       0
     27  Key Material Generation    50       0       0
     28  SSL3 Handshake Hash Test   50       0       0
     29  TLS1 Handshake Hash Test   50       0       0

  Device 1 statistics:
    Total Number of runs:49
    Runs all passed:49
    Number of timer error:0
    ------------------------------------------------------------
    Test Name                     Passed  Failed  Did-not-run
    ------------------------------------------------------------
      0  Power-on Crypto chip sel    1       0       0
      1  Power-on Crypto chip key    1       0       0
      2  Hash Test Case 1           50       0       0
      3  Hash Test Case 2           50       0       0
      4  Hash Test Case 3           50       0       0
      5  Hash Test Case 4           50       0       0
```

```
 6  SSL3 MAC Test Case 1       50      0       0
 7  SSL3 MAC Test Case 2       50      0       0
 8  TLS1 MAC Test Case 1       50      0       0
 9  TLS1 MAC Test Case 2       50      0       0
10  DES Server Test            50      0       0
11  DES Encrypt Test 1         50      0       0
12  DES Decrypt Test 1         50      0       0
13  DES Encrypt Test 2         50      0       0
14  DES Decrypt Test 2         50      0       0
15  ARC4 Test Case 1           50      0       0
16  ARC4 Test Case 2           50      0       0
17  ARC4 Test Case 3           50      0       0
18  ARC4 State Test Case 1     49      0       0
19  ARC4 State Test Case 2     49      0       0
20  ARC4 State Test Case 3     49      0       0
21  ARC4 State Test Case 4     49      0       0
22  HMAC Test Case 1           49      0       0
23  HMAC Test Case 2           49      0       0
24  Random Bytes Generation    49      0       0
25  RSA Encrypt/Decrypt Test   49      0       0
26  Master Secret Generation   49      0       0
27  Key Material Generation    49      0       0
28  SSL3 Handshake Hash Test   49      0       0
29  TLS1 Handshake Hash Test   49      0       0
```

This example shows how to display PKI certificate authentication and authorization statistics:

```
ssl-proxy# show ssl-proxy stats pki auth
Authentication request timeout:240 seconds
Max in process:100 (requests)
Max queued before dropping:0 (requests)
Certificate Authentication & Authorization Statistics:
  Requests started:2
  Requests finished:2
  Requests pending to be processed:0
  Requests waiting for CRL:0
  Signature only requests:0
  Valid signature:0
  Invalid signature:0
  Total number of invalid certificates:0
  Approved with warning (no crl check):2
  Number of times polling CRL:0
  No certificates present:0
  Failed to get CRL:0
  Not authorized (e.g. denied by ACL):0
  Root certificates not self-signed:0
  Verify requests failed (e.g. expired or CRL operation failed):0
  Unknown failure:0
  Empty certificate chain:0
  No memory to process requests:0
  DER encoded certificates missing:0
  Bad DER certificate length:0
  Failed to get key from certificate:0
  Issuer CA not in trusted CA pool:0
  Issuer CA certificates not valid yet:0
  Expired issuer CA certificates:0
  Peer certificates not valid yet:0
  Expired peer certificates:0
```

This example shows how to display PKI peer certificate cache statistics:

```
ssl-proxy# show ssl-proxy stats pki cache
Peer certificate cache size:0 (entries), aging timeout:30 (minutes)
Peer certificate cache statistics:
  In use:0 (entries)
  Cache hit:0
  Cache miss:0
  Cache allocated:0
  Cache freed:0
  Cache entries expired:0
  Cache error:0
  Cache full (wrapped around):0
  No memory for caching:0
```

Collecting Crash Information

The crash-info feature collects information necessary for developers to fix software-forced resets. Enter the **show ssl-proxy crash-info** command to collect software-forced reset information. You can retrieve only the latest crash-info in case of multiple software-forced resets. The **show ssl-proxy crash-info** command takes 1 to 6 minutes to complete the information collection process.

Note The **show stack** command is not a supported command to collect software-forced reset information on the SSL Service Module.

The following example shows how to collect software-forced reset information:

```
ssl-proxy# show ssl-proxy crash-info

===== SSL SERVICE MODULE - START OF CRASHINFO COLLECTION =====


------------- COMPLEX 0 [FDU_IOS] ----------------------

NVRAM CHKSUM:0xEB28
NVRAM MAGIC:0xC8A514F0
NVRAM VERSION:1

++++++++++ CORE 0 (FDU) ++++++++++++++++++++++

   CID:0
   APPLICATION VERSION:2003.04.15 14:50:20 built for cantuc
   APPROXIMATE TIME WHEN CRASH HAPPENED:14:06:04 UTC Apr 16 2003
   THIS CORE DIDN'T CRASH
   TRACEBACK:222D48 216894
   CPU CONTEXT  ----------------------------

$0 :00000000, AT :00240008, v0 :5A27E637, v1 :000F2BB1
a0 :00000001, a1 :0000003C, a2 :002331B0, a3 :00000000
t0 :00247834, t1 :02BFAAA0, t2 :02BF8BB0, t3 :02BF8BA0
t4 :02BF8BB0, t5 :00247834, t6 :00000000, t7 :00000001
s0 :00000000, s1 :0024783C, s2 :00000000, s3 :00000000
s4 :00000001, s5 :0000003C, s6 :00000019, s7 :0000000F
t8 :00000001, t9 :00000001, k0 :00400001, k1 :00000000
gp :0023AE80, sp :031FFF58, s8 :00000019, ra :00216894
LO :00000000, HI :0000000A, BADVADDR :828D641C
EPC :00222D48, ErrorEPC :BFC02308, SREG :34007E03
Cause 0000C000 (Code 0x0):Interrupt exception
```

```
CACHE ERROR registers  -------------------

CacheErrI:00000000, CacheErrD:00000000
ErrCtl:00000000, CacheErrDPA:0000000000000000

   PROCESS STACK -----------------------------
      stack top:0x3200000

   Process stack in use:

   sp is close to stack top;

   printing 1024 bytes from stack top:

031FFC00:06405DE0 002706E0 0000002D 00000001  .@]`.'.`...-....
031FFC10:06405DE0 002706E0 00000001 0020B800  .@]`.'.`..... 8.
031FFC20:031FFC30 8FBF005C 14620010 24020004  ..|0.?.\.b..$...
...........
...........
...........
FFFFFFD0:00000000 00000000 00000000 00000000 ................
FFFFFFE0:00627E34 00000000 00000000 00000000 .b~4............
FFFFFFF0:00000000 00000000 00000000 00000006 ................


===== SSL SERVICE MODULE - END OF CRASHINFO COLLECTION =======
```

Enabling Debugging

This sections describes how to debug PKI transactions and different module processors:

- Debugging PKI, page 4-25
- Debugging FDU, PKI, SSL, and TCP Processors, page 4-27

Debugging PKI

To display debug information about public key infrastructure (PKI) transactions, perform this task:

Command	Purpose
`ssl-proxy# [no] debug crypto pki {messages | transactions}`	Displays PKI debug messages. The **messages** keyword displays messages about the actual data being sent and received during public key infrastructure (PKI) transactions. The **transactions** keyword displays debug messages pertaining to PKI certificates. The messages show status information during certificate enrollment and verification.

The following example, which authenticates and enrolls a CA, contains sample output for the **debug crypto pki transactions** command:

```
ssl-proxy(config)# crypto ca authenticate msca

Certificate has the following attributes:
Fingerprint:A5DE3C51 AD8B0207 B60BED6D 9356FB00
% Do you accept this certificate? [yes/no]:y

ssl-proxy# debug crypto pki transactions

00:44:00:CRYPTO_PKI:Sending CA Certificate Request:
GET /certsrv/mscep/mscep.dll/pkiclient.exe?operation=GetCACert&message=msca HTTP/1.0

00:44:00:CRYPTO_PKI:http connection opened
00:44:01:CRYPTO_PKI:HTTP response header:
 HTTP/1.1 200 OK
Server:Microsoft-IIS/5.0
Date:Fri, 17 Nov 2000 18:50:59 GMT
Content-Length:2693
Content-Type:application/x-x509-ca-ra-cert

Content-Type indicates we have received CA and RA certificates.

00:44:01:CRYPTO_PKI:WARNING:A certificate chain could not be constructed while selecting
certificate status

00:44:01:CRYPTO_PKI:WARNING:A certificate chain could not be constructed while selecting
certificate status

00:44:01:CRYPTO_PKI:Name:CN = msca-rootRA, O = Cisco System, C = US
00:44:01:CRYPTO_PKI:Name:CN = msca-rootRA, O = Cisco System, C = US
00:44:01:CRYPTO_PKI:transaction GetCACert completed
00:44:01:CRYPTO_PKI:CA certificate received.
00:44:01:CRYPTO_PKI:CA certificate received.

ssl-proxy(config)# crypto ca enroll msca
%
% Start certificate enrollment ..
% Create a challenge password. You will need to verbally provide this
   password to the CA Administrator in order to revoke your certificate.
   or security reasons your password will not be saved in the configuration.
   Please make a note of it.

Password:
Re-enter password:

% The subject name in the certificate will be:Router.cisco.com
% Include the router serial number in the subject name? [yes/no]:n
% Include an IP address in the subject name? [yes/no]:n
Request certificate from CA? [yes/no]:y
% Certificate request sent to Certificate Authority
% The certificate request fingerprint will be displayed.
% The 'show crypto ca certificate' command will also show the fingerprint.
Router(config)#

00:44:29:CRYPTO_PKI:transaction PKCSReq completed
00:44:29:CRYPTO_PKI:status:
00:44:29:CRYPTO_PKI:http connection opened
00:44:29:CRYPTO_PKI: received msg of 1924 bytes
00:44:29:CRYPTO_PKI:HTTP response header:
 HTTP/1.1 200 OK
Server:Microsoft-IIS/5.0
Date:Fri, 17 Nov 2000 18:51:28 GMT
```

```
Content-Length:1778
Content-Type:application/x-pki-message

00:44:29:CRYPTO_PKI:signed attr:pki-message-type:
00:44:29:13 01 33
00:44:29:CRYPTO_PKI:signed attr:pki-status:
00:44:29:13 01 30
00:44:29:CRYPTO_PKI:signed attr:pki-recipient-nonce:
00:44:29:04 10 B4 C8 2A 12 9C 8A 2A 4A E1 E5 15 DE 22 C2 B4 FD
00:44:29:CRYPTO_PKI:signed attr:pki-transaction-id:
00:44:29:13 20 34 45 45 41 44 42 36 33 38 43 33 42 42 45 44 45 39 46
00:44:29:34 38 44 33 45 36 39 33 45 33 43 37 45 39
00:44:29:CRYPTO_PKI:status = 100:certificate is granted
00:44:29:CRYPTO__PKI:All enrollment requests completed.
00:44:29:%CRYPTO-6-CERTRET:Certificate received from Certificate Authority
```

Debugging FDU, PKI, SSL, and TCP Processors

A virtual terminal server (VTS) is built into the SSL Service Module for debugging different processors (FDU, PKI, SSL, TCP) on the module.

Note Use the TCP debug commands only to troubleshoot basic connectivity issues under little or no load conditions (for instance, when no connection is being established to the virtual server or real server).

If you use TCP debug commands, the TCP module displays large amounts of debug information on the console, which can significantly slow down module performance. Slow module performance can lead to delayed processing of TCP connection timers, packets, and state transitions.

From a workstation or PC, make a Telnet connection to one of the VLAN IP addresses on the module to reach the FDU (port 2001), TCP (port 2002), and SSL (port 2003) processor on the SSL Services Module.

To display debugging information for the different processors, perform this task:

Command	Purpose
ssl-proxy# [**no**] **debug ssl-proxy** {**fdu** | **pki** [**auth** | **ca-pool**]| **ssl** | **tcp**} [*type*]	Turns on or off the debug flags for the specified system component.

After you make the Telnet connection, enter the **debug ssl-proxy** {**fdu** | pki | **ssl | tcp**} command from the SSL Services Module console. One connection is sent from a client and displays the logs found in TCP console.

The following example shows how to display the log for TCP states for a connection and verify the debugging state:

```
ssl-proxy# debug ssl-proxy tcp state
ssl-proxy# show debugging
STE Mgr:
  STE TCP states debugging is on
```

The following example shows the output from the workstation or PC:

```
Conn 65066 state CLOSED --> state SYN_RECEIVED
Conn 65066 state SYN_RECEIVED --> state ESTABLISHED
Conn 14711 state CLOSED --> state SYN_SENT
Conn 14711 state SYN_SENT --> state ESTABLISHED
Conn 14711 state ESTABLISHED --> state CLOSE_WAIT
Conn 65066 state ESTABLISHED --> state FIN_WAIT_1
Conn 65066 state FIN_WAIT_1 --> state FIN_WAIT_2
Conn 65066 state FIN_WAIT_2 --> state TIME_WAIT
Conn 14711 state CLOSE_WAIT --> state LAST_ACK
Conn 14711 state LAST_ACK --> state CLOSED
###############Conn 65066 state TIME_WAIT --> state CLOSED
```