CHAPTER 26

# Working with Profiles

This chapter describes profiles. It covers these topics:



## About Profiles

A profile is a named collection of rule group and active security settings that determine how traffic is processed and validated for a virtual web application. The system includes built-in profiles, which you can supplement with your own, application-specific profiles.

In general, implementing web application security will involve creating a profile for each class of traffic handled at the ACE XML Gateway. The profile settings include whether a given rule is off or on and values for its configurable settings. It also specifies the action resulting from a rule match, whether to block the rule or continue processing.

The built-in profiles are the Pass-through profile and PCI Compliance profile. The built-in profile settings cannot be modified directly. Instead, to make modifications to the settings of a built-in profile, you will need to create a new profile based on it and modify the settings of the newly created profile. A profile you create can be applied in a virtual web application directly or it can similarly serve as a settings template for creating additional profiles.

To view settings for a profile, click the **Profiles** link in the navigation menu and then click on the name of the profile to view. For built-in profiles, you can view rule settings by clicking the **view** link next to the name of the rule or active security feature. For a user-created profile, you can view or modify the profile rule settings by clicking the **edit** link next to the rule in the profile page. A profile has three types of message processing and validation rules: active security features, message rewrite rules, and message inspection rules.

# Built-In Profiles

A built-in profile is a profile with a preset rule configuration. The configuration is usually intended to address a particular set of requirements, such as the PCI compliance requirements. While the rule settings for a built-in profile cannot be modified directly, you can create new profiles based on a built-in profile, which can then be modified as needed.

Built-in profiles are part of the system's based configuration. The base configuration can be updated, resulting in additional built-in profiles or enhancements to the existing built-in profiles. Base configuration updates must be Cisco-certified, and cannot be modified directly.

The default base configuration contains these built-in profiles:

- Pass-through Profile
- PCI Compliance

## Pass-through Profile

The Pass-through profile is a built-in profile in which all rules are disabled, making it in effect a "blank" profile. The pass-through profile is useful for performing initial system testing. Since it is designed not to affect traffic in any way, it can be used to test connectivity or other initial configuration settings. It can also serve as a basis for creating new profiles.

## PCI Compliance

The PCI Compliance profile provides base settings intended to help meet elements of the requirements specified by the Payment Card Industry DSS (Data Security Standard). The profile helps protect against cross-site scripting (XSS) attacks (PCI 6.5.4), buffer overflow attacks (PCI 6.5.5), and injection flaws, such as SQL injection (PCI 6.5.6). It also specifies credit card number rewriting rule on responses. This rewrite rule is intended to prevent customer credit card data from being inadvertently transmitted in responses.

For details on the profile's settings, click on the profile name in the Profiles page.

**Note** While you cannot change the built-in PCI Compliance profile, you can create a profile based on it, which can be modified as desired.

For more information on PCI compliance, see the Cisco PCI Solution for Retail 2.0 Design and Implementation Guide. Also see https://www.pcisecuritystandards.org/.

# New Profile

To define and use a customized profile configuration, you create a new profile. Creating a profile involves several steps—first you create the profile with its initial settings, such as its name and description. You can then customize the rule settings for the profile, enabling the rules and security actions desired and setting their parameter values and severity levels.

To create a new profile:

**Step 1** Click the **Profiles** link in the navigation menu.

**Step 2** Click the **New Profile** button.

**Step 3** Configure the profile using the settings described in the following table.

*Table 26-1 Profile settings*

| Label | Description |
|---|---|
| Profile Name | A unique, descriptive name for the profile. This name is used to identify the profile in the policy. |
| Description | An optional description of the profile. This value can help to document the profile within the web console to other web console users. The value of the description does not appear outside the console. |
| Copy Settings From | Uses as the initial rule and active security settings for the profile those of an existing profile. Note that the settings from the source profile are propagated to the new profile only at the time the profile is created; that is, subsequent changes to a profile used as the source are not automatically propagated to profiles generated from it.<br>To create the profile without preconfigured settings, keep the default selection, none. |

**Step 4** Click the **Create Profile** button.

The new profile is created based on the settings you specify and its settings page appears.

**Step 5** From the profile settings page, modify the rule and active security features as desired for the profile. Click the **edit** link next to a rule to enable it or reconfigure it in the context of this profile.

After configuring the active security and inspection and rewriting rules in the profile, you can apply the profile in virtual web applications.

# Active Security Features

The active security features in a profile perform specialized message processing and security tasks, such as data overflow defense, HTTP header processing, and cookie encryption/decryption.

**Note** Unlike rules or signatures, active security features cannot be added to the system or otherwise customized except as provided for in their configuration settings.

In general, for virtual web applications in monitor mode, the ACE XML Gateway applies the security rules to traffic but does not block message that violate the rules. Most active security features remain in effect for virtual web applications in monitor mode, including HTTP header processing, HTTP exception mapping, and cookie security. However, data overflow defense and referer enforcement are subject to monitor mode; that is, when triggered, they do not cause message blocking although an event is logged.

In general, to view or modify active security settings for a profile, click the **Profiles** link in the navigation menu and then click on the name of the profile to view. Next to the active security feature name, click the **view** link (for a built-in profile) or the **edit** link (for a user-created profile) to access its settings.

The following sections provide more information on each security feature.

# HTTP Header Processing

As a reverse proxy, the ACE XML Gateway processes and populates certain HTTP headers in messages automatically. For example, it populates the `Date` and `Content-Length` headers with appropriate values when it processes a message. However, other types of HTTP headers found in messages are passed through without modification. Rather than passing headers through, you can have the ACE XML Gateway modify, add, or remove specific headers in messages.

The HTTP Header processing page provides controls for configuring the HTTP headers that often need to be manipulated at a reverse proxy, such as the `Server` and `X-Forwarded-For` header. In addition, you can configure special handling for any HTTP header identified by name. HTTP header processing can be specified on either the request or response side.

The values of certain types of headers cannot be set in the policy. In general, these headers are populated with values determined at runtime, such as the `Date` and `Content-Length` header. In addition, header passthrough does not apply to what are considered “hop-by-hop” headers, which are significant only in the context of a single transport-level connection, such as:

- Accept-Encoding
- Connection
- Keep-Alive
- Proxy-Authenticate
- Proxy-Authorization
- TE
- Trailers
- Transfer-Encoding
- Upgrade

The following table describes the HTTP Header processing settings.

*Table 26-2 HTTP Header processing settings*

| Label | Description |
|---|---|
| Insert "X-Forwarded-For" header with client's IP address | Reverse proxy servers often use the `X-Forwarded-For` HTTP header to identify the client that originated the request to the backend system. You can use this header to indicate to the backend application the IP address that appeared as the request source as received by the ACE XML Gateway. The header is added as specified by one of these options:<br>• **if the header does not already exist** – Adds the header only if it does not appear in the message. If the header is already present, the header is passed through with its original value.<br>• **replacing any existing value** – Removes the existing header and adds the custom header.<br>• **in addition to any existing value** – Adds the header whether or not it is already in the message. If a header with this name already exists in the message, there will be two instances of the header.<br>• **appending to any existing value** – If the header already exists, appends a comma and the IP address of the source to the existing header. If the message does not have a header, it is added. (This option is suggested for best interoperability, particularly with the widely used Squid open source web proxy.) |
| Insert Client SSL Certificate DN in header named | Inserts as a new header the distinguished name of the subject of a client SSL certificate included in the message. The DN value is added to a header with the name you specify. The header is added as specified by these options:<br>• **if the header does not already exist** – Adds the header only if it does not appear in the message. If the header is already present, the header is passed through with its original value.<br>• **replacing any existing value** – Removes an existing header and adds the custom header.<br>• **in addition to any existing value** – Adds the header regardless of whether it is already present in the message. If present, there will be two instances of this header.<br>• **appending to any existing value** – If the header already exists, appends a comma and the certificate DN to the existing header. If the message does not have a header, it is added.<br>For the certificate DN value to be added to the header by this option, the SSLVerifyClient setting must be set to optional_no_ca value, as specified in the I/O Process Settings page. If it is not, the header will be added with a blank value. |
| Rewrite “Host” header with destination server hostname | If selected, replaces the Host header value in requests with the name of the destination backend host, as derived from the destination server specified for the virtual web application definition. |

| Label | Description |
|---|---|
| Custom Header Processing | For request messages, performs the following action on the named HTTP headers. Configure header processing by specifying these values:<br>• In the first field, enter the name of the header to be processed.<br>• The operation menu specifies how the header is to be processed:<br>– **strip** – Removes all instances of the named header you specify from the message. Note that this does not imply that the header is expected; that is, it is not an error if absent.<br>– **set if empty** – If the named header does not exist, inserts it with the value specified. If it does exist, does nothing. The resulting message will have at least one header with the name you configure.<br>– **replace value** – Strips any existing occurrences of the named header and inserts a new instance of it with the value specified.<br>– **add value** – Inserts a new instance of the header with the name and value specified. This option does not affect existing headers with the same name. Therefore, it will result in a message with at least one but possibly multiple headers with the same name.<br>– **append** – Appends a comma followed by the configured value to an existing header with the same name. If the request does not have the named header, it is added. If it has multiple headers with the same name, the value is appended to only one of the headers.<br>The operations are performed in the order in which they appear in the interface.<br>• In the text field that follows the operation (unless the **strip** option is selected), enter the desired value. This field supports dynamic values in the form of Reactor expressions, e.g., $(REQUEST_HEADER['Date'])<br>For more information on the Reactor expression syntax, see the *Cisco ACE XML Gateway User Guide*. |
| Replace "Server" header value with | By default, in responses handled by virtual web applications, the Server header value received from the backend system is passed through to the outgoing response.<br>Alternatively, you can have the ACE XML Gateway rewrite the `Server` header to the value specified in this field. |

| Label | Description |
|---|---|
| Custom Header Processing | For response messages, performs the following action on the named HTTP headers. Configure header processing by specifying these values:<br>• In the first field, enter the name of the header to be processed.<br>• The operation menu specifies how the header is to be processed:<br>– **strip** – Removes all instances of the named header you specify from the message. Note that this does not imply that the header is expected; that is, it is not an error if absent.<br>– **set if empty** – If the named header does not exist, inserts it with the value specified. If it does exist, does nothing. The resulting message will have at least one header with the name you configure.<br>– **replace value** – Strips any existing occurrences of the named header and inserts a new instance of it with the value specified.<br>– **add value** – Inserts a new instance of the header with the name and value specified. This will not affect existing occurrences with the same header name, so it may result in a message with at least one but possibly multiple headers the name.<br>– **append** – Appends a comma followed by the configured value to an existing header with the same name. If the response does not have the named header, it is added. If it has multiple headers with the same name, the value is appended to only one of the headers.<br>The operations are performed in the order in which they appear in the interface.<br>• In the text field that follows the operation (unless the **strip** option is selected), enter the desired header value. |

# HTTP Exception Mapping

An HTTP exception is a response message that signals an error or other unexpected event in the course of request processing. The exception may result from an error in the request itself or an error in the backend system processing or network. In some cases, HTTP exceptions passed from the backend application can contain sensitive information to potential attackers that hackers can use, such as a web server stack trace. By mapping exceptions at the Cisco ACE XML Gateway, you can ensure that only generic error messages are passed to the client.

When exception mapping is enabled, rather than returning specific error information, the ACE XML Gateway returns the response you configure. You can configure mapping of all 400 and 500 errors to a generic 500 error, for instance, or configure a specific response for some errors.

The following table describes the HTTP Exception processing settings.

*Table 26-3 HTTP Exception processing settings*

| Label | Description |
|---|---|
| For server errors (status code 400 and above) not specified below | Use this option to map any error response from the backend system that has an HTTP status code of 400 or greater to a generic error response to be returned to the client. By default, such responses are passed through to the client.<br>The generic HTTP 500 status code response reports a server error with the following description: "The server encountered an internal error and was unable to complete your request".<br>You can enable this error mapping option along with the custom response mapping options below it. The custom response configuration takes precedence over this generic mapping for a response with an error code to which both are applicable.<br>If this option is set for pass through and there are specific status codes mapped to an exception, the `Server` header value for error responses received by clients will vary—those passed through will have the `Server` value set by the backend system, while mapped errors will have the `Server` value set in the ACE XML Gateway configuration, if specified by the policy configuration. |
| Status Codes | Rather than mapping all error responses to a generic response, you can configure a custom response message for specific HTTP error codes received from the backend system.<br>To configure this behavior, in the **Status Codes** field, type the numeric error status codes of responses to be mapped to the custom response. You can enter the numbers individually or as a range, such as "400, 403, 500-599". Ranges and single values must be separated by a comma.<br>When deployed, responses from the backend network with the specified error code will be mapped to the response you configure for delivery to the client. This configuration takes precedence over the generic mapping setting, if also configured. |
| Status Code | The status code for the outgoing response message, with the options of passing through the status code from the incoming response or of using a pre-set status code. |
| Content-Type | The content encoding type for the outgoing response message, with text/html by default. |
| Other Headers | Any other HTTP headers that you want to be included in the response.<br>Note that certain HTTP headers cannot be specified in this field. The Date and Content-Length header values, for instance, are passed through from the backend response and cannot be manually specified here. Also, the Server header value is either passed through from the backend system (the default) or populated by a more specific configuration in the HTTP header processing settings. |
| Response Body | The body of the HTTP response to be sent to the client. |

# Referrer Enforcement

The *referrer* HTTP request header (or `Referer`, as incorrectly spelled in the standard) indicates the address of the resource from which the request obtained the Request-URI. The referrer enforcement feature can protect against a type of attack called Cross Site Request Forgery (CSRF).

A common example of a CSRF attack involves a user who has an active browser session to a sensitive web application, such as an online banking application. If, while the authenticated session remains active, the user can be induced to click on a link on another web site that initiates an operation against the bank application, such as a funds transfer to the attacker's account, the operation can be fulfilled. The link may be presented on a web site controlled by the attacker, in an email, or elsewhere, such as on a public forum.

Such attacks can be avoided by restricting messages based on the host indicated in the `Referer` header. If the host identified by the referrer value is not the host of the web application itself, the request can be rejected. This blocks requests in which the request URL for performing a sensitive operation was obtained elsewhere, such as from a link on a third-party web site or in an email.

The following table describes the referrer enforcement settings.

***Table 26-4 Referrer enforcement settings***

| Label | Description |
|---|---|
| If the "Referer" header is present, require that its value matches requested host name | If selected, requires requests that contain a `Referer` header to have as its value a URI with a hostname that matches the one in the request URL. In other words, the referrer must identify the web application addressed by the request, not an external application or web site. |
| Never check "Referer" header on GET requests | Since HTTP GET requests are not usually used in this type of attack, you can disable checking for GET requests. The checking will apply to POST requests or incoming messages with other HTTP methods. |
| Monitor Mode | When selected, if the host identified in the `Referer` header does not match the request URL, the event is logged but the message is allowed. |

# HTTP Cookie Security

Web applications often use HTTP cookies to store information regarding a particular user or session. The server embeds the cookie in the response sent to the client, and the browser returns the cookie unchanged in subsequent requests. It's possible for cookies to contain private information or to form the basis of a session hijacking attack. In most cases, the cookie content should not be modified except by the backend application.

Whether cookie security is enabled or not, cookies in messages are subject to the data overflow and HTTP header processing settings you configure in the profile. With cookie security enabled, the ACE XML Gateway can apply cookie-specific validation measures and ensure the security of cookies by encrypting or signing cookie before delivery to the client.

After processing a cookie, when the ACE XML Gateway receives the cookie in subsequent requests from the client, it checks the signature or decrypts the cookie before forwarding it to the backend web application. The ACE XML Gateway can thereby ensure that the cookie has not been modified or viewed from the Gateway to the client.

When cookie security is enabled, the ACE XML Gateway removes cookies that are found to be invalid based on its inability to verify the signature, decrypt the cookie, or validate the cookie for correctness. (When cookie security is disabled, cookies are propagated through the ACE XML Gateway unchanged.) Cookies are considered invalid if their values contain unclosed quotes, duplicate attributes, and certain forbidden characters, such as commas (“,”), semi-colons (“;”), double-quotes “(”)”, equals (“=”), and whitespaces.

**Note** Cookie security should not be used with applications that use Javascript to set or modify cookies at the client. For example, client-side Javascript may set a cookie to indicate the browser type to the backend application. Since client-modified cookies will fail validation or decryption, cookie security should be disabled in such cases. Note that with cookie signature enabled, new cookies added at the client will be dropped by the Gateway; however, with encryption enabled, new cookies will be accepted. In either case, cookies that are modified at the client are dropped.

Cookie header validation occurs when cookie security is explicitly enabled through the use of cookie encryption or signing. The following table describes the settings for the active cookie security feature.

***Table 26-5 Cookie security features***

| Label | Description |
|---|---|
| Sign (HMAC-SHA1) | Select this option to have cookies in the response digitally signed before being sent to the client and validated upon being returned in subsequent requests. Digital signatures help to ensure the integrity of cookies. It can prevent cookies that have been maliciously tampered with (for example, in a session spoofing attempt) from being forwarded to protected applications. If the cookie signature is invalid, the cookie is stripped from the request.<br><br>The Cisco ACE XML Gateway signs the cookie using the keyed-Hash Message Authentication Code (HMAC or KHMAC) based on a secret key you specify in the passphrase field of the policy configuration. The same signing passphrase needs to be used by any ACE XML Gateway cluster that may receive requests with cookies that were processed by another ACE XML Gateway cluster.<br><br>The effects of cookie signing should be considered if also using data overflow defense in this profile. When signing a cookie, the Cisco ACE XML Gateway adds an extra header (a signature cookie) to the outbound message. The signature is always eight characters in length, while the name of the cookie header is three characters longer. Therefore, the total size of the signature cookie header will be larger than the size of the original cookie if the original cookie value attribute is 10 characters or less. Take, for instance, the following cookie:<br><br>Cookie: NAME=123456789a<br><br>If signed, the message will include a signature cookie in the following form (not an actual signature):<br><br>Cookie: NAMESig=12345678<br><br>As shown, the cookie value is always represented by eight characters. However, three characters are also added to the name. |

| Label | Description |
|---|---|
| Encrypt (AES) | Select this option to have the Cisco ACE XML Gateway encrypt cookies in messages prior to delivery to the client. When it receives the encrypted cookie in a subsequent request from the client, it decrypts the cookie prior to delivery to the backend application.<br><br>Cookie encryption serves to obscure cookie data from the client. Note that cookies can be generated by clients as well as servers. If it receives a request with an unencrypted cookie (which suggests a client-generated cookie), the ACE XML Gateway allows it through. It's important to note, therefore, that while in general encryption can provide a form of integrity protection, in this case, because the ACE XML Gateway doesn't require every cookie to be encrypted, encryption cannot be relied on to enforce cookie integrity.<br><br>The Cisco ACE XML Gateway encrypts the cookie using AES (Advanced Encryption Standard), a symmetric key-based cipher encryption standard. The cipher text is generated with the secret key you specify in the passphrase field.<br><br>The effects of cookie encryption should be considered if also using data overflow defense, which can impose a limit on the size of the cookie headers. For encryption, cookies will become larger by a number of bytes proportional to the size of the original cookie value. Specifically, the length of the encrypted cookie can be anticipated using the following formula. Since they are encrypted separately, take the length of each of the cookie's name and value and round that value up to the next multiple of 16. Then divide by 3 (rounding up if there's a remainder) and multiply by 4.<br><br>For example, if the name is 10 characters, 10 + 1 = 11, round up to 16, divide by 3 and round up = 6, multiply by 4 = 24. Some name/value example lengths, with their post-encryption sizes:<br><br>• 0 – 15 becomes 24 characters<br>• 16 – 31 becomes 44 characters<br>• 32 – 47 becomes 64 characters<br><br>Use the result of this calculation to determine the appropriate data overflow settings to use. |
| Cookies with Passphrase | The secret key for encrypting or signing cookies. The passphrase you enter should be at least five characters long, and can include any combination of numbers, letters or special characters.<br><br>While the passphrase may be 5 characters long, for best security, you should enter a passphrase that is significantly longer. Ideally, it should be around 20 characters in length.<br><br>When the policy is deployed, the Manager transmits the passphrase to all Cisco ACE XML Gateways in the cluster. If separately managed clusters may receive subsequent requests from the same client, those clusters should be configured with the same passphrase. |

# Data Overflow Defense

Data overflow defense allows you to configure security based on the size or number of various attributes in a message. Messages that do not comply with the data overflow defense settings can be blocked or passed through with an event logged at a configurable severity level.

**Note** The processing settings for the virtual web application itself can affect the size of message components. For instance, cookie signing or encryption can introduce new headers or enlarge existing headers. The data overflow defenses are applied to the message after cookie processing. For more information, see "HTTP Cookie Security" section on page 26-253.

The following table describes the data overflow defense settings.

*Table 26-6 Data overflow defense settings*

| Label | Description |
|---|---|
| Enforce the following data limits on requests | Select this option to enable data overflow protection and make the configuration settings for data overflow defense available. |
| Maximum Number of HTTP Headers | The number of HTTP headers permitted in messages. |
| Maximum Size of Any HTTP Header | The maximum size of any HTTP header, in bytes. This does not apply to cookies. |
| Maximum Cookie Size | The maximum size of any HTTP cookie or encrypted cookie headers in bytes. |
| Maximum Total HTTP Header Size | The maximum size of all the HTTP headers together, including cookies, in KB. |
| Maximum Size of Request URL | The maximum size of request URLs, in bytes. This values includes the requested hostname, resource, and all URL parameters. |
| Maximum Size of GET Query String | The maximum size of GET query string in the URL, in bytes. Query string appears in the request following a question mark symbol, such as: http://hostname/path/page?query_string |
| Maximum Number of Request Arguments | Maximum number of arguments in a GET or POST request. In GET requests, arguments appear as ampersand-separated parameters to the URL, such as http://example.com/path/page?name1=value1&name2=value2<br>In POST requests, they appear as similar name-value pairs in the body of the request. |
| Maximum Size of Any Argument | Maximum size of any one argument in a POST or GET request, including the size of the name and value of the argument, in bytes. |
| Maximum Total Size of Request Body | The maximum total size of the POST body in a request, in bytes. |
| Monitor mode | If selected, requests that violate the data overflow settings result in an event being logged at the configured severity level, but are not blocked. |

| Label | Description |
|---|---|
| Event Log | Controls the severity level of the log event resulting from this action being triggered when it is applied in monitor mode, either at the rule group level or at the virtual web application level.<br>By default, these events are logged at the Warning level. However, when the rule is applied in monitor mode, it may be appropriate to have a reduced severity level. |
| Response | If a message violates the data overflow defense settings, the action the ACE XML Gateway should take. Options are:<br>• Return an HTTP error response with status code 400, Client Error<br>• Return an HTTP error response with status code 500, Server Error<br>• Return a custom HTTP error response you configure.<br>• Allow message is similar to monitor mode; a signature match event is reported but the message is allowed to pass through. |

# Message Rewrite Rules

While a message inspection rule operates on the entire message, in either blocking or allowing it through, a rewrite rule modifies the message by replacing matched content before passing it through. Also, while message inspection rules operate on requests, message rewrite rules operate on responses.

Message rewrite rules help to ensure that backend applications do not emit sensitive information, such as user credit card numbers or social security numbers. The portion of the response that matches the signature in a rewrite rule is replaced by the replacement character. A message rewrite rule can match multiple instances of the signature pattern in a response.

When enabled in a policy, message rewrite rules modify messages even if the virtual web application is set to monitor-mode only. That is, message rewriting occurs whether the virtual web application is in monitor or enabled mode.

### Content Rewriting and Response Compression

Special considerations exist if configuring content replacement for responses that may be compressed. When it receives responses that have been compressed by backend systems, the Cisco ACE XML Gateway can pass through the responses. However, message rewriting rules will not work on compressed responses.

If content replacement is important to your system and the backend system performs response compression, you will need to ensure that the Gateway receives uncompressed responses by stripping the HTTP header used to indicate acceptance of compressed responses (ACCEPT-ENCODING) from outgoing requests.

You can specify header stripping in the virtual web application by configuring HTTP header processing in which the ACCEPT-ENCODING header is removed from outgoing requests. As a result, the response from the backend server will not be compressed.

To enable and configure rewrite rules, click the **edit** link next to the rewrite rule group in the profile. Rewrite rules have the following settings.

*Table 26-7 Rewrite Rule settings*

| Label | Description |
|---|---|
| Rule Set Mode | Whether the rule group will be enabled in the virtual web applications that use this profile. Enabling this option displays the rules in this rule group, which can be individually enabled or disabled.<br>The options are:<br>• Enabled – The rules in this rule group are applied to message traffic for virtual web applications that use this profile.<br>• Disabled – The rules are not applied to traffic for this profile.<br>Note that enabled rewrite rules apply even if the virtual web application is in monitor mode. |
| view rule set details | Displays the source code of the current rule group. |
| Rewrite Rules | Each rewrite rule in the group can be individually enabled or disabled. If enabled, the rule is applied to message traffic. When a rule is triggered by a signature match, each character of the matched text is replaced by a replacement character for the rule. The character is indicated in the Rewrite Char. column for the rule, which you can view by clicking the **view rule set details** link. |

# Message Inspection Rules

Message inspection rules examine requests for potentially malicious content. The rules use signatures to identify content of interest in messages directed to a web application. Built-in message inspection rules exist, for example, to detect command injection or cross site scripting attacks. When the content is detected, the ACE XML Gateway can block the request or allow it through with an event logged.

For a message-inspection rule set, you can specify a strictness level to be applied in the profile—basic, moderate, or strict. The severity level controls which rules in the set are enabled; only rules that have the same or lesser severity are enabled. Choosing moderate severity, for instance, enables the moderate and basic rules in the rule set. Instead of a specific level, you can manually choose which rules to enable with the custom option.

To enable and configure rewrite rules, click the **edit** link next to the message inspection rule group in the profile. Inspection rules have the following settings.

*Table 26-8 Inspection Rule settings*

| Label | Description |
|---|---|
| Rule Set Mode | Whether the rule group will be enabled in the virtual web applications that use this profile. Enabling this option displays the rules in this rule group at a particular severity level.<br>The options are:<br>• Enabled – The rules in this rule group are applied to message traffic for applications that use this profile.<br>• Monitor – If a message triggers any enabled rule, the event is logged at the severity level specified in the rule configuration (Warning, by default), but the message is not blocked.<br>• Disabled – The rules in this rule group are not applied to traffic for this profile. |
| Level | The severity level of the rules in the group that you want to be applied. Each rule is described by its severity level. The severity is usually differentiated by the number and scope of signatures in the rule or the scope of the message to be inspected. Choosing a severity level for the group causes the ACE XML Gateway to apply only the rules in the group that have the same severity level.<br>The levels, in increasing order of severity, are basic, moderate, and strict. Choose **Custom** to choose directly which rules are enabled. |
| Exemptions | Exemptions allow you to fine-tune how rules are applied. An exemption excludes certain signatures from evaluation. If a message matches an exempted signature, the match is ignored. For instance, it may make sense to exempt the signature that looks for a single tick (‘) in parameters that represent user names, since the parameter may have the following legitimate value: /path?lastname=o’neill<br>For these cases, you can exempt the lastname parameter from the single-tick pattern signature. The same effect can be achieved with custom signatures, however, using the exemption feature simplifies the configuration.<br>An exemption can be specified in the base profile, as performed here, or (as may be appropriate in most cases) in the profile of a modifier to the virtual web application.<br>In the exemption configuration, indicate the scope of the exemption, whether it should apply to the entire request, to parameters, or to HTTP headers. Also, indicate whether a specific signature or all signatures are to be exempted.<br>For a named signature, indicate the signature by signature identifier, such as:<br>**In parameter**: lastname **ignore signature**: CrossScriptXSS.52<br>To get the signature identifiers for the signatures in the policy, click **View Signatures** from the Rules & Signatures page.<br>If the exemption applies to a message, the following item is logged at the information level: CROSSITESCRIPT.CrosSiteScript1:CrossScriptXSS.52:REQUEST_GETPARAM['firstname'] detected by a Limit, but an override deactivated it. |

| Label | Description |
|---|---|
| Event Log | Specifies the severity level of the log event resulting from this rule being triggered when it is applied in monitor mode, either at the rule group level or at the virtual web application level.<br><br>By default, these events are logged at the Warning level. However, when the rule is applied in monitor mode, it may be appropriate to have a reduced severity level for the rule. |
| Response | Use to configure the response message returned to the client in the event of a rule signature match. Options are:<br><br>• Return an HTTP error response with status code 400, Client Error<br>• Return an HTTP error response with status code 500, Server Error<br>• Return a custom error response you configure<br>• Allow message is similar to monitor mode; a signature match event is reported but the message is allowed to pass through. |