ProxySG Content Policy Language Reference

Version 7.2.x

Guide Revision: 1/25/2021

Symantec
A Division of Broadcom

Symantec, a Division of Broadcom

Legal Notice

Monday, January 25, 2021

Table of Contents

Overview

The Blue Coat ProxySG appliance's Content Policy Language (CPL) is a programming language with its own concepts and rules. Refer to the following topics:

Concepts

In Symantec documentation, policy refers to configuration values and rules applied to render decisions on authentication requirements, access rights, quality of service, or content transformations (including rewrites and off-box services that should be used to process the request or response). Often, the policy references system configuration for the default values for some settings and then evaluates rules to see if those settings should be overridden.

CPL is a language for specifying the policy rules for the ProxySG appliance. Primarily, it controls the following:

- User authentication requirements
- Access to web-related resources
- Cached content
- Various aspects of request and response processing
- Access logging

You can create policy rules using either the Visual Policy Manager (VPM), which is accessible through the Management Console, or by composing CPL.

Before reading sample CPL or trying to express your own policies in CPL, Symantec recommends that you understand the fundamental concepts underlying policy enforcement in the ProxySG appliances. This section provides an overview of important concepts.

Transactions

A transaction consists of a request for service and any associated response for the purposes of policy evaluation and enforcement. In most cases, a transaction is created for each unique request for service, and the transaction exists for the time taken to process the request and deliver the response.

The transaction serves the following purposes:

- Exposes request and response state for testing during policy evaluation.

  This provides the ability to test various aspects of a request, such as the IP address of the client and the URL used, or the response, such as the contents of any HTTP headers.

- Ensures policy integrity during processing.

  The lifetime of a transaction could be relatively long, especially if a large object is fetched over slow networks and subject to off-box processing services such as content filtering and virus scanning. During this time, changes to configuration or policy rules may occur, which would result in altering the policy decisions that affect a transaction. If a

request was evaluated against one version of policy, and some time later the associated response were evaluated against a different version of policy, the outcome would be unpredictable and possibly inconsistent.

The transaction ensures that both the request and the response are evaluated against the version of policy that was current when the transaction was created. To ensure that new policy is respected, long-lived transactions such as those involved in streaming, or large file downloads, are re-evaluated under new policy. Re-evaluation applies to both the request and response, and any resulting new decisions that cannot be honored (such as new authentication requirements) result in transaction termination.

- Maintains policy decisions relevant to request and response processing.
- Various types of transactions are used to support the different policy evaluation requirements of the individual protocols: administrator, cache, and proxy transactions.
- In a few special cases, multiple transactions can be created for a single request. For example, if an HTTP request is made via the SOCKS proxy (on port 1080 of the appliance), it is possible for two transactions to be created: a SOCKS proxy transaction, and an HTTP proxy transaction. To see these transactions, turn on policy tracing. A new entry is added to the policy trace file for each transaction.

Policy Model

Each transaction begins with a default set of decisions, many of which are taken from the system configuration. These defaults include such things as forwarding hosts or SOCKS gateways. The most important default decision affects whether or not requests should be allowed or denied.

The defaults for the various transaction types are:

- Administrator transaction– the default is to deny requests.

  By default, administration is only available through one of the methods that bypasses policy evaluation. These are:

  - accessing the CLI through the serial console
  - accessing the CLI through RSA authenticated SSH
  - logging into the Management Console or CLI using the console credentials

  Specific rights must be granted through policy to enable other administration methods.

- Cache transactions–the default is to allow requests.

  These requests originate from the appliance itself, and are used primarily to maintain the state of content. Additional policy can be added to specifically deny requests for specific content, and to distinguish content management requests from other cache transactions.

- Proxy transactions–the default is taken from system configuration.

  For new appliances, the default is to deny all requests.

The correct approach to writing <proxy > layer policy depends on whether or not the default is to allow or deny requests. The default proxy policy is configurable and represents the starting point for writing policy to control proxy transactions. The default proxy policy is reported at the top of every policy listing generated by the appliance, as follows:

```
; Default proxy policy is DENY
```

This line in a policy listing is a CPL comment, defining the starting point for proxy policy.

Role of CPL

CPL is the language used to express policy that depends on the runtime evaluation of each transaction. Policy is written in CPL, installed on the appliance, and is evaluated during request processing to override any default decisions taken from configuration.

CPL Basics

The following sections provide an overview of CPL. Detailed specifications for each of the following elements is available in the reference portion of this manual.

Comments

Any line starting with a semicolon (;) is a comment.

A semicolon following a space or tab introduces a comment that extends to the end of the line, as follows:

```
; This is a comment.
```

This is not to be confused with trigger pattern expression or property settings where a semicolon appears inside quotation marks (" ").

You can insert comments anywhere in policy.

Rules

A policy rule consists of a condition and a number of property settings, written in any order. Rules are generally written on a single line, but can be split across lines using a special line continuation character. When a rule is evaluated, the condition is tested for that particular transaction. If the condition evaluates to true, all of the listed property settings are executed and evaluation of the current layer ends. When a rule evaluates to true for a transaction, the rule is said to match. If the rule evaluates to false, it is said to miss.

In turn, a condition is a Boolean combination of trigger expressions. Triggers are individual tests that can be made against components of the request ("url=" on page 298), response ("response.x_header.header_name=" on page 256), related user ("user=" on page 312,"group=" on page 140), or system state ("time=" on page 292).

With a few notable exceptions, triggers test one aspect of request, response, or associated state against a Boolean expression of values.

For the conditions in a rule, each of the triggers is logically anded together. In other words, the condition is only true if each one of the trigger expressions is true.

Properties are settings that control transaction processing, such as deny, or the handling of the object, such as `cache(no)`, indicating that the object is not to be cached locally. At the beginning of a transaction, all properties are set to their default values. As the policy is evaluated in sequence, rules that match might set a property to a particular value. A property retains the final value setting when evaluation ends, and the transaction is processed accordingly. Properties that are not set within the policy maintain their default values.

The logical form of a policy rule could be expressed as:

```
if condition is true then set all listed properties as specified
```

The following is an example of a simple policy rule:

```
url.domain=example.com time=0900..1700 exception(policy_denied)
```

It states that the "exception()" on page 402 property is set to `policy_denied` if both of the following triggers test true:

- The request is made for a page from the domain example.com
- The request is made between 9 a.m. and 5 p.m.

Notes

- CPL triggers have the form *trigger_name=pattern_expressio*n
- CPL properties have the form *property_name*(*setting*), except for a few imperative gestures such as `allow` and deny.
- The text in policy rules is case-insensitive, with a few exceptions identified in the following chapters.
- Non-ASCII characters cannot occur within a CPL source file.
- Policy listings are normalized in several ways. First, condition and action definitions which may appear anywhere in the source, will be grouped following the policy rules. Second, the order of the conditions and properties on a rule may change, since the CPL compiler always puts a deny or allow at the beginning of the rule, and orders conditions to optimize evaluation. Finally, several phrases are synonyms for phrases that are preferred. In the output of show policy, the preferred form is listed instead of the synonym.

  Four such synonyms are:

  - `exception(authorization_failed)`, which is a synonym for the preferred deny.unauthorized
  - `force_exception(authorization_failed)`, which is a synonym for the preferred force_deny.unauthorized
  - `exception(policy_denied)`, which is a synonym for the preferred deny
  - `exception(no)`, which is a synonym for the preferred allow.
- More complex Boolean expressions are allowed for the pattern_expression in the triggers. For example, the second part of the condition in the simple rule shown above could be “the request is made between 9 a.m. and noon or between 1 p.m. and 5 p.m”, expressed as:

  ```
  ... time=(0900..1200 || 1300..1700) ...
  ```

  Boolean expression are built from the specific values allowed with the trigger, and the Boolean operators ! (not), && (and), || (or) and () for grouping. More details are found in the Trigger Reference chapter. Alternative values may also be separated by a comma—this is often more readable than using the ‘||’ operator. For example, the following rule will deny service to requests for pages in either one of the two domains listed.

  ```
  url.domain=(example.com, another.com) deny
  ```

- Lines can be split using '\' as a line continuation character. The '\' must be the last character on the line and be preceded by space or Tab. For example:

```
url.domain=example.com time=0900..1700 \
 deny
```

Do not use a semicolon to add comments within such a continued line: everything following the semicolon, including text on the continued lines, will be treated as part of the comment. For example:

```
url.domain=example.com \ ; misplaced comment
 deny
```

becomes

```
url.domain=example.com ; misplaced comment deny
```

In other words, the effect was to continue the comment.

Quoting

Certain characters are considered special by CPL and have meaning as punctuation elements of the language. For example = (equal) separates a trigger name from its associated value, and blank space separates expressions in a rule. To use a value that contains one of these characters, the value must be quoted with either single (') or double (") quotation marks, so that the special characters are not interpreted as punctuation. Text within single quotation marks can include any character other than a single quotation mark. Text within double quotation marks can include any character other than a double quotation mark. Here are some examples of where quoting is necessary:

```
user="John Doe" ; value contains a space
url="www.example.com/script.cgi?param=value" ; value contains '='
deny( "You don't have access to that page!" ) ; several special chars
```

The full list of characters that should be quoted when they appear can be found in the reference manual. Note that you can quote any string in CPL without affecting interpretation, even if the quotes are not strictly needed. For convenience, you can quote any value that consists of more than letters and/or numbers.

```
user="john.doe" ; quotes not required, but can be used
```

> **Tip:** Within a "define action" on page 628 or "define url_rewrite" on page 662 statement, you must use double quotes ("), not single quotes (') to delimit a string.

Layers

A policy layer is a CPL construct used to evaluate a set of rules and reach one decision. Separating decisions helps control policy complexity, and is done through writing each decision in a separate layer. Each layer has the form:

```
<layer_type [label]> [layer_condition] [layer_properties] ...
 layer_content
```

where:

- *layer_type*: Defines transactions evaluated against this policy, and restricts the triggers and properties allowed in the rules used in the layer. For more information, see "Understanding Layers" on page 50.
- *label*: A user-defined, unique identifier or quoted string that identifies the purpose of the layer. The label is displayed in compiled policy, policy traces, and policy coverage. Layer labels need only be unique per layer type.

  > **Note:** If policy contains layers with the same name, installing policy results in the message: "Warning: Coverage may not be consistent across policy versions: duplicate layer label". Assign unique labels to layers to easily identify policy rules and ensure the continuity of cumulative policy coverage statistics. See KB article 165841 for more information on Policy Coverage.

- *layer_condition*: List of triggers, all of which must evaluate to true before the layer content is evaluated.
- *layer_properties*: List of properties that will become the default settings for those properties for any rule matched in the layer. These can be overridden by explicitly setting a different value for that property in a specific rule within the layer.
- *layer_content*: List of rules, possibly organized in sections. A layer must contain at least one rule.

  Collectively, the *layer_condition* and *layer_properties* are often referred to as a layer guard expression.

  If a rule has the logical form “if (condition is true) then set properties”, a layer has the form:

```
if (layer_condition is true) then
 {
   if (rule1_condition is true) then
set layer_properties then set rule1 properties
 else if (rule2_condition is true) then
set layer_properties then set rule2 properties
 else if (rule3_condition is true) then
set layer_properties then set rule3 properties
 ...
}
```

  Within a layer, the first rule that matches terminates evaluation of that layer.

  Layers within a policy are evaluated from top to bottom, with rules in later layers taking precedence over rules in earlier layers.

  In CPL, all policy rules are written in a layer. A rule cannot appear in policy preceding any layer header.

Sections

The rules in layers can optionally be organized in one or more sections, which is a way of grouping rules together. A section consists of a section header followed by a list of rules. Performance benefits are most noticeable when the number of rules in the section are greater than 10.

A typical section has the form:

```
[section_type  [label]] [section_condition] [section_properties]
  section_content
```

A policy substitution section has the form:

```
[string[.case_sensitive] substitution-expression [label] [section_condition] [section_properties]
 section_content
```

where:

- *section_type*: Defines the syntax of the rules used in the section, and the evaluation strategy used to evaluate those rules. The square brackets [ ] surrounding the section name (and optional label) are required.
- *label*: A user-defined, unique identifier or quoted string that identifies the purpose of the section. The label is displayed in compiled policy, policy traces, and policy coverage. Section labels need only be unique within the layer.
- *section_condition*: List of triggers, all of which must evaluate to true before the section content is evaluated.
- *section_properties*: List of properties that will become the default settings for those properties for any rule matched in the section. These override any layer property defaults and can in turn be overridden by explicitly setting a different value for that property in a rule within the section.
- `.case_sensitive`: Indicates a case-sensitive match. By default, the match is case-insensitive.
- *section_content*: List of rules, each of which begins with a string criterion followed by other CPL triggers and properties. The string criterion is compared against the value of the *substitution_expression* for the current transaction. Multiple rules may begin with the same string criterion. Rules with matching string criteria will be evaluated in order. For valid substitution expressions, refer to the *ProxySG Log Fields and CPL Substitutions Reference*: https://techdocs.broadcom.com/content/broadcom/techdocs/us/en/symantec-security-software/web-and-network-security/proxysg/7-2/proxysg-log-fields-and-substitutions.html

> **Note:** Policy substitution sections should only be used in the <Proxy> and <Cache> layers.

Collectively, the section_condition and section_properties are often referred to as a section guard expression.

A layer with sections has the logical form:

```
if (layer_condition is true) then
 {
```

```
   if (section1_condition is true then
     {
if (rule1A_condition is true) then
            set layer_properties then section_properties then rule1A properties
        else if (rule1B_condition is true) then
           set layer_properties then section_properties then set rule1B properties
     ....
   }
else if (section2_condition is true then
  {
       if (rule2A_condition is true) then
              set layer_properties then section_properties then rule2A properties
       else ...
    }
 ...
}
```

Definitions

Two types of definitions are used in CPL:

- Named definitions that are explicitly referenced by policy
- Anonymous definitions that apply to all policy evaluation and are not referenced directly in rules .

Named Definitions

There are various types of named definitions. Each definition is given a user defined name that is then used in rules to refer to the definition. This section highlights a few of the definition types, as an overview of the topic. See "Destinations" on page 627 for details.

Subnet Definitions

Subnet definitions are used to define a list of IP addresses or IP subnet masks that can be used to test any of the IP addresses associated with the transaction, for example, the client's address or the request's destination address.

Condition Definitions

Condition definitions can include any triggers that are legal in the layer referencing the condition. The condition= trigger is the exception to the rule that triggers can test only one aspect of a transaction. Since conditions definitions can include other triggers, condition= triggers can test multiple parts of the transaction state. Also, condition definitions allow for arbitrary Boolean combinations of trigger expressions.

Category Definitions

Category definitions are used to extend vendor content categories or to create your own. These categories are tested (along with any vendor defined categories) using the category= trigger.

Action Definitions

An action takes arguments and is wrapped in a named action definition block. Actions are turned on or off for a transaction through setting "action()" on page 337. The action property has syntax that allows for individual actions to be turned on and off independently. When the action definition is turned on, any actions it contains operate on their respective arguments.

Transformer Definitions

A transformer definition is a kind of named definition that specifies a transformation that is to be applied to an HTTP response. There are three types:"define url_rewrite" on page 662, "define active_content" on page 630, and "define javascript" on page 644.

Anonymous Definitions

Two types of anonymous definitions modify policy evaluation, but are not referenced by any rules. These definitions serve to restrict DNS and Reverse-DNS lookups and are useful in installations where access to DNS or Reverse-DNS resolution is limited or problematic.

Referential Integrity

Policy references many objects defined in system configuration, such as authentication realms, forward hosts, SOCKS gateways, and the like. CPL enforces the integrity of those references by ensuring that the entities named in policy exist and have appropriate characteristics at the time the policy is compiled. During runtime, any attempts to remove a configured object that is referenced by currently active policy will fail.

To remove a configured entity, such as a realm, that is referenced by policy, new policy must be installed with all references to that realm removed. New transactions will open against a version of policy that does not require the realm. Once all outstanding transactions that required reference to the realm have completed, the realm can be removed from configuration.

Substitutions

The actions used to rewrite the URL request or to modify HTTP request headers or HTTP response headers often need to reference the values of various elements of the transaction state when constructing the new URL or header value. CPL provides support for various substitutions, which will expand at runtime to the indicated transaction value. Substitutions have the form:

```
$(name)
```

For example, the substitution `$(user)` expands to the authenticated user name associated with the transaction. If policy did not require that user to authenticate, the substitution expands to an empty string.

Substitutions can also be used directly in the values specified to some CPL properties, such as when setting text in a message that will be displayed to users.

Substitutions are available for a variety of purposes. For a categorized list of the substitutions available, refer to the *ProxySG Log Fields and CPL Substitutions Reference*: https://techdocs.broadcom.com/content/broadcom/techdocs/us/en/symantec-security-software/web-and-network-security/proxysg/7-2/proxysg-log-fields-and-substitutions.html

Writing Policy Using CPL

A policy file is the unit of integration used to assemble policy.

Policy written in CPL is stored in one of four files on the appliance. These files are the following:

- VPM: This file is reserved for use by the Visual Policy Manager.
- Local: When the VPM is not being used, the Local file will typically contain the majority of the policy rules for a system. When the VPM is being used, this file might be empty, it might include rules for advanced policy features that are not available in the VPM, or it might otherwise supplement VPM policy.
- Central: For users with a single ProxySG appliance, this file is where you can manually define policy statements; an alternative to Local policy. If you have multiple appliances, Central policy is a way for you to manage common policy among several appliances in your network and generate a CPL file, hosted on a server, that's accessible to all appliances. Each appliance configured with a remote URL regularly checks and updates policy if an update is available.
- Forward: The Forward policy file is normally used for all Forward policy, although you can use it to supplement any policy created in the other three policy files.

Each of the files may contain rules and definitions, but an empty file is also legal. (An empty file specifies no policy and has no effect on the appliance.)

Cross file references are allowed but the definitions must be installed before the references, and references must be removed before definitions are removed.

The final installed policy is assembled from the policy stored in the four files by concatenating their contents. The order of assembly of policy files is configurable. The appliance processes all requests in the following order by default:

- CachePulse (if enabled)
- Landlord
- VPM
- Local
- Tenant (tenant policy or default tenant policy, if enabled)
- Central
- Forward

> **Tip:** If you import policy that was created externally (for example, by a third-party tool) into a policy file, the ProxySG appliance validates the policy file's contents for syntax errors but cannot check for formatting or typographical mistakes. Such errors may result in unintended behavior after the policy is loaded. Symantec strongly recommends that you review the CPL in imported policy for correctness before installing the file.

Authentication and Denial

One of the most important timing relationships to be aware of is the relation between authentication and denial. Denial can be done either before or after authentication, and different organizations have different requirements. For example, suppose an organization requires the following:

Protection from denial of service attacks by refusing traffic from any source other than the corporate subnet.

The user name of corporate users is to be displayed in access logs, even when the user request has been denied.

The following example demonstrates how to choose the correct CPL properties. First, the following is a sample policy that is not quite correct:

```
define subnet corporate_subnet
  10.10.12.0/24
end

<Proxy>
  client.address=!corporate_subnet deny ; filter out strangers
  authenticate(MyRealm) ; this has lower precedence than deny

<Proxy>
; user names will NOT be displayed in the access log for the denied requests
  category=Gambling exception(content_filter_denied)
```

In this policy, requests coming from outside the corporate subnet are denied, while users inside the corporate subnet are asked to authenticate.

Content categories are determined from the request URL and can be determined before authentication. Deny has precedence over authentication, so this policy denies the user request before the user is challenged to authenticate. Therefore, the user name is not available for access logging. Note that the precedence relation between deny and authenticate does not depend on the order of the layers, so changing the layer order will not affect the outcome.

The property `force_authenticate()`, however, has higher precedence than `deny`, so the following amended policy ensures that the user name is displayed in the access logs:

```
define subnet corporate_subnet
  10.10.12.0/24
end
```

```
<Proxy>
  client.address=!corporate_subnet deny ; filter out strangers
  force_authenticate(MyRealm) ; this has higher precedence than deny

<Proxy>
; user names will be displayed in the access log for the denied requests
  category=Gambling exception(content_filter_denied)
```

The timing for authentication over the SOCKS protocol is different. If you are using the SOCKS authentication mechanism, the challenge is issued when the connection is established, so user identities are available before the request is received, and the following policy would be correct.

```
define subnet corporate_subnet
  10.10.12.0/24
end

<Proxy>
  client.address=!corporate_subnet deny ; filter out strangers
  socks.authenticate(MyRealm) ; this happens earlier than the category test

<Proxy>
; user names be displayed in the access log for the denied requests
  category=Gambling exception(content_filter_denied)
```

Note that this only works for SOCKS authenticated users.

Installing Policy

Policy is installed by installing one of the four policy files (VPM, Local, Central, Forward). Installing one new file causes the most recent versions of the other three files to be loaded, the contents concatenated in the order specified by the current configuration, and the resulting complete policy compiled.

If any compilation errors are detected, the new policy file is not installed and the policy in effect is unchanged.

Refer to the *Visual Policy Manager Reference* or *Web Visual Policy Manager Reference* for specific instructions on installing a policy file.

CPL General Use Characters and Formatting

The following characters and formatting have significance within policy files in general, outside of the arguments used in condition expressions, the values used in property statements, and the arguments used in actions.

Character	Example	Significance
Semicolon (;)	`; Comment` `<Proxy> ; Comment`	Used either in-path or at the beginning of a line to introduce text to be ignored during policy evaluation. Commonly used to provide comments.

Character	Example	Significance
Newline	`deny server_url.scheme=mms deny server_url.domain=xyz.com`	CPL expects most constructs (layers, sections, rules, definitions) to begin on a new line. When not preceded by a line continuation character, a newline terminates a layer header, section header, the current rule, clause within a defined condition, or action within an action definition.
Line Continuation	`\`	A line continuation character indicates that the current line is part of the previous line.
Whitespace	`< proxy >` `weekday = ( 3 || 7 ) deny`	Used to enhance readability. Whitespace can be inserted between tokens, as shown in this example, without affecting processing. In addition, quoted strings can include whitespace. However, numeric ranges, such as weekday = 1..7, cannot contain whitespace.
Angle brackets (< >)	`<Proxy>`	Used to mark layer headings.
Square brackets ([ ])	`[Rule]`	Used to mark section names.
Equal sign (=)	`server_url.scheme=mms`	Used to indicate the value a condition is to test.
Parentheses ( )	`max_bitrate(no)`	Used to enclose the value that a property is to be set to, or group components of a test.

Troubleshooting Policy

When installed policy does not behave as expected, use policy tracing to understand the behavior of the installed policy.

Tracing records additional information about a transaction and re-evaluates the transaction when it is terminated; however, it does not show the timing of evaluations through transaction processing. The extra processing required significantly impacts performance, so do not enable tracing in production environments unless you need to reproduce and diagnose a problem. If tracing is used on a system in production, attempt to restrict which transactions are traced. For example, you can trace only requests from a test workstation by defining the tracing rules as conditional on a "client.address=" on page 87 trigger that tests for that workstation's IP address.

For more information on generating and retrieving policy trace, see "Testing and Troubleshooting" on page 732.

While policy traces can show the rule evaluation behavior, they do not show the final effect of policy actions like HTTP header or URL modifications. To see the result of these policy actions it is often useful to actually view the packets sent and received. The PCAP facility can be used in conjunction with tracing to see the effect of the actions set by the matching rules.

Upgrade/Downgrade

Specific upgrade and downgrade issues will be mentioned in the release notes accompanying your version of SGOS. This section highlights general upgrade downgrade issues related to policy written in CPL.

CPL Syntax Deprecations

Older language constructs may be replaced with new constructs of equal or greater power; however, this means that support for old language constructs will eventually be dropped to help maintain the runtime efficiency of evaluation. As part of the migration strategy, the CPL compilation warnings might include warnings regarding the use of deprecated constructs. This class of warning is special, and indicates use of a CPL language element that will not be supported in the next major release. Eliminate deprecation warnings by migrating the policy identified by the warning to more modern syntax, which is usually indicated in the warning message. Attempts to upgrade to the next major release might fail, or result in a failure to load policy, unless all deprecation warnings are eliminated.

Conditional Compilation

Occasionally, you might have to write policy for different products, software versions, or enforcement domains, which require different CPL. CPL provides the following conditional compilation directives:

To test the software version:

`release.version=`*`version_number_range`*

where the *`version_number_range`* is an integer or range of versions, such as `min..max`. Numeric conditions can use Boolean expressions and double periods (..), meaning an inclusive numeric range. Numeric ranges cannot use whitespace.

Express min and max in the format:

`MAJOR.MINOR.DOT.PATCH`

where *`MINOR`*, *`DOT`* , and *`PATCH`* are optional.

For example, protect policy applicable to 6.6.x versions up to 6.6.5.4 with:

```
#if release.version=6.6..6.6.5.4
; guarded rules
...
#endif
Protect policy introduced in 6.7.x with:
#if release.version=6.7..
; guarded rules
...
#endif
```

To test the product:

`product=product_abbreviation`

where the product_abbreviation is either `asg` for the Advanced Secure Gateway appliance or `sg` for the ProxySG appliance.

For example, protect Advanced Secure Gateway-specific policy with:

```
#if product=asg
; guarded rules
...
#endif
```

Protect ProxySG-specific policy with:

```
#if product=sg
; guarded rules
...
#endif
```

To test the enforcement domain:

`enforcement=enforcement_domain`

where *enforcement_domain* is either `appliance` for the ProxySG appliance or `wss` for Symantec Web Security Service

For example, protect appliance-specific policy with:

```
#if enforcement=appliance
; guarded rules
...
#endif
```

Protect cloud-specific policy with:

```
#if enforcement=wss
; guarded rules
...
#endif
```

There is no universal directive because all policy rules that are not protected with either the appliance or wss directive are enforced both on the appliance and in Web Security Services.

Glossary

Term	Definition
action	A class of definitions. CPL has two general classes of actions: request or response modifications and notifications. An action takes arguments (such as the portion of the request or response to modify) and is wrapped in a named action definition block. When the action definition is turned on by the policy rules, any actions it contains operate on their respective arguments.
<Admin> layer	One of the layer types allowed in a policy. Used to define policy rules that control access to the Management Console and command line interface (CLI).
admin transaction	Encapsulation of a request to manage the appliance for the purposes of policy evaluation. Policy in <Admin> layers applies to admin transactions. Additionally, if the user is explicitly proxied to the appliance, a proxy transaction will also be created for the request.
allow	The preferred short form of exception(no), a property setting that indicates that the request should be granted. A default rule for the proxy policy layer. You have two choices: allow or deny. Deny prevents any access to the appliance; allow permits full access to the appliance.
<Cache> layer	One of the layer types allowed in a policy. Used to list policy rules that are evaluated during a cache or proxy transaction.
<DNS-Proxy> layer	One of the layer types allowed in a policy. Used to list policy rules that are evaluated during a cache or proxy transaction.
<Diagnostic> layer	One of the layer types allowed in a policy. Used to list policy rules that are evaluated during a DNS proxy transaction.
cache transaction	Encapsulation of a request, generated by the appliance and directed at an upstream device, for the purposes of maintaining content in the local object store.
Central Policy File	For users with a single ProxySG appliance, this file is where you can manually define policy statements; an alternative to Local policy. If you have multiple appliances, Central policy is a way for you to manage common policy among several appliances in your network and generate a CPL file, hosted on a server, that's accessible to all appliances. Each appliance configured with a remote URL regularly checks and updates policy if an update is available.
condition	A Boolean combination of trigger expressions that yields true or false when evaluated.
default policy	The default settings for various transaction properties taken from configuration. An important example is the default proxy policy that is configurable to either allow or deny.
definition	A definition binds a user-defined label to a condition, a content category, a transformation or a group of actions.

Term	Definition
deny	The preferred short form of `exception(policy_denied)`, a property setting that indicates that the request should be refused.
Evaluation order	The order in which the four policy files—Central, Local, VPM, and Forward—are evaluated. When a file is evaluated last, the policy rules and the related configuration settings it specifies can override any settings triggered in the other files. The order of evaluation of the Central, Local, and VPM policy files is configurable using the policy order CLI command or the Management Console. The Forward file is always last in the evaluation order.
`<Exception>` layer	One of the layer types allowed in a policy. Exception layers are evaluated when an exception property is set, forcing transaction termination. Policy in an exception layer gives the administrator a final chance to modify the properties (such as headers) of the response (exception) object, just as they would get a chance to modify the properties of an object returned from the origin server or from cache.
`<Forward>` layer	One of the layer types allowed in a policy. `<Forward>` layers are only evaluated when the current transaction requires an upstream connection.
Forward Policy File	A file that you maintain to supplement any policy described in the other three policy files. It is normally used for forwarding policy. The Forward policy file is always last in the evaluation order. Forwarding policy is generally distinct and independent of other policies, and is often used as part of maintaining network topologies. Forwarding policy can also be created and maintained through the Visual Policy Manager.
layer	A CPL construct for expressing the rules for a single policy decision. Multiple layers can be used to make multiple decisions. Layers are evaluated in top to bottom order. Decisions made by later layers can override decisions made by earlier layers. Layer evaluation terminates on the first rule match. Five layer types exist. The layer type defines the transactions evaluated against this policy and restricts the triggers and properties allowed in the rules used in the layer. Each of the five types of layers are allowed in any policy file.
Local Policy File	A file you create and maintain on your network for policy specific to one or more appliances. This is the file you would normally create when writing CPL directly with a text editor, for use on some subset of the appliances in your organization.
Match	When a rule is evaluated, if all triggers evaluate to true, then all properties specified are set. This is often referred to as a rule Match (for example in policy tracing.)
Miss	When a rule is evaluated, if any trigger evaluates to false, all properties specified are ignored. This is often referred to as a rule Miss (for example in policy tracing.)
N/A	The rule can't be evaluated for this transaction and is being skipped. N/A happens, for example, when you try to apply a streaming condition to an FTP transaction.
OK	Specifies no action on a policy rule.
policy files	Any one of files that contain CPL. When the policy is installed, the contents of each of the files is concatenated according to the evaluation order.

Term	Definition
policy trace	A listing of the results of policy evaluation. Policy tracing is useful when troubleshooting policy.
property	A CPL setting that controls some aspect of transaction processing according to its value. CPL properties have the form property(setting). At the beginning of a transaction, all properties are set to their default values, many of which come from the configuration settings.
<Proxy> layer	One of the layer types allowed in a policy, used to list policy rules that control access to proxy services configured on the appliance. Rules in the <Proxy> layer include user authentication and authorization requirements, time of day restrictions, and content filtering.
proxy transaction	A transaction created for each request received over the proxy service ports configured on the appliance. The proxy transaction covers both the request and its associated response, whether fetched from the origin server or the local object store.
request transformation	A modification of the request for an object (either the URL or Headers). This modification might result in fetching a different object, or fetching the object through a different mechanism.
response transformation	A modification of the object being returned. This modification can be to either the protocol headers associated with the response sent to the client, or a transformation of the object contents itself, such as the removal of active content from HTML pages.
rule	A list of triggers and property settings, written in any order. A rule can be written on multiple lines using a line continuation character. If the rule matches (all triggers evaluate to true), all properties will be set as specified. At most one rule per layer will match. Layer evaluation terminates on the first rule match.
<SSL> layer	One of the layer types allowed in a policy. Used to list policy rules with SSL triggers and properties, not including SSL interception; creates conditions for SSL interception.
<SSL-Intercept> layer	One of the layer types allowed in a policy. Used to list policy triggers and properties that define the conditions under which SSL connections are intercepted and decrypted or tunneled.
section	A way of grouping rules of like syntax together. Sections consist of a section header that defines the section type, followed by policy rules. The section type determines the allowed syntax of the rules, and an evaluation strategy.
transaction	An encapsulation of a request to the appliance together with the resulting response that can be subjected to policy evaluation. The version of policy current when the transaction starts is used for evaluation of the complete transaction, to ensure consistent results.

Term	Definition
Visual Policy Manager file	A file created and stored on an individual appliance by the Visual Policy Manager. The VPM allows you to create policies without writing CPL directly. Because the VPM supports a subset of CPL functionality, you might want to supplement any policy in a VPM file with rules in the Local policy file. If you have a new appliance, the VPM file is empty. VPM files can be shared among various appliances by copying the VPM files to a Web server and then using the Management Console or the CLI from another appliance to download and install the files.

Additional SGOS Documentation

In addition to the following, Symantec provides other deployment guides targeted for specific solutions. For these and other documents, refer to: https://techdocs.broadcom.com/content/broadcom/techdocs/us/en/symantec-security-software/web-and-network-security.html

Document	Overview
SGOS Upgrade/Downgrade Guide https://techdocs.broadcom.com/content/broadcom/techdocs/us/en/symantec-security-software/web-and-network-security/proxysg/7-2/admin-guide.html	Steps for upgrading or downgrading SGOS. Also covers behavior changes and policy deprecations.
SGOS Administration Guide https://techdocs.broadcom.com/content/broadcom/techdocs/us/en/symantec-security-software/web-and-network-security/proxysg/7-2/admin-guide.html	Detailed information for configuring and managing the ProxySG appliance.
Command Line Interface Reference https://techdocs.broadcom.com/content/broadcom/techdocs/us/en/symantec-security-software/web-and-network-security/proxysg/7-2/command-line-interface-reference.html	Commands available in the ProxySG appliance CLI and how to use them to perform configuration and management tasks.
ProxySG Web Visual Policy Manager Reference https://techdocs.broadcom.com/content/broadcom/techdocs/us/en/symantec-security-software/web-and-network-security/proxysg/7-2/web-vpm-policy.html	How to create and implement policy in the ProxySG appliance's web-based Visual Policy Manager, including layer interactions, object descriptions, and advanced tasks.
Legacy Visual Policy Manager Reference https://techdocs.broadcom.com/content/broadcom/techdocs/us/en/symantec-security-software/web-and-network-security/proxysg/7-2/legacy-vpm-policy.html	How to create and implement policy in the ProxySG appliance's legacy Visual Policy Manager.
Required ports, protocols, and services for the ProxySG appliance https://techdocs.broadcom.com/content/broadcom/techdocs/us/en/symantec-security-software/web-and-network-security/proxysg/7-2/ports-reference.html	Basic configurations, and some commonly used options, for ports and protocols.
ProxySG Security Best Practices https://techdocs.broadcom.com/content/broadcom/techdocs/us/en/symantec-security-software/web-and-network-security/proxysg/7-2/security-best-practices.html	Best-effort security considerations for your ProxySG deployment.

Document	Overview
SSL Proxy Deployment Guide https://techdocs.broadcom.com/content/broadcom/techdocs/us/en/symantec-security-software/web-and-network-security/proxysg/7-2/ssl-proxy-deployment.html	Best practices for deploying the SSL proxy. The SSL proxy improves visibility into SSL traffic, allowing security policies and logging to be applied to encrypted requests and responses, and can enhance performance by caching encrypted data.
Reverse Proxy Deployment Guide https://techdocs.broadcom.com/content/broadcom/techdocs/us/en/symantec-security-software/web-and-network-security/proxysg/7-2/reverse-proxy-deployment-guide.html	How to deploy a ProxySG appliance as a front-end for Internet-based users to access secure application, content, and web servers.
Web Application Firewall Solutions Guide https://techdocs.broadcom.com/content/broadcom/techdocs/us/en/symantec-security-software/web-and-network-security/proxysg/7-2/web-application-firewall-solutions-guide.html	How to configure Symantec WAF solution to protect your web servers, accelerate web content, and simplify operation.
Secure Web Gateway - Content Analysis Policy Best Practices Improvement https://techdocs.broadcom.com/content/broadcom/techdocs/us/en/symantec-security-software/web-and-network-security/proxysg/7-2/swg-content-analysis-policy-best-practices.html	Provides a secure and customizable policy model for bypassing content scanning to improve the user experience where needed or save resources by excluding low-risk/high-volume traffic. This document identifies weak policy conditions and is intended to reduce risk by using different sets of policy conditions.

Managing CPL

Policy written in CPL is composed of transactions that are placed into rules and tested against various conditions.

Refer to the following topics for more information:

- "Understanding Transactions and Timing" on the next page
- "Understanding Layers" on page 50
- "Understanding Sections" on page 56
- "Defining Policies" on page 59
- "Best Practices" on page 63

Understanding Transactions and Timing

Transactions are classified as in several different types, described below.Only a subset of layer types, conditions, properties, and actions is appropriate for each of these transaction types.

> **Note:** The <DNS-Proxy> and <SSL-Intercept> layers can also be defined and executed in tenant-specific policy.

<Admin> Transactions

An administrator transaction evaluates policy in <Admin> layers. The policy is evaluated in two stages:

- Before the authentication challenge.
- After the authentication challenge.

If an administrative user logs in to the ProxySG Management Console, and the administrator's Web browser is proxied through that same appliance, then a proxy transaction is created and <Proxy> policy is evaluated before the administrator transaction is created and <Admin> policy is evaluated. In this case, it is possible for an administrator to be denied access to the Management Console by proxy policy.

Policy is not evaluated for serial console access, RSA authenticated SSH access, managers logged in using the console account credentials, or SNMP traffic.

<Cache> Transactions

Cache transactions are initiated by the appliance to load or maintain content in the local object store during adaptive refresh or pipelining, or as a result of a content distribute CLI command. These might be HTTP, FTP, or streaming media transactions. Because no specific user is associated with these transactions, content related policy is evaluated for cache transactions, but user related policy is not evaluated.

A cache transaction evaluates policy in <Cache> and <Forward> layers. The <Forward> layers are only evaluated if an origin server must be contacted to complete the transaction.

The following is a list of cache transactions:

- A content distribute transaction that is initiated by the content distribute CLI command. A content distribute transaction may use one of the following protocols: HTTP, HTTPS, Real Media, or Windows Media. This type of transaction may be preceded by a separate Administrator transaction, since the administrator must be authenticated and authorized to use the command.
- Pipeline transactions (HTTP only).

- Advertisement transactions (HTTP only).
- If-modified-since transactions (HTTP only).
- Refresh transactions (HTTP only).
- ICP transactions.

Cache transactions have no client identity since they are generated internally by the appliance, and they do not support authentication or authorization. Therefore, they do not support conditions such as "client.address=" on page 87 and "group=" on page 140 or the "authenticate()" on page 348 property.

An HTTP cache transaction is examined in two stages:

- Before the object is retrieved from the origin server.
- After the object is retrieved.

<DNS-Proxy> Transactions

When a client connects to one of the DNS proxy service ports configured on the appliance, a `<DNS-Proxy>` transaction is created to cover both the request and its associated response.

A `<DNS-Proxy>` transaction evaluates policy in `<DNS-Proxy>` layers only. Policy in other layers does not affect `<DNS-Proxy>` transactions.

Policy for `<DNS-Proxy>` transactions is evaluated in two stages:

- After the DNS request is received
- After the DNS response is available.

<Exception> Transaction

Exception transactions include `<Proxy>` transactions that have been terminated by an exception.

<Forwarding> Transactions

A `<Forwarding>` transaction is created when the appliance needs to evaluate forwarding policy before accessing a remote host and no proxy or cache transaction is associated with this activity. Examples include sending a heart-beat message, and downloading an installable list from an HTTP server.

A `<Forwarding>` transaction only evaluates policy in `<Forward>` layers.

<Proxy> Transactions

When a client connects to one of the proxy service ports configured on the secure proxy appliance, a proxy transaction is created to cover both the request and its associated response. Note that requests for DNS proxy services are handled separately from requests for higher level services; see the following <DNS-Proxy> transactions section.

A proxy transaction evaluates policy in <Proxy>, <Cache>, <Forward>, and <Exception> layers. The <Forward> layers are only evaluated if the transaction reaches the stage of contacting an origin server to satisfy the request (this is not the case if the request is satisfied by data served from cache, or if the transaction is terminated by an exception). The <Exception> layers are only evaluated if the transaction is terminated by an exception.

Each of the protocol-specific proxy transactions has specific information that can be tested—information that may not be available from or relevant to other protocols. HTTP Headers and Instant Messaging buddy names are two examples of protocol-specific information.

Other key differentiators among the proxy transaction subtypes are the order in which information becomes available and when specific actions must be taken, as dictated by the individual protocols. Variation inherent in the individual protocols determines timing, or the sequence of evaluations that occurs as the transaction is processed.

The following table summarizes the policy evaluation order for protocol-specific proxy transactions.

Transaction Type	Policy is Evaluated....
Tunneled TCP	Before the connection is established to the OCS.
HTTP proxy	Before the authentication challenge. After the authentication challenge, but before the requested object is fetched. Before making an upstream connection, if necessary. After the object is fetched
FTP over HTTP	Before the authentication challenge. After the authentication challenge, but before the required FTP commands are executed. Before making an upstream connection, if necessary. After the object is fetched.
Transparent FTP	Policy is examined before the requested object is fetched.

Transaction Type	Policy is Evaluated....
Real Media streaming	Before the authentication challenge. After the authentication challenge, but before getting object information. Before making an upstream connection, if necessary. After the object information is available, but before streaming begins. After streaming begins (this evaluation can be done multiple times, for example after playback is paused and restarted).
Windows Media MMS streaming	Before the authentication challenge. Before making an upstream connection, if necessary. After the authentication challenge but before getting object information. After the object information is available, but before streaming begins. After streaming begins (this evaluation can be done multiple times, for example after playback is paused and restarted).
Windows Media HTTP streaming	Before the authentication challenge. After the authentication challenge, but before the requested object is fetched. Before making an upstream connection, if necessary. (Up to this point it is similar to an HTTP transaction.) What happens at this stage depends on negotiations with the origin server: After the origin server is contacted, if the User Agent header denotes the Windows Media player and the server supports Microsoft streaming HTTP extensions, it finishes like an MMS transaction: Object information is available at this stage but streaming has not begun. If the User-Agent header is not a Windows Media player or the server does not support Microsoft streaming extensions, it finishes like an HTTP transaction: The requested object is fetched, and policy is evaluated

Some conditions cannot be evaluated during the first stage; for example, the user and group information will not be known until stage two. Likewise, the response headers and MIME type are unavailable for testing until stage three. For conditions, this is known as the earliest available time.

Policy decisions can have similar timing considerations, but this is known as the latest commit time. In this example, the requirement to authenticate must be known at stage one, and a forwarding host or gateway must be determined by stage three.

<SSL> Transactions

Two types of <SSL> transactions exist:

- `<SSL>`: This includes cache and proxy transactions, except for `<SSL-Intercept>` transactions. Note that in VPM, `<SSL>` transactions are referred to as SSL access transactions.
- `<SSL-Intercept>`: A kind of proxy transaction whose purpose is to decide whether or not to intercept and decrypt an SSL connection, or leave it encrypted and simply tunnel it.

<Tenant> Transactions

This layer type is found only in the CPL-based Landlord policy slot. Use of this policy slot is exclusively for the purpose of determining the identify of a tenant in a multi-tenant policy configuration. For more information on multi-tenant policy, refer to the *Multi-Tenant Policy Deployment Guide* at MySymantec.

Timing

As stated in the discussion of proxy transactions, various portions of the transaction information become available at different points in the evaluation, and each protocol has specific requirements for when each decision must be made. The CPL triggers and properties are designed so that wherever possible, the policy writer is shielded from the variations among protocols by making the timing requirements imposed by the CPL accommodate all the protocols. Where this is not possible (because using the most restrictive timing causes significant loss of functionality for the other protocols), protocol-specific triggers have been introduced. When evaluated against other protocols, these triggers return the not applicable value or N/A. This results in the rule being skipped (the expression evaluates to false, no matter what it is). It is possible to explicitly guard such rules so that they are only evaluated against appropriate transactions.

The variation in trigger and property timings implies that within a policy rule a conflict is possible between a condition that can only be tested relatively late in the evaluation sequence and a property that must be set relatively early in the evaluation sequence. Such a rule results in a compile-time error.

For example, here is a rule that would be incorrect for evaluating any transaction:

If the user is in group xyz, require authentication.

The rule is incorrect because group membership can only be determined after authentication and the rule tests group membership and specifies the authentication realm, a property that must be set before the authentication challenge can be issued. The following code illustrates this incorrect rule and the resulting message at compile time:

```
group=xyz authenticate(MyRealm)
Error: Late condition guards early action: 'authenticate(MyRealm)'
```

It is, however, correct for the authentication requirement to be conditional on the client addresss ("client.address=" on page 87) or proxy port ("proxy.port=" on page 200), as these can be determined at the time the client connection is established and therefore are available from the beginning of a proxy transaction.

For the HTTP protocol, "authenticate()" on page 348 can be conditional on the URL ("url=" on page 298), but for MMS streaming, only the Host portion of the URL can be tested (url.host=). Refer to the outline of the evaluation model for Windows Media transactions in "<Proxy> Transactions" on page 46.

Understanding Layers

ProxySG policy supports the following layer types. The layer type determines the validity of CPL gestures that can be used in the layer, and the kinds of transaction its rules can act upon.

> **Tip:** Not all of the policy gestures available are permitted within each layer type. Using an unsupported gesture results in compile-time errors. Refer to the specific policy gesture in this document to determine which layers are supported.

<Admin> Layer

The `<Admin>` layer defines policy that:

- Controls access to the Management Console and the CLI. Policy is not evaluated for serial console access or SNMP traffic.
- Is executed by Administrator transactions.
- Specifies an authentication realm.
- Allows or denies administrative access based on the client's IP address, credentials, and type of administrator access requested (read or write).
- Performs any additional logging for administrative access.

When traffic is explicitly proxied, it arrives at the `<Admin>` layer with the client IP address set to the appliance's IP address; thus, the "client.address=" on page 87 condition is not useful for explicitly proxied traffic.

<Cache> Layer

The `<Cache>` layer defines policy that:

- Is evaluated during both cache and proxy transactions.
- Is executed by both cache and proxy transactions. Since cache transactions have no concept of a client, all `<Cache>` layer policy is clientless, so you cannot test client identity using "client.address=" on page 87, "user=" on page 312, "group=" on page 140, and the like.

  Certain types of policy must be applied consistently to both proxy and cache transactions to preserve cache consistency. Such policy must not be conditional on client identity or time of day, and belongs in a `<Cache>` layer. Examples include the following:

- Response virus scanning.
- Cache control policy (other than "bypass_cache()" on page 373).
- Modifications to request headers, if the modification affects the content returned by the Web server, and the content is cached.
- Rewrites of the request URL that modify the server URL but not the cache URL. (Place rewrites of the request URL that change the cache and server URL to the same value in a `<Proxy>` layer.)

- Only the following properties are safe to make conditional on time or client identity in a `<Cache>` layer:
  - Pipelining
  - Tracing, logging
  - Freshness checks
  - Redirection
  - Content transforms

<Diagnostic> Layer

The `<Diagnostic>` layer defines policy that:

- Includes diagnostic information about the transaction. Policy rules in this layer have no effect on traffic.
- Allows transaction variables and can test variables set in other types of layers. See "Variables" on page 671.
- Aids with policy forensics without affecting the behavior of the regular policy.

<DNS-Proxy> Layer

The `<DNS-Proxy>` layer defines policy that controls only DNS transactions.

<Exception> Layer

The `<Exception>` layer defines policy that:

- Is evaluated when a proxy transaction is terminated by an exception. Exceptions can be caused by a bad request (for example, the request URL names a non-existent server) or by setting "deny" on page 392 or "exception()" on page 402 in policy.
- Can be used to control how access logging is performed for exceptions, such as `authentication_failed`.
- Can be used to modify the HTTP response headers in the exception page that is sent to the client.

<Forward> Layer

The <Forward> layer defines policy that:

- Is evaluated when the current transaction requires an upstream connection (forward policy will not be evaluated for a cache hit). Forwarding policy is generally distinct and independent of other policies, and is often used as part of maintaining network topologies.
- Uses "server_url=" on page 269 tests rather than "url=" on page 298 tests so that they are guaranteed to honor any policy that rewrites the URL.

<Proxy> Layer

The <Proxy> layer defines policy that:

- Evaluates only proxy transactions, except for DNS transactions. <DNS-Proxy> transactions is distinct from policy for other proxy services.
- Authenticates and authorizes users' requests for service over one of the configured proxy service ports. Proxy layer policy involves both client identity and content.

<SSL> Layer

The <SSL> layer defines policy that:

- Stores additional SSL triggers and properties that are unrelated to SSL interception. This layer is evaluated by all cache and proxy transactions except the <SSL-Intercept> and <DNS-Proxy> transactions.
- Creates interception conditions for SSL connections.

<SSL-Intercept> Layer

The <SSL-Intercept> layer defines policy that:

- Lists policy triggers and properties that define the conditions under which SSL connections are intercepted and decrypted or tunneled. Only the conditions, actions, and properties essential to make the tunnel or intercept decision are allowed in this layer.

> **Note:** In order to specify a CA Certificate List (CCL), configure the <SSL-Intercept> layer to match a specific server name indication (SNI). The matched SNI then uses the certificate store defined by the CCL-ID. See the "server.certificate.validate.ccl()" on page 538 property. If no host URL or IP address is configured, the policy is applied to all intercepted connections.

<Tenant> Layer

The `<Tenant>` layer defines policy that:

- Is used only in the landlord policy slot in multi-tenant deployments to define tenant criterion. This layer type is not available in any other policy.
- Is used in deployments that make use of multi-tenant policy as an optional mechanism to set the criterion for a tenant and name the tenant policy slot.

<Tenant> layer syntax

```
<Tenant>
  tenant_criterion_rules
```

where:

- Rules are triggered by a CPL condition that is expected to match the requests of multiple clients.
- Only the action of tenant is permitted to follow a condition.
- Only one `<Tenant>` layer is required. All tenant criterion rules are expected to exist in this layer.
- Use of the landlord policy slot and `<Tenant>` layer is optional and supersedes the use of the CLI command `#(config general) multi-tenant policy criterion` *criterion*.
- Unlike the CLI command, which uses the criterion value as the tenant ID, rules in the `<Tenant>` layer can define a text string that sets the tenant ID.

Layer Syntax

This syntax applies to all layers except the `<Tenant>` layer.

```
<layer_type ["comment"]> [layer_type_condition][layer_type_properties] ...
  layer_type_rules
```

where:

- *comment*: Identifier or quoted string that is displayed in policy traces to identify the purpose of the layer.
- *layer_type_condition*: List of triggers, all of which must evaluate to true before the layer content is evaluated. For more information on using conditions, see "Conditions" on page 65. See "Layer Guards" on the next page.
- *layer_type_properties*: List of properties set if any of the rules in the layer match. These act as defaults, and can be overridden by property settings in specific rules in the layer. For more information on using properties, see "Properties" on page 332. See "Layer Guards" on the next page.
- *layer_type_rules*: List of rules representing the decisions to be made by this policy layer.

Layer Guards

When writing policy, you might put the same conditions or properties in each rule in a layer, such as the following:

```
<Proxy>
 group=general_staff url.domain=competitor.com/jobs deny
 group=general_staff url.host=bad_host deny
 group=general_staff condition=whatever deny
 ; etc.
 group=general_staff allow
```

To make policy cleaner and easier to read, you can factor out the common elements into guard expressions. Notice that the common elements are `group=general_staff` and `deny`. The following is the same policy, expressed as a layer employing a guard expression:

```
<Proxy> group=general_staff deny
 url.domain=competitor.com/jobs
 url.host=bad_host
 condition=whatever
 ; etc.
 allow
```

Note that the explicit allow overrides the deny specified in the layer guard. This is an instance of a rule-specific property setting overriding the default property settings specified in a guard expression.

Timing

The late guards early timing errors that can occur within a rule can arise across rules in a layer. When a trigger cannot yet be evaluated, policy also has to postpone evaluating all following rules in that layer (since if the trigger turns out to be true and the rule matches, then evaluation stops for that layer. If the trigger turns out to be false and the rule misses, then evaluation continues for the rest of the rules in that layer, looking for the first match). Thus a rule inherits the earliest evaluation point timing of the latest rule above it in the layer.

For example, consider the following rule:

```
group=xyz authenticate(MyRealm)
```

This rule would result in a timing conflict error:

```
Error: Late condition guards early action: 'authenticate(MyRealm)'
```

Similarly, consider the following layer:

```
<Proxy>
  group=xyz deny
  authenticate(MyRealm)
```

This rule would result in a timing conflict error:

```
Error: Late condition 'group=xyz' guards early action: 'authenticate(MyRealm)'
```

This also extends to guard expressions, as the guard condition must be evaluated before any rules in the layer:

```
<Proxy> group=xyz deny
  authenticate(MyRealm)
```

This rule would result in a timing conflict error:

```
Error: Late condition 'group=xyz' guards early action: 'authenticate(MyRealm)'
```

Understanding Sections

The rules in layers can optionally be organized in one or more sections, which is a way of grouping rules together. A section consists of a section header followed by a list of rules.

Four sections types are supported in a standard CPL file:

- `[Rule]`
- `[url]`
- `[url.domain]`
- `[server_url.domain]`

Three of the section types, `[url]`, `[url.domain]`, and `[server_url.domain]`, provide optimization for URL tests. The names for these sections correspond to the CPL URL triggers used as the first test for each rule in the section, that is "url=" on page 298, and server_url.domain=. The `[url.regex]` section provides factoring and organization benefits, but does not provide any performance advantage over using a `[Rule]` section and explicit url.regex= tests.

To give an example, the following policy layer is created:

```
<Proxy>
  url.domain=abc.com/sports deny
  url.domain=nbc.com/athletics deny
  ; etc, suppose it's a substantial list
  url.regex="sports|athletics" access_server(no)
  url.regex="\.mail\." deny
  ; etc
  url=www.symantec.com/internal group=!symantec_employees deny
  url=www.symantec.com/proteus group=!symantec_development deny
; etc
```

This can be recast into three sections:

```
<Proxy>
  [url.domain]
  abc.com/sports deny
  nbc.com/athletics deny
  ; etc.
[Rule]
url.regex="sports|athletics" access_server(no)
url.regex="\.mail\." deny
[url]
www.symantec.com/internal group=!symantec_employees deny
www.symantec.com/proteus group=!symantec_development deny
```

Notice that the first thing on each line is not a labelled CPL trigger, but is the argument for the trigger assumed by the section type. Also, after the first thing on the line, the rest of the line is the familiar format.

The performance advantage of using the [url], [url.domain], or [server_url.domain] sections is measurable when the number of URLs being tested reaches roughly 100. Certainly for lists of several hundred or thousands of URLs, the performance advantage is significant.

When no explicit section is specified, all rules in a layer are assumed to be in a [Rule] section. That is, the first example is equivalent to:

```
<Proxy>
  [Rule]
  url.domain=abc.com/sports deny
  url.domain=nbc.com/athletics deny
  ; etc, suppose it's a substantial list
  url.regex="sports|athletics" access_server(no)
  url.regex="\.mail\." deny
  ; etc
  url=www.symantec.com/internal group=!symantec_employees deny
  url=www.symantec.com/proteus group=!symantec_development deny
  ; etc
```

[Rule]

The [Rule] section type is used to logically organize policy rules into a section, optionally applying a guard to the contained rules. The [Rule] section was so named because it can accept all rules in a policy. If no section is specified, all rules in a layer are assumed to be in a [Rule] section.

- Use [Rule] sections to clarify the structure of large layers. When a layer contains many rules, and many of the rules have one or more conditions in common, you may find it useful to define [Rule] sections.
- Rules in [Rule] sections are evaluated sequentially, top to bottom. The time taken is proportional to the number of rules in the section.
- [Rule] sections can be used in any layer.

[url]

The [url] section type is used to group a number of rules that test the URL. The [url] section restricts the syntax of rules in the section. The first token on the rule line is expected to be a pattern appropriate to a "url=" on page 298 trigger. The trigger name is not included.

Rules in [url] sections are evaluated through hash table techniques, with the result that the time taken is not dependent on the number of rules in the section.

[url] sections are not allowed in <Admin> or <Forward> layers.

[url.domain]

The [url.domain] section is used to group a number of rules that test the URL domain. The [url.domain] section restricts the syntax of rules in the section. The first token on the rule line is expected to be a pattern appropriate to a url.domain= trigger. The trigger name is not included.

Rules in [url.domain] sections are evaluated through hash table techniques, with the result that the time taken is not dependent on the number of rules in the section.

[url.domain] sections are not allowed in <Admin> or <Forward> layers.

[url.regex]

The [url.regex] section is used to test the URL. The [url.regex] section restricts the syntax of rules in the section. The first token on the rule line is expected to be a pattern appropriate to a url.regex= trigger. The trigger name is not included. The [url.regex] section replaces the [Regex] section used in previous versions of CPL.

- Rules in [url.regex] sections are evaluated sequentially, top to bottom. The time taken is proportional to the number of rules in the section.
- [url.regex] sections are not allowed in <Admin>, <DNS-Proxy>, or <Forward> layers.

[server_url.domain]

The [server_url.domain] section is used to test the domain of the URL used to fetch content from the origin server. The [server_url.domain] section restricts the syntax and rules in the section. The first token on the rule line is expected to be a pattern appropriate to a server_url.domain= trigger. The trigger name is not included.

[server_url.domain] sections contain policy rules very similar to [url.domain] sections except that these policy rules test the server_url, which reflects any rewrites to the request URL.

- Rules in [server_url.domain] sections are evaluated through hash table techniques, with the result that the time taken is not dependent on the number of rules in the section.
- [server_url.domain] sections are allowed in <Proxy>, <Cache>, <Forward>, <SSL>, and <SSL-Intercept> layers.

Section Guards

Just as you can with layers, you can improve policy clarity and maintainability by grouping rules into sections and converting the common conditions and properties into guard expressions that follow the section header. A guard expression allows you to take a condition that applies to all the rules and put the common condition next to the section header, as in [Rule] group=sales.

Guards are essentially a way of factoring out common sets of triggers and properties, to avoid having to repeat them each time.

Defining Policies

Refer to these guidelines for defining policies using CPL.

- Write an explicit layer header (such as `<Proxy>`, `<Cache>`, `<Admin>`) before writing any rules or sections. The only constructs that should occur before the first layer header are the condition-related definitions and comments.
- Ensure all sections occur within layers. Do not begin a policy file with a section, such as `[Rule]`.
- Use `[Rule]` sections only when needed.
- Avoid empty or badly-formed policy. While some CPL may look well-formed, make sure it actually does something.

While the following example appears like correct CPL, it actually has no effect. It has a layer header and a [Rule] section header, but no rule lines. As no rules exist, no policy exists either:

```
<Admin> group=Administrators
 [Rule] allow
```

Correct policy that allows access for the group “administrators” would be:

```
<Admin>
  group=Administrators allow
```

In the following example, the layer is deceptive because only the first rule can ever be executed:

```
<Proxy>
  authenticate(MyRealm) ; this rule is unconditional
  ;all following rules are unreachable
  Group=administrator allow
  Group=clerk time=0900..1700 allow
  deny
```

At most, one rule is executed in any given layer. The first one that meets the conditions is acted upon; all other rules in the layer are ignored. To execute more than one rule, use more than one layer. To correctly define the above policy, two layers are required:

```
<Proxy>
  authenticate(MyRealm)

<Proxy>
  Group=administrator allow
  Group=clerk time=0900..1700 allow
  deny
```

Blacklists and Whitelists

For administrative policy, the intention is to be cautious and conservative, emphasizing security over ease of use. The assumption is that everything not specifically mentioned is denied. This approach, referred to as the whitelist approach, is common in corporations concerned with security or legal issues above access. Organizations that want to extend this concept to general Internet access select a default proxy policy of deny as well.

In the second approach, the idea is that by default everything is allowed. Some requests might be denied, but really that is the exception. This is known as blacklist policy because it requires specific mention of anything that should be denied (blacklisted). Blacklist policy is used by organizations where access is more important than security or legal responsibilities.

This second approach is used for cache transactions, but can also be common default proxy policy for organizations such as Internet service providers.

Blacklists and whitelists are tactical approaches and are not mutually exclusive. The best overall policy strategy is often to combine the two approaches. For example, starting from a default policy of deny, one can use a whitelist approach to explicitly filter-in requests that ought to be served in general (such as all requests originating from internal subnets, while leaving external requests subject to the default `DENY`). Further policy layers can then apply more specific restrictions in a blacklist mode to filter-out unwanted requests (such as those that fail to conform to content filtering policies).

Whitelisting and blacklisting can also be used not simply to allow or deny service, but also to subject certain requests to further processing. For example, not every file type presents an equal risk of virus infection or rogue executable content. One might choose to submit only certain file types (presumably those known to harbor executable content) to a virus scanner (blacklist), or virus-scan all files except for a whitelist of types (such as image files) that may be considered safe.

General Rules and Exceptions to a General Rule

When writing policy many organizations use general rules, and then define several exceptions to the rule. Sometimes, you might find exceptions to the exceptions. Exceptions to the general rule can be expressed either:

- Through rule order within a layer
- Through layer order within the policy.

Using Rule Order to Define Exceptions

When the policy rules within a layer are evaluated, remember that evaluation is from the top down, but the first rule that matches will end further evaluation of that layer. Therefore, the most specific conditions, or exceptions, should be defined first. Within a layer, use the sequence of most-specific to most-general policy.

The following example is an exception defined within a layer. A company wants access to payroll information limited to Human Resources staff only. The administrator uses membership in the `HR_staff` group to define the exception for HR staff, followed by the general policy:

```
<Proxy>
  ; Symantec uses groups to identify HR staff, so authentication is required
```

```
  authenticate(MyRealm)

define condition payroll_location
  url=hr.my_company.com/payroll/
end

<Proxy> condition=payroll_location
  allow group=HR_staff ; exception
  deny ; general rule
```

This approach requires that the policy be in one layer, and because layer definitions cannot be split across policy files, the rule and the exceptions must appear in the same file. That may not work in cases where the rules and the exceptions are maintained by different groups.

However, this is the preferred technique, as it maintains all policy related to the payroll files in one place. This approach can be used in either blacklist or whitelist models (see "Blacklists and Whitelists" on the previous page) and can be written so that no security holes are opened in error. The example above is a whitelist model, with everything not explicitly mentioned left to the default rule of deny.

Using Layer Ordering to Define Exceptions

Since later layers override earlier layers, general rules can be written in one layer, with exceptions written in following layers, put specific exceptions later in the file.

The Human Resources example could be rewritten as:

```
<Proxy>
  ; Symantec uses groups to identify HR staff, so authentication is required
  authenticate(MyRealm)

define condition payroll_location
  url=hr.my_company.com/payroll/
end

<Proxy>
  condition=payroll_location deny ; general rule

<Proxy>
  condition=payroll_location allow group=HR_staff ; exception
```

Notice that in this approach, some repetition is required for the common condition between the layers. In this example, the `condition=payroll_location` must be repeated. It is very important to keep the exception from inadvertently allowing other restrictions to be undone by the use of `allow`.

As the layer definitions are independent, they can be installed in separate files, possibly with different authors. Definitions, such as the payroll location condition, can be located in one file and referenced in another. When linking rules to definitions in other files, file order is not important, but the order of installation is. Definitions must be installed before policy that references them will compile. Keeping definitions used across files in only one of the files, rather than spreading them out, will eliminate

the possibility of having changes rejected because of interlocking reference problems. Note that when using this approach, exceptions must follow the general rule, and you must be aware of the policy file evaluation order as currently configured. Changes to the policy file evaluation order must be managed with great care.

Remember that properties maintain any setting unless overridden later in the file, so you could implement general policy in early layers by setting a wide number of properties, and then use a later layer to override selected properties.

Avoid Conflicting Actions

Although policy rules within a policy file can set the action property repeatedly, turning individual actions on and off for the transaction being processed, the specific actions can conflict.

- If an action-definition block contains two conflicting actions, a compile-time error results. This conflict would happen if, for example, the action definition consisted of two"request_redirect()" on page 613 actions.
- If two different action definitions are executed and they contain conflicting actions, it is a run-time error; a policy error is logged to the event log, and one action is arbitrarily chosen to execute.

The following describes the potential for conflict between various actions:

- Two transform actions of the same type conflict.
- Two"rewrite()" on page 615 actions conflict.

Making Policy Definitive

You can make policy definitive two ways. The first is to put that policy into the file; that is, last in the evaluation order. (Remember that the forward file is by default the last policy file.) For example, suppose that service was limited to the corporate users identifiable by subnet. Placing a rule such as the following at the end of the Forward file ensures that no other policy overrides this restriction through accidental use of allow.

```
<Proxy>
  client.address=!my_subnet deny
```

Although not usually used for this purpose, the fact that the forward file is always last, and the order of the other three files is configurable, makes this the appropriate location for definitive policy in some organizations.

An alternate method has been provided for definitive denial. While a "deny" on page 392 or an "exception()" on page 402 property can be overridden by a subsequent allow in later rules, CPL provides "force_deny()" on page 410 and "force_exception ()" on page 412. CPL does not offer `force_allow`, so while the error returned to the user may be reset by subsequent "force_ deny()" on page 410or "force_exception()" on page 412 gestures, the ultimate effect is that the request is denied. Thus these properties provide definitive denial regardless of where they appear in policy.

Best Practices

- Express separate decisions in separate layers.

  As policy grows and becomes more complex, maintenance becomes a significant issue. Maintenance will be easier if the logic for each aspect of policy is separate and distinct.

  Try to make policy decisions as independent as possible, and express each policy in one layer or two adjacent layers.

- Be consistent with the model.

  Set the default proxy policy according to your policy model and then use blacklist or whitelist approaches as appropriate.

  The recommended approach is to begin with a default proxy policy of deny in configuration. Allow requests in early layers and deny requests in later layers. Ensure that all layers that allow requests precede any layers that deny requests. The following is a simple illustration of this model:

```
define subnet corporate_subnet
  10.10.12.0/24
end

; First, explicitly allow access to our users
<Proxy>
  client.address=corporate_subnet ALLOW

; Next, impose any authentication requirements
<Proxy>
  authenticate(corp_realm)  ; all access must be authenticated

; And now begin to filter-out unwanted requests
<Proxy>
  url.domain=forbidden.com deny
  category=(Gambling, Hacking, Chat) deny
  ; more layers…
```

- Expose only what is necessary.

  Often, it may be useful to keep the rule logic and the condition definitions separate so that the rules can be maintained by one group, but the definitions by another. The rules may contain exception details or other logic that should not be modified; however, the conditions, such as affected groups or content, may change frequently. With careful separation of the rules and the conditions, the rules can be expressed in the local policy file, and users unfamiliar with CPL can update the condition definitions through the VPM.

  When using this technique, installation order is important. Condition definitions must be available before policy referencing those conditions will compile, so the conditions you want to expose for general use must be defined in the VPM before they are referenced in the Local policy file.

Include the access_server(no) property to prevent subsequent connections to the OCS after a rule that denies a request. Otherwise, policy might inadvertently allow outbound request data to a restricted site although client receives a "blocked" response status from the appliance. For details and policy examples, refer to TECH246261:

http://www.symantec.com/docs/TECH246261

Conditions

A condition is an expression that yields true or false when evaluated. Conditions can appear in:

- Policy rules
- As guards in section and layer headers, such as `[Rule] group=("bankabc\hr" || "cn=humanresources,ou=groups,o=westernnational")`
- Definition blocks

Condition Syntax

A condition has the following form:

`condition=pattern-expression`

A condition is the name of a condition variable. It can be simple, such as "url=" on page 298, or it can contain sub-object specifiers and modifiers, as in `url.path.case_sensitive` or `request.header.Cookie`. A condition cannot contain white space.

A pattern expression can be either:

- A simple pattern, which is matched against the condition value.
- A Boolean combination of simple patterns, or a parenthesized, comma-separated list of simple patterns.

A pattern expression can be any of the following:

- String: A string argument must be quoted if it contains whitespace or other special characters. An example condition expression is `category=”self help”`.

  You can optionally specify a substitution expression with the .exact, .substring, .prefix, and .suffix string modifiers. Policy tests the transaction when the later event occurs:

  - the trigger condition is available
  - the substitution expression is available
  - Single argument: Conditions such as live= take only a single argument, in this case, yes or no.
  - Boolean expressions: Conditions such as `server_url.scheme=` can list one or more arguments together with Boolean operators; for example, `server_url.scheme=!http`.
  - Integer or range of integers: Numeric conditions can use Boolean expressions and double periods (..), meaning an inclusive numeric range. Numeric ranges cannot use whitespace. The minute= condition is used to show

examples of ranges:

- minute=10..40–From 10 minutes to 40 minutes after the hour.
- minute=10..–From 10 minutes after the hour to the end of the hour.
- minute=..40–From the beginning of the hour to 40 minutes after the hour.
- minute=40..10–From 40 minutes after the hour, to 10 minutes after the next hour.

◦ Regular expressions: Some header-related conditions and two URL-related conditions take regular expressions. For more information about writing regular expressions, see "Regex Reference" on page 703.

The following is Backus-Naur Form (BNF) grammar:

- condition ::= condition "=" expression
- condition ::= identifier | identifier "." word
- expression ::= term | list
- list ::= "(" ((pattern ",")* pattern)? ")"
- disjunction ::= conjunction | disjunction "||" conjunction
- conjunction ::= term | conjunction "&&" term
- term ::= pattern | "(" disjunction ")" | "!" term
- pattern ::= word | 'string' | "string"
- word ::= sequence of characters not including whitespace, & | ( ) < > [ ] ; ! = " '
- string ::= sequence of characters that may including whitespace, & | ( ) < > [ ] ; ! =. The characters " and ' may be enclosed within a string delimited by the alternate delimiter.

Pattern Types

Different conditions support different pattern syntaxes.

A pattern for a Boolean condition has one of the following forms:

```
Boolean ::= yes | no | true | false | on | off
```

The pattern for a numeric condition can be either an integer or a range of integers. Numeric patterns cannot contain white space.

condition=I
Test if condition == I.

condition=I..J
Test if condition >= I and condition <= J (where I <= J). For example, time=0900..1700 tests if the time is between 9:00 and 17:00 inclusive.

condition=J..I
Test if condition >= J or condition <= I (where J > I). For example, minute=45..15 tests if the minute of the hour is between 45 and 15 inclusive.

condition=I..
Test if condition >= I. For example, bitrate=56k.. tests if the bitrate is greater than or equal to 56000.

condition=..J
Test if condition <= J. For example, bitrate=..56k tests if the bitrate is less than or equal to 56000.

Some conditions have IP address patterns. This can be either a literal IP address, such as 1.2.3.4, or an IP subnet pattern, such as 1.2.0.0/16, or a name defined by a define subnet statement.

Some conditions have regex patterns. This is a Perl 5 regular expression that matches a substring of the condition value; it is not an anchored match unless an anchor is specified as part of the pattern.

Unavailable Conditions

Some conditions can be unavailable in some transactions. If a condition is unavailable, then any condition containing that condition is false, regardless of the pattern expression. For example, if the current transaction is not authenticated (that is, the "authenticate()" on page 348 property was set to no), then the user condition is unavailable. This means that `user=kevin` and `user=!kevin` are both false.

A condition can be false either because the pattern does not match the condition value, or because the condition is unavailable. Policy rule-tracing distinguishes these two cases, using `miss` for the former and `N/A` for the latter.

Layer Type Restrictions

Each condition is restricted as to the types of layers in which it can be used. A direct use of a condition in a forbidden layer results in a compile-time error. Indirect use of a condition in a forbidden layer (by way of "condition=" on page 113 and "define condition" on page 637) also results in a compile time error.

Global Restrictions

To allow suppression of DNS and RDNS lookups from policy, the following restrictions are supported. These restrictions have the effect of assuming a no_lookup modifier for appropriate url, url.host, and url.domain tests. The restrictions also apply to lookups performed by on-box content category lookups. For more information on DNS and RDNS restrictions, see "restrict dns" on page 667 and "restrict rdns" on page 669.

restrict dns domain_list end

- Applies to all layers.
- Applies to all transactions.
- If the domain specified in a URL matches any of the domain patterns specified in domain_list, no DNS lookup is performed for any server_url=, server_url.address=, server_url.domain=, or server_url.host= test.

  If a lookup is required to evaluate the condition, the condition evaluates to false.

restrict rdns subnet_list end

- Applies to all layers.
- Applies to all transactions.
- If the requested URL specifies the host in IP form, no RDNS lookup is performed to match any server_url=, server_url.domain=, or server_url.host= condition.

  If a lookup is required to evaluate the condition, the condition evaluates to false.

admin.access=

Test the administrative access method required by the current administrative transaction.

If write access is required, then policy is evaluated with `admin.access=write` to determine if the administrator is allowed to modify the configuration. For example, administrative policy is evaluated to determine if a CLI user is permitted to enter Enable mode, or when attempting to save changes from the Management Console. If only read access is required, then policy is evaluated with `admin.access=read` to determine if the administrator is permitted to have read-only access.

> **Note:** All administrative transactions require either read or write access; thus, a condition such as `admin.access=(read,write)` is always true and can be deleted without changing the meaning of a CPL rule.

Syntax

```
admin.access={read|write}
```

Layer and Transaction Notes

- Layers: `<Admin>` , `<Diagnostic>`
- Transactions: administrator

Example

Control administrative access for two classes of users, Read_only_admins and Full_admins. A user must be the specified user or belong to the specified group for the rule to match. In cases where a user is in both groups, they inherit the larger set of permissions.

```
; user paul and everyone in the operations group are full admins
define condition Full_admins
  user=paul
  group=operations
end

; user george and everyone in the help desk group are read-only admins
define condition Read_only_admins
  user=george
  group=help_desk
end

<Admin>
  authenticate(my_realm)
```

```
<Admin>
  allow condition=Full_admins
  allow condition=Read_only_admins admin.access=read
deny
```

ami.config.threat-protection.malware-scanning=

> **Tip:** Use the Access Security Policy layer with the Policy Services subscription to configure threat protection in version 7.x. This policy gesture still compiles as of version 7.1.1.1, but it has no effect on policy. Refer to the "Using Policy Services" chapter in the *SGOS Administration Guide*, and the *ProxySG Security Best Practices* document for details.

Specifies the rules that are invoked when malware scanning is enabled on the appliance. Your settings in configuration invoke the corresponding sections of the threat protection policy file which are then compiled for use on the appliance.

Syntax

`ami.config.threat-protection.malware-scanning.`*`configuration_setting`*`={'(`*`type_name "value`*`")'|yes|no}`

where:

- *`configuration_setting`*: Supported configuration setting.
- *`type_name`*: Type of setting value as defined in configuration.

Configuration setting	Type name
level	`BC-Malware-Scanning-Scan-Level`
secure-connection	`BC-Malware-Scanning-Secure-Connection`
failure-mode	`BC-Malware-Scanning-Failure-Mode`

- *`value`*: Value of configuration setting to match.
- yes: Malware scanning is enabled.
- no: Malware scanning is disabled.

Layer and Transaction Notes

- Layers: `<Admin>`, `<Cache>`, `<Diagnostic>`, `<DNS-Proxy>`, `<Exception>`, `<Forward>`, `<Proxy>` , `<SSL>`, `<SSL-Intercept>`
- Transactions: all

Example

Create user policy that extends the threat-protection of low-risk files. Files are not virus-scanned when there is a private URL and the malware scanning level is set to high-performance.

```
<Cache>
  ami.config.threat-protection.malware-scanning.level='(BC-Malware-Scanning-Scan-Level "high-
performance")'
  url.host.is_private=yes response.icap_service.secure_connection(no)
```

appliance.id=

Tests the serial number of the appliance. Use this condition to write policy that applies to specific appliances. For example, use this when there is a common source of policy for multiple appliances and you want to support appliance-specific behaviors.

Syntax

`appliance.id[.`*`string_modifier`*`][.case_sensitive]=`*`pattern`*

where:

- *`string_modifier`*: A supported string modifier:
  - `exact` - match the string exactly
  - `prefix` - match the start of a string
  - `regex` - (default if no string modifier is specified) regular expression match; see "Regex Reference" on page 703
  - `substring` - match part of a string
  - `suffix` - match the end of a string
- `case_sensitive` - Perform case-sensitive match.
- *`pattern`*: Type-appropriate test expression, such as a quote-delimited string expression or a regular expression.

Layer and Transaction Notes

- Layers: `<Cache>` , `<Diagnostic>`, `<Proxy>`
- Transactions: all

Example

Define two conditions corresponding to two data centers, in Paris and London respectively. Two conditions are defined to return different Deny messages based on users' locations.

```
define condition datacenter1
  appliance.id=(1709140012,0805060002)
end

define condition datacenter2
  appliance.id=1709140012
end
```

```
<Proxy>
  condition=datacenter1 DENY("Accès refusé")
  condition=datacenter2 DENY("Access denied")
```

attribute.*name*=

Tests if the current transaction is authenticated in a RADIUS or LDAP realm, and if the authenticated user has the specified attribute with the specified value. This condition is unavailable if the current transaction is not authenticated (that is, the authenticate property is set to no).

If you reference more than one realm in your policy, you can disambiguate attribute tests by combining them with a realm= test. This can reduce the number of extraneous queries to authentication services for attribute information that does not pertain to that realm.

Syntax

`attribute.name=value`

where:

- *name*: A RADIUS or LDAP attribute. RADIUS realms are always case-sensitive; for LDAP, case-sensitivity depends on realm definition. **Show RADIUS values**

RADIUS Attribute Name	CPL Gesture Name	Type (Possible Value)
Blue-Coat-Group	attribute.Blue-Coat-Group	String (Max length:20)
3GPP-Allocate-IP-Type	attribute.3GPP-Allocate-IP-Type	Octet-string (Max length:1)
3GPP-Camel-Charging-Info	attribute.3GPP-Camel-Charging-Info	Octet-string (Max length:32)
3GPP-Charging-Characteristics	attribute.3GPP-Charging-Characteristics	String (Max length:4)
3GPP-Charging-ID	attribute.3GPP-Charging-ID	Integer
3GPP-Charging-Gateway-Address	attribute.3GPP-Charging-Gateway-Address	IP Address (IPV4)
3GPP-Charging-Gateway-IPv6-Address	attribute.3GPP-Charging-Gateway-IPv6-Address	IP Address (IPV6)
3GPP-GGSN-Address	attribute.3GPP-GGSN-Address	IP Address (IPV4)
3GPP-GGSN-IPv6-Address	attribute.3GPP-GGSN-IPv6-Address	IP Address (IPV6)
3GPP-GGSN_MCC-MNC	attribute.3GPP-GGSN_MCC-MNC	String (Max length:6)
3GPP-GPRS-Negotiated-QoS-Profile	attribute.3GPP-GPRS-Negotiated-QoS-Profile	Octet-string (Max length:35)
3GPP-IMEISV	attribute.3GPP-IMEISV	String (Max length:14)
3GPP-IMSI	attribute.3GPP-IMSI	String (Max length:15)
3GPP-IMSI-MCC-MNC	attribute.3GPP-IMSI-MCC-MNC	String (Max length:6)
3GPP-IPv6-DNS-Servers	attribute.3GPP-IPv6-DNS-Servers	Octet-string (Max length:240)

RADIUS Attribute Name	CPL Gesture Name	Type (Possible Value)
3GPP-MS-TimeZone	attribute.3GPP-MS-TimeZone	Octet-string (Max length:2)
3GPP-Negotiated-DSCP	attribute.3GPP-Negotiated-DSCP	Octet-string (Max length:1)
3GPP-NSAPI	attribute.3GPP-NSAPI	String (Max length:1)
3GPP-Packet-Filter	attribute.3GPP-Packet-Filter	Octet-string (Max length:32)
3GPP-PDP-Type	attribute.3GPP-PDP-Type	Enum (0:IPv4, 1:PPP, 2:IPv6)
3GPP-RAT-Type	attribute.3GPP-RAT-Type	Octet-string (Max length:1)
3GPP-Selection-Mode	attribute.3GPP-Selection-Mode	String (Max length:1)
3GPP-Session-Stop-Indicator	attribute.3GPP-Session-Stop-Indicator	Octet-string (Max length:1)
3GPP-SGSN-Address	attribute.3GPP-SGSN-Address	IP Address (IPV4)
3GPP-SGSN-IPv6-Address	attribute.3GPP-SGSN-IPv6-Address	IP Address (IPV6)
3GPP-SGSN-MCC-MNC	attribute.3GPP-SGSN-MCC-MNC	String (Max length:6)
3GPP-User-Location-Info	attribute.3GPP-User-Location-Info	Octet-string (Max length:32)
3GPP-Teardown-Indicator	attribute.3GPP-Teardown-Indicator	Octet-string (Max length:1)
Acct-Authentic	attribute.Acct-Authentic	Enum (1:RADIUS, 2:Local)
Acct-Delay-Time	attribute.Acct-Delay-Time	Integer
Acct-Input-Octets	attribute.Acct-Input-Octets	Integer
Acct-Input-Packets	attribute.Acct-Input-Packets	Integer
Acct-Link-Count	attribute.Acct-Link-Count	Integer
Acct-Multi-Session-ID	attribute.Acct-Multi-Session-ID	String (Max length:20)
Acct-Output-Octets	attribute.Acct-Output-Octets	Integer
Acct-Output-Packets	attribute.Acct-Output-Packets	Integer
Acct-Session-ID	attribute.Acct-Session-ID	String (Max length:20)
Acct-Session-Time	attribute.Acct-Session-Time	Integer
Acct-Status-Type	attribute.Acct-Status-Type	Enum (1:Start, 2:Stop,3:Interim-Update, 4:Unassigned (4), 5:Unassigned (5), 6:Unassigned (6), 7:Accounting-On, 8:Accounting-Off, 9:Tunnel-Start, 10:Tunnel-Stop, 11:Tunnel-Reject, 12:Tunnel-Link-Start, 13:Tunnel-Link-Stop, 14:Tunnel-Link-Reject)

RADIUS Attribute Name	CPL Gesture Name	Type (Possible Value)
Acct-Terminate-Cause	attribute.Acct-Terminate-Cause 가Ê	Enum (1:User Request, 2:Lost Carrier, 3:Lost Service, 4:Idle Timeout, 5:Session Timeout, 6:Admin Reset, 7:Admin Reboot, 8:Port Error, 9:NAS Error, 11:NAS Request, 11:NAS Reboot, 12:Port Unneeded, 13:Port Preempted, 14:Port Suspended, 15:Service, Unavailable 16:Callback, 17:User Error)
Callback-ID	attribute.Callback-ID	String (Max length:20)
Callback-Number	attribute.Callback-Number	String (Max length:20)
Called-Station-ID	attribute.Called-Station-ID	String (Max length:20)
Calling-Station-ID	attribute.Calling-Station-ID	String (Max length:20)
CHAP-Challenge	attribute.CHAP-Challenge	String (Max length:20)
CHAP-Password	attribute.CHAP-Password	String (Max length:20)
Class	attribute.Class	String (Max length:20)
Filter-ID	attribute.Filter-ID	String (Max length:20)
Framed-AppleTalk-Link	attribute.Framed-AppleTalk-Link	Integer
Framed-AppleTalk-Network	attribute.Framed-AppleTalk-Network	Integer
Framed-AppleTalk-Zone	attribute.Framed-AppleTalk-Zone	String (Max length:20)
Framed-Compression	attribute.Framed-Compression	Enum (0:None, 1:Van Jacobsen-Header-Compression, 2:IPX-Header-Compression)
Framed-IP-Address	attribute.Framed-IP-Address	IP Address (IPV4)
Framed-IP-Netmask	attribute.Framed-IP-Netmask	IP Address (IPV4)
Framed-IPv6-Route	attribute.Framed-IPv6-Route	IP Address (IPV6)
Framed-IPX-Network	attribute.Framed-IPX-Network	Integer
Framed-MTU	attribute.Framed-MTU	Integer
Framed-Pool	attribute.Framed-Pool	String (Max length:20)
Framed-Protocol	attribute.Framed-Protocol	Enum (1:PPP, 2:SLIP, 3:ARAP, 4:Gandalf, 5:Xylogics)
Framed-Route	attribute.Framed-Route	String (Max length:20)
Framed-Routing	attribute.Framed-Routing	Enum (0:None, 1:Send, 2:Listen)
Idle-Timeout	attribute.Idle-Timeout	Integer

RADIUS Attribute Name	CPL Gesture Name	Type (Possible Value)
Login-LAT-Group	attribute.Login-LAT-Group	String (Max length:20)
Login-LAT-Node	attribute.Login-LAT-Node	String (Max length:20)
Login-LAT-Port	attribute.Login-LAT-Port	String (Max length:20)
Login-LAT-Service	attribute.Login-LAT-Service	String (Max length:20)
Login-IP-Host	attribute.Login-IP-Host	IP Address (IPV4)
Login-IPv6-Host	attribute.Login-IPv6-Host	IP Address (IPV6)
Login-Service	attribute.Login-Service	Enum (0:Telnet, 1:Rlogin, 2:TCP Clear, 3:PortMaster, 4:LAT, 5:X25-PAD, 6:X25-T3POS, 7:Unassigned)
Login-TCP-Port	attribute.Login-TCP-Port	Integer (0-65535)
Message-Authenticator	attribute.Message-Authenticator	String (Max length:20)
NAS-Identifier	attribute.NAS-Identifier	String (Max length:20)
NAS-IP-Address	attribute.NAS-IP-Address	IP Address (IPV4)
NAS-IPv6-Address	attribute.NAS-IPv6-Address	IP Address (IPV6)
NAS-Port	attribute.NAS-Port	Integer
NAS-Port-Type	attribute.NAS-Port-Type	Enum (0:Async, 1:Sync, 2:ISDN Sync, 3:ISDN Async V.120, 4:ISDN Async V.110, 5:Virtual, 6:PIAFS, 7:HDLC, 8:X.25, 9:X.75, 10:G.3, 11:SDSL, 12:ADSL-CAP, 13:ADSL-DMT, 14:IDSL, 15:Ethernet, 16:xDSL, 17:Cable, 18:Wireless Other)
Port-Limit	attribute.Port-Limit	Integer
Proxy-State	attribute.Proxy-State	String (Max length:20)
Reply-Message	attribute.Reply-Message	String (Max length:20)
Service-Type	attribute.Service-Type	Enum (1:Login, 2:Framed, 3:Callback Login, 4:Callback Framed, 5:Outbound, 6:Administrative, 7:NAS Prompt, 8:Authenticate Only, 9:Callback NAS Prompt, 10:Call Check)
Session-Timeout	attribute.Session-Timeout Integer State attribute.State	String (Max length:20)
Termination-Action	attribute.Termination-Action	Enum (0:Default)

RADIUS Attribute Name	CPL Gesture Name	Type (Possible Value)
Tunnel-Assignment-ID	attribute.Tunnel-Assignment-ID	String (Max length:20)
Tunnel-Medium-Type	attribute.Tunnel-Medium-Type	Tag-Enum (1:IPv4, 2:IPv6, 3:NSAP, 4:HDLC, 5:BBN, 6:802, 7:E.163, 8:E.164, 9:F.69, 10:X.121, 11:IPX, 12:AppleTalk, 13:Decnet IV, 14:Banyan Vines)
Tunnel-Private-Group-ID	attribute.Tunnel-Private-Group-ID	String (Max length:20)
Tunnel-Type	attribute.Tunnel-Type	Tag-Enum (1:PPTP, 2:L2F, 3:L2TP, 4:ATMP, 5:VTP, 6:AH, 7:IP-IP, 8:MIN-IP-IP, 9:ESP, 10:GRE, 11:DVS)
User-Name attribute.User-Name String (Max length:20)		
User-Password	attribute.User-Password	String (Max length:20)

- *value*: An attribute value.

Layer and Transaction Notes

- Layers: `<Admin>`, `<Forward>`, `<Proxy>`, `<SSL-Intercept>`
- Transactions: all proxies, administrator

Note: When used in the `<forward>` layer, this condition can evaluate to NULL (shown in a trace as N/A) if no authenticated client exists. Rules containing these conditions can be guarded by authenticated= to preserve normal logic.

Note: This condition cannot be combined with the "authenticate()" on page 348 or "socks.authenticate()" on page 554 properties.

Example 1

Use the value of the ContentBlocking attribute associated with a user to select which content categories to block.

```
<Proxy>
authenticate(LDAPRealm)

<Proxy> exception(content_filter_denied)
  attribute.ContentBlocking=Adult category=(Sex, Nudity, Mature, Obscene/Extreme)
```

```
  attribute.ContentBlocking=Violence category=(Criminal_Skills, Hate_Speech)
  ...
```

Example 2

Use the attribute property to determine permissions associated with RADIUS authentication.

```
define condition ProxyAllowed
  attribute.ServiceType=(2,6,7,8)
end

<Proxy>
authenticate(RADIUSRealm)
```

Example 3

Restrict non-authorized users.

```
<Proxy>
  deny condition=!ProxyAllowed
```

Example 4

Override a previous denial and grant access to authorized users.

```
<Proxy>
  allow condition=ProxyAllowed
```

See Also

- Conditions: "authenticated=" on the facing page, "group=" on page 140,"has_attribute.name=" on page 142, "http.transparent_authentication=" on page 180, "realm=" on page 208, "user=" on page 312, "user.domain=" on page 316, "user.x509.issuer=" on page 322, "user.x509.serialNumber=" on page 323, "user.x509.subject=" on page 325
- Properties: "authenticate()" on page 348, "authenticate.force()" on page 354, "check_authorization()" on page 376, "exception()" on page 402, "socks.authenticate()" on page 554, "socks.authenticate.force()" on page 556

authenticated=

If authentication is requested for the current transaction, tests whether the requested realm's credentials have been verified.

> **Note:** This condition cannot be combined with the "authenticate()" on page 348 property.

Syntax

```
authenticated={yes|no}
```

where:

- yes: Requested realm's credentials are verified.
- no: Requested realm's credentials are not verified or authentication is not requested.

Layer and Transaction Notes

- Layers: <Admin>, <Exception>, <Forward>, <Proxy> , <SSL>, <SSL-Intercept>
- Transactions: all proxies, administrator

Example 1

Grant access to a specific site only to users authenticated in any domain. All other users are unauthenticated.

```
<Proxy>
  client.address=10.10.10.0/24 authenticate(LDAPRealm)
  client.address=10.10.11.0/24 authenticate(NTLMRealm)
  client.address=10.10.12.0/24 authenticate(LocalRealm)
```

Example 2

Restrict unauthorized users to override previously granted access.

```
<Proxy> server_url.domain=xyz.com
 deny authenticated=no
```

Example 3

Grant access and override a previous denial, assuming a deny in a previous layer.

```
<Proxy> server_url.domain=xyz.com
 allow authenticated=yes
```

See Also

- Conditions: "attribute.name=" on page 75, "group=" on page 140, "has_attribute.name=" on page 142, "has_ attribute.name=" on page 142, "http.transparent_authentication=" on page 180, "realm=" on page 208, "user=" on page 312. "user.domain=" on page 316
- Properties: "authenticate()" on page 348, "authenticate.force()" on page 354, "check_authorization()" on page 376, "socks.authenticate()" on page 554, "socks.authenticate.force()" on page 556

bitrate=

Tests if a streaming transaction requests bandwidth within the specified range or an exact match. When providing a range, either value can be left empty, implying either no lower or no upper limit on the test. Bitrate can change dynamically during a transaction, so this policy is re-evaluated for each change. The numeric pattern used to test the `bitrate=` condition cannot contain whitespace.

Syntax

`bitrate={`*range*`|`*exact_rate*`}`

where:

- *range*: A bandwidth range of bits specified with a lower limit, an upper limit, or both:
  - `..`*upper_limit* - the maximum bandwidth
  - *lower_limit*`..` - the minimum bandwidth
  - *lower_limit*`..`*upper_limit* - bandwidth within the specified range
- *exact_rate*: Exact bandwidth, in bits, to test.

> **Tip:** Express kilobits (1000x) as *integer*`k` and megabits (1,000,000x) as *integer*`m`.

> **Note:** To test an inverted range, a shorthand expression is available. Instead of writing `bitrate=(..28.8k|56k..)` to indicate bit rates from 0 to 28.8k and from 56k and higher, the policy language recognizes `bitrate=56k..28.8k` as equivalent.

Layer and Transaction Notes

- Layers: `<Cache>`, `<Proxy>`
- Transactions: streaming proxies

Example 1

Set a deny rule, and then allow some users access to streams of a certain bandwidth.

```
; deny service for bit rates above 56k
deny bitrate=!0..56k

; allow members of the Sales group access to streams up to 2 megabits
; all others are limited to 56K bit streams
```

```
<Proxy>
 authenticate(NTLMRealm)


<Proxy>
; deny sales access to streams over 2M bits
deny group=sales bitrate=!0..2m

; deny non-sales access to streams over 56K bits
deny group=!sales bitrate=!0..56k
```

Example 2

Override a previous deny rule to grant access to streams to authorized users.

```
; this rule assumes that the users are by default denied
; and overrides this to grant access to authorized users

<Proxy> ; Use this layer to override a deny in a previous layer
; Grant everybody access to streams up to 56K, sales group up to 2M
allow bitrate=..56K
allow group=sales bitrate=..2M
```

See Also

- Conditions: "live=" on page 192, "streaming.client=" on page 283, "streaming.content=" on page 284
- Properties: "access_server()" on page 335, "max_bitrate()" on page 494, "streaming.transport()" on page 569

category=

Tests the content categories of the requested URL as assigned by policy definitions or an enabled content filter database. You cannot use `category=` to test the category assigned by off-box content filtering services; those services have their own policy that must be managed separately.

Content categories can be assigned to URLs by policy (see "define category" on page 635), a local database you maintain, or a third-party database.

If a URL is assigned more than one category, this condition is `true` if it matches any of the assigned categories.

System-Defined Statuses

The following statuses are system-defined:

- `none`: The request is uncategorized by all enabled content filter providers. If even one content filter provider returns a category for the requested site, policy rules containing a `*.category=none` condition will not match.
- `pending`: The categorization request has not been processed or is not complete.
- `unavailable`: A content filter provider is selected in configuration, but an error occurs in determining the category. Other categories can still be assigned directly by policy. This categorization might be a result of a missing database.
- `unlicensed`: A content filter provider license is expired. When this status is true, the `unavailable` status is also true.

Renamed Category Warnings

If CPL includes a category that has been renamed in the currently downloaded database, policy warnings occur at compilation time. The following is an example of the warning:

```
Deprecation warning: 'old_category'; 'old_category' has been replaced by 'new_category'
due to Category name updated and will no longer be accepted after Sat, 11 Jul 2020 00:00:00 UTC.
Please switch to the new name before then.
```

To ensure that policy performs as intended, rename all instances of the affected category in CPL and re-apply policy by the specified date.

Syntax

```
category={status|category_name1[,category_name2,...]|category_group1[,category_group2,...]}
```

where:

- *`status`*: One of the following system-defined statuses: `none`, `pending`, `unavailable`, and `unlicensed`.
- *`category_name`*: Category names defined by policy or the selected content filter provider.
- *`category_group`*: Category group names defined by policy or Blue Coat provider if enabled.

The list of currently valid category names and category group names is available both through the Management Console and CLI.

Layer and Transaction Notes

- Layers: <Cache>, <Exception>, <Proxy> , <SSL>, <SSL-Intercept>
- Transactions: all proxies

Example

Deny requests for games or sports-related content and return the specified exception page.

```
<Proxy>
; Tests true if the request is in one of these categories.
  category=(Sports, Games) exception(content_filter_denied)
  category=unavailable exception(content_filter_unavailable); Fail closed
```

See Also

- Conditions: "server_url.category=" on page 272
- Properties: "exception()" on page 402

client.address=

Tests the IP address of the client. The expression can include an IP address or subnet or the label of a subnet definition block.

> **Note:** If a user is explicitly proxied to the appliance, <Proxy> layer policy applies even if the URL destination is an administrative URL for the appliance itself, and should therefore also be covered under <Admin> layer policy. However, when the client.address= condition is used in an <Admin> layer, clients explicitly proxied to the appliance appear to have their client IP address set to the IP address of the appliance.

Syntax

client.address={*ip_address*|*ip_address_range*|*ip_address_wildcards*|*subnet*|*subnet_label*}

where:

- *ip_address*: Client IP address; for example, 10.25.198.0
- *ip_address_range*: IP address range; for example, 192.0.2.0-192.0.2.255
- *ip_address_wildcards*: IP address specified using wildcards in any octet(s); for example, 10.25.*.0 or 10.*.*.0
- *subnet*: Subnet specification; for example, 10.25.198.0/24
- *subnet_label*: Label of a "define subnet" on page 656 block that binds a number of IP addresses or subnets

Layer and Transaction Notes

- Layers: <Admin>, <Cache>, <Diagnostic>, <DNS-Proxy>, <Exception>, <Forward>, <Proxy> , <SSL>, <SSL-Intercept>, <Tenant>
- Unavailable if the transaction is not associated with a client.

Example 1

Blacklist a workstation.

```
<Proxy>
  client.address=10.25.198.0 deny
```

Example 2

Use the client address to select the authentication realm for administration of the appliance.

```
<Admin>
  client.address=10.25.198.0/24 authenticate(LDAPRealm)
  client.address=10.25.199.0/24 authenticate(NTLMRealm)
  authenticate(LocalRealm) ; Everyone else
```

See Also

- Conditions: "client.interface=" on page 109, "client.protocol=" on page 111, "proxy.address=" on page 198, "proxy.port=" on page 200
- Definitions: "define subnet" on page 656
- Information on wildcards: http://www.symantec.com/docs/TECH241521
- Information on IP address ranges: http://www.symantec.com/docs/TECH241929

client.address.country=

Returns the country from which traffic originates, based on the client IP address.

If the geolocation feature is disabled or the database is not yet downloaded, this condition will always return “Unavailable” for the country name. If the database is enabled but not licensed, this condition will always return “Unlicensed” for the country name.

> **Tip:** If you receive an “Obsolete country name” warning when installing policy, replace the outdated country name in policy with the suggested name in the message.

Syntax

`client.address.country="country_name"`

where:

- *country_name*: Name of the country to look up. You can also use the code name specified in the database, for example, “CA” for Canada.

Layer and Transaction Notes

- Layers: <Admin>, <Cache>, <Diagnostic>, <DNS-Proxy>, <Exception>, <Forward>, <Proxy> , <SSL>, <SSL-Intercept>
- Unavailable if the transaction is not associated with a client

Example 1

Accept client connections only from specific countries.

```
; Only accept client connections from North America
<Proxy>
  allow client.address.country=(US, CA)
  deny("Restricted location: $(x-cs-client-ip-country)")
```

Example 2

Accept client connections only from specific countries, considering the original IP address of forwarded requests.

```
; Only accept traffic from North America with support for proxied traffic
; with client address in X-Forwarded-For
<Proxy>
  client.effective_address("$(request.header.X-Forwarded-For)")
```

```
<Proxy>
  client.effective_address.country=(US, CA) OK
  deny("Restricted location: $(x-cs-client-effective-ip-country)")
```

See Also

- Conditions: "client.address=" on page 87, "client.effective_address=" on page 101, "client.effective_address.country=" on page 102
- Properties: "client.effective_address.connection()" on page 384, "client.effective_address.request()" on page 386

client.address.login.count=

Test the number active logins at the current IP address.

This condition is used to test how many logins are active on the current IP address. It can be used to manage the maximum number of logins per IP address.

Syntax

`client.address.login.count={`*range*`|`*exact*`}`

where:

- *range*: A range specified with a lower limit, an upper limit, or both:
  - `..`*upper_limit* - the highest number of logins
  - *lower_limit*`..` - the lowest number of logins
  - *lower_limit*`..`*upper_limit* - number of logins within the specified range
- *exact*: Exact number of logins.

Layer and Transaction Notes

- Layers: `<Admin>`, `<Exception>`, `<Forward>`, `<Proxy>` , `<SSL>`, `<SSL-Intercept>`
- Transactions: all

Example

Log out other logins if there is more than one login at a specified address.

```
<Proxy>
  client.address.login.count=2.. client.address.login.log_out_other(yes)
```

client.certificate.common_name=

Test the common name of the client certificate in an SSL transaction.

Test the common name extracted from the X.509 certificate offered by the client while establishing an SSL connection.

Syntax

```
client.certificate.common_name[.string_modifier][.case_sensitive]=pattern
client.certificate.common_name.length=value
```

where:

- *string_modifier*: A supported string modifier:
  - exact - match the string exactly
  - prefix - match the start of a string
  - regex - (default if no string modifier is specified) regular expression match; see "Regex Reference" on page 703
  - substring - match part of a string
  - suffix - match the end of a string
- case_sensitive - Perform case-sensitive match.
- *pattern*: Type-appropriate test expression, such as a quote-delimited string expression or a regular expression.

> **Tip:** The pattern expression supports substitutions. You can specify a substitution expression with the .exact, .substring, .prefix, and .suffix string modifiers where they are available.

Layer and Transaction Notes

- Layers: <SSL>
- Transactions: HTTPS forward and reverse proxy transactions

Example

Use the condition for consent certificates.

```
<SSL> ssl.proxy_mode=https-reverse-proxy
  OK client.certificate.common_name="Yes decrypt my content"
  FORCE_DENY
```

client.certificate.requested=

Tests whether or not the server has requested SSL client certificate authentication.

When the SSL proxy establishes a connection with the server and the server requests an SSL client certificate, this condition is set to yes; else, it is set to no. This condition is NULL for transactions that do not involve an SSL connection to the client.

When the appliance evaluates this condition, it uses a list of requesting servers (a Client Certificate Requested list) to determine if a client certificate was requested during both an initial handshake and renegotiation. As long as this condition exists in policy, the appliance can automatically detect servers that request a client certificate during renegotiation and maintain the Client Certificate Requested list.

Syntax

```
client.certificate.requested={yes|no}
```

Layer and Transaction Notes

- Layers: <SSL-Intercept>
- Transactions: SSL intercept

Example

Enable SSL proxy interception only when a client certificate is not requested by the server.

```
<SSL-Intercept>
; If the server requests a client certificate, tunnel the SSL traffic via SSL proxy
  client.certificate.requested=yes ssl.forward_proxy(no)

; Otherwise, intercept SSL traffic using HTTPS forward proxy.
  ssl.forward_proxy(https)
```

client.certificate.subject=

Test the subject field of the client certificate in an SSL transaction.

Test the subject field extracted from the X.509 certificate offered by the client while establishing an SSL connection.

This condition is used mainly for consent certificates.

Syntax

```
client.certificate.subject[.string_modifier][.case_sensitive]=pattern
client.certificate.subject.length=value
```

where:

- *string_modifier*: A supported string modifier:
  - exact - match the string exactly
  - prefix - match the start of a string
  - regex - (default if no string modifier is specified) regular expression match; see "Regex Reference" on page 703
  - substring - match part of a string
  - suffix - match the end of a string
- case_sensitive - Perform case-sensitive match.
- *pattern*: Type-appropriate test expression, such as a quote-delimited string expression or a regular expression.

> **Tip:** The pattern expression supports substitutions. You can specify a substitution expression with the .exact, .substring, .prefix, and .suffix string modifiers where they are available.

Layer and Transaction Notes

- Layers: <SSL>
- Transactions: HTTPS forward and reverse proxy transactions

Example

Use the condition for consent certificates.

```
<SSL> ssl.proxy_mode = https-reverse-proxy
  OK client.certificate.subject="Yes decrypt my content"
  FORCE_DENY
```

client.certificate.subject_directory_attribute=

Tests the subject directory attribute field extracted from an X.509 certificate offered by a client while establishing an SSL connection. Per section 3.3.2 of RFC3739, the directory attribute field in an X.509 certificate contains information such as the client's country of origin or residence, gender or place of birth.

This condition does not apply to transactions that do not involve an SSL connection to the client or do not contain subject directory attributes.

Syntax

`client.certificate.subject_directory_attribute.`*`attribute_name`*`[.`*`modifier`*`]=`*`pattern`*

where:

- *`attribute_name`*: name defined in a subject_directory_attribute definition.
- *`modifier`*: One of the following:
  - `exists` - Attribute exists in the certificate
  - `count` - Number of times the attribute appears in the certificate
  - *`string_modifier`* - supported string modifier:
    - `exact` - match the string exactly
    - `prefix` - match the start of a string
    - `regex` - (default if no string modifier is specified) regular expression match; see "Regex Reference" on page 703
    - `substring` - match part of a string
    - `suffix` - match the end of a string
- *`pattern`*: Type-appropriate test expression, such as a quote-delimited string expression or a regular expression.

> **Tip:** The pattern expression supports substitutions. You can specify a substitution expression with the .exact, .substring, .prefix, and .suffix string modifiers where they are available.

Layer and Transaction Notes

- Layers: `<Proxy>` , `<SSL>`
- Transactions: HTTP proxy

Example

In an organization where client workstations identify themselves with X.509 certificates, one user from the UK branch office is visiting the US office and needs access to web resources from the UK intranet. While these UK resources are technically accessible from the US office, employees in the US are not permitted to access them. The following policy ensures that the visiting UK user has the access he needs.

```
<Proxy>
  client.certificate.subject_directory_attribute.country=UK
```

In a similar case, the branch manager for this organization's Canada and UK offices also requires access to specific resources. These resources require that the user is from both CA (Canada) and UK (United Kingdom) the combination of policy list matching and the count modified can be used. The following will match the case where the 'country' attribute has exactly two values: CA and UK.

```
<Proxy>
  client.certificate.subject_directory_attribute.country=(CA && UK) client.certificate.subject_
directory_attribute.country.count=2
```

client.connection.dscp=

Test the client-side inbound DSCP value.

Syntax

`client.connection.dscp=`*`dscp_value`*

where:

- *`dscp_value`*: One of the following:
  - decimal value between 0 and 63
  - `best-effort`
  - a class: `af11 | af12 | af13 | af21 | af22 | af23 | af31 | af32 | af33 | af41 | af42 | af43 | cs1 | cs2 | cs3 | cs4 | cs5 | cs6 | cs7 | ef`

Layer and Transaction Notes

- Layers: `<DNS-Proxy>`, `<Forward>`, `<Proxy>`
- Transactions: all

Example

The first QoS policy rule tests the client inbound QoS/DSCP value against 50, and deny if it matches; the second QoS policy rule tests the client inbound QoS/DSCP value against best-effort, and deny if it matches.

```
<Proxy>
  deny client.connection.dscp=50

<Proxy>
  deny client.connection.dscp=best-effort
```

client.connection.negotiated_cipher=

Test the cipher suite negotiated with a securely connected client.

Syntax

`client.connection.negotiated_cipher={`*`cipher-suite`*`|none}`

where:

- *`cipher-suite`*: Cipher suite(s) that the appliance supports. Refer to https://www.symantec.com/docs/TECH247556 for cipher suites shipped with the appliance.
- none: No supported cipher suite.

Layer and Transaction Notes

- Layers: `<SSL>`
- Transactions: all proxies

Example

Deny clients that are not using one of the specified cipher suites and access-log clients that are not using secure connections.

```
<SSL>
  ALLOW client.connection.negotiated_cipher=(TLS_AES_128_GCM_SHA256 || TLS_AES_256_GCM_SHA384 || TLS_
CHACHA20_POLY1305_SHA256
  DENY

<SSL>
  client.connection.negotiated_cipher=none access_log[unsecure_log1](yes)
```

client.connection.negotiated_cipher.strength=

Test the cipher strength negotiated with a securely connected client.

Syntax

`client.connection.negotiated_cipher.strength=`*`level`*

where:

- *`level`*: One of the following: low, medium, high, none

> **Note:** OpenSSL sets standards for cipher suite strength levels. Refer to OpenSSL for information.

Layer and Transaction Notes

- Layers: `<SSL>`
- Transactions: all proxies

Example

Deny clients that do not have at least a medium cipher strength. Allow clients using FTP irrespective of their cipher strength.

```
<SSL>
 ALLOW client.connection.negotiated_cipher.strength=(medium||high)
 ALLOW url.scheme=ftp
DENY
```

client.connection.negotiated_ssl_version=

Test the SSL version negotiated with a securely connected client.

Syntax

```
client.connection.negotiated_ssl_version={SSLV3|TLSV1|TLSV1.1|TLSV1.2|TLS1.3}
```

Layer and Transaction Notes

- Layers: <Proxy> , <SSL>
- Transactions: all proxies

Example

Test for connections secured with TLS 1.3.

```
<SSL>
  client.connection.negotiated_ssl_version=TLS1.3
```

client.effective_address=

Compares the effective client IP address against an IP address or subnet. If the effective client IP address that is extracted is not valid, the client IP address is used instead ("client.address=" on page 87).

Syntax

`client.effective_address={ip_address|ip_address_range|ip_address_wildcards|subnet|subnet_label}`

where:

- *ip_address*: Effective IP address; for example, 10.25.198.0
- *ip_address_range*: IP address range; for example, 192.0.2.0-192.0.2.255
- *ip_address_wildcards*: IP address specified using wildcards in any octet(s); for example, 10.25.*.0 or 10.*.*.0
- *subnet*: Subnet specification; for example, 10.25.198.0/24
- *subnet_label*: Label of a subnet definition block that binds a number of IP addresses or subnets

Layer and Transaction Notes

- Layers: <Admin>, <Cache>, <Diagnostic>, <DNS-Proxy>, <Exception>, <Forward>, <Proxy> , <SSL>, <SSL-Intercept>, <Tenant>
- Unavailable if the transaction is not associated with a client.

Example

Allow traffic only from a specified IP address.

```
; only allow traffic from 192.0.2.0
<Proxy>
  client.effective_address=192.0.2.0 allow
  deny
```

See Also

- Conditions: "client.address=" on page 87, "client.effective_address.country=" on the next page
- Properties: "client.effective_address.connection()" on page 384, "client.effective_address.request()" on page 386
- Definitions: "define subnet" on page 656
- Information on wildcards: http://www.symantec.com/docs/TECH241521
- Information on IP address ranges: http://www.symantec.com/docs/TECH241929

client.effective_address.country=

Returns the country that traffic came from, based on the effective client IP address.

If the geolocation feature is disabled or the database is not yet downloaded, this condition will always return “Unavailable” for the country name. If the database is enabled but not licensed, this condition will always return “Unlicensed” for the country name.

If you receive an “Obsolete country name” warning when installing policy, replace the outdated country name in policy with the suggested name in the message.

Syntax

```
client.effective_address.country=country_name
```

where:

- *country_name*: Name of the country to look up. You can also use the code name specified in the databases, for example, “CA” for Canada.

Layer and Transaction Notes

- Layers: <Admin>, <Cache>, <Diagnostic>, <DNS-Proxy>, <Exception>, <Forward>, <Proxy> , <SSL>, <SSL-Intercept>
- Unavailable if the transaction is not associated with a client

Example 1

Allow all if unlicensed. Only allow traffic from Canada if licensed.

```
<Proxy>
  client.effective_address.country=Unlicensed allow
  client.effective_address.country=CA allow
  deny
```

Example 2

Allow all if geolocation is disabled or database is not downloaded.

```
<Proxy>
  client.address.country=Unavailable allow
```

See Also

- Conditions: "client.effective_address=" on page 101
- Properties: "client.effective_address.connection()" on page 384, "client.effective_address.request()" on page 386

client.[effective_]address.ip_reputation=

Test the confidence level of the specified IP reputation category, for the IP address or effective IP address.

Syntax

```
client.[effective_]address.ip_reputation[.category1[,category2,...]=range
```

where:

- *category*: IP reputation category; if categories are not specified, the rule considers all existing categories.
- *range*: Confidence level expressed as one of the following:
  - *..level* - confidence level is less than or equal to the specified level
  - *level..* - confidence level is greater than or equal to the specified level
  - *level1..level2* - confidence level is between the specified levels, inclusive

Layer and Transaction Notes

- Layers: <Proxy>
- Transactions: current HTTP proxy transaction
- Unavailable if the transaction is not associated with a client

Example

Deny transactions based on IP reputation categories.

```
; deny transactions when confidence is 7 or greater that the IP is spam
<Proxy>
  client.address.ip_reputation.spam=7.. DENY
```

See Also

- Properties: "client.ip_reputation.category()" on page 389
- *SGOS Administration Guide*, “Control Traffic Based on Client IP Reputation”

client.effective_address.is_overridden=

Evaluates to yes if the effective client IP address has been changed to another IP address.

Syntax

```
client.effective_address.is_overridden={yes|no}
```

Layer and Transaction Notes

- Layers: <Admin>, <Cache>, <Diagnostic>, <DNS-Proxy>, <Exception>, <Forward>, <Proxy> , <SSL>, <SSL-Intercept>

Example

Only allow traffic if effective client IP address was not overridden

```
<Proxy>
  client.effective_address.is_overridden=yes
  deny
```

See Also

- Conditions: "client.address=" on page 87, "client.effective_address.country=" on page 102
- Properties: "client.effective_address.connection()" on page 384, "client.effective_address.request()" on page 386

client.host=

Test the hostname of the client (obtained through RDNS).

Syntax

`client.host[.string_modifier][.length]={hostname|domain_suffix|criterion}`

where:

- *string_modifier*: A supported string modifier:
  - `exact` - match the string exactly
  - `prefix` - match the start of a string
  - `regex` - regular expression match; see "Regex Reference" on page 703
  - `substring` - match part of a string
  - `suffix` - match the end of a string
- *hostname*: Host name.
- *domain-suffix*: Domain suffix, such as ".com".
- *criterion*: Type-appropriate test expression, such as a quote-delimited string expression or a regular expression.

> **Tip:** The pattern expression supports substitutions. You can specify a substitution expression with the .exact, .substring, .prefix, and .suffix string modifiers where they are available.

Layer and Transaction Notes

- Layers: `<Admin>`, `<Exception>`, `<Forward>`, `<Proxy>` , `<SSL>`, `<SSL-Intercept>`, `<Tenant>`
- Transactions: all proxies, excluding DNS

Example

Set an allow rule, followed by rules to deny users that meet the specified criteria.

```
<Proxy>
  ALLOW

<Proxy>
; deny all users that do not have an RDNS name that ends with symantec.com
  DENY client.host=!".symantec.com"
```

```
; deny all users that have test in their RDNS name
  DENY client.host.substring="test"
; deny all users that have an RDNS name that ends with example.symantec.com
; This is meant to include bexample.symantec.com and b.example.symantec.com.
  DENY client.host.suffix="example.symantec.com"
; dent all users that have numbers in their RDNS name.
  DENY client.host.regex="[0-9]*"
; deny all users that have an RDNS name that begins with fnord.
  DENY client.host.prefix="fnord."
```

client.host.has_name=

Test the status of the RDNS performed to determine "client.host=" on page 106.

Syntax

```
client.host.has_name={yes|no|restricted|refused|nxdomain|error}
```

Layer and Transaction Notes

- Layers: <Admin>, <Exception>, <Forward>, <Proxy> , <SSL>, <SSL-Intercept>
- Transactions: all proxies, excluding DNS proxy

Example

Deny all users from subnetA if their RDNS name could not be resolved. Deny all users from subnetB if they have no RDNS name, but allow them if their RDNS name lookup failed because of a DNS lookup error. The inclusion of the client's address in an RDNS restriction is a lookup error.

```
define subnet subnetA
  10.10.10.0/24
end

define subnet subnetB
  10.9.9.0/24
end

<Proxy>
  DENY

<Proxy>
  ALLOW client.address=subnetA client.host.has_name=yes

; for users in 'subnetB' nxdomain is the only error to prevent
  ALLOW client.address=subnetB client.host.has_name=!nxdomain
```

client.interface=

Test the interface used to service a user request.

This condition allows an administrator to define policy that applies to all traffic received at a given physical or virtual interface on the appliance.

Syntax

`client.interface=`*`adapter`*`[:`*`interface`*`[.`*`vlan`*`]]`

Layer and Transaction Notes

- Layers: `<Admin>`, `<Exception>`, `<Forward>`, `<Proxy>` , `<SSL>`, `<SSL-Intercept>`, `<Tenant>`
- Transactions: all proxies

Example

Allow the users in the 345 VLAN, connected to interface 0:0, to access only symantec.com.

```
<Proxy>
  client.interface=0:0.345 url.domain="symantec.com" allow
  client.interface=0:0.345 force_deny
```

See Also

- Conditions: "client.address=" on page 87, "client.protocol=" on page 111, "has_client=" on page 145, "proxy.address=" on page 198, "proxy.port=" on page 200

client.interface.routing_domain=

Test the routing domain associated with the interface used to service a user request.

This condition allows an administrator to define policy that applies to all traffic that goes through the specified routing domain.

Syntax

```
client.interface.routing_domain=routing_domain
```

Layer and Transaction Notes

- Layers: <Admin>, <Exception>, <Forward>, <Proxy> , <SSL>, <SSL-Intercept>, <Tenant>
- Transactions: all proxies

Example

Allow transactions from users in Midwest routing domain.

```
<Proxy>
  client.interface.routing_domain=midwest allow
```

See Also

- Conditions: "client.address=" on page 87, "client.protocol=" on the facing page, "has_client=" on page 145, "proxy.address=" on page 198, "proxy.port=" on page 200

client.protocol=

Test the protocol used between the client and the appliance.

Syntax

`client.protocol=`*`protocol`*

where:

- *protocol*: A supported protocol:

  http | https | ftp | tcp | socks | mms | rtsp | icp | aol-im | msn-im | yahoo-im | dns | telnet | epmapper | ssl | dns | rtmp | rtmpt | rtmpe | rtmpte | sip | sips | ms-turn

  Note about protocols

  - tcp specifies a tunneled transaction.
  - `client.protocol=dns` is valid in the `<DNS-Proxy>` layer only.
  - ms-turn, sip, and sips is returned either as a result of"force_protocol()" on page 414 or as a result of successful detection of one of these protocols for which "detect_protocol()" on page 395 was enabled. The protocol sips can also be returned as a result of the ssl.forward_proxy(sips) policy. The "tunneled=" on page 297 condition will report no for these types of traffic, even though the traffic is effectively still tunneled by the TCP tunnel or STunnel proxies.
    - SIP detection applies when protocol detection is enabled on an HTTP CONNECT, SOCKS CONNECT, or TCP Tunnel proxy connection where the protocol running inside the tunnel is SIP.
    - SIPS detection applies on SSL traffic with protocol detection enabled on a connection handled by the STunnel proxy, where the traffic was decrypted and determined to be SIP.

Layer and Transaction Notes

- Layers: `<Exception>`, `<Forward>`, `<Proxy>` , `<SSL>`, `<SSL-Intercept>`
- Transactions: all proxies
- Tests false if the transaction is not associated with a client.

Example

Do not detect protocol for the specified domain and allow only SSL transactions.

```
<proxy>
  url.domain=example.com detect_protocol(none)
  DENY client.protocol=!ssl tunneled=yes
```

See Also

- Conditions: "client.address=" on page 87, "client.interface=" on page 109, "proxy.address=" on page 198, "proxy.port=" on page 200

condition=

Tests if the specified defined condition is true.

Syntax

`condition=`*`condition_label`*

where:

- *`condition_label`*: Name of custom condition as defined in a "define condition" on page 637, "define url condition" on page 658, or "define url.domain condition" on page 660 definition block.

Layer and Transaction Notes

- Layers: `<Admin>`, `<Cache>`, `<Diagnostic>`, `<DNS-Proxy>`, `<Exception>`, `<Forward>`, `<Proxy>` , `<SSL>`, `<SSL-Intercept>`
- The defined conditions that are referenced may have usage restrictions, as they must be evaluated in the layer from which they are referenced.

Example 1

Deny access to client 1.2.3.4 for any http request through proxy port 8080.

```
define condition qa
  client.address=1.2.3.4 proxy.port=8080
end

<Proxy>
  condition=qa client.protocol=http deny
```

Example 2

Restrict access to internal sites to specific groups using nested conditions.

```
define condition restricted_sites
  url.domain=internal.my_co.com
end

define condition has_full_access
  group=admin,execs,managers
end

define condition forbidden
  condition=restricted_sites condition=!has_full_acesss
end
```

```
<Proxy>
  authenticate(My_realm)

<Proxy>
  condition=forbidden deny
```

Example 3

Deny access to specified URLs.

```
define url condition test
  http://www.x.com time=0800..1000
  http://www.y.com month=1
  http://www.z.com hour=9..10
end

<Proxy>
  condition=test deny
```

See Also

- Properties: "action()" on page 337
- Definitions: "define condition" on page 637, "define url condition" on page 658, "define url.domain condition" on page 660

console_access=

Tests if the current request is destined for the <Admin> layer. This test can be used to distinguish access to the Management Console by administrators who are explicitly proxied to the appliance being administered. The test can be used to guard transforms that should not apply to the Management Console. This cannot be used to test Telnet sessions, as they do not go through a <Proxy> layer.

Syntax

```
console_access={yes|no}
```

Layer and Transaction Notes

- Layers: <Cache>, <Exception>, <Proxy> , <SSL>
- Transactions: HTTP proxy

Example

Deny requests that are not destined for <Admin> layer.

```
<Proxy>
  console_access=no deny
```

See Also

- Conditions: "admin.access=" on page 69

content_management=

Tests if the current request is a content management transaction.

Syntax

```
content_management={yes|no}
```

Layer and Transaction Notes

- Layers: <Cache>, <Forward> , <SSL>
- Transactions: all proxies

See Also

- Conditions: "category=" on page 85, "ftp.method=" on page 139, "http.method=" on page 155, "server_url=" on page 269
- Properties: "http.request.version()" on page 475, "http.response.version()" on page 479

data_leak_detected=

Tests true if the current transaction contains the header string `data_leak_detected`. This header string is added to the header during ICAP scanning if server is a DLP server.

Syntax

```
data_leak_detected={yes|no}
```

Layer and Transaction Notes

- Layers: `<Exception>`, `<Proxy>`

Example

If a data leak is detected, return the specified exception page.

```
<Proxy>
  data_leak_detected=yes exception(DLP_exception)
```

See Also

- Conditions: "virus_detected=" on page 326

date=

Tests true if the current time is within the specified range. The comparison is made against local time unless UTC is specified.

Syntax

`date[.utc]=`*`date_range`*

where:

- *`date_range`*: Date range expressed in the form *`YYYYMMDD..YYYYMMDD`* or *`MMDD..MMDD`*.

Layer and Transaction Notes

- Layers: `<Admin>`, `<Cache>`, `<Diagnostic>`, `<DNS-Proxy>`, `<Exception>`, `<Forward>`, `<Proxy>` , `<SSL>`, `<SSL-Intercept>`
- Transactions: all

> **Note:** Using time-related conditions to control caching behavior in a `<Cache>` layer may cause thrashing of the cached objects.

See Also

- Conditions: "day=" on the facing page, "hour=" on page 147, "minute=" on page 193, "month=" on page 195, "time=" on page 292, "weekday=" on page 328, "year=" on page 330

day=

Tests if the day of the month is in the specified range or an exact match. The appliance's configured date and time zone are used to determine the current day of the month. To specify the UTC time zone, use the form day.utc=. The numeric pattern used to test the day condition cannot contain whitespace.

Syntax

`day[.utc]={[first_day]..[last_day]|exact_day}`

where:

- *first_day*: Integer from 1 to 31, indicating the first day of the month that will test true. If left blank, day 1 is assumed.
- *last_day*: Integer from 1 to 31, indicating the last day of the month that will test true. If left blank, day 31 is assumed.
- *exact_day*: Integer from 1 to 31, indicating the day of the month that will test true.

> **Note:** To test against an inverted range, such as days early and late in the month, a shorthand expression is available. Whereas `day=(..5|25..)` specifies the first 5 days of the month and last few days of the month, the policy language also recognizes `day=25..5` as the same.

Layer and Transaction Notes

- Layers: `<Admin>`, `<Cache>`, `<Diagnostic>`, `<DNS-Proxy>`, `<Exception>`, `<Forward>`, `<Proxy>` , `<SSL>`, `<SSL-Intercept>`
- Transactions: all

> **Note:** Using time-related conditions to control caching behavior in a `<Cache>` layer may cause thrashing of the cached objects.

Example 1

Test for New Year's Day.

```
<Proxy>
 day=1 month=1
```

Example 2

Allow access to a special event site only during the days of the event. Restrict access during non-event times.

```
<Proxy> url=http://www.xyz.com/special_event

; The next line matches, but does nothing if allow is the default
; year=2023 month=7 day=23..25 ; During the event
; deny Any other time
; This form of the rule assumes access is generally denied, and grants access during
; the special event.

<Proxy> url=http://www.xyz.com/special_event
  allow year=2023 month=7 day=23..25 ; During the event
```

See Also

- Conditions: "date=" on page 118, "hour=" on page 147, "minute=" on page 193, "month=" on page 195, "time=" on page 292, "weekday=" on page 328, "year=" on page 330

dns.client_transport=

Test the transport protocol of a proxied DNS query

Syntax

```
dns.client_transport={https|tcp|udp}
```

Layer and Transaction Notes

- Layers: <DNS-Proxy>
- Transactions: DNS proxy

Example

Refuse all DNS queries that use the TCP protocol unless the query is coming from the subnet 10.9.8.0/24.

```
<DNS-Proxy>
 client.address=!10.9.8.0/24 dns.client_transport=tcp dns.respond(refused)
```

dns.request.address=

Test the address of a PTR type DNS query (also known as RDNS).

Syntax

`dns.request.address={`*`ip_address`*`|`*`ip_address_range`*`|`*`ip_address_wildcards`*`|`*`subnet`*`|`*`subnet_label`*`}`

where:

- *`ip_address`*: Client IP address; for example, 10.25.198.0
- *`ip_address_range`*: IP address range; for example, 192.0.2.0-192.0.2.255
- *`ip_address_wildcards`*: IP address specified using wildcards in any octet(s); for example, 10.25.*.0 or 10.*.*.0
- *`subnet`*: Subnet specification; for example, 10.25.198.0/24
- *`subnet_label`*: Label of a "define subnet" on page 656 block that binds a number of IP addresses or subnets

Layer and Transaction Notes

- Layers: `<DNS-Proxy>`
- Transactions: DNS proxy

Example

Refuse all DNS PTR queries for addresses in the 10.10.0.0/16 subnet. Respond with host1.example.com to DNS PTR queries for 10.9.8.1, and with host2.example.com to DNS PTR queries for 10.9.8.2.

```
<DNS-Proxy>
  dns.request.address=10.10.0.0/16 dns.respond(refused)
  dns.request.address=10.9.8.1 dns.respond.ptr("host1.example.com")
  dns.request.address=10.9.8.2 dns.respond.ptr("host2.example.com")
```

See Also

- Information on wildcards: http://www.symantec.com/docs/TECH241521
- Information on IP address ranges: http://www.symantec.com/docs/TECH241929

dns.request.category=

Test the URL category of either the DNS queried hostname or IP address. The content filtering category is associated with hostnames in IPv6 (AAAA) DNS queries in addition to IPv4 (A) DNS queries.

> **Note:** Additional RDNS/DNS lookups are not performed to categorize the DNS query.

System-Defined Statuses

The following statuses are system-defined:

- `none`: The request is uncategorized by all enabled content filter providers. If even one content filter provider returns a category for the requested site, policy rules containing a `*.category=none` condition will not match.
- `pending`: The categorization request has not been processed or is not complete.
- `unavailable`: A content filter provider is selected in configuration, but an error occurs in determining the category. Other categories can still be assigned directly by policy. This categorization might be a result of a missing database.
- `unlicensed`: A content filter provider license is expired. When this status is true, the `unavailable` status is also true.

Renamed Category Warnings

If CPL includes a category that has been renamed in the currently downloaded database, policy warnings occur at compilation time. The following is an example of the warning:

```
Deprecation warning: 'old_category'; 'old_category' has been replaced by 'new_category'
due to Category name updated and will no longer be accepted after Sat, 11 Jul 2020 00:00:00 UTC.
Please switch to the new name before then.
```

To ensure that policy performs as intended, rename all instances of the affected category in CPL and re-apply policy by the specified date.

Syntax

```
dns.request.category={status|category_name1[,category_name2,...]|category_group1[,category_
group2,...]}
```

where:

- *status*: One of the following system-defined statuses: `none`, `pending`, `unavailable`, and `unlicensed`.
- *category_name*:Category names defined by policy or the selected content filter provider.
- *category_group*: Category group names defined by policy or Blue Coat provider if enabled.

The list of currently valid category names and category group names is available both through the Management Console and CLI.

Layer and Transaction Notes

- Layers:<DNS-Proxy>
- Transactions: DNS proxy

Example

Refuse all DNS type A queries from the Engineering subnet for category HR_intranet_servers, and all DNS type AAAA queries from the HR subnet for category Engineering_intranet_servers.

```
define category HR_intranet_servers
  hr1.example.com
  hr2.example.com
end

define category Engineering_intranet_servers
  eng1.example.com
  engweb.example.com
end

define subnet Engineering
  10.10.0.0/16
end

define subnet HR
  10.9.0.0/16
end

<DNS-Proxy> dns.request.type=A
  client.address=Engineering dns.request.category=HR_intranet_servers dns.respond(refused)

<DNS-Proxy> dns.request.type=AAAA client.address=HR dns.request.category=Engineering_intranet_servers
dns.respond(refused)
```

dns.request.class=

Test the QCLASS of the DNS query

Syntax

```
gdns.request.class={any|ch|hs|in|none|value}
```

where:

- *value*: Integer from 0 to 65535.

Layer and Transaction Notes

- Layers: <DNS-Proxy>
- Transactions: DNS proxy

Example

Refuse all DNS traffic that does not use the QCLASS IN

```
<DNS-Proxy>
  dns.request.class=!IN dns.respond(refused)
```

dns.request.name=

Test the QNAME in the question section of the DNS query.

Syntax

`dns.request.name[.`*`string_modifier`*`][.length]={`*`hostname`*`|`*`domain_suffix`*`|`*`criterion`*`}`

where:

- *`string_modifier`*: A supported string modifier:
  - `exact` - match the string exactly
  - `prefix` - match the start of a string
  - `regex` - (default if no string modifier is specified) regular expression match; see "Regex Reference" on page 703
  - `substring` - match part of a string
  - `suffix` - match the end of a string
- *`hostname`*: Host name.
- *`domain-suffix`*: Domain suffix, such as ".com".
- *`criterion`*: Type-appropriate test expression, such as a quote-delimited string expression or a regular expression.

Layer and Transaction Notes

- Layers: `<DNS-Proxy>`
- Transactions: DNS proxy

Example

Refuse all queries for hostnames that end with example.com, and permit queries for host1.example.com.

```
<DNS-Proxy>
  dns.request.name=.example.com dns.respond(refused)

<DNS-Proxy>
  dns.request.name=host1.example.com dns.respond(auto)
```

dns.request.opcode=

Test the OpCode in the header of the DNS query.

Syntax

`dns.request.opcode={query|status|notify|update|`*`opcode`*`}`

where:

- *opcode*: OpCode from 0 to 15.

Layer and Transaction Notes

- Layers: `<DNS-Proxy>`
- Transactions: DNS proxy

Example

Refuse all DNS traffic that does not use the QUERY Opcode.

```
<DNS-Proxy>
  dns.request.opcode=!QUERY dns.respond(refused)
```

dns.request.type=

Test the QTYPE of the DNS query.

Syntax

`dns.request.type={qtype|value}`

where:

- *qtype*: One of the following supported QTYPEs:

  A, CNAME, MR, HINFO, RP, RT, KEY, LOC, SRV, CERT, OPT, RRSIG, UID, TSIG, MAILA, NS, SOA, NULL, MINFO, AFSDB, NSAP, PX, NXT, ATMA, A6, APL, NSEC, GID, IXFR, ALL, MD, MB, WKS, MX, X25, NSAP-PTR, GPOS, EID, NAPTR, DNAME, DS, DNSKEY, UNSPEC, AXFR, MF, MG, PTR, TXT, ISDN, SIG, AAAA, NIMLOC, KX, SINK, SSHFP, UINFO, TKEY, MAILB

- *value*: Integer from 0 to 65535.

Layer and Transaction Notes

- Layers: `<DNS-Proxy>`
- Transactions: DNS proxy

Example

Test for the specified QTYPE.

```
<DNS-Proxy>
  dns.request.type=CNAME
```

dns.response.a=

Test the addresses from the A RRs in the DNS response.

Syntax

`dns.response.a={ip_address|ip_address_range|ip_address_wildcards|subnet|subnet_label}`

where:

- *ip_address*: Client IP address; for example, 10.25.198.0
- *ip_address_range*: IP address range; for example, 192.0.2.0-192.0.2.255
- *ip_address_wildcards*: IP address specified using wildcards in any octet(s); for example, 10.25.*.0 or 10.*.*.0
- *subnet*: Subnet specification; for example, 10.25.198.0/24
- *subnet_label*: Label of a "define subnet" on page 656 block that binds a number of IP addresses or subnets

Layer and Transaction Notes

- Layers: `<DNS-Proxy>`
- Transactions: DNS proxy

Example

If the response address in the DNS response is 10.9.8.7, change it to 10.10.10.10.

```
<DNS-Proxy>
  dns.response.a=10.9.8.7 dns.respond.a(10.10.10.10)
```

See Also

- Property: "dns.respond.a()" on page 398
- Information on wildcards: http://www.symantec.com/docs/TECH241521
- Information on IP address ranges: http://www.symantec.com/docs/TECH241929

dns.response.aaaa=

Matches with type AAAA RR in the answer section of the DNS response.

> **Note:** The DNS proxy caches IPv6 AAAA records.

Syntax

`dns.response.aaaa={`*`ip_address`*`[/`*`prefix_length`*`]}`

where:

- *`ip_address`*: Client IP address or subnet specification
- *`prefix_length`*: Prefix length of subnet mask

Layer and Transaction Notes

- Layers: `<DNS-Proxy>`
- Transactions: DNS proxy

Example

If the DNS response from server contains an AAAA RR equaling 2001::1, it sends a DNS response to the client with an AAAA RR (2001::2). This policy allows the appliance to re-write the AAAA RR record returned from the DNS server.

```
<DNS-Proxy>
  dns.response.aaaa=2001::1 dns.respond.aaaa(2001::2)
```

See Also

- Properties: "dns.respond.aaaa()" on page 399

dns.response.cname=

Test the string values from the CNAME RRs in the DNS response.

Syntax

`dns.request.name[.`*`string_modifier`*`][.length]={`*`hostname`*`|`*`domain_suffix`*`|`*`criterion`*`}`

where:

- *`string_modifier`*: A supported string modifier:
  - `exact` - match the string exactly
  - `prefix` - match the start of a string
  - `regex` - (default if no string modifier is specified) regular expression match; see "Regex Reference" on page 703
  - `substring` - match part of a string
  - `suffix` - match the end of a string
- *`hostname`*: Host name.
- *`domain-suffix`*: Domain suffix, such as ".com".
- *`criterion`*: Type-appropriate test expression, such as a quote-delimited string expression or a regular expression.

Layer and Transaction Notes

- Layers: `<DNS-Proxy>`
- Transactions: DNS proxy

Example

Refuse all DNS queries that have .example.com in any of the CNAME RRs, and permit host1.example.com.

```
<DNS-Proxy>
  dns.response.cname=.example.com dns.respond(refused)

<DNS-Proxy>
  dns.response.cname=host1.example.com dns.respond(auto)
```

dns.response.code=

Test the numeric response code of the proxied DNS query's response.

Syntax

`dns.response.code={`*`code`*`|`*`value`*`}`

where:

- *code*: One of the following response codes:

  noerror,formerr, servfail, nxdomain, notimp, refused, yxdomain, yxrrset, nxrrse, notauth, notzone

- *value*: Integer from 0 to 15.

Layer and Transaction Notes

- Layers: `<DNS-Proxy>`
- Transactions: DNS proxy

Example

A DNS server that routinely responds with yxdomain, but some client machines do not handle that response code gracefully. Convert the yxdomain response to nxdomain.

```
<DNS-Proxy>
  dns.response.code=yxdomain dns.respond(nxdomain)
```

See Also

- Properties: "dns.respond()" on page 397

dns.response.nodata=

Test whether the DNS response had no RRs.

Syntax

```
dns.response.nodata={yes|no}
```

Layer and Transaction Notes

- Layers: <DNS-Proxy>
- Transactions: DNS proxy

Example

A DNS server that routinely sends back empty responses. Some clients fail to handle this gracefully. Temporarily convert empty responses to nxdomain.

```
<DNS-Proxy>
  dns.response.nodata=yes dns.respond(nxdomain)
```

See Also

- Properties: "dns.respond()" on page 397

dns.response.ptr=

Test the hostname values from the PTR RRs in the DNS response.

Syntax

`dns.request.name[.string_modifier][.length]={hostname|domain_suffix|criterion}`

where:

- *string_modifier*: A supported string modifier:
  - `exact` - match the string exactly
  - `prefix` - match the start of a string
  - `regex` - (default if no string modifier is specified) regular expression match; see "Regex Reference" on page 703
  - `substring` - match part of a string
  - `suffix` - match the end of a string
- *hostname*: Host name.
- *domain-suffix*: Domain suffix, such as ".com".
- *criterion*: Type-appropriate test expression, such as a quote-delimited string expression or a regular expression.

Layer and Transaction Notes

- Layers: `<DNS-Proxy>`
- Transactions: DNS proxy

Example

Test for responses with symantec.com in the PTR field of the RR record.

```
<DNS-Proxy>
  dns.response.ptr=.symantec.com
```

effective_date=

Specifies the set of WAF rules selected by the enclosing "define application_protection_set" on page 632 definition. Symantec delivers WAF rule updates for the Blacklist and Analytics Filter engines through the Web Application Protection (WAP) subscription. This condition allows WAF administrators to control rule selection and usage based on the date the rules were added.

For example, rules qualified in a pre-production environment can be set to block-mode, while new rules can be set to monitor-mode. This enables an organization to take advantage of new rules immediately without introducing new false-positives that block legitimate requests. See the "Example " below below.

After the new rules are qualified, `effective_date=` can be migrated forward in the production environment, thereby setting the new rules into block-mode.

> **Note:** This condition is only valid when a WAP (Web Application Protection) subscription is present on the ProxySG appliance.

Syntax

`effective_date=`*`date_range`*

where:

- *`date_range`*: Date or a date range that determines rule set selection. If the condition is not specified, all rules are selected.

Layer and Transaction Notes

- Layers: `<Proxy>`
- Transactions: all proxies

Example

Select WAF rules added up to 2021-09-01 and block the transactions. Selects WAF rules added on 2021-09-02 or later and monitor the transactions.

```
define application_protection_set WAF_QualifiedRules
  engine=blacklist effective_date=..20210901
end

<Proxy>
  http.request.detection.WAF_QualifiedRules(block
```

```
define application_protection_set WAF_NewRules
  engine=blacklist effective_date=20210902..
end

<Proxy>
  http.request.detection.WAF_NewRules(monitor)
```

exception.id=

Tests whether the exception being returned to the client is the specified exception. It can also be used to determine whether the exception being returned is a built-in or user-defined exception.

Built-in exceptions are handled automatically by the appliance but special handling can be defined within an `<Exception>` layer. Special handling is most often required for user-defined exceptions.

Syntax

`exception.id=`*`exception_id`*

where:

- *`exception_id`*: Name of a built-in exception of the form *`exception_id`*, or name of a user-defined exception in the form *`user_defined.exception_id`*.

In addition to testing the identity of exceptions set by the "exception()" on page 402 property, `exception.id=` can also test for exceptions returned by other CPL gestures, such as `policy_denied` returned by the "deny" on page 392 property and `policy_redirect` returned by the "redirect()" on page 611 action.

Layer and Transaction Notes

- Layers: `<Exception>`
- Transactions: all proxies

Example

Catch some commonly generated exceptions are caught. Appropriate subnet and action and category definitions are assumed.

```
<Proxy> url.domain=partner.my_co.com action.partner_redirect(yes) ; action contains redirect( )


<Proxy> url.domain=internal.my_co.com force_deny client.address!=mysubnet
  authenticate(my_realm)

<Proxy> deny.unauthorized
  url.domain=internal.my_co.com/hr group=!hr;
; and other group/user restrictions ...

<Proxy> category=blocked_sites
  exception(user_defined.restricted_content)
; could probably have used built in content_filter_denied

; Custom handling for some built-in exceptions
```

```
<Exception>
; thrown by authenticate( ) if there is a realm configuration error
  exception.id=configuration_error action.config_err_alerts(yes)
; thrown by deny.unauthorized
  exception.id=authorization_failed action.log_permission_failure(yes)
; thrown by deny or force_deny
  exception.id=policy_denied action.log_interloper(yes)

<Exception> exception.id=user_defined.restricted_content
; any policy required for this user defined exception
...
```

See Also

- Properties: "authenticate()" on page 348, "authenticate.force()" on page 354,"deny" on page 392, "deny.unauthorized()" on page 394, "exception()" on page 402
- Actions: "redirect()" on page 611

ftp.method=

Tests FTP and FTPS request methods against any of a well-known set of methods. A CPL parse error is given if an unrecognized method is specified.

This condition evaluates to true if the request method matches any of the methods specified. It evaluates to NULL if the request is not an FTP/FTPS protocol request.

Syntax

`ftp.method=`*`ftp_method`*

where:

- *`ftp_method`*: FTP and FTPS request methods. **Show FTP/S methods**
  ABOR, ACCT, ALLO, APPE, AUTH, CDUP, CWD, DELE, EPRT, EPSV, HELP, LIST, MDTM, MKD, MODE, NLST, NOOP, PASS, PASV, PBSZ, PORT, PROT, PWD, REST, RETR, RMD, RNFR, RNTO, SITE, SIZE, SMNT, STOR, STOU, STRU, SYST, TYPE, USER, XCUP, XCWD, XMKD, XPWD, XRMD, OPEN

Layer and Transaction Notes

- Layers: `<Cache>`, `<Exception>`, `<Proxy>`
- Transactions: FTP proxy, implicit and explicit FTPS proxy transactions

Example

Deny FTP uploads for the specified users/groups.

```
<Proxy>
  realm=LDAP user="CN=Users1,CN=Users2,dc=test,dc=local1,dc=local2" ftp.method=(STOR,STOU) deny
```

See Also

- Conditions: "category=" on page 85, "content_management=" on page 116, "http.method=" on page 155, "server_url=" on page 269, "socks.method=" on page 279

group=

Tests if the client is authenticated, and the client belongs to the specified group. If both of these conditions are met, the result is true. In addition, the "realm=" on page 208 condition can be used to test whether the user is authenticated in the specified realm. This condition is unavailable if the current transaction is not authenticated; that is, the"authenticate()" on page 348 property is set to no.

If you reference more than one realm in your policy, consider disambiguating group tests by combining them with a "realm=" on page 208 test. This reduces the number of extraneous queries to authentication services for group information that does not pertain to that realm.

Syntax

`group=group_name`

where:

- *group_name*: Name of a group in the default realm. The required form, and the name attribute's case-sensitivity, depends on the type of realm. Refer to the documentation for your authentication realm for details.

Layer and Transaction Notes

- Layers: <Admin>, <Exception>, <Forward>, <Proxy> , <SSL>, <SSL-Intercept>
- Transactions: all proxies, administrator

> **Note:** When used in the <forward> layer, this condition can evaluate to NULL (shown in a trace as N/A) if no authenticated client exists. Rules containing these conditions can be guarded by authenticated= to preserve normal logic.

Example

Allow a group of administrators in each realm to administer the appliance.

```
; Test if user is authenticated in group all_staff and specified realm.
realm=corp group=all_staff

; Sample group tests for each type of realm.
; <Admin> layer uses a whitelist model by default.

define condition RW_Admin
  realm=LocalRealm group=RWAdmin
  realm=NTLMRealm group=xyz-domain\cache_admin
  realm=LDAPRealm group="cn=cache_admin, ou=groups, o=xyz"
; The RADIUSRealm uses attributes, and this can be expressed as follows:
```

```
  realm=RADIUSRealm attribute.ServiceType=8
end

<Admin>
  client.adress=10.10.1.250/28 authenticate(LocalRealm)
  client.adress=10.10.1.0/24 authenticate(NTLMRealm)
  client.adress=10.10.2.0/24 authenticate(LDAPRealm)
  client.adress=10.10.3.0/24 authenticate(RADIUSRealm)

<Admin>
  allow condition=RW_Admin admin.access=(read||write)
```

See Also

- Conditions: "authenticated=" on page 81,"has_attribute.name=" on the next page, "http.transparent_authentication=" on page 180, "realm=" on page 208, "user=" on page 312, "user.domain=" on page 316, "user.x509.issuer=" on page 322, "user.x509.serialNumber=" on page 323, "user.x509.subject=" on page 325
- Properties: "authenticate()" on page 348, "authenticate.force()" on page 354, "check_authorization()" on page 376, "socks.authenticate()" on page 554, "socks.authenticate.force()" on page 556

has_attribute.*name*=

Tests if the current transaction is authenticated in an LDAP or RADIUS realm and if the authenticated user has the specified attribute. If the attribute specified is not configured in the LDAP schema and yes is used in the expression, the condition always yields false. This condition is unavailable if the current transaction is not authenticated (that is, the"authenticate()" on page 348 property is set to no.

If you reference more than one realm in your policy, consider disambiguating has_attribute.*name*= tests by combining them with a "realm=" on page 208 test. This reduces the number of extraneous queries to authentication services for attribute information that does not pertain to that realm.

> **Note:** This condition is incompatible with Novell eDirectory servers. If the name attribute is configured in the LDAP schema, then all users are reported by the eDirectory server to have the attribute, regardless of whether they actually do. This can cause unpredictable results.

Syntax

```
has_attribute.name={yes|no}
```

where:

- *name*: LDAP or RADIUS attribute. Case-sensitivity for the attribute name depends on the realm definition in configuration. Supported RADIUS attributes are:
  - Callback-ID
  - Callback-Number
  - Filter-ID
  - Framed-IP-Address
  - Framed-IP-Netmask
  - Framed-MTU
  - Framed-Pool
  - Framed-Protocol
  - Framed-Route
  - Idle-Timeout
  - Login-LAT-Group

- Login-LAT-Node
- Login-LAT-Port
- Login-LAT-Service
- Login-IP-Host
- Login-TCP-Port
- Port-Limit
- Service-Type
- Session-Timeout
- Tunnel-Assignment-ID
- Tunnel-Medium-Type
- Tunnel-Private-Group-ID
- Tunnel-Type
- Blue-Coat-Group

Layer and Transaction Notes

- Layers: `<Admin>`, `<Exception>` (RADIUS only), `<Forward>` (RADIUS only), `<Proxy>`
- Transactions: proxy, administration (LDAP); all (RADIUS)

Example 1

Allow users to access the proxy if they have the RADIUS attribute Callback-Number. The attribute could have any value, including null.

```
<Proxy>
  authenticate(RADIUSRealm)

<Proxy>
  allow has_attribute.Callback-Number=yes
```

Example 2

Allow users to access the proxy if they have the LDAP attribute ProxyUser. The attribute could have any value, including null. Generally this kind of policy is established in the first proxy layer, and sets up either a blacklist or whitelist model, as required.

```
<Proxy>
  authenticate(LDAPRealm)
```

```
; Setting up a whitelist model
<Proxy>
  deny has_attribute.ProxyUser=no

; Setting up a blacklist model
<Proxy>
  allow has attribute.ProxyUser=yes
  deny
```

See Also

- Conditions:"attribute.name=" on page 75, "authenticated=" on page 81, "group=" on page 140, "http.transparent_authentication=" on page 180, "realm=" on page 208, "user=" on page 312, "user.domain=" on page 316
- Properties: "authenticate()" on page 348, "authenticate.force()" on page 354, "check_authorization()" on page 376

has_client=

Test whether the current transaction has a client. This can be used to guard conditions that depend on client identity in a <Forward> layer.

Syntax

```
has_client={yes|no}
```

Layer and Transaction Notes

- Layers: <Cache>, <Forward>, <SSL>
- Transactions: all

Example

Forward clientless connections to the specified forwarding host.

```
<Forward>
  has_client=no forward.fail_open(no) forward(upstream_host)
```

See Also

- Conditions: "client.address=" on page 87, "client.interface=" on page 109, "client.protocol=" on page 111, "proxy.address=" on page 198, "proxy.port=" on page 200, "streaming.client=" on page 283
- Properties: "access_server()" on page 335, "max_bitrate()" on page 494, "streaming.transport()" on page 569

health_check=

Tests whether a transaction belongs to a health check.

This trigger tests whether the current transaction is a health check transaction or not. Optionally, the trigger tests whether the transaction is that of a specific health check.

Syntax

`health_check={yes|no|`*`health_check_name`*`}`

where:

- *`health_check_name`*: Name of a specific health check. When specifying a user-defined health check, the prefix `user.` is optional. In all other cases the complete health check name, including its prefix, must be entered.

Layer and Transaction Notes

- Layers: `<Forward>` , `<SSL>`
- Transactions: all

Example

Prevent any forwarding for a user-defined health check user upstream.

```
<Forward>
  health_check=user.upstream forward(no)
```

See Also

- Conditions: "is_healthy.health_check=" on page 184

hour=

Tests if the time of day is in the specified range or an exact match.

The current time is determined by the appliance's configured clock and time zone by default, although the UTC time zone can be specified by using the form `hour.utc=`. The numeric pattern used to test the `hour=` condition contains no whitespace.

> **Note:** Any range of hours or exact hour includes all the minutes in the final hour. See the Example section.

Syntax

`hour[.utc]={[`*`first_hour`*`]..[`*`last_hour`*`]|`*`exact_hour`*`}`

where:

- *`first_hour`*: Two digits (*nn*) in 24-hour time format representing the first hour in a range; for example, 09 means 9:00 a.m. If left blank, midnight (00) is assumed–exactly 00:00 a.m.
- *`last_hour`*: Two digits (*nn*) in 24-hour time format representing the last full hour in a range; for example, 17 specifies 5:59 p.m. If left blank, 23 is assumed (23:59 p.m.).
- *`exact_time`*: Two digits (*nn*) in 24-hour time format representing an exact, full hour.

Layer and Transaction Notes

- Layers: `<Admin>`, `<Cache>`, `<Diagnostic>`, `<DNS-Proxy>`, `<Exception>`, `<Forward>`, `<Proxy>` , `<SSL>`, `<SSL-Intercept>`
- Transactions: all

> **Note:** Using time-related conditions to control caching behavior in a `<Cache>` layer may cause thrashing of the cached objects.

Example 1

Test for 3:00 a.m. to 1:59 p.m. UTC.

```
hour.utc=03..13
```

Example 2

Restrict access to various sites during business hours.

```
; Restrict access to external sites during business hours.
; This rule assumes that the user has access that must be restricted.
<Proxy>
; Internal site always available, no action required
  server_url.domain=xyz.com
; Restrict other sites during business hours
  deny weekday=1..5 hour=9..16
; If a previous rule had denied access, then this rule could provide an exception.
```

http.connect=

Tests whether an HTTP CONNECT tunnel is in use between the appliance and the client.

Syntax

```
http.connect={yes|no}
```

Layer and Transaction Notes

- Layers: <Exception>, <Forward>, <Proxy> , <SSL>
- Transactions: all proxies

Example

```
<Proxy>
  http.connect=yes
```

http.connect.host=

Tests the hostname (the host value in the first line of the HTTP CONNECT request) obtained from the original HTTP CONNECT request URL. This condition supports all substitution variables. For example, you can use the `$(url.host)` substitution variable to compare the value of the url.host against the value specified by this condition. For more information on substitution variables, refer to the *ProxySG Log Fields and CPL Substitutions Reference*: https://techdocs.broadcom.com/content/broadcom/techdocs/us/en/symantec-security-software/web-and-network-security/proxysg/7-2/proxysg-log-fields-and-substitutions.html.

Syntax

`http.connect.host.`*`string_modifier`*`=`*`criterion`*

where:

- *`string_modifier`*: A supported string modifier:
  - `exact` - match the string exactly
  - `prefix` - match the start of a string
  - `regex` - (default if no string modifier is specified) regular expression match; see "Regex Reference" on page 703
  - `substring` - match part of a string
  - `suffix` - match the end of a string
- *`criterion`*: Type-appropriate test expression, such as a quote-delimited string expression or a regular expression.

Layer and Transaction Notes

- Layers: `<Cache>`, `<Exception>`, `<Proxy>` , `<SSL>`, `<SSL-Intercept>`
- Transactions: all proxies

Example

```
<Proxy>
 http.connect.host=!"$(url.host)" Deny
```

http.connect.host.category=

Tests the category of the host name in the HTTP CONNECT request.

System-Defined Statuses

The following statuses are system-defined:

- none: The request is uncategorized by all enabled content filter providers. If even one content filter provider returns a category for the requested site, policy rules containing a `*.category=none` condition will not match.
- pending: The categorization request has not been processed or is not complete.
- unavailable: A content filter provider is selected in configuration, but an error occurs in determining the category. Other categories can still be assigned directly by policy. This categorization might be a result of a missing database.
- unlicensed: A content filter provider license is expired. When this status is true, the unavailable status is also true.

Renamed Category Warnings

If CPL includes a category that has been renamed in the currently downloaded database, policy warnings occur at compilation time. The following is an example of the warning:

```
Deprecation warning: 'old_category'; 'old_category' has been replaced by 'new_category'
due to Category name updated and will no longer be accepted after Sat, 11 Jul 2020 00:00:00 UTC.
Please switch to the new name before then.
```

To ensure that policy performs as intended, rename all instances of the affected category in CPL and re-apply policy by the specified date.

Syntax

```
http.connect.host.category={status|category_name1[,category_name2,...]|category_group1[,category_
group2,...]}
```

where:

- *status*: One of the following system-defined statuses: none, pending, unavailable, and unlicensed.
- *category_name*: Category names defined by policy or the selected content filter provider.
- *category_group*: Category group names defined by policy or Blue Coat provider if enabled.

  The list of currently valid category names and category group names is available both through the Management Console and CLI.

Layer and Transaction Notes

- Layers: <Cache>, <Exception>, <Proxy> , <SSL>, <SSL-Intercept>
- Transactions: proxy transactions with a connect request URL

Example

Deny requests for games or sports-related content and return the specified exception page.

```
<Proxy>
; Tests true if the request is in one of these categories.
  http.connect.host.category=(Sports, Games) exception(content_filter_denied)
  http.connect.host.category=unavailable exception(content_filter_unavailable); Fail closed
```

http.connect.port=

Tests the port (the port value in the first line of the HTTP CONNECT request) obtained from the original HTTP CONNECT request URL.

Syntax

`http.connect.port=`*`port_number`*

Layer and Transaction Notes

- Layers: `<Exception>`, `<Forward>`, `<Proxy>` , `<SSL>`, `<SSL-Intercept>`, `<Tenant>`
- Transactions: all proxies

Example

```
<Proxy>
  http.connect.port=8080
```

http.connect.User-Agent=

Tests which user agent is used to initiate an explicit proxy HTTP CONNECT request.

> **Note:** You cannot use this condition to match a User-Agent header in HTTP transactions. Use the request.header.User-Agent= condition to match HTTP transactions.

Syntax

`http.connect.User-Agent[.`*`string_modifier`*`][.case_sensitive]=`*`criterion`*

where:

- *`string_modifier`*: A supported string modifier:
  - `exact` - match the string exactly
  - `prefix` - match the start of a string
  - `regex` - (default if no string modifier is specified) regular expression match; see "Regex Reference" on page 703
  - `substring` - match part of a string
  - `suffix` - match the end of a string
- `case_sensitive` - Perform case-sensitive match.
- *`criterion`*: Type-appropriate test expression, such as a quote-delimited string expression or a regular expression.

Layer and Transaction Notes

- Layers: `<Proxy>` , `<SSL-Intercept>`
- Transactions: explicitly proxied HTTPS transactions

Example

Inspect the User-Agent header for specific browser and intercept SSL if matched.

```
<SSL-Intercept>
  http.connect.User-Agent.exact="my_browser" ssl.forward_proxy(yes)
```

See Also

- Conditions: "live=" on page 192, "streaming.client=" on page 283, "streaming.content=" on page 284
- Properties: "access_server()" on page 335, "max_bitrate()" on page 494, "streaming.transport()" on page 569

http.method=

Tests HTTP request methods against any of a common set of HTTP methods. The condition matches to true if the request method matches any of the methods specified, and to NULL if the request is not an HTTP protocol request.

Syntax

`http.method=`*`method`*

where:

- *`method`*: One of the following methods:

  GET, CONNECT, DELETE, HEAD, POST, PUT, TRACE, OPTIONS, TUNNEL, LINK, UNLINK , PATCH, PROPFIND, PROPPATCH, MKCOL, COPY, MOVE, LOCK, UNLOCK, MKDIR, INDEX, RMDIR, COPY, MOVE

Layer and Transaction Notes

- Layers: `<Cache>`, `<Exception>`, `<Proxy>`
- Transactions: HTTP proxy, SSL-terminated HTTPS transactions

Example

Bypass authentication for POST requests.

```
<Proxy>
  http.method=POST authenticate(no) allow
```

See Also

- Conditions: "admin.access=" on page 69,"ftp.method=" on page 139, "http.method.custom=" on the next page, "http.method.regex=" on page 157, "socks.method=" on page 279
- Properties: "http.request.version()" on page 475, "http.response.version()" on page 479

http.method.custom=

Tests the HTTP protocol method versus custom values.

Syntax

`http.method.custom=`*`string`*

where:

- *`string`*: A custom value.

Layer and Transaction Notes

- Layers: `<Cache>`, `<Exception>`, `<Proxy>`
- Transactions: HTTP proxy, SSL-terminated HTTPS transactions

Example

Permit GET, PUT, and the specified custom method. Deny all other HTTP methods.

```
<Proxy>
  ALLOW http.method=(GET||POST)
  ALLOW http.method.custom=MYMETHOD1
  DENY
```

See Also

- Conditions: "http.method=" on the previous page, "http.method.regex=" on the facing page

http.method.regex=

Test the HTTP method using a regular expression.

Syntax

`http.method.regex=`*`regular_expression`*

where:

- *`string`*: A custom value.

Layer and Transaction Notes

- Layers: `<Cache>`, `<Exception>`, `<Proxy>`
- Transactions: HTTP proxy, SSL-terminated HTTPS transactions

Example

Deny any HTTP method that contains a decimal number.

```
<Proxy>
  DENY http.method.regex="[0-9]+"
```

See Also

- Conditions: "http.method=" on page 155, "http.method.custom=" on the previous page

http.request.apparent_data_type=

Used to test HTTP POST body data against apparent type.

This condition allows you to control the types of files submitted during an HTTP POST. Unlike MIME or file extension-based policies, the appliance determines apparent data type by examining the first few bytes of the HTTP POST request body data.

> **Note:** Recommended for Reverse Proxy deployments, where files are uploaded through the appliance to a back-end server.

Syntax

`http.request.apparent_data_type=`*`data_type`*

where:

- *`data_type`*: One of the data types in the following table; specify using either the label or the file extension:

*Added in 7.2.4.1.

Label	Description	Common Extensions
7ZIP*	7-Zip archive	.7z
ACE*	ACE archive	.ace
ARJ*	ARJ archive	.arj
BMP	BMP image	.bmp
BZ2	BZip2 archive	.bz2, .tbz2
CAB	MS Cab archive	.cab
COMPRESS*	compress compressed file	.Z (different from .z)
CPIO*	cpio archive	.cpio
DAA*	Direct Access Archive	.daa
EGG*	.EGG archive	.egg
EML*	raw email	.eml, .mht, .mhtml
EXE	MS application	.exe
FLASH	Adobe Flash	.swf, flv
GIF	GIF image	.gif
GZIP	GZIP compressed file	.gz
HTML	HTML file	.html
ICC	ICC profile	.icc

Label	Description	Common Extensions
JPG	JPEG image	.jpg
LHA*	LHA archive	.lha, .lzh
LZIP*	Lzip compressed file	.lz
MACH-O*	macOS application or library	
MSDOC	Microsoft document	.doc, .docx, .xls, .xlsx, .ppt, .pptx, .msi, .vba
MRAR	multi-part RAR	.partX.rar
MZIP	multi-part ZIP	.zip.xxx
PDF	Portable Document Format file	.pdf
PNG	PNG image	.png
RAR	RAR archive	.rar
RTF	Rich text document	.rtf
TAR	TAR archive	.tar
TIF	TIFF image	.tif
TNEF*	file encoded in Microsoft Transport-Neutral Encapsulation Format	.dat, .tnef
TTF	True-Type font	.ttf
TXT	plain text	.txt
UUE*	file encoded with uuencode or xxencode	.uu, .uue, .xx, .xxe
XAR*	Extensible Archive Format	.mpkg, .pkg, .xar
XML	XML file	.xml
XZ*		.xz
ZIP	ZIP archive	.zip

Layer and Transaction Notes

- Layers: `<Proxy>`
- Transactions: HTTP and decrypted HTTPS POST requests

Example

Allow users to upload images to the website, but only in JPG format.

```
<Proxy>
  ALLOW http.request.apparent_data_type=JPG
  DENY
```

See Also

- Conditions: "http.response.apparent_data_type=" on page 174, "request.icap.apparent_data_type=" on page 235, "response.icap.apparent_data_type=" on page 249
- Properties: "http.request.apparent_data_type.allow()" on page 438, "http.request.apparent_data_type.deny()" on page 441

http.request[*attribute*]=

Tests up to the first 8k of the body for the specified argument names and values within HTTP requests located in the query string, post body (URL-encoded and Multipart-Form encoded formats), cookie (excluding Google Analytic cookies with names beginning with "__utm"); against a regular expression, string, or an integer (when using the `count` modifier).

The search occurs after the names and values are normalized. The normalization process performs decoding specified in the "http.request.normalization.default()" on page 470 property, but does not modify the content when it is sent upstream. The search process applies to body content in URL-encoded and Multipart-Form encoded formats; it does not apply to key value pairs in XML or JSON body content.

Syntax

`http.request[`*`attribute1,attribute2, …`*`][.`*`string_modifier`*`][.case_sensitive]=`*`pattern`*

where:

- *`attribute:`* Comma-separated list of the predefined content sources:

Name	Value
`name` - All argument names found in the URL query string, post body (URL-encoded and multipart-form encoded formats), or cookie.	`value` - All named and unnamed argument values found in URL query string, post body (URL-encoded and multipart-form encoded formats), or cookie.
`query_arg_name` - All argument names found in the URL query string.	`query_arg` - All named and unnamed argument values found in the URL query string.
`arg_name` - All argument names found in both the URL query string and the post body (URL-encoded and multipart-form encoded formats).	`arg` - All named and unnamed argument values found in both the URL query string and the post body (URL-encoded and multipart-form encoded formats).
`cookie_name` - All argument names found in all Cookie and Cookie2 headers.	`cookie` - All named and unnamed argument values found in all Cookie and Cookie2 headers.
`post_arg_name` - All argument names found in the post body (URL-encoded and Multipart-form encoded formats)	`post_arg` - All named and unnamed argument values found in the post body (URL-encoded and Multipart-form encoded formats).
`header_name` - All header names.	`header` - All header values.
`path` - Path of the URL. This attribute does not have `name=value` format.	

> **Tip:** In this case, square brackets `[]` do not denote optional parameters, but are literal text to be entered.

- *string_modifier*: A supported string modifier:
  - count - number of occurrences of the specified attribute
  - exact - match the string exactly
  - prefix - match the start of a string
  - regex - (default if no string modifier is specified) regular expression match; see "Regex Reference" on page 703
  - substring - match part of a string
  - suffix - match the end of a string

    If you do not specify a modifier, the condition tests for an exact pattern match.
- case_sensitive - Perform case-sensitive match. By default, the test is case-insensitive.
- *pattern*: Type-appropriate test expression, such as a quote-delimited string expression or a regular expression.

Layer and Transaction Notes

- Layers: <Proxy>
- Transactions: HTTP proxy

Example 1

Perform regex scan for case-insensitive pattern "bad" in any name or value in cookie values and cookie names.

```
<Proxy>
  http.request[cookie_name,cookie].regex="bad"
```

Example 2

Perform regex scan for "bad" in any name or value within any parameter; a URL encoded query string will match the rule: “http://mydomain.com/path?%42%41%44=value”

```
<Proxy>
  http.request[name,value].regex="bad"
```

Example 3

Reject HTTP requests with scores equal to or greater than 20.

```
<Proxy>
  http.request[arg].count=20.. deny
```

See Also

- Conditions:"http.request.data=" on page 167
- Properties: "http.request.detection.other()" on page 454, "http.request.normalization.default()" on page 470
- Definitions: "define application_protection_set" on page 632

http.request.body.inspection_size_exceeded=

Tests if any content in a HTTP request body has not been scanned by Web Application Firewall (WAF) content nature detection engines. When enabled, this condition applies only under the following circumstances:

- WAF engines are configured in policy and the request data is a type for which engines are enabled. For example, the request body includes SQL statements and the SQL injection engine is enabled.
- The total size of unchunked, uncompressed request body data exceeds the value specified in the http.request.body.inspection_size() property.

For information on WAF engines, refer to the *Web Application Firewall Solutions Guide*.

Syntax

```
http.request.body.inspection_size_exceeded={yes|no}
```

Layer and Transaction Notes

- Layers: <Cache>, <Proxy>
- Transactions: HTTP requests that have a request body

Example

Deny a request if the request body has data that WAF engines have not scanned.

```
<Proxy>
  http.request.body.inspection_size_exceeded=yes DENY
```

See Also

- Properties: "http.request.body.inspection_size()" on page 446

http.request.body.max_size_exceeded=

Used in conjunction with the property, "http.request.body.max_size()" on page 448, this condition is used only in <Exception> layers. It allows administrators to take actions to log when a request exceeds the maximum body size for an HTTP request.

Syntax

```
http.request.body.max_size_exceeded={yes|no}
```

Layer and Transaction Notes

- Layers: <Exception>
- Transactions: requests that match rules restricting the size of an HTTP request body using the property "http.request.body.max_size()" on page 448

Example

When the body of an HTTP request exceeds 5 MB, it will be denied and reported in the custom log, ‘large_request’:

```
<Proxy>
  http.request.body.max_size(5242880)

<Exception>
  http.request.body.max_size_exceeded=yes access_log[large_request](yes)
```

See Also

- Properties: "http.request.body.max_size()" on page 448

http.request.body.size=

Used to test HTTP requests that include body content of a specific size.

This condition allows you to control the size of HTTP request transactions based on the size of the HTTP body content (in bytes). Unlike "http.request.body.max_size()" on page 448, this condition does not enforce a maximum.

Syntax

`http.request.body.size={range|N}`

where:

- *range*: A range specified with a lower limit, an upper limit, or both:
  - `..upper_limit` - the highest number of bytes
  - `lower_limit..` - the lowest number of bytes
  - `lower_limit..upper_limit` - number of bytes within the specified range
- *N*: Exact number of bytes.

Layer and Transaction Notes

- Layers: `<Cache>`, `<Exception>`, `<Proxy>`
- Transactions: HTTP requests with body content

> **Note:** Evaluated after the origin content server (OCS) responds. To deny similar requests before the OCS responds, use "http.request.body.max_size()" on page 448 instead.

Example

When the body content size of an HTTP request transaction exceeds 10 MB, output a new entry to custom log 'large_request'.

```
<Proxy>
  http.request.body.size=10485760.. access_log[large_request](yes)
```

See Also

- Properties: "http.request.body.max_size()" on page 448

http.request.data=

Inspect the contents of the body of an HTTP request. Content filters can use up to 65536 bytes from the HTTP request body to match.

If you have enabled WAF engines in policy and policy also includes the "http.request.body.inspection_size()" on page 446 property, WAF engines use the greatest specified value for scanning. For information on WAF engines, refer to the *Web Application Firewall Solutions Guide*.

> **Note:** The content filter could also use the URL of an application to determine the application name by using the "request.application.name=" on page 215 condition.

Syntax

`http.request.data.N[.string_modifier][.case_sensitive]=criterion`

where:

- *N*: Exact number of HTTP request body bytes to inspect, from 1..65536.
- *string_modifier*: A supported string modifier:
  - `exact` - match the string exactly
  - `prefix` - match the start of a string
  - `regex` - (default if no string modifier is specified) regular expression match; see "Regex Reference" on page 703
  - `substring` - match part of a string
  - `suffix` - match the end of a string
- `case_sensitive` - Perform case-sensitive match.
- *criterion*: Type-appropriate test expression, such as a quote-delimited string expression or a regular expression.

Layer and Transaction Notes

- Layers: `<Cache>`, `<Proxy>`
- Transactions: HTTP proxy

Example

Deny the HTTP request if a "sql" string exists in the HTTP request body, which may help parse the content of the HTTP POST body.

```
<Proxy>
  http.request.data.65536.substring="sql" DENY
```

See Also

- Conditions: "request.application.name=" on page 215
- Properties: "http.request.body.inspection_size()" on page 446

http.request.detection.result.application_protection_set=

Allows you to define policy actions based on the results of WAF application protection content nature detection engine scanning decisions. When a WAF application protection scan rule results in a block or monitor result, you can use this condition to perform an action such as additional logging to manually identify the content of the request.

For information on WAF engines, refer to the *Web Application Firewall Solutions Guide*.

Syntax

```
http.request.detection.result.application_protection_set=[block|monitor]
```

Layer and Transaction Notes

- Layers: <Proxy>
- Transactions: all transactions that have already been processed by a WAF application protection scan rule

Example

When a WAF application protection set action results in a block or monitor result, log the full header and body.

```
define application_protection_set SecurityEngines
  engine=injection.sql
  engine=xss
end

<Proxy>
  http.request.detection.SecurityEngines (monitor)

<Proxy>
  http.request.detection.result.application_protection_set=(monitor||block) http.request.log_details
[header,body] (yes)
```

See Also

- Conditions: "http.request.detection.result.validation=" on the next page
- Properties: "http.request.log_details()" on page 464
- Definitions: "define application_protection_set" on page 632

http.request.detection.result.validation=

Allows you to define policy actions based on the results of WAF validation decisions. When a WAF validation rule results in a block or monitor result, you can use this condition to perform an action such as additional logging to manually identify the content of the request.

For information on WAF engines, refer to the *Web Application Firewall Solutions Guide*.

Syntax

```
http.request.detection.result.validation=[block|monitor]
```

Layer and Transaction Notes

- Layers: `<Proxy>`
- Transactions: all transactions that have already been processed by a WAF validation rule via "http.request.detection.other()" on page 454:
  - `http.request.detection.other.null_byte(monitor|block)`
  - `http.request.detection.other.invalid_encoding(monitor|block)`
  - `http.request.detection.other.invalid_form_data(monitor|block)`
  - `http.request.detection.other.invalid_json(monitor|block)`
  - `http.request.detection.other.multiple_encoding(monitor|block)`
  - `http.request.detection.other.multiple_header(monitor|block)`
  - `http.request.detection.other.threshold_exceeded(monitor|block)`

Example

When a WAF validation results in a block or monitor result, log the full header and body.

```
<Proxy>
  http.request.normalization.default(auto)

<Proxy>
  http.request.detection.other.invalid_form_data(monitor)

<Proxy>
  http.request.detection.result.validation=(monitor||block) http.request.log_details[header,body]
(yes)
```

See Also

- Conditions: "http.request.detection.result.application_protection_set=" on page 169
- Properties: "http.request.body.inspection_size()" on page 446, "http.request.log_details()" on page 464

http.request.version=

Tests the version of HTTP used by the client in making the request to the appliance.

Syntax

```
http.request.version={0.9|1.0|1.1|2}
```

Layer and Transaction Notes

- Layers: <Cache>, <Exception>, <Proxy>
- Transactions: HTTP and SSL-terminated HTTPS transactions

Example

Allow requests using HTTP/2.

```
<Proxy>
  ALLOW  http.request.version=2
```

See Also

- Conditions: "http.response.version=" on page 179

http.request_line.regex=

Test the HTTP protocol request line.

Syntax

`http.request_line.regex=`*`regular_expression`*

For details on regex matches, see "Regex Reference" on page 703

Layer and Transaction Notes

- Layers: `<Exception>`, `<Proxy>`
- Transactions: HTTP and SSL-terminated HTTPS transactions

Example

By default, the appliance allows the HTTP request line to contain leading and trailing white space. It allows tab characters to be used in place of space characters, and it allows multiple space characters to occur between tokens. But according to a strict interpretation of the HTTP specification, there cannot be leading or trailing white space, do use tabs, and only a single space can appear between tokens.

The following policy enforces these syntax restrictions.

```
<Proxy>
  DENY("bad HTTP request line") http.request_line.regex="\t|(^ )|( $)|(  )"
```

http.response.apparent_data_type=

Used to test HTTP requests that include data, matching on the apparent type of that content.

This condition allows you to control the types of files being requested by users after contacting the Origin Content Server (OCS), as identified by apparent data type. Unlike MIME or file extension-based policies, the appliance determines the apparent data type by examining the first few bytes of a response from an OCS.

Syntax

`http.response.apparent_data_type=`*`data_type`*

- *`data_type`*: One of the data types in the following table; specify using either the label or the file extension:

*Added in 7.2.4.1.

Label	Description	Common Extensions
7ZIP*	7-Zip archive	.7z
ACE*	ACE archive	.ace
ARJ*	ARJ archive	.arj
BMP	BMP image	.bmp
BZ2	BZip2 archive	.bz2, .tbz2
CAB	MS Cab archive	.cab
COMPRESS*	compress compressed file	.Z (different from .z)
CPIO*	cpio archive	.cpio
DAA*	Direct Access Archive	.daa
EGG*	.EGG archive	.egg
EML*	raw email	.eml, .mht, .mhtml
EXE	MS application	.exe
FLASH	Adobe Flash	.swf, flv
GIF	GIF image	.gif
GZIP	GZIP compressed file	.gz
HTML	HTML file	.html
ICC	ICC profile	.icc
JPG	JPEG image	.jpg
LHA*	LHA archive	.lha, .lzh
LZIP*	Lzip compressed file	.lz
MACH-O*	macOS application or library	

Label	Description	Common Extensions
MSDOC	Microsoft document	.doc, .docx, .xls, .xlsx, .ppt, .pptx, .msi, .vba
MRAR	multi-part RAR	.partX.rar
MZIP	multi-part ZIP	.zip.xxx
PDF	Portable Document Format file	.pdf
PNG	PNG image	.png
RAR	RAR archive	.rar
RTF	Rich text document	.rtf
TAR	TAR archive	.tar
TIF	TIFF image	.tif
TNEF*	file encoded in Microsoft Transport-Neutral Encapsulation Format	.dat, .tnef
TTF	True-Type font	.ttf
TXT	plain text	.txt
UUE*	file encoded with uuencode or xxencode	.uu, .uue, .xx, .xxe
XAR*	Extensible Archive Format	.mpkg, .pkg, .xar
XML	XML file	.xml
XZ*		.xz
ZIP	ZIP archive	.zip

Layer and Transaction Notes

- Layers: `<Cache>`, `<Proxy>`
- Transactions: HTTP and decrypted HTTPS transactions

Example

Deny the request if the first few bytes of the response data indicate that this is an Adobe Flash file.

```
<Proxy>
  deny http.response.apparent_data_type=FLASH
```

See Also

- Conditions: "http.request.apparent_data_type=" on page 158, "request.icap.apparent_data_type=" on page 235, "response.icap.apparent_data_type=" on page 249

http.response.code=

Tests true if the current transaction is an HTTP transaction and the response code received from the origin server is as specified.

Syntax

`http.response.code=`*`response_code`*

Layer and Transaction Notes

- Layers: `<Cache>`, `<Exception>`,`<Proxy>`
- Transactions: HTTP and SSL-terminated HTTPS transactions

Example

Disable attack detection when the proxy receives HTTP response 403 from the destination server.

```
<Proxy>
  http.response.code=403 attack_detection.failure_weight(0)
```

See Also

- Conditions: "http.request.version=" on page 172,"http.response.version=" on page 179
- Properties: "http.response.version=" on page 179

http.response.data=

Test the first few bytes of HTTP response data.

This trigger causes HTTP to wait until N bytes of response data have been received from the origin server (or the end of file, whichever comes first). Then, the first N bytes of response data are compared to the string pattern on the right side of the condition.

Syntax

`http.response.data.`*`N`*`[.`*`string_modifier`*`]=`*`pattern`*

where:

- *`N`*: Number of bytes, from 1..256
- *`string_modifier`*: A supported string modifier:
  - `exact` - match the string exactly
  - `prefix` - match the start of a string
  - `regex` - (default if no string modifier is specified) regular expression match; see "Regex Reference" on page 703
  - `hex` - hex signature match
  - `substring` - match part of a string
  - `suffix` - match the end of a string
- *`pattern`*: Type-appropriate test expression, such as a quote-delimited string expression or a regular expression.

> **Note:** As .regex values in policy can be costly in terms of system resources, Symantec recommends using .hex whenever possible.
> In a .hex value, "\" introduces two hex characters. To include a literal "\" in the hex value, use "\\" (two backslash characters). For information on how to identify the hex value for a given file extension, refer to http://www.garykessler.net/library/file_sigs.html.

Layer and Transaction Notes

- Layers: `<Cache>`, `<Exception>`, `<Proxy>`
- Transactions: HTTP proxy

Example 1

Deny the request if the first 2 bytes of the response data indicate that this is probably a Windows executable file.

```
<Proxy>
  DENY http.response.data.2.case_sensitive="MZ"
```

Example 2

Examine the first 16 bytes of each response and look for the hex value that pertains to a PNG file,and deny it.

```
<Proxy>
  http.response.data.16.hex="\89PNG\0D\0A\1A\0A\00\00\00\0DIHDR"
```

http.response.version=

Tests the version of HTTP used by the origin content server to deliver the response to the appliance.

Syntax

```
http.response.version={0.9|1.0|1.1|2}
```

Layer and Transaction Notes

- Layers: <Cache>, <Exception>, <Proxy>
- Transactions: HTTP and SSL-terminated HTTPS transactions

Example

Allow responses using HTTP/2.

```
<Proxy>
  ALLOW http.response.version=2
```

See Also

- Conditions: "http.response.code=" on page 176, "http.request.version=" on page 172

http.transparent_authentication=

Tests whether HTTP uses transparent proxy authentication for the current request.

Use with the "authenticate()" on page 348 or "authenticate.force()" on page 354 property.

Syntax

```
http.transparent_authentication={yes|no}
```

Layer and Transaction Notes

- Layers:<Exception>, <Proxy>
- Transactions: HTTP and SSL-terminated HTTPS transactions

See Also

- Conditions: "attribute.name=" on page 75, "authenticated=" on page 81, "group=" on page 140, "has_attribute.name=" on page 142, "realm=" on page 208, "user=" on page 312, "user.domain=" on page 316
- Properties: "authenticate()" on page 348, "authenticate.force()" on page 354, "authenticate.mode()" on page 360, "check_authorization()" on page 376

http.websocket=

The WebSocket protocol provides simultaneous two-way communications channels over a single TCP connection by detecting the presence of a proxy server and tunneling communications through the proxy.

To upgrade an HTTP connection to a newer HTTP version or use another protocol such as WebSocket, a client sends a request with Upgrade, Connection, and other relevant headers.

Syntax

```
http.websocket={yes|no}
```

Layer and Transaction Notes

- Layers: `<Exception>`, `<Forward>`, `<Proxy>`
- Transactions: HTTP and HTTPS

Example

Authenticate using the specified realm for WebSocket transactions.

```
<Proxy>
  http.websocket=yes authenticate(realm1)
```

See Also

- Conditions: "client.protocol=" on page 111

icap_*method*.header.*header_name*=

Inspect ICAP response headers to make policy decisions based on their contents.

Because external services are not supported in MACH5, this feature is not useful in MACH5 deployments.

Syntax

`icap_*method*.header.*header_name*[.*string_modifier*][.case_sensitive]=*pattern*`

where:

- *string_modifier*: A supported string modifier:
  - `as_number`- a positive integer, or range of numbers, representing the header value converted to an unsigned 32-bit integer
  - `exact` - match the string exactly
  - `length` - an integer between 0..8192 representing the length of the header value
  - `prefix` - match the start of a string
  - `regex` - (default if no string modifier is specified) regular expression match; see "Regex Reference" on page 703
  - `substring` - match part of a string
  - `suffix` - match the end of a string
- `case_sensitive` - Perform case-sensitive match, when applicable.
- *pattern*: Type-appropriate test expression, such as a quote-delimited string expression or a regular expression.

> **Note:** Because non-standard headers are common, policy written using string modifiers could end up testing headers that contain strings. In these cases, strings are ignored and the policy takes no effect.

Layer and Transaction Notes

- Layers: `<Cache>`, `<Exception>`, `<Proxy>`
- Transactions: HTTP proxy

Example

The following policy denies the uploading and downloading of executable (EXE) files. Through an ICAP scan, ProxyAV can determine the apparent data type of the object and return this information in an ICAP header.

```
<Proxy>
  request.icap_service(my_icap_reqmod)

<Cache>
  response.icap_service(my_icap_respmod)

<Proxy>
  DENY icap_reqmod.header.X-Apparent-Data-Types="EXE"
  DENY icap_respmod.header.X-Apparent-Data-Types="EXE"
```

is_healthy.*health_check*=

Tests whether a health check is reporting as healthy.

Syntax

`is_healthy.*health_check*={yes|no}`

where:

- *health_check*: Name of a health check.

> **Tip:** When specifying a user defined health check, the prefix `user.` is optional, and will be added automatically. In all other cases the complete health check name, including its prefix, must be entered.

Layer and Transaction Notes

- Layers: <Cache>, <Forward>, <Proxy> , <SSL>
- Transactions: all

Example

Consider a user defined health check user.upstream set up to test access to the Internet while using a forwarding proxy named internet_main. This could be a composite health check testing several Internet sites. In this example, when Internet access fails then all traffic is directed to use a different forwarding proxy called internet_backup.

The health check must continue to evaluate the Internet access using internet_main. This requires the health check transaction be identified and consistently routed. Otherwise, the health check oscillates between using the two proxies.

```
<Forward>
  ; Normally forward through 'internet_main'.
  is_healthy.user.upstream=yes forward(internet_main)
  ; Health check must use 'internet_main'.
  health_check=user.upstream forward(internet_main)
  ; Otherwise, use the backup.
  forward(internet_backup)
```

See Also

- Conditions: "health_check=" on page 146

is_set.variable.*name*=

Tests if the specified variable has been set in another layer, including a layer of a different type. When a variable is set, it overrides the default value. See " Variables Syntax and Usage" on page 671 for information on setting variables.

Syntax

```
is_set.variable.name={yes|no}
```

Layer and Transaction Notes

- Layers: <Cache>, <Diagnostic>, <Exception>, <Forward>, <Proxy> , <SSL>, <SSL-Intercept>
- Transactions: HTTP proxy

Example

The `url.threat_risk.effective_level` variable is set in the <Proxy> layer. In this example, the condition in the <Cache> layer would evaluate to true and responses from websites in the `risky_sites` condition are scanned with the specified ICAP service. If the ICAP service is not healthy, the transaction is denied.

```
define condition risky_sites
  server_url.domain=external_site1 server_url.extension=(doc,html)
  server_url.domain=external_site2 response.header.Content-Type=(application/pdf)
end

<Proxy>
    condition=risky_sites variable.url.threat_risk.effective_level(9)

<Cache>
  is_set.variable.url.threat_risk.effective_level=yes response.icap_service(ICAP_service, failed_
closed)
```

iterator=

When referenced inside of an "iterate()" on page 603 block, this condition compares the text string passed against the current string value being iterated over.

Syntax

`iterator[.`*`string_modifier`*`][.case_sensitive]=`*`pattern`*

where:

- *`string_modifier`*: A supported string modifier:
  - `exact` - match the string exactly
  - `prefix` - match the start of a string
  - `regex` - (default if no string modifier is specified) regular expression match; see "Regex Reference" on page 703
  - `substring` - match part of a string
  - `suffix` - match the end of a string
- `case_sensitive` - Perform case-sensitive match, when applicable.
- *`pattern`*: Type-appropriate test expression, such as a quote-delimited string expression or a regular expression.

Layer and Transaction Notes

- Use only inside of an "iterate()" on page 603 block.

Example

Delete occurrences of "Sample" prefix in the Cookie request header.

```
define action DeleteSampleCookies
  iterate(request.header.Cookie)
    iterator.prefix="Sample" iterator.delete()
  end
end

<Proxy>
  action.DeleteSampleCookies(yes)
```

See Also

- Action: "iterate()" on page 603

ldap.attribute.ldap_*attribute*=

Compares strings with the value of the LDAP attribute obtained from the user's entry.

Syntax

`ldap.attribute.ldap_`*`attribute`*`[.`*`string_modifier`*`][.case_sensitive]=`*`pattern`*

where:

- *`attribute`*: LDAP attribute name.
- *`string_modifier`*: A supported string modifier:
  - `exact` - match the string exactly
  - `prefix` - match the start of a string
  - `regex` - (default if no string modifier is specified) regular expression match; see "Regex Reference" on page 703
  - `substring` - match part of a string
  - `suffix` - match the end of a string
- `case_sensitive` - Perform case-sensitive match, when applicable.
- *`pattern`*: Type-appropriate test expression, such as a quote-delimited string expression or a regular expression.

Layer and Transaction Notes

- Layers: `<Admin>`, `<Exception>`, `<Forward>`, `<Proxy>` , `<SSL>`, `<SSL-Intercept>`
- Transactions: all

Example

Deny user access to the proxy under certain conditions.

```
<Proxy>
  authenticate(LDAPRealm)

<Proxy>
  DENY ldap.attribute.user_type.suffix="_restricted"
  DENY category=gambling ldap.attribute.web_permission.regex=!".*gambling.*"
  DENY ldap.attribute.proxy_user="John.Smith"
  ALLOW
```

See Also

- Conditions: "has_attribute.name=" on page 142, "ldap.attribute.ldap_attribute.as_number=" on the facing page, "ldap.attribute.ldap_attribute.count=" on page 190, "ldap.attribute.ldap_attribute.exists=" on page 191

ldap.attribute.ldap_*attribute*.as_number=

Converts the value of the attribute to an unsigned 32-bit integer, and then allows numerical tests to be done.

Syntax

`ldap.attribute.ldap_`*`attribute`*`.as_number=`*`integer`*

where:

- *`attribute`*: LDAP attribute name.
- *`integer`*: A single integer.

Layer and Transaction Notes

- Layers: `<Admin>`, `<Exception>`, `<Forward>`, `<Proxy>` , `<SSL>`, `<SSL-Intercept>`
- Transactions: all

Example

Allow users with high priority only to access the proxy.

```
<Proxy>
  authenticate(LDAPRealm)

<Proxy>
  ALLOW ldap.attribute.UserPriority.as_number=0
DENY
```

See Also

- Conditions: "has_attribute.name=" on page 142, "ldap.attribute.ldap_attribute=" on page 187, "ldap.attribute.ldap_attribute.count=" on the next page, "ldap.attribute.ldap_attribute.exists=" on page 191

ldap.attribute.ldap_*attribute*.count=

Tests the number of values in a list for the named attribute.

Syntax

`ldap.attribute.ldap_`*`attribute`*`.count=`*`integer`*

where:

- *`attribute`*: LDAP attribute name.
- *`integer`*: A single integer.

Layer and Transaction Notes

- Layers: `<Admin>`, `<Exception>`, `<Forward>`, `<Proxy>` , `<SSL>`, `<SSL-Intercept>`
- Transactions: all

Example

Deny access to a restricted host for users that have warnings in their profile.

```
<Proxy>
  authenticate(LDAPRealm)

<Proxy>
  DENY url.hostname=rewards.MyCompanyName.com ldap.attribute.Warnings.count=!0
  ALLOW
```

See Also

- Conditions: "has_attribute.name=" on page 142, "ldap.attribute.ldap_attribute=" on page 187, "ldap.attribute.ldap_attribute.as_number=" on the previous page, "ldap.attribute.ldap_attribute.exists=" on the facing page

ldap.attribute.ldap_*attribute*.exists=

Checks if the named attribute exists in the user's entry.

Syntax

`ldap.attribute.ldap_attribute.exists={yes|no}`

Layer and Transaction Notes

- Layers: `<Admin>`, `<Exception>`, `<Forward>`, `<Proxy>` , `<SSL>`, `<SSL-Intercept>`
- Transactions: all

Example

Allow users to access the proxy if they have the LDAP attribute ProxyUser. The attribute could have any value, even null.

```
<Proxy>
  authenticate(LDAPRealm)

<Proxy>
  ALLOW ldap.attribute.ProxyUser.exists=yes
  DENY
```

See Also

- Conditions: "has_attribute.name=" on page 142, "ldap.attribute.ldap_attribute=" on page 187, "ldap.attribute.ldap_attribute.as_number=" on page 189, "ldap.attribute.ldap_attribute.count=" on the previous page

live=

Tests if the streaming content is a live stream.

Syntax

```
live={yes|no}
```

Layer and Transaction Notes

- Layers: <Cache>, <Proxy>
- Transactions: streaming transactions

Example

Restrict access to live streams during morning hours.

```
<Proxy>
  deny live=yes time=1200..0800
```

See Also

- Conditions: "bitrate=" on page 83, "streaming.client=" on page 283, "streaming.content=" on page 284
- Properties: "access_server()" on page 335, "max_bitrate()" on page 494, "streaming.transport()" on page 569

minute=

Tests if the minute of the hour is in the specified range or an exact match. By default, the system clock and time zone are used to determine the current minute.

To specify the UTC time zone, use the form minute.utc=. The numeric pattern used to test the minute condition can contain no whitespace.

Syntax

`minute[.utc]={[range]|exact_minute}`

where:

- *range*: A range specified with the earliest minute, the latest minute, or both:
  - *first_minute* - Integer from 0 to 59, indicating the first minute of the hour that tests true. If left blank, minute 0 is assumed.
  - *last_minute* - Integer from 0 to 59, indicating the last minute of the hour that tests true. If left blank, minute 59 is assumed.
  - *first_minute..last_minute* - Number of minutes within the specified range.
- *exact_minute*: Integer from 0 to 59, indicating the minute of each hour that tests true.

> **Note:** To test against an inverted range, such as a range that crosses from one hour into the next, the following shorthand expression is available. While `minute=(..14|44..)` specifies the first 15 minutes and last 15 minutes of each hour, the policy language also recognizes `minute=44..14` as equivalent.

Layer and Transaction Notes

- Layers: `<Admin>`, `<Cache>`, `<Diagnostic>`, `<DNS-Proxy>`, `<Exception>`, `<Forward>`, `<Proxy>` , `<SSL>`, `<SSL-Intercept>`
- Transactions: all

> **Note:** Using time-related conditions to control caching behavior in a `<Cache>` layer may cause thrashing of the cached objects.

Example

Test for the first 5 minutes of every hour.

```
<Proxy>
  minute=0..5
```

See Also

- Conditions: "date=" on page 118, "day=" on page 119, "hour=" on page 147, "month=" on the facing page, "time=" on page 292, "weekday=" on page 328, "year=" on page 330

month=

Tests if the month is in the specified range or an exact match. By default, the system date and time zone are used to determine the current month. To specify the UTC time zone, use the form month.utc=. The numeric pattern used to test the month condition can contain no whitespace.

Syntax

`month[.utc]={[first_month]..[last_month]|exact_month}`

where:

- *first_month*: Integer from 1 to 12, where 1 specifies January and 12 specifies December, specifying the first month that tests true. If left blank, January (month 1) is assumed.
- *last_month*: Integer from 1 to 12, where 1 specifies January and 12 specifies December, specifying the first month that tests true. If left blank, December (month 12) is assumed.
- *exact_month*: Integer from 0 to 12, indicating the month that tests true.

> **Note:** To test against an inverted range, such as a range that crosses from one year into the next, the following shorthand expression is available. While `month=(..6|9..)` specifies September through June, the policy language also recognizes `month=9..6` as equivalent.

Layer and Transaction Notes

- Layers: `<Admin>`, `<Cache>`, `<Diagnostic>`, `<DNS-Proxy>`, `<Exception>`, `<Forward>`, `<Proxy>` , `<SSL>`, `<SSL-Intercept>`
- Transactions: all

> **Note:** Using time-related conditions to control caching behavior in a `<Cache>` layer may cause thrashing of the cached objects.

Example

Test for a holiday season.

```
define condition year_end_holidays
  month=12 day=25..
  month=1 day=1
end_condition year_end_holidays
```

See Also

- Conditions: "date=" on page 118, "day=" on page 119, "hour=" on page 147, "minute=" on page 193, "time=" on page 292, "weekday=" on page 328, "year=" on page 330

p2p.client=

Test the type of peer-to-peer client in use.

Syntax

```
p2p.client={yes|no|bittorrent|edonkey|fasttrack|gnutella}
```

Layer and Transaction Notes

- Layers: <Exception>, <Forward>, <Proxy>
- Transactions: all proxies

Example

Test for Gnutella client.

```
<Proxy>
  p2p.client=gnutella
```

proxy.address=

Tests the destination address of the incoming IP packet.

If the transaction was explicitly proxied, this condition tests the IP address the client used to reach the proxy. The destination IP address is either the IP address of the NIC on which the request arrived or a virtual IP address. This is intended for situations where the proxy has a range of virtual IP addresses.

If the transaction was transparently proxied, this condition tests the destination address contained in the IP packet.

> **Note:** This test might not be equivalent to testing server_url.address=. The server_url.address= and "proxy.address=" above conditions test different addresses in the case where a proxied request is transparently intercepted: server_url.address= contains the address of the origin server, and "proxy.address=" above contains the address of the upstream proxy through which the request is to be handled.

> **Note:** "client.interface=" on page 109 functions correctly for transparent transactions.

Syntax

`proxy.address={ip_address|ip_address_range|ip_address_wildcards|subnet_label}`

where:

- *ip_address*: Client IP address or subnet specification; for example, 10.25.198.0/24
- *ip_address_range*: IP address range; for example, 192.0.2.0-192.0.2.255
- *ip_address_wildcards*: IP address specified using wildcards in any octet(s); for example, 10.25.*.0 or 10.*.*.0
- *subnet_label*: Label of a "define subnet" on page 656 block that binds a number of IP addresses or subnets

Layer and Transaction Notes

- Layers: <Admin>, <DNS-Proxy>, <Exception>, <Forward>, <Proxy> , <SSL>, <SSL-Intercept>, <Tenant>
- Transactions: HTTP proxy

Example

Deny service through proxy within the subnet 1.2.3.x.

```
<Proxy>
  proxy.address=1.2.3.0/24 deny
```

See Also

- Conditions: "client.address=" on page 87, "client.interface=" on page 109, "client.protocol=" on page 111, "proxy.port=" on the next page
- Definitions: "define subnet" on page 656
- Information on wildcards: http://www.symantec.com/docs/TECH241521
- Information on IP address ranges: http://www.symantec.com/docs/TECH241929

proxy.port=

Tests if the IP port used by a request is within the specified range or an exact match. The numeric pattern used to test the `proxy.port=` condition cannot contain whitespace.

If the transaction was explicitly proxied, this tests the IP port that the client used to reach the proxy.

If the transaction was transparently proxied, `proxy.port=` tests which port the client thinks it is connecting to on the upstream proxy device or origin content server (OCS). If the client thinks it is connecting directly to the OCS, but the transaction is transparently proxied and the port number the client specified by the client in the request URL is consistent and not falsified, then `proxy.port=` and server_url.port= test the same value.

> **Note:** Because the appliance default configuration passes through tunneled traffic, some changes must be made to begin transparent port monitoring. Only proxy ports that have been configured and enabled can be tested using the `proxy.port=` condition. For example, if the transparent FTP service on port 21 is either not configured or disabled, a policy rule that includes `proxy.port=21` has no effect.

Syntax

`proxy.port={`*`range`*`|`*`port_number`*`}`

where:

- *range*: Range of port numbers to be tested. Specify the lowest and highest values of the range, separated by (..). The values must be integers from 1 through 65535. For example, a valid port range is 8080..8082.
- *port_number*: Single port number; for example, 80. Can be a number between 1 and 65535.

Layer and Transaction Notes

- Layers: `<Admin>`, `<DNS-Proxy>`, `<Exception>`, `<Forward>`, `<Proxy>` , `<SSL>`, `<SSL-Intercept>`. `<Tenant>`
- Transactions: all proxies

Example

Deny URL through the range of port numbers.

```
<Proxy>
  url=http://www.example.com proxy.port=8080..8082 deny
```

See Also

- Conditions: "client.address=" on page 87, "client.interface=" on page 109,"client.protocol=" on page 111, "proxy.address=" on page 198, server_url.port=

random=

Specify the number of samples to randomly test. Use in conjunction with other conditions.

Syntax

`random=n`

where:

- *n*: The condition evaluates to true approximately 1/*n* times for the policy rule.

Layer and Transaction Notes

- Layers: `<Diagnostic>`
- Transactions: all

Example

Trace a random 1% sample of matching requests to piratebay.org.

```
<Diagnostic> trace.request(yes)
  random=100 url.domain=//piratebay.org/
```

raw_url.host.regex=

Test the value of the host component of the raw request URL.

The raw_url.host= condition is the original character string used to specify the host in the HTTP request. It is different from the url.host= string because the following normalizations are not applied:

- Conversion to lower case. For example, "WWW.SomeDomain.COM" -> "www.somedomain.com".
- Trailing dot is stripped from domain name. For example, "www.example.com." -> "www.example.com".
- IP addresses in non-standard form are converted to a decimal dotted quad. For example, "0xA.012.2570" -> "10.10.10.10".

For information, see "Regex Reference" on page 703.

Syntax

raw_url.host.regex=*regular_expression*

Layer and Transaction Notes

- Layers: <Cache>, <Exception>, <Proxy>
- Transactions: all proxies

Example

Reject request as invalid if host is an IP address in non-standard form.

```
<Proxy>
  exception(invalid_request) url.host.is_numeric=yes raw_url.host.regex=("(^|\.)0[^.]" ||
!"\..*\..*\.")
```

raw_url.path.regex=

Test the value of the path component of the raw request URL.

The `raw_url.path.regex=` condition tests the original character string used to specify the path in the HTTP request. It is different from the url.path.regex= condition because the following normalizations are not applied:

- If path and query are both missing, the path is set to "/". For example, "http://abc.com" -> "http://abc.com/".
- Double slashes in the path are normalized to single slashes. For example, "http://abc.com/a//b.gif" -> "http://abc.com/a/b.gif".
- The path components "." and ".." are removed. For example, "http://abc.com/a/./b.gif" -> "http://abc.com/a/b.gif" and "http://abc.com/a/../b.gif" -> "http://abc.com/b.gif".
- Unnecessary % escape sequences are replaced by the characters they encode. For example, "http://abc.com/%64%65%66.gif" -> "http://abc.com/def.gif".

For information, see "Regex Reference" on page 703.

Syntax

`raw_url.path.regex=`*`regular_expression`*

Layer and Transaction Notes

- Layers: `<Cache>`, `<Exception>`, `<Proxy>`
- Transactions: all proxies

Example

Reject request as invalid if path encodes letters and digits using hex escape sequences, as this might be an attempt to evade content filtering policy.

```
<Proxy>
  exception(invalid_request) raw_url.path.regex="\%(3[0-9]|[46][1-9a-fA-F]|[57][0-9aA])"
```

raw_url.pathquery.regex=

Test the value of the path and query component of the raw request URL.

The `raw_url.pathquery.regex=` condition tests the original character string used to specify the path and query in the HTTP request. It is different from the path and query tested by the url.regex= condition because the following normalizations are not applied:

- If path and query are both missing, the path is set to "/". For example, "http://abc.com" -> "http://abc.com/".
- Double slashes in the path are normalized to single slashes. For example, "http://abc.com/a//b.gif" -> "http://abc.com/a/b.gif".
- The path components "." and ".." are removed. For example, "http://abc.com/a/./b.gif" -> "http://abc.com/a/b.gif" and "http://abc.com/a/../b.gif" -> "http://abc.com/b.gif".
- Unnecessary % escape sequences are replaced by the characters they encode. For example, "http://abc.com/%64%65%66.gif" -> "http://abc.com/def.gif".

For information, see "Regex Reference" on page 703.

Syntax

`raw_url.pathquery.regex=`*`regular_expression`*

Layer and Transaction Notes

- Layers: `<Cache>`, `<Exception>`, `<Proxy>`
- Transactions: all proxies

Example

Reject request as invalid if pathquery encodes letters and digits using hex escape sequences, as this might be an attempt to evade content filtering policy.

```
<Proxy>
  exception(invalid_request) raw_url.pathquery.regex="\%(3[0-9]|[46][1-9a-fA-F]|[57][0-9aA])"
```

raw_url.query.regex=

Tests the original character string used to specify the query in the HTTP request. It is different from url.query.regex= because the following normalization is not applied:

Unnecessary % escape sequences are replaced by the characters they encode, such as "http://abc.com/search?q=%64%65%66" -> "http://abc.com/search?q=def".

Syntax

`raw_url.query.regex=`*`regular_expression`*

Layer and Transaction Notes

- Layers: `<Admin>`, `<Cache>`, `<DNS-Proxy>`, `<Exception>`, `<Forward>`, `<Proxy>` , `<SSL>`, `<SSL-Intercept>`
- Transactions: HTTP proxy

Example

Reject request as invalid if query encodes letters and digits using hex escape sequences, as this might be an attempt to evade content filtering policy.

```
<Proxy>
  exception(invalid_request) raw_url.query.regex="\%(3[0-9]|[46][1-9a-fA-F]|[57][0-9aA])"
```

raw_url.regex=

Test the value of the raw request URL.

The raw_url.regex= condition is the request URL without any normalizations applied. The ProxySG appliance normalizes URLs in order to better enforce policy. However, there are instances where testing the raw form is desirable, such as using CPL to detect that a URL contained the signature of an exploit that was removed during normalization.

For information, see "Regex Reference" on page 703.

Syntax

raw_url.regex=*regular_expression*

Layer and Transaction Notes

- Layers: <Cache>, <Exception>, <Proxy>
- Transactions: all proxies

Example

Reject request as invalid if URL encodes letters and digits using hex escape sequences, as this might be an attempt to evade content filtering policy.

```
<Proxy>
  exception(invalid_request) raw_url.regex="\%(3[0-9]|[46][1-9a-fA-F]|[57][0-9aA])"
```

realm=

Tests if the client is authenticated and if the client has logged into the specified realm. If both of these conditions are met, the response is true. In addition, the "group=" on page 140 condition can be used to test whether the user belongs to the specified group. This condition is unavailable if the current transaction is not authenticated (for example, "authenticate()" on page 348 is set to no).

If you reference more than one realm in your policy, consider disambiguating user, group and attribute tests by combining them with a realm=test. This reduces the number of extraneous queries to authentication services for group, user or attribute information that does not pertain to that realm.

Syntax

`realm=`*`authentication_realm`*

Layer and Transaction Notes

- Layers: `<Admin>`, `<Exception>`, `<Forward>`, `<Proxy>` , `<SSL>`, `<SSL-Intercept>`
- Transactions: all proxies, administrator

> **Note:** When used in the `<forward>` layer, this condition can evaluate to NULL (shown in a trace as N/A) if no authenticated client exists. Rules containing these conditions can be guarded by authenticated= to preserve normal logic.

Example 1

Tests if the user has logged into realm corp and is authenticated in the specified group.

```
<Proxy>
  realm=corp group=all_staff
```

Example 2

Distinguish the policy applied to two groups of users--corp's employees, and their corporate partners and clients. These two groups will authenticate in different realms.

```
<Proxy>
  client.address=10.10.10/24 authenticate(corp) ; The corporate realm
  authenticate(client) ; Company partners & clients
<Proxy> realm=corp ; Rules for corp employees
  allow url.domain=corp.com ; Unrestricted internal access
  category=(violence, gambling) exception(content_filter_denied)
<Proxy> realm=client ; Rules for business partners & clients
```

```
  allow group=partners url=corp.com/partners ; Restricted to partners
  allow group=(partners, clients) url=corp.com/clients ; Both groups allowed
  deny
; Additional layers would continue to be guarded with the realm, so that only
; the ‘client’ realm would be queried about the ‘partners’ and ‘clients’ groups.
```

See Also

- Conditions: "authenticated=" on page 81, "group=" on page 140, "has_attribute.name=" on page 142, "http.transparent_authentication=" on page 180, "user=" on page 312, "user.domain=" on page 316, "user.x509.issuer=" on page 322, "user.x509.serialNumber=" on page 323, "user.x509.subject=" on page 325
- Properties: "authenticate()" on page 348, "authenticate.force()" on page 354, "check_authorization()" on page 376

release.id=

Tests the software release ID of the appliance.

Syntax

`release.id=`*`release_id`*

Layer and Transaction Notes

- Layers: `<Admin>`, `<Cache>`, `<Diagnostic>`, `<DNS-Proxy>`, `<Exception>`, `<Forward>`, `<Proxy>` , `<SSL>`, `<SSL-Intercept>`

Example

True if you are running a version of SGOS whose release ID is 231461 or later.

```
<Proxy>
  release.id=231461..
```

See Also

- Conditions: "release.version=" on the facing page

release.version=

Tests the software release ID of the appliance.

Syntax

`release.version=`*`release_version`*

where:

- *`release_version`*: Release version, in the form:

  `major_#.minor_#.dot_#.patch_#`

  Each number must be in the range 0 to 255. The *`major_#`* is required; less significant portions of the version might be omitted and default to 0.

Layer and Transaction Notes

- Layers: `<Admin>`, `<Cache>`, `<Diagnostic>`, `<DNS-Proxy>`, `<Exception>`, `<Forward>`, `<Proxy>` , `<SSL>`, `<SSL-Intercept>`

Example

True if you are running SGOS 7.1 or later.

```
<Proxy>
  release.version=7.1..
```

See Also

- Conditions: "release.id=" on the previous page

request.application.*attribute*=

Tests the specified web application attributes for a URL. You must have a valid CASB Audit subscription (included in an Intelligence Services bundle) and enable the Application Classification and Application Attributes services to receive the CASB AppFeed with these attributes.

> **Note:** When writing policy rules based on Business Readiness Rating (BRR) note that the CASB AppFeed applies its own Default BRR; it does not apply tenants' BRR modified from Symantec CloudSOC. TECH247736 (http://www.symantec.com/docs/TECH247736) describes this behavior.

Possible Values of Attributes

When writing policy that uses application attributes, you can ensure that the CPL parameters are valid by identifying an application attribute's possible values. Use the following command:

```
#(config application-attributes)view possible-values attribute
```

For example, to determine the possible values for the ISO 27001 attribute, issue the command:

```
#(config application-attributes)view possible-values iso_27001
True/False
```

Based on this output, you can include a condition such as the following in policy:

```
request.application.iso_27001=true
```

If policy includes an invalid attribute value, a warning appears at policy compile time and you must update policy for it compile correctly.

Renamed Attribute Warnings

If CPL includes an attribute that has been renamed in the currently downloaded database, policy warnings occur at compilation time. The following is an example of the warning:

```
Deprecation warning: 'old_attribute'; 'old_attribute' has been replaced by 'new_attribute'
due to Too obscure and will no longer be accepted after Sat, 27 Jun 2020 00:00:00 UTC.
Please switch to the new name before then.
```

To ensure that policy performs as intended, rename all instances of the affected attribute in CPL and re-apply policy by the specified date.

Syntax

```
request.application.attribute=value
```

where:

- *attribute*: Name of an attribute. After you enable the service, the appliance downloads the attributes list. The list is located at:

  https://*appliance_IP_address*:8082/ApplicationClassification/Attributes

- *value*: One of the following, depending on the attribute:
  - An integer or range of integers. For example, `request.application.default_brr=10..50` tests for a Business Readiness Rating between 10 and 50.
  - An alphanumeric string. For example, `request.application.founding_year=2000` tests for a founding year of 2000.
  - Boolean, such as Yes/No and True/False. For example, `request.application.csa_compliant=yes` tests whether the URL is CSA compliant.
  - a system-defined value:
    - none - The specified attribute is undefined or does not exist for the given application.
    - unavailable - A non-licensing problem exists with the Application Attributes database or with accessing the service, or the Application Attributes service is disabled.
    - unlicensed - The appliance does not have a valid Application Classification license.

  For example, `request.application.uses_ssl=none` tests if there is no Uses SSL attribute for the application.

Layer and Transaction Notes

- Layers: `<Proxy>` , `<SSL>`, `<SSL-Intercept>`
- Transactions: all transactions where a URL is available

Example

Deny requests to URLs where the associated web application does not use SSL.

```
define condition ssl_att
  request.application.uses_ssl=false
end condition ssl_att

<Proxy>
  condition=ssl_att Deny
```

request.application.group=

Tests the specified web application groups for a URL. You must have a valid CASB Audit subscription (included in an Intelligence Services bundle) and enable the Application Classification service to use this condition.

Modifications to the CASB database are automatically provided in updates via the subscription feed. You do not have to update policy when this occurs, but if CPL includes an application that will be renamed, policy compilation emits a message about the change and when the change will take effect. If CPL includes an application that is already removed, a deprecation message appears at compile time.

You can use the `#(config application-classification)view groups` CLI command to list the supported application groups, and the `#(config application-classification)view applications group` *group_name* command to identify the web applications belonging to the specified group. Note that applications can belong to more than one group.

Syntax

```
request.application.group={group_name1[,group_name2..]}
```

where:

- *group_name*: Name defined in the Application Classification database. Group names that include spaces must be enclosed within quotation marks. A single group is not required to be enclosed within parentheses.

Layer and Transaction Notes

- Layers: `<Cache>`, `<Exception>`, `<Proxy>` , `<SSL>`, `<SSL-Intercept>`
- Transactions: proxy transactions with a request URL

Example 1

Deny requests to all web applications within the specified group.

```
<Proxy>
  DENY request.application.group="File Transfer"
```

Example 2

Allow requests to all web applications within the specified groups.

```
<Proxy>
  ALLOW request.application.group=(ERP, "Time Tracking")
```

See Also

- Conditions: "request.application.name=" on the facing page, "request.application.operation=" on page 217

request.application.name=

Test the Blue Coat content filter application name of the request URL.

> **Note:** The content filter could also use HTTP request body data to determine the application name using the http.request.data= condition.

Syntax

`request.application.name=`*`application_name`*

where:

- *application_name*: Application name defined by the WebFilter database. Use the **#show content-filter bluecoat applications** CLI command to list the supported application names.

Layer and Transaction Notes

- Layers: <Cache>, <Exception>, <Proxy> , <SSL>, <SSL-Intercept>
- Transactions: proxy transactions with a request URL

Example 1

Block access to Hotmail.

```
<Proxy>
  request.application.name=Hotmail deny
```

Example 2

Allow LinkedIn access but deny Hotmail.

```
; Show an error page displaying the application name
define string my_format_string
 > request.application.name= $(url.application.name)
   > and the details: $(exception.details)
end

<Proxy>
  request.application.name=Linkedin allow ; no explicit allows
  request.application.name=Hotmail exception( policy_denied, "To prevent data loss, personal email is
prohibited by management.", my_format_string )
```

Example 3

Block all social networking sites except for Facebook.

```
<Proxy>
  request.application.name=Facebook allow
  url.category=social-networking deny
```

See Also

- Conditions: "http.request.data=" on page 167,"request.application.operation=" on the facing page, "url.category=" on page 307

request.application.operation=

Test the Blue Coat content filter application operation of the request URL.

Syntax

`request.application.operation=`*`operation`*

where:

- *operation*: Operation name defined by the WebFilter database. Use the `#show content-filter bluecoat operations` CLI command to list the supported application operations.

Layer and Transaction Notes

- Layers: `<Cache>`, `<Exception>`, `<Proxy>` , `<SSL>`, `<SSL-Intercept>`
- Transactions: proxy transactions with a request URL

Example

Allow attachment downloads, but block attachment uploads for all applications.

```
; Allow attachment downloads but not uploads.
; Show an error page displaying the application operation
define string my_format_string
  > request.application.operation= $(request.application.operation)
    > and the details: $(exception.details)
end


<Proxy>
  request.application.operation=”Download Attachment” allow ; no explicit allows
  request.application.operation=”Upload Attachment” exception( policy_denied, "Attachment uploads are
not allowed.", my_format_string )
```

See Also

- Conditions: "request.application.name=" on page 215, "url.category=" on page 307

request.header.content-length.as_number=

This condition is used to test the value of the HTTP Content-Length request header.

The condition modifier `.as_number` allows you to configure rules based on the actual number of bytes in the Content-Length header. This is an alternative to the `.regex` condition modifier, which can lead to performance issues.

Syntax

`request.header.content-length.as_number={`*number*`|`*range*`}`

where:

- *number*: Number of bytes.
- *range*: Expressed as *N..N*. A value of 1000..2000 will trigger the rule for requests providing a Content-Length value between 1000 and 2000 bytes. See Condition Syntax in this guide for information on using a double period with integer-based values.

Layer and Transaction Notes

- Layers: `<Cache>`, `<Proxy>`
- Transactions: transactions where the user agent making requests provides a valid Content-Length HTTP request header

Example

Deny the request when the Content-Length HTTP request header indicates a body size that is greater than 10 MB.

```
<Proxy>
  request.header.content-length.as_number=10485760.. DENY
```

request.header.*header_name*=

Tests the specified request header (header_name) against a regular expression or a string. Any recognized HTTP request header can be tested. For custom headers, use "request.x_header.header_name=" on page 241 instead. For streaming requests, only the User-Agent header is available.

Syntax

`request.header.header_name[.string_modifier]=pattern`

where:

- *header_name*: Recognized HTTP header. For a complete list of recognized headers, see "Recognized HTTP Headers" on page 700.
- *string_modifier*: A supported string modifier:
  - `exact` - match the string exactly
  - `prefix` - match the start of a string
  - `regex` - (default if no string modifier is specified) regular expression match; see "Regex Reference" on page 703
  - `substring` - match part of a string
  - `suffix` - match the end of a string
- *pattern*: Type-appropriate test expression, such as a quote-delimited string expression or a regular expression.

Layer and Transaction Notes

- Layers: `<Admin>`, `<Cache>`, `<Exception>`, `<Forward>`, `<Proxy>`
- Transactions: HTTP proxy, streaming transactions; for streaming requests, only the User-Agent header is available

Example 1

Deny access when request is sent with the `Pragma-no-cache` header.

```
<Proxy>
  deny url=http://www.symantec.com request.header.Pragma.exact=”no-cache”
```

Example 2

Detect signature cookies.

```
define action delete_all_unsigned_cookies
  iterate(request.header.Cookie)
```

```
    iterator.prefix="BCSIG_"
    request.header.Cookie.exact=!"$(iterator:rewrite(([^=]*)=(.*), BCSIG_$(1)=$(2:concat
($(client.address)):hmac))" iterator.delete()
  end
end

<Proxy>
  iterator.delete_all_unsigned_cookies(yes)
```

See Also

- Conditions: "request.header.header_name.address=" on the facing page, "request.x_header.header_name=" on page 241, "request.x_header.header_name.address=" on page 242,"response.header.header_name=" on page 248
- Properties: "transform.data_type()" on page 581
- Actions: "append()" on page 595, "delete()" on page 598, "delete_matching()" on page 600, "rewrite()" on page 615, "set ()" on page 619, "transform" on page 622

request.header.*header_name*.address=

Tests if the specified request header can be parsed as an IP address; otherwise, false. If parsing succeeds, then the IP address extracted from the header is tested against the specified IP address. The expression can include an IP address or subnet, or the label of a "define subnet" on page 656 block. This condition can only be used with the client-ip and X-Forwarded-For headers.

Syntax

`request.header.`*`header_name`*`.address=[`*`ip_address`*`|`*`subnet`*`|`*`subnet_label`*`}`

where:

- *`header_name`*: Recognized HTTP header. For a complete list of recognized headers, see "Recognized HTTP Headers" on page 700.
- *`ip_address`*: Client IP address; for example, 10.25.198.0
- *`subnet`*: Subnet specification; for example, 10.25.198.0/24
- *`subnet_label`*: Label of a "define subnet" on page 656 block that binds a number of IP addresses or subnets

Layer and Transaction Notes

- Layers: `<Cache>`, `<Proxy>`
- Transactions: HTTP proxy, streaming transactions; for client-ip and X-Forwarded-For headers only

Example

Allow forwarded requests with the specified NATed client IP address.

```
<Proxy>
  request.header.X-Forwarded-For.address=198.51.10.0 url.domain=test.com allow

<Proxy>
  request.header.X-Forwarded-For.address=198.51.10.0/24 url.domain=test.com allow
```

See Also

- Conditions: "request.header.header_name=" on page 219, "request.x_header.header_name=" on page 241, "request.x_header.header_name.address=" on page 242,"response.header.header_name=" on page 248
- Properties: "transform.data_type()" on page 581
- Actions: "append()" on page 595, "delete()" on page 598, "delete_matching()" on page 600, "rewrite()" on page 615, "set()" on page 619, "transform" on page 622

request.header.*header_name*.count=

Test the number of header values in the request for the specified header name.

Syntax

`request.header.`*`header_name`*`.count=={`*`range`*`|`*`exact`*`}`

where:

- *range*: A range specified with a lower limit, an upper limit, or both:
  - `..`*`upper_limit`* - the highest number of header values
  - *`lower_limit`*`..` - the lowest number of header values
  - *`lower_limit`*`..`*`upper_limit`* - number of header values within the specified range
- *exact_rate*: Exact number of header values.

Layer and Transaction Notes

- Layers: <Cache>, `<Exception>`, `<forwarding>`, `<Proxy>`
- Transactions: HTTP proxy

Example

Deny abnormal HTTP requests with two or more host headers.

```
<Proxy>
  DENY("Too many Host headers") request.header.Host.count=2..
```

request.header.*header_name*.exists=

Test whether the specified request header exists.

Syntax

`request.header.`*`header_name`*`.exists={yes|no}`

where:

- *header_name*: Recognized HTTP header. For a complete list of recognized headers, see "Recognized HTTP Headers" on page 700.

Layer and Transaction Notes

- Layers: `<Exception>`, `<Proxy>`
- Transactions: HTTP proxy

Example

Test whether `Accept` header exists in the request.

```
<Proxy>
  request.header.Accept.exists=yes
```

request.header.*header_name*.length=

Test the total length of the header values for the specified header name.

Syntax

`request.header.`*`header_name`*`.length=`*`Length`*

where:

- *Length*: Number of bytes, from 0 to 8192.

Layer and Transaction Notes

- Layers: <Exception>, <Forward>, <Proxy>
- Transactions: HTTP proxy

Example

Deny HTTP requests with more than 2K of cookie data.

```
<Proxy>
  DENY("Too much Cookie data") request.header.Cookie.length=2048..
```

request.header.Origin.url.category=

Test the content filter categories associated with the hostname in the `Origin` request header.

System-Defined Statuses

The following statuses are system-defined:

- `none`: The request is uncategorized by all enabled content filter providers. If even one content filter provider returns a category for the requested site, policy rules containing a `*.category=none` condition will not match.
- `pending`: The categorization request has not been processed or is not complete.
- `unavailable`: A content filter provider is selected in configuration, but an error occurs in determining the category. Other categories can still be assigned directly by policy. This categorization might be a result of a missing database.
- `unlicensed`: A content filter provider license is expired. When this status is true, the `unavailable` status is also true.

Renamed Category Warnings

If CPL includes a category that has been renamed in the currently downloaded database, policy warnings occur at compilation time. The following is an example of the warning:

```
Deprecation warning: 'old_category'; 'old_category' has been replaced by 'new_category'
due to Category name updated and will no longer be accepted after Sat, 11 Jul 2020 00:00:00 UTC.
Please switch to the new name before then.
```

To ensure that policy performs as intended, rename all instances of the affected category in CPL and re-apply policy by the specified date.

Syntax

```
request.header.Origin.url.category={status|category_name1[,category_name2,...]|category_group1
[,category_group2,...]}
```

where:

- *status*: One of the following system-defined statuses: `none`, `pending`, `unavailable`, and `unlicensed`.
- *category_name*: Category names defined by policy or the selected content filter provider.
- *category_group*: Category group names defined by policy or Blue Coat provider if enabled.

  The list of currently valid category names and category group names is available both through the Management Console and CLI.

Layer and Transaction Notes

- Layers: `<Exception>`, `<Proxy>`
- Transactions: proxy transactions with a request URL

Example

Access-log requests when the Origin hostname is associated with the specified categories.

```
<Proxy>
  request.header.Origin.url.category=Sport,Gambling access_log.default(yes)
```

See Also

- Conditions: "request.header.Origin.url.threat_risk.level=" on the facing page

request.header.Origin.url.threat_risk.level=

Test the Threat Risk Level associated with the hostname in the `Origin` request header.

Syntax

`request.header.Origin.url.`*`risk_level`*`={status|`*`risk_level1`*`[,`*`risk_level2`*`,...]}`

where:

- *`status`*: One of the following system-defined statuses: `none`, `pending`, `unavailable`, and `unlicensed`.
- *`risk_level`*: One of the following:
  - An integer or range of integers from 1 to 10, representing the severity level of the threat risk, for example:
    - `10` – Threat risk level 10
    - `8..10` – Threat risk levels 8 through 10
    - `7..` – Threat risk level 7 and higher
    - `..4` – Threat risk level 4 and lower
  - `0` to override the threat risk assigned to the URL.

Refer to the following tables for descriptions of these threat risk levels:

Level(s)	Description
1, 2	Low. The URL has an established history of normal behavior and has no future predictors of threats; however, this level should be evaluated by other layers of defense (such as Content Analysis and Malware Analysis).
3, 4	Medium-Low. The URL has an established history of normal behavior, but is less established than URLs in the Low group. This level should be evaluated by other layers of defense (such as Content Analysis and Malware Analysis).
5, 6	Medium. The URL is unproven; there is not an established history of normal behavior. This level should be evaluated by other layers of defense (such as Content Analysis and Malware Analysis) and considered for more restrictive policy.
7, 8, 9	Medium-High. The URL is suspicious; there is an elevated risk. Symantec recommends blocking at this level.
10	High. The URL is confirmed to be malicious. Symantec recommends blocking at this level.
0	Override the threat risk associated with the URL.
none	

Layer and Transaction Notes

- Layers: `<Exception>`, `<Proxy>`
- Transactions: proxy transactions with a request URL

Example

Deny requests from the specified group when the Origin hostname has threat risk level 7 through 10.

```
define condition user_group1
  realm=SAML1 group="group1"
end condition user_

<Proxy>
  condition=group1 request.header.Origin.url.threat_risk.level=7..10 deny
```

See Also

- Conditions: "request.header.Origin.url.category=" on page 225

request.header.Referer.url=

Test if the URL specified by the Referer header matches the specified criteria. The basic `request.header.Referer.url=` test attempts to match the complete Referer URL against a specified pattern. The pattern might include the scheme, host, port, path and query components of the URL. If any of these is not included in the pattern, then the corresponding component of the URL is not tested and can have any value.

Specific portions of the Referer URL can be tested by applying URL component modifiers to the condition. In addition to component modifiers, optional test type modifiers can be used to change the way the pattern is matched.

This condition is unavailable if the Referer header is missing, or if its value cannot be parsed as a URL. If the Referer header contains a relative URL, the requested URL is used as a base to form an absolute URL prior to testing.

Syntax

```
request.header.Referer.url.string.modifier=pattern
request.header.Referer.url[.case_sensitive][.no_lookup]=prefix_pattern
request.header.Referer.url.domain[.case_sensitive][.no_lookup]=domain_suffix_pattern
request.header.Referer.url.address={ip_address|ip_address_range|ip_address_wildcards|subnet|subnet_
label}
request.header.Referer.url.extension[.case_sensitive]=[.]filename_extension
request.header.Referer.url.host.has_address={yes|no|restricted|refused|nxdomain|error}
request.header.Referer.url.host[.exact]=host
request.header.Referer.url.host.[prefix|substring|suffix]=string
request.header.Referer.url.host.has_name={yes|no|restricted|refused|nxdomain|error}
request.header.Referer.url.is_absolute={yes|no}
request.header.Referer.url.host.is_numeric={yes|no}
request.header.Referer.url.host.no_name={yes|no}
request.header.Referer.url.path[.case_sensitive]=string
request.header.Referer.url.path[.substring|.suffix][.case_sensitive]=string
request.header.Referer.url.path.regex[.case_sensitive]=regular_expression
request.header.Referer.url.port={range|exact}
request.header.Referer.url.query.regex[.case_sensitive]=regular_expression
request.header.Referer.url.scheme=url_scheme
```

where:

- all options are identical to "url=" on page 298, except for the URL being tested. For more information, see "url=" on page 298.

> **Tip:** The `request.header.Referer.url=` condition is identical to "url=" on page 298, except for the lack of a "define url condition" on page 658 and [url] or [url.domain] sections.

Layer and Transaction Notes

- Layers: <Exception>, <Proxy>
- Transactions: HTTP proxy

Example

Test if the Referer URL includes this pattern, and block access.

```
; Relative URLs, such as docs subdirectories and pages, will match.
deny request.header.Referer.url=http://www.example.com/docs

; Test if the Referer URL host’s IP address is a match.
request.header.Referer.url.address=10.1.198.0

; Test whether the Referer URL includes company.com as domain.
request.header.Referer.url.domain=company.com

; Test whether the Referer URL includes .com.
request.header.Referer.url.domain=.com

; Test if the Referer URL includes this domain-suffix pattern, and block service.
; Relative URLs, such as docs, subdirectories and pages, will match.
deny request.header.Referer.url.domain=company.com/docs

; examples of the use of .Referer.url.extension=
request.header.Referer.url.extension=.txt
request.header.Referer.url.extension=(.htm, .html)
request.header.Referer.url.extension=(img, jpg, jpeg)

; How request.header.Referer.url.has_address and request.header.Referer.url.has_name test for DNS
responses
request.header.Referer.url.has_address=yes ; DNS lookup succeeded
request.header.Referer.url.has_address=no ; DNS lookup failed
request.header.Referer.url.has_address=restricted ; lookup was restricted because the host or address
was included in a "restrict dns" on page 667 or "restrict rdns" on page 669 definition
request.header.Referer.url.has_name=refused ; DNS server refused response
request.header.Referer.url.has_name=nxdomain ; domain does not exist
request.header.Referer.url.has_name=error ; DNS error response was received

; Trigger the specified exception when the forward DNS lookup of the Referer hostname fails
request.header.Referer.url.has_address=no exception(policy_denied,"DNS failure")

; Trigger the specified exception when the reverse DNS lookup of the Referer URL address fails
request.header.Referer.url.host.has_name=no exception(policy_denied,"RDNS lookup failure")

; This example matches the first Referer header value and doesn’t match the second
```

```
; from the following two requests:
; 1) Referer: http://1.2.3.4/test
; 2) Referer: http://www.example.com
<Proxy>
  request.header.Referer.url.host.is_numeric=yes

; In the example below we assume that 1.2.3.4 is the IP of the host mycompany
; The condition will match the following two requests if the reverse DNS was successful:
; 1) Referer: http://1.2.3.4/
; 2) Referer: http://mycompany.com/
; If the reverse DNS fails then the first request is not matched
<Proxy>
  request.header.Referer.url.host.regex=mycompany

; .Referer.url.path tests
; The following .Referer.url.path strings would all match the example Referer URL:
; Referer: http://www.example.com/cgi-bin/query.pl?q=test#fragment
request.header.Referer.url.path="/cgi-bin/query.pl?q=test"
request.header.Referer.url.path="/cgi-bin/query.pl"
request.header.Referer.url.path="/cgi-bin/"
request.header.Referer.url.path="/cgi" ; partial components match too
request.header.Referer.url.path="/" ; Always matches regardless of URL.

; Testing the Referer URL port
request.header.Referer.url.port=80
```

See Also

- Conditions:"server_url=" on page 269, "url=" on page 298,
- Definitions: "define subnet" on page 656
- Information on wildcards: http://www.symantec.com/docs/TECH241521
- Information on IP address ranges: http://www.symantec.com/docs/TECH241929

request.header.Referer.url.category=

Test the content filter categories of the Referer URL.

System-Defined Statuses

The following statuses are system-defined:

- none: The request is uncategorized by all enabled content filter providers. If even one content filter provider returns a category for the requested site, policy rules containing a `*.category=none` condition will not match.
- pending: The categorization request has not been processed or is not complete.
- unavailable: A content filter provider is selected in configuration, but an error occurs in determining the category. Other categories can still be assigned directly by policy. This categorization might be a result of a missing database.
- unlicensed: A content filter provider license is expired. When this status is true, the unavailable status is also true.

Renamed Category Warnings

If CPL includes a category that has been renamed in the currently downloaded database, policy warnings occur at compilation time. The following is an example of the warning:

```
Deprecation warning: 'old_category'; 'old_category' has been replaced by 'new_category'
due to Category name updated and will no longer be accepted after Sat, 11 Jul 2020 00:00:00 UTC.
Please switch to the new name before then.
```

To ensure that policy performs as intended, rename all instances of the affected category in CPL and re-apply policy by the specified date.

Syntax

```
request.header.Referer.url.category={status|category_name1[,category_name2,...]|category_group1
[,category_group2,...]}
```

where:

- *status*: One of the following system-defined statuses: none, pending, unavailable, and unlicensed.
- *category_name*: Category names defined by policy or the selected content filter provider.
- *category_group*: Category group names defined by policy or Blue Coat provider if enabled.

  The list of currently valid category names and category group names is available both through the Management Console and CLI.

Layer and Transaction Notes

- Layers: `<Exception>`, `<Proxy>`
- Transactions: proxy transactions with a request URL

Example

Test for the Sports category.

```
<Proxy>
  request.header.Referer.url.category=Sports
```

See Also

- Conditions: "category=" on page 85, "server.certificate.hostname.category=" on page 262, "url.category=" on page 307

request.header.Referer.url.host.is_private=

Test whether the Referer URL is within the configured private network.

Syntax

```
request.header.Referer.url.host.is_private={yes|no}
```

Layer and Transaction Notes

- Layers: <Exception>, <Proxy>
- Transactions: proxy transactions with a request URL

Example

Tests for a Referer URL within the private network.

```
<Proxy>
  request.header.Referer.url.host.is_private=yes
```

request.icap.apparent_data_type=

This condition allows you to leverage a ProxyAV appliance's ability to extract file archives to identify the types of files being submitted by users during an HTTP POST.

In order for this condition to be effective, request data must first be sent to a ProxyAV appliance. The ProxyAV's scanning result is sent to the ProxySG appliance.

> **Note:** Recommended for Reverse Proxy deployments, where files are uploaded through the appliance to a back-end server.

Syntax

`request.icap.apparent_data_type=`*`data_type`*

where:

- *`data_type`*: One of the data types in the following table; specify using either the label or the file extension:

*Added in 7.2.4.1.

Label	Description	Common Extensions
7ZIP*	7-Zip archive	.7z
ACE*	ACE archive	.ace
ARJ*	ARJ archive	.arj
BMP	BMP image	.bmp
BZ2	BZip2 archive	.bz2, .tbz2
CAB	MS Cab archive	.cab
COMPRESS*	compress compressed file	.Z (different from .z)
CPIO*	cpio archive	.cpio
DAA*	Direct Access Archive	.daa
EGG*	.EGG archive	.egg
EML*	raw email	.eml, .mht, .mhtml
EXE	MS application	.exe
FLASH	Adobe Flash	.swf, flv
GIF	GIF image	.gif
GZIP	GZIP compressed file	.gz
HTML	HTML file	.html

Label	Description	Common Extensions
ICC	ICC profile	.icc
JPG	JPEG image	.jpg
LHA*	LHA archive	.lha, .lzh
LZIP*	Lzip compressed file	.lz
MACH-O*	macOS application or library	
MSDOC	Microsoft document	.doc, .docx, .xls, .xlsx, .ppt, .pptx, .msi, .vba
MRAR	multi-part RAR	.partX.rar
MZIP	multi-part ZIP	.zip.xxx
PDF	Portable Document Format file	.pdf
PNG	PNG image	.png
RAR	RAR archive	.rar
RTF	Rich text document	.rtf
TAR	TAR archive	.tar
TIF	TIFF image	.tif
TNEF*	file encoded in Microsoft Transport-Neutral Encapsulation Format	.dat, .tnef
TTF	True-Type font	.ttf
TXT	plain text	.txt
UUE*	file encoded with uuencode or xxencode	.uu, .uue, .xx, .xxe
XAR*	Extensible Archive Format	.mpkg, .pkg, .xar
XML	XML file	.xml
XZ*		.xz
ZIP	ZIP archive	.zip

Note: New-style Microsoft documents, (.DOCX, .PPTX and so on) use a zip-style format. ProxyAV version 3.5 will treat them as such and report the apparent data type as ZIP.

Layer and Transaction Notes

- Layers: `<Cache>`, `<Proxy>`
- Transactions: all HTTP and decrypted HTTPS POST requests that have been processed by a ProxyAV Appliance running version 3.5 or later

> **Note:** An ICAP Request Modification rule is required to handle the appliance-to-ICAP communication.

Example

A reverse proxy administrator would like to prevent users on the Internet from uploading executable files to a website hosted behind the appliance. The website permits uploads of zip files as well, so the administrator wants to ensure that ZIP files containing EXE files are also rejected. The policy below first sends .ZIP files to be ICAP scanned, then in the later layer and rule, it acts on the results of that scan to identify the types of files in that zip file.

```
<Cache>
  http.request.apparent_data_type=ZIP request.icap_service(req)
  DENY http.request.apparent_data_type=EXE

<Cache>
  DENY http.request.apparent_data_type=ZIP request.icap.apparent_data_type=EXE
```

See Also

- Conditions: "http.request.apparent_data_type=" on page 158, "http.response.apparent_data_type=" on page 174, "response.icap.apparent_data_type=" on page 249
- Properties: "http.request.apparent_data_type.allow()" on page 438, "http.request.apparent_data_type.deny()" on page 441

request.icap.error_code=

Tests which ICAP error occurred during request scanning. Rules that include this condition apply only to transactions involving ICAP scanning.

Syntax

```
request.icap.error_code={any|none|icap_error}
```

where:

- *any*: Any ICAP error.
- *none*: No ICAP error.
- *icap_error*: One of the following ICAP errors:

  antivirus_load_failure, antivirus_license_expired, antivirus_engine_error, connection_failure, decode_error file_ extension_blocked, insufficient_space, internal_error, max_file_size_exceeded, max_total_size_exceeded, max_ total_files_exceeded password_protected, request_timeout, scan_timeout, server_error, server_unavailable

Layer and Transaction Notes

- Layers: <Exception>, <Proxy>
- Transactions: HTTP transactions (proxy, refresh, pipeline), FTP proxy transactions

Example

In this example, the request.icap_service() property specifies to use the ICAP service named virus_scan, and includes the fail_open argument so that requests are sent out if the service is unhealthy. If no ICAP error occurs (none) or if the scanning service determine that the file is password-protected (password_protected), users are allowed or denied access to the files as determined by later policy rules. If any other ICAP error occurs, a user-defined exception page named virus_scan_failure is returned instead.

```
<Proxy>
  request.icap_service(virus_scan, fail_open)

<Proxy>
  request.icap.error_code=!(none, password_protected) exception(virus_scan_failure)
```

request.raw_headers.count=

Test the total number of HTTP request headers.

This condition tests the total number of raw HTTP request headers, as defined by the request.raw_headers.regex condition.

Syntax

`request.raw_headers.count=`*`number`*

where:

- *number*: Number from 0 to 8192.

Layer and Transaction Notes

- Layers: `<Exception>`, `<Proxy>`
- Transactions: HTTP proxy

Example

Reject the request if it contains more than 40 request headers.

```
<Proxy>
  exception(invalid_request) request.raw_headers.count=40..
```

request.raw_headers.regex=

Test the value of all HTTP request headers with a regular expression and without any normalizations applied. The appliance normalizes URLs to better enforce policy; however, testing the raw form may be preferred in some cases, such as using CPL to detect that the HTTP header contained an injection attack.

This condition allows you to test the complete, unaltered HTTP request header text, which includes the header names, delimiters and header values. It iterates over all of the raw HTTP request headers. If the specified regular expression matches one of these strings, then the condition is true.

Each raw header is a string consisting of a header line concatenated with zero or more continuation lines. The initial header line consists of a header name, followed by colon, followed by the header value, if any, followed by newline. The header value may have leading and trailing whitespace. Each continuation line begins with a space or tab, followed by additional text which is part of the header value, followed by a newline. Therefore, each raw header string contains a minimum of one newline, plus an additional newline for each continuation line.

Here is how certain regex patterns work in the context of request.raw_headers.regex:

- "." matches any character, including newline.
- "^" only matches at the beginning of the header name.
- "$" only matches at the end of the string. The last character of the string is newline, so "$" will only match after the final newline. You probably want to use "\s*$" instead.
- "\s" matches any white space character, including newline.
- "\n" matches newline.

Syntax

`request.raw_headers.regex=`*`regular_expression`*

Layer and Transaction Notes

- Layers: `<Exception>`, `<Proxy>`
- Transactions: HTTP proxy

Example

Reject the request if it contains a header continuation line. Although this syntax is part of the HTTP standard, it is not normally used, and might not be interpreted correctly by some upstream devices.

```
<Proxy>
  exception(invalid_request) request.raw_headers.regex="\n[ \t]"
```

request.x_header.*header_name*=

Tests the specified custom request header (*header_name*) against a regular expression or a string.

To test recognized headers, use "request.header.header_name=" on page 219 instead, so that typing errors can be caught at compile time. For streaming requests, only the User-Agent header is available.

Syntax

`request.x_header.header_name[.string_modifier]=pattern`

where:

- *header_name*: HTTP header, including custom headers.
- *string_modifier* - supported string modifier:
  - `exact` - match the string exactly
  - `prefix` - match the start of a string
  - `regex` - (default if no string modifier is specified) regular expression match; see "Regex Reference" on page 703
  - `substring` - match part of a string
  - `suffix` - match the end of a string
- *pattern*: Type-appropriate test expression, such as a quote-delimited string expression or a regular expression.

Layer and Transaction Notes

- Layers: `<Cache>`, `<Exception>`, `<Forward>`, `<Proxy>`
- Transactions: HTTP proxy

Example

Deny access to the URL below if the request contains the custom header “Test” and the header has a value of “test1”.

```
<Proxy>
  deny url=http://www.symantec.com request.x_header.Test.exact=”test1”
```

See Also

- Conditions: "request.header.header_name=" on page 219, "request.header.header_name.address=" on page 221, "request.x_header.header_name.address=" on the next page, "response.x_header.header_name=" on page 256
- Actions: "append()" on page 595, "delete()" on page 598, "delete_matching()" on page 600, "rewrite()" on page 615, "set()" on page 619

request.x_header.*header_name*.address=

Tests if the specified request header can be parsed as an IP address; otherwise, false. If parsing succeeds, then the IP address extracted from the header is tested against the specified IP address. The expression can include an IP address or subnet, or the label of a subnet definition block. This condition is intended for use with custom headers other than X-Forwarded-For and Client-IP headers; for these, use"request.header.header_name.address=" on page 221 so that typing any errors can be caught at compile time.

Syntax

```
request.x_header.header_name.address={ip_address|ip_address_range|ip_address_wildcards|subnet|subnet_label}
```

where:

- *ip_address*: Effective IP address; for example, 10.25.198.0
- *ip_address_range*: IP address range; for example, 192.0.2.0-192.0.2.255
- *ip_address_wildcards*: IP address specified using wildcards in any octet(s); for example, 10.25.*.0 or 10.*.*.0
- *subnet*: Subnet specification; for example, 10.25.198.0/24
- *subnet_label*: Label of a subnet definition block that binds a number of IP addresses or subnets

Layer and Transaction Notes

- Layers: <Cache>, <Proxy>

Example

Deny access if the request's custom header "Local" has the value 10.1.198.0.

```
<Proxy>
  deny request.x_header.Local.address=10.1.198.0
```

See Also

- Conditions: "request.header.header_name.address=" on page 221, "request.x_header.header_name=" on the previous page, "response.x_header.header_name=" on page 256
- Actions: "append()" on page 595, "delete()" on page 598, "delete_matching()" on page 600, "rewrite()" on page 615, "set()" on page 619
- Definitions: "define subnet" on page 656

- Information on wildcards: http://www.symantec.com/docs/TECH241521
- Information on IP address ranges: http://www.symantec.com/docs/TECH241929

request.x_header.*header_name*.count=

Test the number of header values in the request for the specified custom request header name.

Syntax

`request.x_header.header_name.count=number`

where:

- *number*: Number between 1 and 8192.

Layer and Transaction Notes

- Layers: <Exception>, <Proxy>
- Transactions: HTTP proxy

Example

Deny abnormal HTTP requests with 2 or more host headers.

```
<Proxy>
  DENY("Too many Host headers") request.header.Host.count=2..
```

See Also

- Conditions: "request.header.header_name.address=" on page 221, "request.x_header.header_name=" on page 241, "request.x_header.header_name.exists=" on the facing page, "request.x_header.header_name.length=" on page 246

request.x_header.*header_name*.exists=

Test whether the specified custom request header exists.

Syntax

`request.x_header.`*`header_name`*`.exists={yes|no}`

Layer and Transaction Notes

- Layers: `<Exception>`, `<Proxy>`
- Transactions: HTTP proxy

Example

Test whether the `Accept` header exists.

```
<Proxy>
   request.x_header.Accept.exists=yes
```

request.x_header.*header_name*.length=

Test the total length of the header values for the specified header name.

Syntax

`request.x_header.`*`header_name`*`.length=`*`number`*

where:

- *number*: Number between 1 and 8192.

Layer and Transaction Notes

- Layers: <Exception>, <Proxy>
- Transactions: HTTP proxy

Example

Deny HTTP requests with more than 2K of cookie data.

```
<Proxy>
  DENY("Too much Cookie data") request.x_header.Cookie.length=2048..
```

response.header.content-length.as_number=

Test the value of the HTTP Content-Length response header.

The condition modifier, .as_number allows you to configure rules based on the actual number of bytes in the Content-Length HTTP response header. This is an alternative to the .regex condition modifier, which can lead to performance issues.

Syntax

`response.header.content-length.as_number={exact|range}`

where:

- *exact*: Exact number of bytes.
- *range*: Ranges are supported when defining as_number separated by .. (a value of 1000..2000 will trigger the rule for requests equal to a size between 1000 and 2000 bytes). See "Condition Syntax" on page 65 for information on using a double period with integer-based values.

Layer and Transaction Notes

- Layers: `<Cache>`, `<Proxy>`
- Transactions: Used where the OCS responding to a proxied request provides a valid Content-Length HTTP response header.

Example

Deny the response when the Content-Length HTTP response header indicates a body size that is greater than 10 MB.

```
<Proxy>
  response.header.content-length.as_number=10485760.. DENY
```

response.header.*header_name*=

Tests the specified response header (*header_name*) against a regular expression or a string.

Any recognized HTTP response header can be tested. For custom headers, use "request.x_header.header_name=" on page 241 instead.

Syntax

`response.header.header_name[.string_modifier]=pattern`

where:

- *header_name*: Recognized HTTP header. For a complete list of recognized headers, see "Recognized HTTP Headers" on page 700.
- *string_modifier*: A supported string modifier:
  - `exact` - match the string exactly
  - `prefix` - match the start of a string
  - `regex` - (default if no string modifier is specified) regular expression match; see "Regex Reference" on page 703
  - `substring` - match part of a string
  - `suffix` - match the end of a string
- *pattern*: Type-appropriate test expression, such as a quote-delimited string expression or a regular expression.

Layer and Transaction Notes

- Layers: `<Cache>`, `<Exception>`, `<Proxy>`

Example

Test if the response's "Content-Type" header has the value "image/jpeg".

```
<Proxy>
  response.header.Content-Type.prefix="image/jpeg( |\t)*($|;)"
```

See Also

- Conditions: "request.header.header_name=" on page 219, "response.x_header.header_name=" on page 256
- Actions: "append()" on page 595, "delete()" on page 598, "delete_matching()" on page 600, "rewrite()" on page 615, "set()" on page 619
- Properties: "access_server()" on page 335, "max_bitrate()" on page 494, "streaming.transport()" on page 569

response.icap.apparent_data_type=

This condition allows you to leverage a ProxyAV Appliance to control the types of files contained in response data for individual and archive files. Specifically, this gesture focuses on the response to an HTTP GET request. In order to match this condition, requests must first be processed by an ICAP response modification rule. After the ProxyAV examines the file, it sends back info (an ICAP header, to be specific) to the appliance describing what files it found. Then policy makes a decision based on this information.

> **Note:** Requires an ICAP response modification rule configured. Policy will fail to compile if this is not present.

Syntax

`response.icap.apparent_data_type=`*`data_type`*

where:

- *`data_type`*: One of the data types in the following table; specify using either the label or the file extension:

*Added in 7.2.4.1.

Label	Description	Common Extensions
7ZIP*	7-Zip archive	.7z
ACE*	ACE archive	.ace
ARJ*	ARJ archive	.arj
BMP	BMP image	.bmp
BZ2	BZip2 archive	.bz2, .tbz2
CAB	MS Cab archive	.cab
COMPRESS*	compress compressed file	.Z (different from .z)
CPIO*	cpio archive	.cpio
DAA*	Direct Access Archive	.daa
EGG*	.EGG archive	.egg
EML*	raw email	.eml, .mht, .mhtml
EXE	MS application	.exe
FLASH	Adobe Flash	.swf, flv
GIF	GIF image	.gif
GZIP	GZIP compressed file	.gz
HTML	HTML file	.html

Label	Description	Common Extensions
ICC	ICC profile	.icc
JPG	JPEG image	.jpg
LHA*	LHA archive	.lha, .lzh
LZIP*	Lzip compressed file	.lz
MACH-O*	macOS application or library	
MSDOC	Microsoft document	.doc, .docx, .xls, .xlsx, .ppt, .pptx, .msi, .vba
MRAR	multi-part RAR	.partX.rar
MZIP	multi-part ZIP	.zip.xxx
PDF	Portable Document Format file	.pdf
PNG	PNG image	.png
RAR	RAR archive	.rar
RTF	Rich text document	.rtf
TAR	TAR archive	.tar
TIF	TIFF image	.tif
TNEF*	file encoded in Microsoft Transport-Neutral Encapsulation Format	.dat, .tnef
TTF	True-Type font	.ttf
TXT	plain text	.txt
UUE*	file encoded with uuencode or xxencode	.uu, .uue, .xx, .xxe
XAR*	Extensible Archive Format	.mpkg, .pkg, .xar
XML	XML file	.xml
XZ*		.xz
ZIP	ZIP archive	.zip

Note: New-style Microsoft documents, (.DOCX, .PPTX and so on) use a zip-style format. ProxyAV version 3.5 will treat them as such and report the apparent data type as ZIP.

Layer and Transaction Notes

- Layers: `<Cache>`, `<Proxy>`
- Transactions: HTTP proxy
- Used to identify the apparent data type in HTTP GET responses for single files and file archives using a configured ProxyAV Appliance running version 3.5 of higher.

Example

A school ProxySG administrator wants to allow students to be able to download RTF documents, but only from educational sources. This deployment also makes use of a ProxyAV Appliance to allowthe appliance to permit users to download RTF files contained in ZIP archives. RTF files from other sources are denied.

```
<Cache>
  http.response.apparent_data_type=ZIP response.icap_service(icap1,fail_closed)

<Cache>
  allow http.response.apparent_data_type=RTF url.category=("Education")
  allow http.response.apparent_data_type=ZIP response.icap.apparent_data_type=RTF url.category=
("Education")
  deny http.response.apparent_data_type=RTF
  deny http.response.apparent_data_type=ZIP response.icap.apparent_data_type=RTF
```

See Also

- Conditions: "http.request.apparent_data_type=" on page 158, "http.response.apparent_data_type=" on page 174

response.icap.error_code=

Test which ICAP error occurred during response scanning. Rules that include this condition apply only to transactions involving ICAP scanning.

Syntax

```
response.icap.error_code={any|none|icap_error}
```

where:

- *any*: Any ICAP error.
- *none*: No ICAP error.
- *icap_error*: One of the following ICAP errors:

  antivirus_load_failure, antivirus_license_expired, antivirus_engine_error, connection_failure, decode_error file_extension_blocked, insufficient_space, internal_error, max_file_size_exceeded, max_total_size_exceeded, max_total_files_exceeded password_protected, request_timeout, scan_timeout, server_error, server_unavailable

Layer and Transaction Notes

- Layers: <Exception>, <Proxy>
- Transactions: HTTP transactions (proxy, refresh, pipeline), FTP proxy transactions

Example

In this example, admin users in the specified group= are allowed access to files if the ICAP service determines that the file is password-protected (password_protected). Any other users are not allowed access to the files and instead receive the content_filter_denied exception with the specified message.

The response.icap_service() property specifies to use the service named cas1, with the default connection setting, and includes the fail_closed argument so that responses are not sent to the client if the service is unhealthy.

```
<Proxy>
  group=admins response.icap.error_code=password_protected ALLOW
response.icap.error_code=password_protected force_exception(content_filter_denied, "Password Protected
Archives are blocked by policy.")

<Cache>
  response.icap_service(cas1, fail_closed) response.icap_service.secure_connection(auto)
```

response.raw_headers.count=

Test the total number of HTTP response headers.

This trigger tests the total number of raw HTTP response headers, as defined by the response.raw_headers.regex trigger.

Syntax

`response.raw_headers.count={`*`range`*`|`*`exact`*`}`

where:

- *range*: A range specified with a lower limit, an upper limit, or both:
  - `..`*`upper_limit`* - the highest number of headers
  - *`lower_limit`*`..` - the lowest number of headers
  - *`lower_limit`*`..`*`upper_limit`* - number of headers within the specified range
- *exact*: Exact number of headers.

Layer and Transaction Notes

- Layers: `<Exception>`, `<Proxy>`
- Transactions: HTTP proxy

Example

Reject the response if it contains 40 or more response headers.

```
<Proxy>
  DENY("Too many response headers") response.raw_headers.count=40..
```

response.raw_headers.length=

Test the total length of all HTTP response headers.

This trigger tests the total number of bytes of HTTP response header data, including the header names, values, delimiters and newlines. The tally does not include the HTTP response line, which is the first line of the response, and it does not include the terminating blank line.

Syntax

`response.raw_headers.length={`*`range`*`|`*`exact`*`}`

where:

- *range*: A range specified with a lower limit, an upper limit, or both:
  - `..`*`upper_limit`* - the greatest length
  - *`lower_limit`*`..` - the lowest length
  - *`lower_limit`*`..`*`upper_limit`* - length within the specified range
- *exact*: Exact number of headers.

Layer and Transaction Notes

- Layers: `<Exception>`, `<Proxy>`
- Transactions: HTTP proxy

Example

Reject the response if it contains more than 4K of response header data.

```
<Proxy>
   DENY("Too much response header data") response.raw_headers.length=4096..
```

response.raw_headers.regex=

Test the value of all HTTP response headers with a regular expression.

This trigger allows you to test the complete, unaltered HTTP response header text, which includes the header names, delimiters and header values. It iterates over all of the raw HTTP response headers. If the specified regular expression matches one of these strings, then the condition is true.

Each raw header is a string consisting of a header line concatenated with zero or more continuation lines. The initial header line consists of a header name, followed by colon, followed by the header value, if any, followed by newline. The header value may have leading and trailing whitespace. Each continuation line begins with a space or tab, followed by additional text which is part of the header value, followed by a newline. Therefore, each raw header string contains a minimum of one newline, plus an additional newline for each continuation line.

Here is how certain regex patterns work in the context of response.raw_headers.regex:

- * "." matches any character, including newline.
- * "^" only matches at the beginning of the header name.
- * "$" only matches at the end of the string. The last character of the string is newline, so "$" only matches after the final newline. You probably want to use "\s*$" instead.
- * "\s" matches any white space character, including newline.
- * "\n" matches newline.

Syntax

`response.raw_headers.regex=`*`regular_expression`*

Layer and Transaction Notes

- Layers: `<Exception>`, `<Proxy>`
- Transactions: HTTP proxy

Example

Reject the response if it contains a header continuation line. Although this syntax is part of the HTTP standard, it is not normally used, and might not be interpreted correctly by some downstream devices.

```
<Proxy>
  exception(invalid_response) response.raw_headers.regex="\n[ \t]"
```

response.x_header.*header_name*=

Tests the specified response header (header_name) against a regular expression or a string. For HTTP requests, any response header can be tested, including custom headers. For recognized HTTP headers, use "response.header.header_name=" on page 248 instead so that typing errors can be caught at compile time.

Syntax

`response.x_header.header_name[.string_modifier]=pattern`

where:

- *header_name*: HTTP header, including custom headers.
- *string_modifier* - supported string modifier:
  - `exact` - match the string exactly
  - `prefix` - match the start of a string
  - `regex` - (default if no string modifier is specified) regular expression match; see "Regex Reference" on page 703
  - `substring` - match part of a string
  - `suffix` - match the end of a string
- *pattern*: Type-appropriate test expression, such as a quote-delimited string expression or a regular expression.

Layer and Transaction Notes

- Layers: `<Cache>`, `<Exception>`, `<Proxy>`
- Transactions: HTTP proxy

Example

Test if the custom header “Security” has the value of “confidential".

```
<Proxy>
  response.x_header.Security.substring="confidential"
```

See Also

- Conditions: "request.x_header.header_name=" on page 241, "response.header.header_name=" on page 248
- Actions: "append()" on page 595, "delete()" on page 598, "delete_matching()" on page 600, "rewrite()" on page 615, "set()" on page 619

risk_score=

Allows you to specify the risk score-based trigger to set an action based on the cumulative risk score that a client reaches for a given transaction. Keep in mind all SQL injection violations have a weight of 10. These are internal values that cannot be altered.

An action, such as denying the request, is taken when the specified risk_score limits are met.

Syntax

`risk_score=`*`risk_score_value`*

where:

- *`risk_score_value`*: Threshold at which a specified action is triggered.

Layer and Transaction Notes

- Layers: `<Proxy>`
- Transactions: HTTP proxy

Example

Block the connection when the client hits one or more violations, assuming a default risk-score value of 10 for each violation.

```
<Proxy>
  risk_score=10.. deny
```

See Also

- Properties: "risk_score.maximum()" on page 530, "risk_score.other()" on page 531

saml.attribute.*attribute_name*=

Compares strings with the value of the SAML assertion attribute obtained from the user's entry.

Syntax

`saml.attribute.`*`attribute_name`*`[.string_modifier][.case_sensitive]=`*`pattern`*

where:

- *`attribute_name`*: SAML assertion attribute.
- *`string_modifier`*: A supported string modifier:
  - `exact` - match the string exactly
  - `prefix` - match the start of a string
  - `regex` - (default if no string modifier is specified) regular expression match; see "Regex Reference" on page 703
  - `substring` - match part of a string
  - `suffix` - match the end of a string

    If you do not specify a modifier, the condition tests for an exact pattern match.
- `case_sensitive` - Perform case-sensitive match. By default, the test is case-insensitive.
- *`pattern`*: Type-appropriate test expression, such as a quote-delimited string expression or a regular expression.

Layer and Transaction Notes

- Layers: `<Forward>`, `<Proxy>` , `<SSL>`, `<SSL-Intercept>`
- Transactions: all

Example

This policy authenticates users against the SAML realm and denies certain users under specific conditions.

```
<Proxy>
  authenticate(SAMLRealm)

<Proxy>
  DENY saml.attribute.user_type.suffix="_restricted"
  DENY category=gambling saml.attribute.web_permission.regex=!".*gambling.*"
  DENY saml.attribute.proxy_user="John.Smith"
  ALLOW
```

See Also

- For information on SAML realm configuration, refer to the “SAML Authentication” chapter in the *SGOS Administration Guide*.

server.certificate.hostname=

Test the hostname of an SSL server certificate.

Test the hostname extracted from the X.509 certificate returned by the server while establishing an SSL connection.

Syntax

```
server.certificate.hostname={domain_name|domain_suffix_pattern}
server.certificate.hostname[.string_modifier]=pattern
```

where:

- *domain_name*: One or more domain labels, separated by dots, such as 'example.com' or 'www.example.com'. It matches the hostname exactly.
- *domain_suffix_pattern*: Domain name prefixed with a '.', and matches the suffix of the hostname on label boundaries. The patterns '.com', '.example.com' and '.www.example.com' all match the domain name 'www.example.com'.
- *string_modifier*: A supported string modifier:
  - `exact` - match the string exactly
  - *`list`* - match the name of a configured domain list definition
  - `prefix` - match the start of a string
  - `length` - an integer representing the length of the hostname
  - `regex` - (default if no string modifier is specified) regular expression match; see "Regex Reference" on page 703
  - `substring` - match part of a string
  - `suffix` - match the end of a string

    If you do not specify a modifier, the condition tests for an exact pattern match.
- *pattern*: Type-appropriate test expression, such as a quote-delimited string expression or a regular expression.

> **Tip:** The pattern expression supports substitutions. You can specify a substitution expression with the .exact, .substring, .prefix, and .suffix string modifiers where they are available.

Layer and Transaction Notes

- Layers: `<Proxy>` , `<SSL>`, `<SSL-Intercept>`
- Transactions: HTTPS forward and reverse proxy transactions, SSL Intercept transactions, SSL tunnel transactions

Example

Use both string and domain-suffix-pattern patterns to direct traffic to a specific log.

```
define condition special_site
; A string pattern must match exactly.
; This pattern will not match
;
; www.somehost.example.com
;
; or wildcard patterns returned in the certificate
; such as:
;
; *.example.com
;
  server.certificate.hostname=somehost.example.com
; This domain-suffix pattern will match
;
; xyz.com
; www.xyz.com
; mailer.xyz.com
; www.mailer.xyz.com
;
;and so on. Note that this will match when
; the server certificate contains wildcards such as
;
; *.xyz.com
; www*.xyz.com
;
  server.certificate.hostname=.xyz.com
end

<Proxy>
  ALLOW condition=special_site access_log(special_log)
```

See Also

- Conditions: "server.certificate.hostname.category=" on the next page, "server.certificate.subject=" on page 264
- Properties: "server.certificate.validate()" on page 536, "server.certificate.validate.check_revocation()" on page 539, "server.certificate.validate.ignore()" on page 540

server.certificate.hostname.category=

Test the content filter categories of the hostname extracted from the X.509 certificate returned by the server while establishing an SSL connection.

System-Defined Statuses

The following statuses are system-defined:

- none: The request is uncategorized by all enabled content filter providers. If even one content filter provider returns a category for the requested site, policy rules containing a `*.category=none` condition will not match.
- pending: The categorization request has not been processed or is not complete.
- unavailable: A content filter provider is selected in configuration, but an error occurs in determining the category. Other categories can still be assigned directly by policy. This categorization might be a result of a missing database.
- unlicensed: A content filter provider license is expired. When this status is true, the unavailable status is also true.

Renamed Category Warnings

If CPL includes a category that has been renamed in the currently downloaded database, policy warnings occur at compilation time. The following is an example of the warning:

```
Deprecation warning: 'old_category'; 'old_category' has been replaced by 'new_category'
due to Category name updated and will no longer be accepted after Sat, 11 Jul 2020 00:00:00 UTC.
Please switch to the new name before then.
```

To ensure that policy performs as intended, rename all instances of the affected category in CPL and re-apply policy by the specified date.

Syntax

```
server.certificate.hostname.category={status|category_name1[,category_name2,...]|category_group1
[,category_group2,...]}
```

where:

- *status*: One of the following system-defined statuses: none, pending, unavailable, and unlicensed.
- *category_name*: Category names defined by policy or the selected content filter provider.
- *category_group*: Category group names defined by policy or Blue Coat provider if enabled.

  The list of currently valid category names and category group names is available both through the Management Console and CLI.

Layer and Transaction Notes

- Layers: <SSL>
- Transactions: all proxies

Example

This example uses both URL content filtering category definitions and categories provided by a content filtering vendor to restrict SSL access to certain sites.

```
define category Internal
  example1.com
  www.example1.org
end

<Proxy> client.is_ssl
  Allow server.certificate.hostname.category=Internal     ; local definition
  Allow server.certificate.hostname.category=(Sports, Games) ; or vendor supplied
  Allow server.certificate.hostname.category=unavailable action.log_content_filter_down(yes)  ; vendor
down - allow but log
  Deny

define action log_content_filter_down
  log_message( "content filter vendor unavailable to test   $(server.certificate.hostname)" )
end
```

See Also

- Conditions: "category=" on page 85, "request.header.Referer.url.category=" on page 232, "server.certificate.hostname.category=" on the previous page, "server.certificate.subject=" on the next page, "url.category=" on page 307
- Properties: "server.certificate.validate()" on page 536, "server.certificate.validate.check_revocation()" on page 539, "server.certificate.validate.ignore()" on page 540

server.certificate.subject=

Test the subject field of an SSL server certificate.

Syntax

```
server.certificate.subject[.string_modifier][.case_sensitive]=pattern
server.certificate.subject.length=value
```

where:

- *string_modifier*: A supported string modifier:
  - exact - match the string exactly
  - prefix - match the start of a string
  - regex - (default if no string modifier is specified) regular expression match; see "Regex Reference" on page 703
  - substring - match part of a string
  - suffix - match the end of a string
- case_sensitive - Perform case-sensitive match.
- *pattern*: Type-appropriate test expression, such as a quote-delimited string expression or a regular expression.

> **Tip:** The pattern expression supports substitutions. You can specify a substitution expression with the .exact, .substring, .prefix, and .suffix string modifiers where they are available.

Layer and Transaction Notes

- Layers: <SSL>, <SSL-Intercept>
- Transactions: HTTPS forward and reverse proxy transactions, SSL intercept transactions, SSL tunnel transactions

Example

Test if the SSL certificate's subject field contains the specified string.

```
<SSL>
  server.certificate.subject=todo
```

server.connection.dscp=

Test the server-side inbound DSCP value.

Syntax

`server.connection.dscp=`*`dscp_value`*

where:

- *`dscp_value`*: One of the following:
  - decimal value between 0 and 63
  - `best-effort`
  - a class: `af11 | af12 | af13 | af21 | af22 | af23 | af31 | af32 | af33 | af41 | af42 | af43 | cs1 | cs2 | cs3 | cs4 | cs5 | cs6 | cs7 | ef`

Layer and Transaction Notes

- Layers: `<Cache>`, `<Forward>`, `<Proxy>`
- Transactions: all proxy

Example

The first QoS policy rule tests the client inbound QoS/DSCP value against 50, and deny if it matches; the second QoS policy rule tests the client inbound QoS/DSCP value against best-effort, and deny if it matches.

```
<Proxy>
  deny server.connection.dscp=50

<Proxy>
  deny server.connection.dscp=best-effort
```

server.connection.negotiated_cipher=

Test the cipher suite negotiated with a secure server.

Syntax

`server.connection.negotiated_cipher={`*`cipher_suite`*`|none}`

where:

- *`cipher-suite`*: Cipher suite(s) that the appliance supports. Refer to https://www.symantec.com/docs/TECH247556 for cipher suites shipped with the appliance.
- none: No supported cipher suite.

Layer and Transaction Notes

- Layers: `<Proxy>` , `<SSL>`
- Transactions: HTTPS forward and reverse proxy transactions, SSL tunnel transactions

Example

Deny requests to servers that are not using one of the specified cipher suites and log access to servers that are not using secure connections in 'unsecure_log1'.

```
<SSL>
  ALLOW server.connection.negotiated_cipher=(TLS_AES_128_GCM_SHA256 || TLS_AES_256_GCM_SHA384 || TLS_
CHACHA20_POLY1305_SHA256)
  DENY

<Proxy>
  server.connection.negotiated_cipher=none access_log[unsecure_log1](yes)
```

server.connection.negotiated_cipher.strength=

Test the cipher strength negotiated with a securely connected server.

Syntax

```
server.connection.negotiated_cipher.strength=level
```

where:

- *level*: One of the following: low. medium, high, none, export

OpenSSL defines the meanings of high, medium, and low. Refer to OpenSSL ciphers (http://www.openssl.org/docs/apps/ciphers.html) for more information. Currently the definitions are:

- high - Cipher suites with key lengths larger than 128 bits
- medium - Cipher suites with key lengths of 128 bits.
- low - Cipher suites using 64- or 56-bit encryption algorithms but excluding export cipher suites.
- export - Cipher suites using 40- and 56- bits algorithms.

Layer and Transaction Notes

- Layers: <Proxy> , <SSL>
- Transactions: all proxies

Example

Allow only server connections that have a medium or high cipher strength.

```
<Proxy>
  DENY server.connection.negotiated_cipher.strength=(low|export)
```

server.connection.negotiated_ssl_version=

Test the SSL version negotiated with a secure server.

Syntax

```
server.connection.negotiated_ssl_version={SSLV3|TLSV1|TLSV1.1|TLSV1.2|TLS1.3}
```

Layer and Transaction Notes

- Layers: <Proxy> , <SSL>
- Transactions: HTTPS forward and reverse proxy transactions, SSL tunnel transactions

Example

Test if the SSL version is SSLv3.

```
<SSL>
  server.connection.negotiated_ssl_version=SSLV3
```

server_url=

Test if a portion of the URL used in server connections matches the specified criteria. The basic `server_url=` test attempts to match the complete, possibly rewritten, request URL against a specified pattern. The pattern may include the scheme, host, port, path and query components of the URL. If any of these is not included in the pattern, then the corresponding component of the URL is not tested and can have any value.

Specific portions of the URL can be tested by applying URL component modifiers to the condition. In addition to component modifiers, optional test type modifiers can be used to change the way the pattern is matched.

> **Note:** This set of tests match against the requested URL, taking into account the effect of any "rewrite()" on page 615 actions. Because any rewrites of the URL intended for servers or other upstream devices must be respected by `<Forward>` layer policy, the "url=" on page 298 conditions are not allowed in `<Forward>` layers. Instead, the equivalent set of `server_url=` tests are provided for use in the `<Forward>` layer. Those tests always take into account the effect of any "rewrite()" on page 615 actions on the URL.

Syntax

```
server_url.string.modifier=pattern
server_url[.case_sensitive][.no_lookup]=prefix_pattern
server_url.domain[.case_sensitive][.no_lookup]=domain_suffix_pattern
server_url.address={ip_address|ip_address_range|ip_address_wildcards|subnet|subnet_label}
server_url.extension[.case_sensitive]=[.]filename_extension
server_url.has_address={yes|no|restricted|refused|nxdomain|error}
server_url.host[.exact]=host
server_url.host.[prefix|substring|suffix]=string
server_url.host.has_name={yes|no|restricted|refused|nxdomain|error}
server_url.host.is_numeric={yes|no}
server_url.host.no_name={yes|no}
server_url.is_absolute={yes|no}
server_url.path[.case_sensitive]=string
server_url.path[.substring|.suffix][.case_sensitive]=string
server_url.path.regex[.case_sensitive]=regular_expression
server_url.port={range|exact}
server_url.query.regex[.case_sensitive]=regular_expression
server_url.scheme=url_scheme
```

where:

- all options are identical to "url=" on page 298, except for the URL being tested.

> **Tip:** The `server_url=` condition is identical to "url=" on page 298, except there is no `define server_url` condition or `[server_url]` section. Most optimization in forwarding is done with `server_url.domain` conditions and sections.

Layer and Transaction Notes

- Layers: `<Cache>`, `<Forward>`, `<Proxy>` , `<SSL>`, `<SSL-Intercept>`
- Transactions: non-adminstrator transactions

Example

Test if the server URL includes the specified pattern, and block access.

```
; Relative URLs, such as docs subdirectories and pages, will match.
server_url=http://www.example.com/docs access_server(no)

; Test if the URL host’s IP address is a match.
server_url.address=10.1.198.0

; Test whether the URL includes company.com as domain.
server_url.domain=company.com

; Test whether the URL includes .com.
server_url.domain=.com

; Test if the URL includes this domain-suffix pattern and block service.
; Relative URLs, such as docs, subdirectories and pages, will match.
server_url.domain=company.com/docs access_server(no)

; Example of the use of server_url.extension=
server_url.extension=.txt
server_url.extension=(.htm, .html)
server_url.extension=(img, jpg, jpeg)

; How server_url.has_address and server_url.has_name test for DNS responses
server_url.has_address=yes ; DNS lookup succeeded
server_url.has_address=no ; DNS lookup failed
server_url.has_address=restricted ; lookup was restricted because the host or address was included in
a "restrict dns" on page 667 or "restrict rdns" on page 669 definition
server_url.has_name=refused ; DNS server refused response
server_url.has_name=nxdomain ; domain does not exist
server_url.has_name=error ; DNS error response was received

; Trigger the specified exception when the forward DNS lookup of the server URL hostname fails
```

```
server_url.has_address=no exception(policy_denied,"DNS failure")

; Trigger the specified exception when the reverse DNS lookup of the server URL address fails
server_url.host.has_name=no exception(policy_denied,"RDNS lookup failure")

; This example matches the first request and doesn't match the second from
; the following two requests:
; http://1.2.3.4/test
; http://www.example.com
<Forward>
  server_url.host.is_numeric=yes

; The following example assumes that 1.2.3.4 is the IP of the host mycompany
; The condition will match the following two requests if the reverse DNS was successful:
; request http://1.2.3.4/
; request http://mycompany.com/
; If the reverse DNS fails then the first request is not matched
<Forward>
  server_url.host.regex=mycompany

; server_url.path tests
; The following server_url.path strings would all match the example URL:
; http://www.example.com/cgi-bin/query.pl?q=test#fragment
server_url.path="/cgi-bin/query.pl?q=test"
server_url.path="/cgi-bin/query.pl"
server_url.path="/cgi-bin/"
server_url.path="/cgi" ; partial components match too
server_url.path="/" ; Always matches regardless of URL.
; testing the url port
server_url.port=80
```

See Also

- Conditions:"content_management=" on page 116, "url=" on page 298
- Definitions: "define subnet" on page 656, "define server_url.domain condition" on page 652
- Properties: "transform.data_type()" on page 581
- Information on wildcards: http://www.symantec.com/docs/TECH241521
- Information on IP address ranges: http://www.symantec.com/docs/TECH241929

server_url.category=

Matches the content categories of the URL that the appliance sends for a user request. If a URL has been rewritten, the condition matches the categories of the rewritten URL instead of the requested URL.

Content categories can be assigned to URLs by policy (see "define category" on page 635), a local database you maintain, or a third-party database.

If a URL is assigned more than one category, this condition is `true` if it matches any of the assigned categories.

System-Defined Statuses

The following statuses are system-defined:

- `none`: The request is uncategorized by all enabled content filter providers. If even one content filter provider returns a category for the requested site, policy rules containing a `*.category=none` condition will not match.
- `pending`: The categorization request has not been processed or is not complete.
- `unavailable`: A content filter provider is selected in configuration, but an error occurs in determining the category. Other categories can still be assigned directly by policy. This categorization might be a result of a missing database.
- `unlicensed`: A content filter provider license is expired. When this status is true, the `unavailable` status is also true.

> **Note:** If `server_url.category=unlicensed` is true, `server_url.category=unavailable` is true.

Dynamic real-time rating is not available for `server_url.category=`; however, if the URL is not rewritten, any rating result for the original request URL is used.

Renamed Category Warnings

If CPL includes a category that has been renamed in the currently downloaded database, policy warnings occur at compilation time. The following is an example of the warning:

```
Deprecation warning: 'old_category'; 'old_category' has been replaced by 'new_category'
due to Category name updated and will no longer be accepted after Sat, 11 Jul 2020 00:00:00 UTC.
Please switch to the new name before then.
```

To ensure that policy performs as intended, rename all instances of the affected category in CPL and re-apply policy by the specified date.

Syntax

```
server_url.category={status|category_name1[,category_name2,...]|category_group1[,category_group2,...]}
```

where:

- *status*: One of the following system-defined statuses: none, pending, unavailable, and unlicensed.
- *category_name*: Category names defined by policy or the selected content filter provider
- *category_group*: Category group names defined by policy or Blue Coat provider if enabled.

  The list of currently valid category names and category group names is available both through the Management Console and CLI.

Layer and Transaction Notes

- Layers: <Cache>, <Exception>, <Forward>, <Proxy> , <SSL>, <SSL-Intercept>
- Transactions: all proxies

Example

For requests to URLs with the Entertainment category, use the specified forwarding host..

```
<Forward>
  server_url.category="Entertainment" forward(host)
```

See Also

- Conditions: "category=" on page 85, "server_url=" on page 269
- Properties: "access_server()" on page 335

server_url.host.is_private=

Test whether the server URL is within the configured private network.

Syntax

```
server_url.host.is_private={yes|no}
```

Layer and Transaction Notes

- Layers: <Cache>, <Forward>, <Proxy> , <SSL>, <SSL-Intercept>
- Transactions: all proxy transactions with a request URL

Example

Test for server URL within the private network.

```
<Proxy>
  server_url.host.is_private=yes
```

service.group=

Test the service group associated with a transaction.

Syntax

`service.group=`*`group_name`*

where:

- *group_name*:Name of service group.

Layer and Transaction Notes

- Layers: <Admin>, <Exception>, <Forward>, <Proxy> , <SSL>, <SSL-Intercept>
- Transactions: all proxy transactions excluding DNS proxy transactions

Example

Forward the request based on service group name.

```
<Forward>
  service.group=standard forward(standard_proxy)
```

See Also

- Conditions: "service.name=" on the next page

service.name=

Test the service name associated with a transaction.

Syntax

```
service.name=service_name
```

where:

- *service_name*:Name of service.

> **Tip:** In the event that CPL refers to an attribute that was not sent in the RADIUS accounting-start packet, comparisons using this attribute will return false and substitutions will emit the empty string. When compiling policy, CPL will check that each Session Monitor attribute specified is part of the Session Monitor's configuration. CPL will emit an error for attributes that are not part of the configuration.

Layer and Transaction Notes

- Layers: <Admin>, <DNS-Proxy>, <Exception>, <Forward>, <Proxy> , <SSL>, <SSL-Intercept>
- Transactions: all proxy transactions excluding DNS proxy transactions

Example

Forward the request based on service name.

```
<Forward>
  service.name=http_internal forward(internal_proxy)
```

See Also

- Conditions: "service.group=" on the previous page

socks=

This condition is true whenever the session for the current transaction involves SOCKS to the client. The socks=yes condition is intended as a way to test whether a request arrived through the SOCKS proxy. It is true for both SOCKS requests that the ProxySG appliance tunnels and for SOCKS requests the appliance accelerates by handing them off to HTTP. In particular, socks=yes remains true even in the resulting HTTP transactions. Other conditions, such as "proxy.address=" on page 198 or "proxy.port=" on page 200, do not maintain a consistent value across the SOCKS transaction and the later HTTP transaction, so they cannot be reliably used to do this kind of cross-protocol testing.

Syntax

```
socks={yes|no}
```

Layer and Transaction Notes

- Layers: <Exception>, <Forward>, <Proxy> , <SSL>
- Transactions: all transactions

Example

Log a message when SOCKS proxy is used.

```
<Proxy>
  socks=yes log_message("SOCKS proxy used for transaction to $(url)")
```

See Also

- Conditions: "socks.accelerated=" on the next page
- Properties:"socks.accelerate()" on page 553 "socks.authenticate()" on page 554, "socks.authenticate.force()" on page 556, "socks_gateway()" on page 557

socks.accelerated=

Test whether the SOCKS proxy will hand off this transaction to other protocol agents for acceleration.

Syntax

```
socks.accelerated={yes|no|http}
```

where:

- yes: True for SOCKS transactions that will hand off to another protocol-specific proxy agent.
- no: Implies the transaction is a SOCKS tunnel.
- http: True if the transaction will be accelerated by the HTTP proxy.

Layer and Transaction Notes

- Layers: <Exception>, <Proxy>
- Transactions: SOCKS proxy

Example

Test whether the SOCKS proxy will hand off the transaction to the HTTP proxy for acceleration.

```
<Proxy>
  socks.accelerated=http
```

See Also

- Conditions: "socks.method=" on the facing page, "socks.version=" on page 280
- Properties:"socks.accelerate()" on page 553, "socks.authenticate()" on page 554, "socks.authenticate.force()" on page 556, "socks_gateway()" on page 557

socks.method=

Tests the SOCKS protocol method name associated with the transaction.

Syntax

```
socks.method={CONNECT|BIND|UDP_ASSOCIATE}
```

Layer and Transaction Notes

- Layers: `<Exception>`, `<Proxy>`
- Transactions: SOCKS proxy

Example

Test whether the transaction uses SOCKS CONNECT method.

```
<Proxy>
  socks.method=CONNECT
```

See Also

- Conditions: "ftp.method=" on page 139, "http.method=" on page 155, "server_url=" on page 269, "socks.version=" on the next page
- Properties:"socks.accelerate()" on page 553"socks.authenticate()" on page 554, "socks.authenticate.force()" on page 556, "socks_gateway()" on page 557

socks.version=

Tests whether the version of the SOCKS protocol used to communicate to the client is SOCKS 4/4a or SOCKS 5. SOCKS 5 has more security and is more highly recommended.

SOCKS 5 supports authentication and can be used to authenticate transactions that may be accelerated by other protocol services.

SOCKS 4/4a does not support authentication. If "socks.authenticate()" on page 554 or "socks.authenticate.force()" on page 556is set during evaluation of a SOCKS 4/4a transaction, that transaction will be denied.

Syntax

socks.version={4|5}

Layer and Transaction Notes

- Layers: `<Exception>`, `<Forward>`, `<Proxy>`
- Transactions: SOCKS proxy; does not apply to administrator transactions

Example

Authenticate SOCKS v5 clients, and allow only a known set of client IP addresses to use SOCKS v4/4a.

```
<Proxy>
 socks.version=5 socks.authenticate(my_realm)
 deny socks.version=4 client.address=!old_socks_allowed_subnet
```

See Also

- Conditions: "socks=" on page 277, "socks.method=" on the previous page
- Properties:"socks.accelerate()" on page 553, "socks.authenticate()" on page 554, "socks.authenticate.force()" on page 556, "socks_gateway()" on page 557

source.port=

Test the port that the client connects from.

Syntax

`source.port=`*`port_number`*

Layer and Transaction Notes

- Layers: `<Admin>`, `<DNS-Proxy>`, `<Forward>`, `<Proxy>` , `<SSL>`, `<SSL-Intercept>`
- Transactions: all proxies

Example

Test for client connections on port 8080.

```
<Admin>
  source.port=8080
```

ssl.proxy_mode=

Test if the appliance is intercepting and decrypting an SSL connection.

Syntax

```
ssl.proxy_mode={yes|no|https-reverse-proxy|https-forward-proxy}
```

where:

- yes: Test for both HTTPS forward and reverse proxy requests.
- no: Do not test if appliance is intercepting and decrypting SSL.
- https-reverse-proxy: Test for HTTPS reverse proxy requests.
- https-forward-proxy: Test for HTTPS forward proxy requests.

Layer and Transaction Notes

- Layers: <Proxy> , <SSL>
- Transactions: all proxies

Example

Test if an HTTPS reverse proxy request is being terminated.

```
<Proxy>
 ssl.proxy_mode=https-reverse-proxy
```

streaming.client=

Tests the client agent associated with the current transaction.

Syntax

`streaming.client={yes|no|`*`streaming_client`*`}`

where:

- *`streaming_client`*: One of the following: `windows_media`, `real_media`, `quicktime`, `flash`, `ms_smooth`, `adobe_hds`, `apple_hls`

Layer and Transaction Notes

- Layers: `<Cache>`, `<Exception>`, `<Forward>`, `<Proxy>`
- Transactions: HTTP and streaming transactions; does not apply to administrator transactions

Example 1

Force Smooth Streaming to be cached, which is useful when a CDN says cacheable MS Smooth traffic HTTP responses are non-cacheable.

```
<Cache>
  streaming.client=ms_smooth force_cache(yes)
```

Example 2

Disable ADN byte compression because most video traffic can't be compressed efficiently.

```
<Forward>
  streaming.client=ms_smooth adn.server.optimize.compress(no)
```

See Also

- Conditions: "bitrate=" on page 83, "live=" on page 192, "streaming.content=" on the next page
- Properties: "access_server()" on page 335, "max_bitrate()" on page 494, "streaming.transport()" on page 569

streaming.content=

Tests the content of the current transaction to determine whether it is streaming media, and to determine the streaming media type.

Syntax

`streaming.content={yes|no|`*`content_type`*`}`

where:

- *`content_type`*: One of the following: *`windows_media`*, *`real_media`*, *`quicktime`*, *`flash`*.

Layer and Transaction Notes

- Layers: `<Cache>`, `<Exception>`, `<Proxy>`
- Transactions: all transactions

Example

For Flash content, enforce the configured volume quota.

```
<Proxy>
 streaming.content=flash variable.volume_quota_enforced(true)
```

See Also

- Conditions: "bitrate=" on page 83, "live=" on page 192, "streaming.client=" on the previous page
- Properties: "access_server()" on page 335, "max_bitrate()" on page 494, "streaming.transport()" on page 569

streaming.rtmp.app_name=

Identifies the name of the Flash application of which the stream is a part.

Syntax

`streaming.rtmp.app_name=`*`string`*

Layer and Transaction Notes

- Layers: `<Cache>`, `<Exception>`, `<Proxy>`
- Transactions: Flash streaming

Example

```
<Proxy>
  streaming.rtmp.app_name=vod
```

See Also

- Conditions: "streaming.rtmp.method=" on the next page, "streaming.rtmp.page_url=" on page 287, "streaming.rtmp.stream_name=" on page 288, "streaming.rtmp.swf_url=" on page 289

streaming.rtmp.method=

Identifies the Flash operation on which policy is being applied.

Syntax

```
streaming.rtmp.method={open|connect|play}
```

Layer and Transaction Notes

- Layers: <Cache>, <Exception>, <Proxy>
- Transactions: Flash streaming

Example

Tests whether the Flash method is play.

```
<Proxy>
  streaming.rtmp.method=play
```

See Also

- Conditions: "streaming.rtmp.app_name=" on the previous page, "streaming.rtmp.page_url=" on the facing page, "streaming.rtmp.stream_name=" on page 288, "streaming.rtmp.swf_url=" on page 289

streaming.rtmp.page_url=

Identifies the URL of the web page that is embedding the Flash plugin.

Syntax

`streaming.rtmp.page_url={`*`url`*`|`*`ip_address_wildcards`*`|`*`ip_address_range`*`}`

where:

- *`url`*: Requested URL.
- *`ip_address_range`*: IP address range; for example, 192.0.2.0-192.0.2.255
- *`ip_address_wildcards`*: IP address specified using wildcards in any octet(s); for example, 10.25.*.0 or 10.*.*.0

Layer and Transaction Notes

- Layers: `<Cache>`, `<Exception>`, `<Proxy>`
- Transactions: Flash proxy

Example

Test for Flash embedded in the specified webpage.

```
<Proxy>
  streaming.rtmp.page_url=http://www.mysite.com/videoplayer/videoplayer.html
```

See Also

- Conditions: "streaming.rtmp.app_name=" on page 285, "streaming.rtmp.method=" on the previous page, "streaming.rtmp.stream_name=" on the next page, "streaming.rtmp.swf_url=" on page 289
- Information on wildcards: http://www.symantec.com/docs/TECH241521
- Information on IP address ranges: http://www.symantec.com/docs/TECH241929

streaming.rtmp.stream_name=

Tests for a specific Flash stream being requested.

Syntax

`streaming.rtmp.stream_name=`*`string`*

Layer and Transaction Notes

- Layers: `<Cache>`, `<Exception>`, `<Proxy>`
- Transactions: Flash proxy

Example

Test for Flash stream with the specified name.

```
<Proxy>
  streaming.rtmp.stream_name=sample.flv
```

See Also

- Conditions: "streaming.rtmp.app_name=" on page 285, "streaming.rtmp.method=" on page 286, "streaming.rtmp.page_url=" on the previous page, "streaming.rtmp.swf_url=" on the facing page

streaming.rtmp.swf_url=

Test for the URL of the SWF file being played within the Flash plugin.

Syntax

`streaming.rtmp.swf_url={`*`url`*`|`*`ip_address_wildcards`*`|`*`ip_address_range`*`}`

where:

- *`url`*: Requested URL.
- *`ip_address_range`*: IP address range; for example, 192.0.2.0-192.0.2.255
- *`ip_address_wildcards`*: IP address specified using wildcards in any octet(s); for example, 10.25.*.0 or 10.*.*.0

Layer and Transaction Notes

- Layers: `<Cache>`, `<Exception>`, `<Proxy>`
- Transactions: Flash proxy

Example

Test for the specified SWF file within the Flash plugin.

```
<Proxy>
  streaming.rtmp.swf_url=http://www.mysite.com/videoplayer/swfs/videoplayer.swf
```

See Also

- Conditions: "streaming.rtmp.app_name=" on page 285, "streaming.rtmp.method=" on page 286, "streaming.rtmp.page_url=" on page 287, "streaming.rtmp.stream_name=" on the previous page,
- Information on wildcards: http://www.symantec.com/docs/TECH241521
- Information on IP address ranges: http://www.symantec.com/docs/TECH241929

supplier.country

After a client makes a request to an Origin Content Server (OCS), the server hostname resolves to one or more IP addresses. You can make policy decisions based on the geographical location of the IP address of the specific host from which the appliance retrieved content for the response. Use this condition to perform actions after initial contact to the host.

> **Note:** Because a connection to the host must succeed before the appliance evaluates this condition, potential security risks exist:
> - POST request data could be sent to a malicious OCS, before the condition is evaluated.
> - When the appliance completes the connection, it receives the first part of the response before the condition is evaluated. Should the condition result in a denial, the appliance will have received at least some of the data before the denial occurs.

If you receive an "Obsolete country name" warning when installing policy, replace the outdated country name in policy with the suggested name in the message.

Syntax

`supplier.country=`*`country_name`*

where:

- *`country_name`*: Full country name. If the country name includes punctuation or a space (such as United States), enclose the name in single quotation marks ('United States'). Alternatively, use the ISO two-letter code name, such as US for the United States.

  To see the list of countries in the geolocation database and their ISO code names, go to https://*IP_address*:*port*/Geolocation/Countries.

Layer and Transaction Notes

- Layers: `cache>`,`<Proxy>` , `<SSL>`
- Transactions: all transactions

Example 1

Scan files originating from China (CN).

```
; Scan files originating from IP addresses in China (CN)
<Proxy>
  supplier.country=CN AV_SCAN
```

Example 2

Analyze the response data type for connections to Russia (RU) and terminates the connection if it determines that the payload is an executable file.

```
; Deny executables originating from IP addresses in Russia (RU)
<Proxy>
  http.response.apparent_data_type=executable supplier.country=RU deny
```

Example 3

This is an example of when this condition is not useful. Unlike the previous two examples, this policy does not rely on post-connection data to make a decision (in this case, a denial).

```
; Deny access to IP addresses in China (CN)
<Proxy>
  supplier.country=CN deny
```

The intention of this policy is to deny connections to China, but until the appliance retrieves the content from the host, the connection is allowed. To write deny policy based on country, use the"supplier.allowed_countries()" on page 570property, which applies before the appliance attempts to connect to the OCS.

See Also

- Properties: "supplier.allowed_countries()" on page 570

time=

Tests if the time of day is in the specified range or an exact match. The current time is determined by the system configured clock and time zone by default, although the UTC time zone can be specified by using the form `time.utc=`. The numeric pattern used to test the time condition can contain no whitespace.

Syntax

`hour[.utc]={[start_time]..[end_time]|exact_time}`

where:

- *start_time*: Four digits (nnnn) in 24-hour time format representing the start of a time range; for example, 0900 specifies 9:00 a.m. If left blank, midnight (0000) is assumed.
- *end_time*: Four digits (nnnn) in 24-hour time format representing the end of a time range; for example, 1700 specifies 5:00 p.m. If left blank, 2359 (11:59 p.m.) is assumed.
- *exact_time*: Four digits (nnnn) in 24-hour time format representing an exact time.

> **Note:** To test against an inverted range, such as a range that crosses from one hour into the next, the following shorthand expression is available. While `time=(..0600|1900..)` species midnight to 6 a.m. and 7 p.m. to midnight, the policy language also recognizes `time=1900..0600` as equivalent.

Layer and Transaction Notes

- Layers: `<Admin>`, `<Cache>`, `<Diagnostic>`, `<DNS-Proxy>`, `<Exception>`, `<Forward>`, `<Proxy>` , `<SSL>`, `<SSL-Intercept>`
- Transactions: all

> **Note:** Using time-related conditions to control caching behavior in a `<Cache>` layer may cause thrashing of the cached objects.

Example

Temporarily restrict access to websites based on time.

```
; Tests for 3 a.m. to 1 p.m. UTC.
time.utc=0300..1300

; Allow access to a particular site only during 9 a.m.
```

```
; to noon UTC (presented in two forms).
; Restrict form:
<Proxy>
  deny url.host=special_event.com time=!0900..1200

; Grant form:
<Proxy>
  allow url.host=special_event.com time=0900..1200

; This example restricts the times during which certain
; stations can log in with administrative privileges.
define subnet restricted_stations
 10.10.10.4/30
 10.10.11.1
end

<Admin> client.address=restricted_stations
  allow time=0800..1800 weekday=1..5 admin.access=(READ||WRITE);
  deny
```

See Also

- Conditions: date[.utc]=, "day=" on page 119, "hour=" on page 147, "minute=" on page 193, "month=" on page 195, "weekday=" on page 328, "year=" on page 330

transaction.field.*name*=

Test any access log fields with type-appropriate criteria. Access log timestamps cannot be tested.

Syntax

`transaction.field.`*`name`*`=`*`type_appropriate_criterion`*

where:

- *name*: Valid access log field name. Refer to *ProxySG Log Fields and CPL Substitutions Reference* for more information.
- *type_appropriate_criterion*: Valid criterion depending on the access log field.

Layer and Transaction Notes

- Layers: `<Diagnostic>`
- Transactions: all

Example

Trace any transaction going to sample_domain.com that takes longer than 3 seconds, based on the `time-taken` access log field value.

```
<Diagnostic> trace.request(yes)
  url.domain=sample_domain.com transaction.field.time-taken=3000..
```

See Also

- Conditions: "transaction.type=" on the facing page
- *ProxySG Log Fields and CPL Substitutions Reference*

transaction.type=

Test for the specified transaction type(s).

> **Tip:** To determine the transaction type when troubleshooting, you can run a policy trace. The trace displays the transaction type near the start of transaction deta:
>
> ```
> start transaction ------------------
> transaction ID=52685 type=ssl.tunnel
> transaction handed off from: 52684
> ..
> ```

Syntax

`transaction.type=`*`type1`*`[,`*`type2`*`,..]`

where:

- *type*: Transaction type. **Show transaction types**
  http.proxy, http.refresh, http.pipeline, http.content-pull, http.document, http.internal, http.diagnostics, http.health-check, https.reverse-proxy, https.forward-proxy, https.refresh, https.pipeline, https.content-pull, https.document, https.diagnostics, https.health-checks, ftp.proxy, ftps, ftp.content-pull, mms.proxy, mms.noauth.proxy, mms.content-pull, mms.internal, rtmp.proxy, rtmp.content-pull, rtsp.proxy, rtsp.content-pull, rtsp.internal, aol-im.proxy, msn-im.proxy, yahoo-im.proxy, socks.proxy, socks.internal, tcp.tunnel, admin.https.cag, admin.http.cag, admin.telnet, admin.serial, admin.ssh, admin.https.init-cfg, admin.https.mgmt, admin.http.mgmt, admin.https.proxy-client, clientless, icp, dns.proxy, telnet.proxy, p2p.proxy, epmapper.proxy, epmapper.tunnel, ssl.intercept, ssl.tunnel, ssl.reverse-proxy, ssl.forward-proxy, msrpc.proxy, cifs.proxy, cifs.content-pull, ocsp.http, ocsp.https, icmp.health-check, tcp.health-check, ssl.health-check, srtr.health-check, authentication.health-check, dns.health-check, hsm.health-check, snmp, syslog.relay, webex.proxy, drtr.document, mapihttp.proxy, sip, sip-ssl, msturn

Layer and Transaction Notes

- Layers: `<Diagnostic>`
- Transactions: all

Example

Log an event log message for IM transactions.

```
define action logIM
  log_message("Using unsupported IM")
end
```

```
<Diagnostic>
  transaction.type=aol-im.proxy,msn-im.proxy,yahoo-im.proxy action.logIM(yes)
```

See Also

- Conditions: "transaction.field.name=" on page 294

tunneled=

Tests if the current transaction represents a tunneled request. A tunneled request is one of:

- TCP tunneled request
- HTTP CONNECT request
- Unaccelerated SOCKS request

> **Note:** HTTPS connections to the management console are not tunneled for the purposes of this test.

Syntax

```
tunneled={yes|no}
```

Layer and Transaction Notes

- Layers: <Exception>, <Forward>, <Proxy> , <SSL>
- Transaction: all proxies

Example

Deny tunneled transactions except when they originate from the corporate subnet.

```
define subnet corporate_subnet
  10.1.2.0/24
  10.1.3.0/24
end

<Proxy>
  deny tunneled=yes client.address=!corporate_subnet
```

See Also

- Conditions: "http.method=" on page 155, "socks.accelerated=" on page 278, url.scheme=
- Properties:"socks.accelerate()" on page 553

url=

Tests if a portion of the requested URL matches the specified criteria. The basic `url=` test attempts to match the complete request URL against a specified pattern. The pattern may include the scheme, host, port, path and query components of the URL. If any of these is not included in the pattern, the corresponding component of the request URL is not tested and can have any value.

Specific portions of the URL can be tested by applying URL component modifiers to the condition. In addition to component modifiers, optional test type modifiers can be used to change the way the pattern is matched.

> **Note:** This set of tests match against the originally requested URL, disregarding the effect of any "rewrite()" on page 615 actions. Because any rewrites of the URL intended for servers or other upstream devices must be respected by `<Forward>` layer policy, the `url=` conditions are not allowed in `<Forward>` layers. Instead, an equivalent set of "server_url=" on page 269 tests are provided for use in the `<Forward>` layer. Those tests always take into account the effect of any "rewrite()" on page 615 actions on the URL.

Syntax

```
url.string.modifier=pattern
url[.case_sensitive][.no_lookup]=prefix_pattern
url.domain[.case_sensitive][.no_lookup]=domain_suffix_pattern
url.address={ip_address|ip_address_range|ip_address_wildcards|subnet|subnet_label}
url.extension[.case_sensitive]=[.]filename_extension
url.has_address={yes|no|restricted|refused|nxdomain|error}
url.host[.exact]=host
url.host.[prefix|substring|suffix]=string
url.host.has_name={yes|no|restricted|refused|nxdomain|error}
url.host.is_numeric={yes|no}
url.host.no_name={yes|no}
url.is_absolute={yes|no}
url.path[.case_sensitive]=string
url.path[.substring|.suffix][.case_sensitive]=string
url.path.regex[.case_sensitive]=regular_expression
url.port={range|exact}
url.query.regex[.case_sensitive]=regular_expression
url.scheme=url_scheme
```

where:

- *prefix_pattern*: URL pattern that includes at least a portion of `scheme://host:port/path`. Accepted prefix patterns include the following:

- scheme://host
- scheme://host:port
- scheme://host:port/path_query
- scheme://host/path_query
- //host
- //host:port
- //host:port/path_query
- //host/path_query
- host
- host:port
- host:port/path_query
- host/path_query
- /path_query

- *domain_suffix_pattern*: URL pattern that includes a domain suffix, as a minimum, using syntax scheme://domain_suffix:port/path. Accepted domain suffix patterns include the following:
  - scheme://domain_suffix
  - scheme://domain_suffix:port
  - scheme://domain_suffix:port/path_query
  - scheme://domain_suffix/path_query
  - //domain_suffix
  - //domain_suffix:port
  - //domain_suffix:port/path_query
  - //domain_suffix/path_query
  - domain_suffix
  - domain_suffix:port
  - domain_suffix:port/path_query
  - domain_suffix/path_query

- *url.scheme*: One of the following: http, https, ftp, mms, rtsp, icp, tcp

  The request URL has the scheme https only in the case of SSL termination. A request URL with the scheme tcp only has a host and a port, and occurs in two cases: when a connection is made to a TCP tunnel service port, and when the CONNECT method is used in an explicitly proxied HTTP request. For example, when the Web browser has an explicit HTTP proxy and the user requests an HTTPS URL, the browser creates a TCP tunnel using the CONNECT method.

- *host*: domain name or IP address. Host names must be complete; for example, url=http://www fails to match a URL such as http://www.example.com. This use of a complete host instead of a domain_suffix (such as example.com) indicates the difference between the url= and url.domain= conditions.

- *domain_suffix*: Pattern which matches either a complete domain name or is a suffix of the domain name, respecting component boundaries. An IP address is not allowed. This use of a domain_suffix pattern instead of a complete host name marks the difference between the url.domain= and url= conditions.

- *port*: Port number, between 1 and 65535.

- *path_query*: Portion of a URL is the string beginning with '/' that follows the host and port, and precedes any URL fragment. A path_query pattern is a string beginning with a '/' that matches the beginning of the path_query.

- *filename_extension*: String representing a filename extension to be tested, optionally preceded by a period (.). A quoted empty string (url.extension="") matches URLs that do not include a filename extension, such as http://example.com/ and http://example.com/test. To test multiple extensions, use parentheses and a comma separator (see the Example section below).

- *regular_expression*: Perl regular expression. The expression must be quoted if it contains whitespace or any of the following: & | ( ) < > { } ; ! . = " '. For more information, see *ProxySG Log Fields and CPL Substitutions Reference.*

Objects with paths relative to the prefix_pattern and domain_suffix_pattern are also considered a match (see the "Example" section).

The following are test modifiers:

- .case_sensitive—By default, all matching is case-insensitive; however, the matches on the path and query portions can be made case-sensitive by using the form url.case_sensitive=.

- .domain—Changes the way the match is performed on the host portion of the URL. The host pattern is a domain_suffix pattern which either matches the hostname exactly, or matches a suffix of the hostname on component boundaries. The host is converted to a domain name by reverse DNS lookup if necessary. For example, the condition url.domain=//example.com matches the request URL http://www.example.com/, but does not match the request URL http://www.myexample.com/.

- .exact—Forces an exact string comparison on the full URL or component.

- .no_lookup—Depending on the form of the request's host and the form of the pattern being matched, a DNS or reverse DNS lookup is performed to convert the request's host before the comparison is made. This lookup can be suppressed by using the .no_lookup= form of the condition. The .no_lookup modifier speeds up policy evaluation, but use of it may

introduce loopholes into your security policy that can be exploited by those who want to bypass your security measures. DNS and reverse DNS lookups can be globally restricted by restrict definitions.

The `.no_lookup` option should only be applied to url, url.host and url.domain conditions.

When applied to the `url.host=` condition, if the URL host was specified as an IP address, the behavior depends on whether the no_lookup modifier was specified. If no_lookup was specified, then the condition is false. If no_lookup was not specified, then a reverse DNS lookup is performed to convert the IP address to a domain name. If the reverse DNS lookup fails, then the condition is false.

> **Note:** If you deny the URL using `url=http://www.sex.com/` for example, you cannot bypass the policy by simply requesting the IP address of www.sex.com, since the underlying DNS lookup exists for that particular condition. If, however, you deny the URL by using `url.exact=http://www.sex.com`, you can bypass the policy by simply using the IP address of that site.

When applied to the `url.host=` condition, this pattern match is always case-insensitive.

- `.prefix`–Test if the string pattern is a prefix of the URL or component.
- `.regex`–Test the URL or component against a regular_expression pattern.

  When applied to the `url.host=` condition, if the URL host was specified as an IP address, the behavior depends on whether the no_lookup modifier was specified. If no_lookup was specified, then the condition is false. If no_lookup was not specified, then a reverse DNS lookup is performed to convert the IP address to a domain name. If the reverse DNS lookup fails, then the condition is false. This leads to the following edge conditions: `url.host.regex=!""` has the same truth value as `url.host.no_name=yes`, and `url.host.regex.no_lookup=!""` has the same truth value as `url.host.is_numeric=yes`.

  When applied to the `url.host=` condition, this pattern match is always case-insensitive.

- `.substring`–Test if the string pattern is a substring of the URL or component. The substring need not match on a boundary (such as a subdomain or path directory) within a component.
- `.suffix`–Test if the string pattern is a suffix of the URL or component. The suffix need not match on a boundary (such as a domain component or path directory) within a URL component.

> **Note:** .prefix, .regex, .substring, and .suffix are string comparisons that do not require a match on component boundaries. For this reason, `url.host.suffix=` differs from the host comparison used in `url.domain=` tests, which does require component level matches.

The URL component modifiers are:

- `.address`—Tests if the host IP address of the requested URL matches the specified IP address, IP subnet, or subnet definition. If necessary, a DNS lookup is performed on the host name. DNS lookups can be globally restricted by a restrict DNS definition.

  The patterns supported by the `url.address=` test are:

  - *`ip_address`*—Host IP address or subnet; for example, 10.1.198.0.
  - *`ip_address_wildcards`*—IP address specified using wildcards in any octet(s); for example, 10.25.*.0 or 10.*.*.0.
  - *`ip_address_range`*—IP address range; for example, 192.0.2.0-192.0.2.255.
  - *`subnet`*—A subnet mask; for example, 10.1.198.0/24.

- `subnet_label`—Label of a subnet definition block that binds a number of IP addresses or subnets.

  The .address modifier is primarily useful when the expression uses either a subnet or a subnet_label. If a literal ip_address is used, then the url.address= condition is equivalent to `url.host=`.

- `.host`—Tests the host component of the requested URL against the following, as specified by the host pattern:

  - In a transparent proxy deployment, the IP address to which the client made the request.
  - In an explicit proxy deployment, the hostname from the HTTP CONNECT request.

  If server name indication (SNI) information is available in explicit and transparent HTTPS proxy connections, the SNI is used for the URL host. If SNI is not implemented on the server, the default behavior specified above occurs.

  The pattern cannot include a forward slash (/) or colon (:). It does not recognize wild cards or suffix matching. Matches are case-insensitive. The default test type is .exact.

  > **Note:** `url.host.exact=` can be tested using hash techniques rather than string matches, and will therefore have significantly better performance than other, string based, versions of the `url.host=` tests.

  Because the host component of a request URL can be either an IP address or a domain name, a conversion is sometimes necessary to allow a comparison.

  - If the expression uses a domain name and the host component of the request URL is an IP address, then the IP address is converted to a domain name by doing a reverse DNS lookup.
  - If the expression uses an IP address and the host component of the request URL is a domain name, then the domain name is converted to an IP address by doing a DNS lookup.

    The `.host` component supports additional test modifiers:

- `.has_name`—This is true if the URL host can be mapped to a hostname.
- `.is_numeric`—This is true if the URL host was specified as an IP address. For some types of transactions (for example, transparent requests on a non-accelerated port), this condition will always be true.
- `.no_name`—This is true if no domain name can be found for the URL host. Specifically, it is true if the URL host was specified as an IP address, and a reverse DNS lookup on this IP address fails, because it returns no name or a network error occurs.

◦ `.path`—Tests the path component of the request URL. By default, the pattern is tested as a prefix of the complete path component of the requested URL, as well as any query component. The path and query components of a URL consist of all text from the first forward slash (/) that follows the host or port, to the end of the URL, not including any fragment identifier. The leading forward slash is always present in the request URL being tested, because the URL is normalized before any comparison is performed. Unless an .exact, .substring, or .regex modifier is used, the pattern specified must include the leading '/' character.

In the following URL example, `?q=test` is the query component used in the comparison and `#fragment` is the ignored fragment identifier:

`http://www.example.com/cgi-bin/query.pl?q=test#fragment`

A URL such as the following is normalized so that a forward slash replaces the missing path component: http://www.example.com becomes http://www.example.com/.

◦ `.port`—Tests if the port number of the requested URL is within the specified range or an exact match. URLs that do not explicitly specify a port number have a port number that is implied by the URL scheme. The default port number is 80 for HTTP, so url.port=80 is true for any HTTP-based URL that does not specify a port.

The patterns supported by the `url.address=` test are:

- *`low_port_number`*—Number at the low end of the range to be tested. Can be a number between 1 and 65535.
- *`high_port_number`*—Number at the high end of the range to be tested. Can be a number between 1 and 65535.
- *`exact_port_number`*—Single port number; for example, 80. Can be a number between 1 and 65535.

The numeric pattern used to test the `url.port=` condition can contain no whitespace.

◦ `.query`—Tests if the regex matches a substring of the query string component of the request URL. If no query string is present, the test is false. As a special case, `url.query.regex=!""` is true if no query string exists.

The query string component of the request URL, if present, consists of all text from the first question mark (?) following the path to the end of the URL. Note that pound (#) characters following the ? are included in the query string for compatibility with certain Web applications. If a query string component exists, it begins with a ? character.

- `.scheme`—Tests if the scheme of the requested URL matches the specified schema string. The comparison is always case-insensitive.

Discussion

The `url=` condition can be considered a convenient way to do testing that would require a combination of the following conditions: `url.scheme=`, `url.host=`, `url.port=`, and `url.path=`.

For example, `url=http://example.com:8080/index.html` is equivalent to:

```
url.scheme=http url.host=example.com url.port=8080 url.path=/index.html
```

If you are testing a large number of URLs using the `url=` condition, consider the performance benefits of a "define url condition" on page 658 block or a `[url]` section.

If you are testing a large number of URLs using the `url.domain=` condition, consider the performance benefits of a"define url.domain condition" on page 660 block or a `[url.domain]` section.

Regular expression matches are not anchored. You may want to use either or both of the ^ and $ operators to anchor the match. Alternately, use the .exact, .prefix, or .suffix form of the test, as appropriate.

Layer and Transaction Notes

- Layers: `<Cache>`, `<Exception>`, `<Proxy>` , `<SSL>`, `<SSL-Intercept>`, `<Tenant>`
- Transactions: all non-administrator transactions

Example

```
; comment text; Test if the URL includes this pattern, and block service
; Relative URLs, such as docs subdirectories and pages, will match.
url=http://www.example.com/docs max_bitrate(no)

; Test if the URL host’s IP address is a match.
url.address=10.1.198.0

; Test whether the URL includes company.com as domain.
 url.domain=company.

; Test whether the URL includes .com.
url.domain=.com

; Test if the URL includes this domain-suffix pattern,
; and block service. Relative URLs, such as docs
; subdirectories and pages, will match.
url.domain=company.com/docs max_bitrate(no)

; examples of the use of url.extension=
```

```
url.extension=.txt
url.extension=(.htm, .html)
url.extension=(img, jpg, jpeg)

; How url.has_address and url.has_name test for DNS responses
url.has_address=yes ; DNS lookup succeeded
url.has_address=no ; DNS lookup failed
url.has_address=restricted ; lookup was restricted because the host or address was included in a
"restrict dns" on page 667 or "restrict rdns" on page 669 definition
url.has_name=refused ; DNS server refused response
url.has_name=nxdomain ; domain does not exist
url.has_name=error ; DNS error response was received
; Trigger the specified exception when the forward DNS lookup of the request URL hostname fails
url.has_address=no exception(policy_denied,"DNS lookup failure")

; Trigger the specified exception when the reverse DNS lookup of the request URL address fails
url.host.has_name=no exception(policy_denied,"RDNS lookup failure")

; This example matches the first request and doesn’t match the second from
; the following two requests:
; http://1.2.3.4/test
; http://www.example.com
<Proxy>
  url.host.is_numeric=yes

; In the example below we assume that 1.2.3.4 is the IP of the host mycompany
; The condition will match the following two requests if the reverse DNS was successful:
; request http://1.2.3.4/
; request http://mycompany.com/
; If the reverse DNS fails then the first request is not matched

<Proxy>
  url.host.regex=mycompany

; url.path tests
; The following server_url.path strings would all match the example URL:
; http://www.example.com/cgi-bin/query.pl?q=test#fragment
url.path="/cgi-bin/query.pl?q=test"
url.path="/cgi-bin/query.pl"
url.path="/cgi-bin/"
url.path="/cgi" ; partial components match too
url.path="/" ; Always matches regardless of URL.
; testing the url port
url.port=80
```

See Also

- Conditions: "category=" on page 85, "console_access=" on page 115, "content_management=" on page 116, "server_url=" on page 269
- Definitions: "define subnet" on page 656, "define server_url.domain condition" on page 652
- Information on wildcards: http://www.symantec.com/docs/TECH241521
- Information on IP address ranges: http://www.symantec.com/docs/TECH241929

url.category=

Test the content filter categories of the request URL. Synonym for "category=" on page 85.

System-Defined Statuses

The following statuses are system-defined:

- none: The request is uncategorized by all enabled content filter providers. If even one content filter provider returns a category for the requested site, policy rules containing a `*.category=none` condition will not match.
- pending: The categorization request has not been processed or is not complete.
- unavailable: A content filter provider is selected in configuration, but an error occurs in determining the category. Other categories can still be assigned directly by policy. This categorization might be a result of a missing database.
- unlicensed: A content filter provider license is expired. When this status is true, the unavailable status is also true.

Renamed Category Warnings

If CPL includes a category that has been renamed in the currently downloaded database, policy warnings occur at compilation time. The following is an example of the warning:

```
Deprecation warning: 'old_category'; 'old_category' has been replaced by 'new_category'
due to Category name updated and will no longer be accepted after Sat, 11 Jul 2020 00:00:00 UTC.
Please switch to the new name before then.
```

To ensure that policy performs as intended, rename all instances of the affected category in CPL and re-apply policy by the specified date.

Syntax

```
url.category={status|category_name1[,category_name2,...]|category_group1[,category_group2,...]}
```

where:

- *status*: One of the following system-defined statuses: none, pending, unavailable, and unlicensed.
- *category_name*: Category names defined by policy or the selected content filter provider.
- *category_group*: Category group names defined by policy or Blue Coat provider if enabled.

  The list of currently valid category names and category group names is available both through the Management Console and CLI.

Layer and Transaction Notes

- Layers: <Cache>, <Exception>, <Proxy> , <SSL>, <SSL-Intercept>
- Transactions: proxy transactions with a request URL

Example

Test for Sports category.

```
<Cache>
  url.category=Sports
```

See Also

- Conditions: "category=" on page 85, "request.header.Referer.url.category=" on page 232, "server.certificate.hostname.category=" on page 262

url.host.is_private=

Test whether the request URL is within the configured private network.

Syntax

```
url.host.is_private={yes|no}
```

Layer and Transaction Notes

- Layers: <Cache>, <Exception>, <Proxy> , <SSL>, <SSL-Intercept>
- Transactions: proxy transactions with a request URL

Example

Test for request URL within the private network.

```
<Proxy>
  url.host.is_private=yes
```

url.threat_risk.level=

The Threat Risk Levels service assigns threat risk levels to URLs according to specific criteria. This condition tests for the threat risk level associated with the URL.

For details on the Threat Risk Levels service, refer to “Analyzing the Threat Risk of a URL” in the *SGOS Administration Guide*.

Syntax

url.threat_risk.level=*risk_level*

where:

- *url*: One of the following, depending on the policy layer:
  - dns.request in the <DNS-Proxy> layer; tests the threat risk level of hostnames in DNS IPv6 queries in addition to DNS IPv4 queries.
  - server.certificate.hostname in the <SSL -Intercept> and <SSL> layers
  - server_url in the <Forward> layer
  - url in the <Proxy> and <Cache> layers
- *risk_level*: One of the following:
  - An integer or range of integers from 1 to 10, representing the severity level of the threat risk, for example:
    - url.threat_risk.level=10 — Threat risk level 10
    - server.certificate.hostname=8..10 — Threat risk levels 8 through 10
    - dns.request.threat_risk.level=7.. — Threat risk level 7 and higher
    - server_url.threat_risk.level=..4 — Threat risk level 4 and lower
  - 0 to override the threat risk assigned to the URL.
  - One of four system-defined values: none, pending, unavailable, and unlicensed.

Refer to the following tables for descriptions of these threat risk levels:

Level(s)	Description
1, 2	Low. The URL has an established history of normal behavior and has no future predictors of threats; however, this level should be evaluated by other layers of defense (such as Content Analysis and Malware Analysis).
3, 4	Medium-Low. The URL has an established history of normal behavior, but is less established than URLs in the Low group. This level should be evaluated by other layers of defense (such as Content Analysis and Malware Analysis).

Level(s)	Description
5, 6	Medium. The URL is unproven; there is not an established history of normal behavior. This level should be evaluated by other layers of defense (such as Content Analysis and Malware Analysis) and considered for more restrictive policy.
7, 8, 9	Medium-High. The URL is suspicious; there is an elevated risk. Symantec recommends blocking at this level.
10	High. The URL is confirmed to be malicious. Symantec recommends blocking at this level.
0	Override the threat risk associated with the URL.
none	

Layer and Transaction Notes

- Layers: <Cache>, <DNS-Proxy>, <Forward>, <Proxy> , <SSL>, <SSL-Intercept>
- Transactions: all transactions

Example

Deny users in the defined group who request URLs with threat risk level 7 through 10.

```
define condition user_group1
  realm=SAML1 group="group1"
end condition user_group1

; deny specified users who request URL with threat risk level 7 through 10
<Proxy>
  condition=group1 url.threat_risk.level=7..10 deny
```

user=

Tests the authenticated username associated with the transaction. This condition is only available if the transaction was authenticated (that is, the "authenticate()" on page 348 property was set to something other than no).

Syntax

`user=user_name`

where:

- *user_name*: User name, as follows:
  - IWA realm: Usernames are case-insensitive.

    For example:

    `user=symantec\mary.jones`

    matches a complete username, and

    `user=mary.jones`

    matches a relative name.

    In IWA this provides the flexibility of matching either a full username or relative username.
  - UNIX (local) realm: Usernames are case-sensitive.
  - RADIUS realm: Username case-sensitivity depends on the RADIUS server's setting. The case-sensitive setting should also be set correctly when defining a RADIUS realm in the appliance.
  - LDAP realm: Username case-sensitivity depends on the LDAP server's setting. The case-sensitive setting should also be set correctly when defining an LDAP realm in the appliance.

    In LDAP this provides the flexibility of matching either a fully qualified domain name or relative username.

    For example:

    `user="cn=mary.jones,cn=sales,dc=symantec,dc=com"`

    OR

    `user="uid=mary.jones,ou=sales,o=symantec"`

    matches a complete username, and

```
user=mary.jones
```

matches a relative name.

Layer and Transaction Notes

- Layers: <Admin>, <Exception>, <Forward>, <Proxy> , <SSL>, <SSL-Intercept>
- Transactions: all proxies, administrator

> **Note:** When used in the <forward> layer, this condition can evaluate to NULL (shown in a trace as N/A) if no authenticated client exists. Rules containing these conditions can be guarded by authenticated= to preserve normal logic.

Example

Test for the specified user.

```
<Proxy>
  user=john.smith
```

See Also

- Conditions: "authenticated=" on page 81,"group=" on page 140, "has_attribute.name=" on page 142, "http.transparent_authentication=" on page 180, "realm=" on page 208, "user.domain=" on page 316, "user.x509.issuer=" on page 322, "user.x509.serialNumber=" on page 323, "user.x509.subject=" on page 325
- Properties: "authenticate()" on page 348, "authenticate.force()" on page 354, "check_authorization()" on page 376,"deny.unauthorized()" on page 394, "socks.authenticate()" on page 554, "socks.authenticate.force()" on page 556, "ssl.forward_proxy()" on page 559

user.authentication_error=

Test what, if any, error was encountered during user authentication.

This condition is used to test what, if any, user authentication error was encountered. It is only evaluated if the encountered error was also specified as a tolerated error.

Syntax

`user.authentication_error{any|none|not_attempted|`*`error`*`1[,`*`error`*`2,...]}`

where:

- any: Evaluates to true if there was an authentication error.
- none: Evaluates to true if there were no authentication errors and authentication was attempted.
- `not_attempted`: Evaluates to true if authentication was not attempted.
- *`error`*: Authentication error. Evaluates to true if one of the specified authentication errors occurs.

Layer and Transaction Notes

- Layers: `<Exception>`, `<Forward>`, `<Proxy>` , `<SSL>`, `<SSL-Intercept>`
- Transactions: all proxies

Example

Redirect a user to a password change page after a password expiration.

```
define action redirect_to_password_change_page
  redirect(302, '.*', 'http://ourcompany.com/password_change')
end

<Proxy>
  authenticate(realm) authenticate.tolerate_error(expired_credentials)

<Proxy>
  user.authentication_error=(expired_credentials) action.redirect_to_password_change_page
```

user.authorization_error=

Test what, if any, error was encountered during user authorization.

This condition is used to test what, if any, user authorization error was encountered. It is only evaluated if the encountered error was also specified as a tolerated error.

Syntax

`user.authorization_error={any|none|not_attempted|`*`error`*`1[`*`error`*`2,...]}`

where:

- any: Evaluates to true if there is an authorization error.
- none: Evaluates to true if there were no authorization errors and authorization was attempted.
- not_attempted: Evaluates to true if authorization was not attempted.
- *error*: Authorization error. Evaluates to true if one of the specified authorization errors occurs.

Layer and Transaction Notes

- Layers: <Exception>, <Forward>, <Proxy> , <SSL>, <SSL-Intercept>
- Transactions: all proxies

Example

Add a user to a default group if authentication succeeded but authorization failed due to a communication error with the authorization server.

```
<Proxy>
  authenticate(realm) authorize.tolerate_error(communication_error)

<Proxy>
  user.authorization_error=(communication_error) authorize.add_group(default_group)
```

user.domain=

Tests if the client is authenticated, the logged-into realm is an NTLM realm, and the domain component of the username is the specified domain. If all of these conditions are met, the response will be true. This condition is unavailable if the current transaction is not authenticated (that is, the "authenticate()" on page 348 property is set to no).

Syntax

`user.domain=`*`windows_domain_name`*

Layer and Transaction Notes

- Layers: `<Admin>`, `<Exception>`, `<Forward>`, `<Proxy>` , `<SSL>`, `<SSL-Intercept>`

> **Note:** When used in the `<forward>` layer, this condition can evaluate to NULL (shown in a trace as N/A) if no authenticated client exists. Rules containing these conditions can be guarded by authenticated= to preserve normal logic.

Example

Test if the user is in domain all-staff.

```
<Proxy>
  user.domain=all-staff
```

See Also

- Conditions: "authenticated=" on page 81,"group=" on page 140, "has_attribute.name=" on page 142, "http.transparent_authentication=" on page 180, "realm=" on page 208,"user=" on page 312 , "user.x509.issuer=" on page 322, "user.x509.serialNumber=" on page 323, "user.x509.subject=" on page 325
- Properties: "authenticate()" on page 348, "authenticate.force()" on page 354, "check_authorization()" on page 376,"deny.unauthorized()" on page 394, "socks.authenticate()" on page 554, "socks.authenticate.force()" on page 556, "ssl.forward_proxy()" on page 559

user.is_guest=

Test whether a user is authenticated as a guest user. Only useful if used in conjunction with an authenticate.guest property.

Syntax

```
user.is_guest={yes|no}
```

Layer and Transaction Notes

- Layers: <Exception>, <Forward>, <Proxy> , <SSL>, <SSL-Intercept>
- Transactions: all proxies

Example

Add a guest user to a default group.

```
<Proxy>
 authenticate.guest(guest_user, 900, realm)

<Proxy>
  user.is_guest=yes authorize.add_group(default_group)
```

user.login.address=

Test the address that user is logged in from.

This condition is used to match the IP address or subnet that the user has logged in from. This may be different than the client IP address if the user is behind a proxy chain. The user login address can be set with the property: authenticate.credentials.address.

Syntax

`user.login.address={`*`ip_address`*`|`*`ip_address_wildcards`*`|`*`subnet`*`|`*`subnet_label`*`}`

where:

- *`ip_address`*: Effective IP address; for example, 10.25.198.0
- *`ip_address_wildcards`*: IP address specified using wildcards in any octet(s); for example, 10.25.*.0 or 10.*.*.0
- *`subnet`*: Subnet specification; for example, 10.25.198.0/24
- *`subnet_label`*: Label of a subnet definition block that binds a number of IP addresses or subnets

Layer and Transaction Notes

- Layers: `<Admin>`, `<Exception>`, `<Forward>`, `<Proxy>` , `<SSL-Intercept>`
- Transactions: all

Example

Allow the user if logged in from the subnet 10.167.146.0/24.

```
<Proxy>
  user.login.address=10.167.146.0/24 allow
```

See Also

- Information on wildcards: http://www.symantec.com/docs/TECH241521
- Information on IP address ranges: http://www.symantec.com/docs/TECH241929

user.login.count=

Test the number active logins of this user.

This condition is used to test how many times the current user is logged in. It can be used to manage the maximum number of times a user is allowed to log in.

Syntax

`user.login.count={`*range*`|`*exact*`}`

where:

- *range*: A range specified with a lower limit, an upper limit, or both:
  - *..upper_limit* - the highest number of logins
  - *lower_limit..* - the lowest number of logins
  - *lower_limit..upper_limit* - number of logins within the specified range
- *exact*: Exact number of logins.

Layer and Transaction Notes

- Layers: `<Admin>`, `<Exception>`, `<Forward>`, `<Proxy>` , `<SSL-Intercept>`
- Transactions: all

Example

If a user is logged in more than once, log user out of all sessions except one.

```
<Proxy>
  user.login.count=2.. user.login.log_out_other(yes)
```

See Also

- Properties: "user.login.log_out_other()" on page 587

user.login.time=

Test the number seconds that the current login has been active.

This condition is used to test how many seconds the current user has been logged in at the current IP address. It can be used to manage the maximum time that a user is allowed to be logged in.

Syntax

`user.login.time={range|exact}`

where:

- *range*: A range specified with a lower limit, an upper limit, or both:
  - *..upper_limit* - the highest number of seconds
  - *lower_limit..* - the lowest number of seconds
  - *lower_limit..upper_limit* - number of seconds within the specified range
- *exact*: Exact number of logins.

Layer and Transaction Notes

- Layers: <Admin>, <Exception>, <Forward>, <Proxy> , <SSL-Intercept>
- Transactions: all

Example

Log out the user if the user has been logged in for more than one hour.

```
<Proxy>
  user.login.time=3600.. user.login.log_out(yes)
```

See Also

- Properties: "user.login.log_out_other()" on page 587

user.regex=

Tests regular-expression-defined portions of authenticated usernames.

This is a source condition used to define a case-sensitive regular expression match against known portions of authenticated user names. This prevents administrators from having to configure long lists of unique user objects. It also provides flexibility for future use, where more usernames with the defined portion are planned.

Syntax

`user.regex=`*`regular_expression`*

Layer and Transaction Notes

- Layers: `<Admin>`, `<Exception>`, `<Forward>`, `<Proxy>` , `<SSL>`
- Transactions: HTTP proxy

Example

Consider a deployment where several Active Directory users have usernames that begin with “admin” (e.g., admin123, adminbobjones). The following example defines a rule for these users without the need to define each specific username.

```
<Proxy>
  ALLOW user.regex=”^admin.*” ; Matches requests with username starting with "admin"
```

See Also

- "Regex Reference" on page 703

user.x509.issuer=

Tests the issuer of the x509 certificate used in authentication to certificate realms. This condition is primarily useful in constructing explicit certificate revocation lists. This condition is only true for users authenticated against a certificate realm.

Syntax

`user.x509.issuer=`*`issuer_DN`*

where:

- *`issuer_DN`*: RFC2253 LDAP distinguished name, appropriately escaped. Comparisons are case-sensitive.

Layer and Transaction Notes

- Layers: `<Admin>`,`<Proxy>`, `<Exception>`, `<Forward>`, `<Proxy>` , `<SSL>`
- Transactions: all proxies

> **Note:** When used in the `<forward>` layer, this condition can evaluate to NULL (shown in a trace as N/A) if no authenticated client exists. Rules containing these conditions can be guarded by authenticated= to preserve normal logic.

Example

Test if the LDAP distinguished name matches the specified string.

```
<Proxy>
  user.x509.issuer=x-group
```

See Also

- Conditions: "authenticated=" on page 81, "group=" on page 140, "has_attribute.name=" on page 142, "http.transparent_authentication=" on page 180, "realm=" on page 208, "user=" on page 312, "user.domain=" on page 316, "user.x509.serialNumber=" on the facing page, "user.x509.subject=" on page 325
- Properties: "authenticate()" on page 348, "authenticate.force()" on page 354

user.x509.serialNumber=

Tests the serial number of the x509 certificate used to authenticate the user against a certificate realm. The `user.x509.serialNumber=` condition is primarily useful in constructing explicit certificate revocation lists. Comparisons are case-insensitive.

Syntax

`user.x509.serialNumber=`*`serial_number`*

where:

- *`serial_number`*: string representation of the certificate's serial number in hexadecimal.

  The string should comprise an even number of characters, so if the number needs an odd number of characters to represent in hex, insert a leading zero. The string can be up to 160 bits in length.

Layer and Transaction Notes

- Layers: `<Admin>`, `<Exception>`, `<Forward>`, `<Proxy>` , `<SSL>`
- Transactions: all proxies

> **Note:** When used in the `<forward>` layer, this condition can evaluate to NULL (shown in a trace as N/A) if no authenticated client exists. Rules containing these conditions can be guarded by authenticated= to preserve normal logic.

Example

Deny the transaction if the x509 certificate was revoked.

```
define condition CRL_deny
  user.x509.serialNumber=019F3C5A343115B6CFF7
  user.x509.subject="site.com"
end

<Proxy>
  condition=CRL_deny log_message("certificate revoked") deny
```

See Also

- Conditions: "authenticated=" on page 81, "group=" on page 140, "has_attribute.name=" on page 142, "http.transparent_authentication=" on page 180, "realm=" on page 208, "user=" on page 312, "user.domain=" on page 316, "user.x509.serialNumber=" on the previous page, "user.x509.subject=" on the facing page
- Properties: "authenticate()" on page 348, "authenticate.force()" on page 354

user.x509.subject=

Tests the subject field of the x509 certificate used to authenticate the user against a certificate realm. The `user.x509.subject=` condition is primarily useful in constructing explicit certificate revocation lists.

Syntax

`user.x509.subject=`*`subject`*

where:

- *`subject`*: RFC2253 LDAP DN, appropriately escaped. Comparisons are case-sensitive.

Layer and Transaction Notes

- Layers: `<Admin>`, `<Exception>`, `<Forward>`, `<Proxy>` , `<SSL>`
- Transactions: all proxies

> **Note:** When used in the `<forward>` layer, this condition can evaluate to NULL (shown in a trace as N/A) if no authenticated client exists. Rules containing these conditions can be guarded by authenticated= to preserve normal logic.

Example

See the example in "user.x509.subject=" above.

See Also

- Conditions: "authenticated=" on page 81, "group=" on page 140, "has_attribute.name=" on page 142, "http.transparent_authentication=" on page 180, "realm=" on page 208, "user=" on page 312, "user.domain=" on page 316,"user.x509.issuer=" on page 322, "user.x509.serialNumber=" on page 323
- Properties: "authenticate()" on page 348, "authenticate.force()" on page 354

virus_detected=

Test whether a virus has been detected. Rules containing this trigger do not match for a transaction that does not involve virus scanning.

Syntax

```
virus_detected={yes|no}
```

Layer and Transaction Notes

- Layers: `<Exception>`, `<Proxy>`
- Transactions: All HTTP transactions (proxy, refresh, pipeline), FTP proxy transactions

Example

Deny request when a virus is detected.

```
<Proxy>
  virus_detected=yes deny
```

webex.site=

Set the WebEx site value for WebEx long-lived HTTP connections.

This tests if the website hosted by WebEx matches the policy. This is only effective if WebEx HTTPS interception and WebEx handoff are enabled. In other cases, the condition value is set to empty string. For example, company1.webex.com and company2.webex.com will have different configurations and allowed features.

Find the webex.site value under Collaboration in the Access Log.

Syntax

`webex.site=`*`site_name`*

Layer and Transaction Notes

- Layers: `<Proxy>`
- Transactions: all proxies

Example

Allow file sharing for WebEx conferences with company 1, but deny file sharing for conferences with other companies.

```
<Proxy>
  request.application.name=WebEx request.application.operation="Upload Files" webex.site="company 1"
ALLOW
  request.application.name=WebEx request.application.operation="Upload Files" DENY
```

weekday=

Tests if the day of the week is in the specified range or an exact match. By default, the appliance's date is used to determine the day of the week. To specify the UTC time zone, use the form `weekday.utc=`. The numeric pattern used to test the `weekday=` condition can contain no whitespace

Syntax

`weekday[.utc]={[range]|exact_weekday}`

where:

- *range*: A range specified with the earliest weekday, the latest weekday, or both:
  - `..last_weekday` - Integer from 1 to 7, where 1 specifies Monday and 7 specifies Sunday, indicating the last day of the week that tests true. If left blank, Sunday is assumed.
  - `first_weekday..` - Integer from 1 to 7, where 1 specifies Monday and 7 specifies Sunday, indicating the first day of the week that tests true. If left blank, Monday is assumed.
  - `first_weekday..last_weekday` - Number of logins within the specified range
- *exact_weekday*: Integer from 1 to 7, where 1 specifies Monday and 7 specifies Sunday, indicating the day of the week that tests true.

> **Tip:** To test a range that wraps from one week into the next, the following shorthand expression is available. While `weekday=(..1|6..)` specifies a long weekend that includes Monday, the policy language also recognizes `weekday=6..1` as equivalent.

Layer and Transaction Notes

- Layers: `<Admin>`, `<Cache>`, `<Diagnostic>`, `<DNS-Proxy>`, `<Exception>`, `<Forward>`, `<Proxy>` , `<SSL>`, `<SSL-Intercept>`
- Transactions: all

> **Note:** Using time-related conditions to control caching behavior in a `<Cache>` layer may cause thrashing of the cached objects.

See Also

- Conditions: date[.utc]=, "day=" on page 119, "hour=" on page 147, "minute=" on page 193, "month=" on page 195, "time=" on page 292, "year=" on the next page

year=

Tests if the year is in the specified range or an exact match. The current year is determined by the date set on the appliance by default. To specify the UTC time zone, use the form `year.utc=`. The numeric pattern used to test the `year=` condition can contain no whitespace.

Syntax

`year[.utc]={`*range*`|`*exact_year*`}`

where:

- *range*: A range specified with the earliest year, the latest year, or both:
- - `..`*last_weekday* - Integer from 1 to 7, where 1 specifies Monday and 7 specifies Sunday, indicating the last day of the week that tests true. If left blank, Sunday is assumed.
  - *first_weekday*`..` - Integer from 1 to 7, where 1 specifies Monday and 7 specifies Sunday, indicating the first day of the week that tests true. If left blank, Monday is assumed.
  - *first_weekday*`..`*last_weekday* - Years within the specified range.
- *exact_year*: Exact year.

> **Tip:** To test against an inverted range of years, the following shorthand expression is available. While `year=(..2019|2025..)` specifies years up to and including 2019, and from 2025 on, the policy language also recognizes `year=2025..2019` as equivalent.

Layer and Transaction Notes

- Layers: `<Admin>`, `<Cache>`, `<Diagnostic>`, `<DNS-Proxy>`, `<Exception>`, `<Forward>`, `<Proxy>` , `<SSL>`, `<SSL-Intercept>`
- Transactions: all

> **Note:** Using time-related conditions to control caching behavior in a `<Cache>` layer may cause thrashing of the cached objects.

Example

Test for years 2020 through 2022.

```
<Proxy>
  year=2020..2022
```

See Also

- Conditions: date[.utc]=, "day=" on page 119, "hour=" on page 147, "minute=" on page 193, "month=" on page 195, "time=" on page 292, "weekday=" on page 328

Properties

A property is a variable that can be set to a value. At the beginning of a transaction, all properties are set to their default values. As each layer in the policy is evaluated in sequence, it can set a property to a particular value. A property retains the final value setting when evaluation ends, and the transaction is processed accordingly. Properties that are not set within the policy maintain their default values.

access_log()

Selects the access log used for this transaction. Multiple access logs can be selected to record a single transaction. Individual access logs are referenced by the name given in configuration. Configuration also determines the format of the each log.

To record entries in the event log, see"log_message()" on page 608 .

Syntax

```
access_log.log_name(yes|no)
access_log(auto|no|log_name_list)
access_log[log_name1,logname2,..]](yes|no)
```

where:

- yes: Enable logging for this transaction to the specified log(s).
- no:Disable logging, either for this transaction or for the specified log(s).
- auto: (Default) Use default log for this protocol.
- *log_name*: Configured access log.

Discussion

Each of the syntax variants has a different role in selecting the list of access logs used to record the transaction:

- access_log() overrides any previous access log selections for this transaction.
- access_log.*log_name*() selects or de-selects the named log, according to the specified value. Any other log selections for the transaction are unaltered.
- access_log[*log_name1,logname2,..*]() selects or de-selects all the logs named in the list, according to the specified value. The selection of logs not named in the list is unaffected.

Layer and Transaction Notes

- Layers: <Cache>, <Exception>, <Forward>, <Proxy>
- Transactions: all proxies

Example

Access-log requests when the Origin hostname is associated with the specified categories.

```
<Proxy>
  request.header.Origin.url.category=Sport,Gambling access_log.default(yes)
```

See Also

- Properties: "log.rewrite.field_id()" on page 491, "log.suppress.field_id( )" on page 493
- Actions: "log_message()" on page 608

access_server()

Determines whether the client can connect to the origin content server (OCS).

Syntax

```
access_server(yes|no)
```

where:

- yes: (Default behavior) Allow the client to receive streaming content directly from the OCS or other upstream device. Set access_server(no) to serve only cached content. This property is ignored for Flash VOD caching because the Flash proxy always checks the OCS for every playspurt. Because part of a stream can be cached, and another part of the same stream can be uncached, access_server(no) can cause a streaming transaction to be terminated after some of the content has been served from the cache.
- no: Prevent subsequent connections to the OCS after a decision to deny the request. Refer to http://www.symantec.com/docs/TECH246261 for details on using this property to prevent outbound request data from inadvertently reaching a restricted site.

Layer and Transaction Notes

- Layers: <Cache>, <DNS-Proxy>, <Forward>, <Proxy>
- Transactions: DNS, HTTP, SOCKS, and streaming transactions

Example

Block access to file-sharing sites and all downloads of executables.

```
<Proxy>
  deny url.category="File Sharing" access_server(no)
  allow

<Proxy>
  deny http.response.apparent_data_type=exe
```

If the rule in the first layer matches (request is to a file-sharing site), access_server(no) prevents the appliance from connecting to the server; thus, the request is denied before the ProxySG appliance has enough information to evaluate the rule in the second policy layer.

If the rule in the first layer doesn't match, the connection to the server is allowed and the rule in the second policy layer is applied (files that appear to be executables are blocked).

See Also

- Conditions: "bitrate=" on page 83, "live=" on page 192, "streaming.client=" on page 283, "streaming.content=" on page 284

action()

Selectively enables or disables the specified "define action" on page 628 block. The default value is no.

> **Note:** Several "define action" on page 628 blocks may be enabled for a transaction. If more than one action selected rewrites the URL or header a specific header, the actions are deemed to conflict and only one will be executed. When detected at runtime, action conflicts will be reported in the event log as a severe event. Action conflicts may also be reported at compilation time.

Syntax

```
action(action_label)
action.action_label(yes|no)
```

where:

- *action_label*: Label of the "define action" on page 628 block to be enabled or disabled.

Discussion

Each of the different syntax variants has a different role in selecting the list of actions applied to the transaction:

- action(*action_label*) enables the specified action block and disables all other action blocks.
- action.*action_label*() enables or disables the specific action block. Any other action block selections for the transaction are unaltered.

Layer and Transaction Notes

- Layers: <Cache>, <Exception>, <Proxy>

Example

See the example in "define action" on page 628.

See Also

- Definitions: "define action" on page 628

adn.connection.dscp()

Control the DSCP value for both upstream and downstream ADN tunneled packets.

Syntax

`adn.connection.dscp(preserve|`*`dscp_value`*`)`

where:

- `preserve`: (Default behavior) ADN proxies (both branches and concentrators) preserve DSCP values of inbound packets into the ADN network. DSCP values of client inbound packets and upstream tunnel packets will be the same. DSCP values of server inbound packets on concentrator and downstream tunnel packets will be the same.
- *`dscp_value`*: DSCP values of upstream tunnel packets will be of the specified value until they are reset by some intermediary device. DSCP values of downstream tunnel packets will be of the same specified value until they are reset by some intermediary device, even when there is an intermediary device that modifies DSCP values of upstream tunnel packets. Values can be one of the following:
  - decimal value between 0 and 63
  - a class: `af11 | af12 | af13 | af21 | af22 | af23 | af31 | af32 | af33 | af41 | af42 | af43 | cs1 | cs2 | cs3 | cs4 | cs5 | cs6 | cs7 | ef`

Layer and Transaction Notes

- Layers: `<Forward>`
- Transactions: all transactions

Example

First DSCP policy rule for ADN tunneled packets sets both the upstream and downstream DSCP value to preserve; similarly, the second policy rule sets DSCP value to 50.

```
<Forward>
  adn.connection.dscp(preserve)

<Forward>
  adn.connection.dscp(50)
```

See Also

- Properties: "client.connection.dscp()" on page 381, "adn.connection.dscp()" above, "server.connection.dscp()" on page 545

adn.server()

Enable or disable ADN service.

Syntax

```
adn.server(yes|no)
```

The default value is taken from the applicable service configuration.

Layer and Transaction Notes

- Layers: <Forward>
- Transactions: all proxies

Example

Disable ADN service.

```
<Forward>
  adn.server(no)
```

See Also

- Properties: "adn.server.optimize()" on the next page

adn.server.optimize()

Control optimization of individual connections based on any policy conditions available in the <Forward> layer. Applies if the application proxy server connection is forwarded over an ADN tunnel.

Syntax

```
adn.server.optimize[.byte_cache,compress](yes|no)
```

where:

- byte_cache: Include byte caching.
- compress: Include ADN compression.
- yes: Data to and from the server is optimized.
- no: Optimization of data to and from the server is disabled.

> **Tip:**
> The default value is taken from the applicable service configuration, which is dependent on the service.

> **Tip:** Square brackets [] are optional if specifying one parameter, but are required when specifying multiple parameters in a comma-separated list.
> The period . is optional when specifying multiple parameters within square brackets.

Layer and Transaction Notes

- Layers: <Forward>
- Transactions: all proxies

Example

Disable optimizations for data flowing in both directions between clients from a specified subnet and a particular service.

```
<Forward>
  client.address=10.10.167.0/24 service.name="XWindows" adn.server.optimize(no)
```

See Also

- Properties: "adn.server.optimize.inbound()" on the facing page, "adn.server.optimize.outbound()" on page 342

adn.server.optimize.inbound()

Control optimization of data received over individual server connections based on any policy conditions available in the <Forward> layer. Applies if the application proxy server connection is forwarded over an ADN tunnel.

Syntax

```
adn.server.optimize.inbound[.byte_cache,compress](yes|no)
```

where:

- byte_cache: Include byte caching.
- compress: Include ADN compression.
- yes: Data from the server is optimized.
- no: Optimization of data from the server is disabled.

> **Tip:**
> The default value is taken from the applicable service configuration, which is dependent on the service.

> **Tip:** Square brackets [] are optional if specifying one parameter, but are required when specifying multiple parameters in a comma-separated list.
> The period . is optional when specifying multiple parameters within square brackets.

Layer and Transaction Notes

- Layers: <Forward>
- Transactions: all proxies

Example

Disable optimization for data from a particular host.

```
<Forward>
  server_url.host=my_host.my_business.com adn.server.optimize.inbound(no)
```

See Also

- Properties:"adn.server.optimize()" on the previous page, "adn.server.optimize.outbound()" on the next page

adn.server.optimize.outbound()

Control optimization of data sent over individual server connections based on any policy conditions available in the <Forward> layer. Applies only if the application proxy server connection is forwarded over an ADN tunnel.

Syntax

```
adn.server.optimize.outbound[.byte_cache,compress](yes|no)
```

where:

- byte_cache: Include byte caching.
- compress: Include ADN compression.
- yes: Data to the server is optimized.
- no: Optimization of data to the server is disabled.

> **Tip:**
> The default value is taken from the applicable service configuration, which is dependent on the service.

> **Tip:** Square brackets [] are optional if specifying one parameter, but are required when specifying multiple parameters in a comma-separated list.
> The period . is optional when specifying multiple parameters within square brackets.

Layer and Transaction Notes

- Layers: <Forward>
- Transactions: all proxies

Example

Disable optimization for data sent to a particular host.

```
<Forward>
  server_url.host=my_host.my_business.com adn.server.optimize.outbound(no)
```

See Also

- Properties: "adn.server.optimize()" on page 340, "adn.server.optimize.inbound()" on the previous page

advertisement()

Determines whether to treat the objects at a particular URL as banner ads to improve performance. If the content is not specific to a particular user or client, then the hit count on the origin server is maintained while the response time is optimized using the following behavior:

- Always serve from the cache if a cached response is available. Ignore any request headers that bypass the cache; for example, Pragma: No-Cache.
- Always cache the response from the origin content server.
- If the request was served from the cache, request the object from the origin server in the background to maintain the origin server's hit count on the ad and also allow ad services to deliver changing ads.

A number of CPL properties affect caching behavior, as listed in the **See Also** section below. Remember that any conflict between their settings is resolved by CPL evaluation logic, which uses the property value that was last set when evaluation ends.

Syntax

```
advertisement(yes|no)
```

The default behavior is no.

Layer and Transaction Notes

- Layers: <Cache>
- Transactions: HTTP proxy, except FTP over HTTP transactions

Example

For the specified URL, do not treat objects as ads.

```
<Cache>
  url.host=intranet.org advertisement(no)
```

See Also

- Properties: "always_verify()" on page 345, "cache()" on page 374, "cookie_sensitive()" on page 390, "pipeline()" on page 500, "refresh()" on page 503, "ttl()" on page 584, "ua_sensitive()" on page 585

allow

Allows the transaction to be served.

`allow` preserves previous decisions, with some exceptions. It can override the "deny" on page 392 and "exception()" on page 402 properties.

This property can be overridden by the "access_server()" on page 335, "authenticate()" on page 348, "deny" on page 392, "exception()" on page 402, "force_deny()" on page 410, and "force_exception()" on page 412 properties and the "redirect()" on page 611 action.

> **Caution:** Because this property can override "deny" on page 392, ensure that security is not compromised when using `allow` in layers evaluated after layers containing `deny`.

Syntax

```
condition allow
```

Layer and Transaction Notes

- Layers: `<Admin>`, `<Cache>`, `<Proxy>` , `<SSL>`
- Transactions: all

Example

Deny unauthenticated clients, but the specified client addresses. The allow overrides the previous denial for matching transactions.

```
<proxy>
  authenticated=no deny

<proxy>
  client.address=10.10.10.0/24 allow
```

See Also

- Properties: "deny" on page 392, "OK" on page 499

always_verify()

Determines whether each request for the objects at a particular URL must be verified with the origin content server. This property provides a URL-specific alternative to the global caching setting always-verify-source. If there are multiple simultaneous accesses of an object, the requests are reduced to a single request to the origin server.

This property is ignored for Flash VOD caching because the Flash proxy will always verify object requests with the OCS. Therefore, even in fully-cached videos, you will see some server bytes statistics.

Syntax

```
always_verify(yes|no)
```

The default behavior is no.

Layer and Transaction Notes

- Layers: <Cache>, <Proxy>
- Transactions: HTTP proxy, except FTP over HTTP transactions

Example

For internal sites, do not verify objects.

```
<Cache>
  url.host=internal.org always_verify(no)
```

See Also

- Properties: "advertisement()" on page 343, "bypass_cache()" on page 373, "cache()" on page 374, "cookie_sensitive()" on page 390,"force_cache()" on page 407, "pipeline()" on page 500, "refresh()" on page 503, "ttl()" on page 584, "ua_sensitive()" on page 585

application.name()

Set a custom name for the application associated with a URL. This value of this property populates the access log field `x-bluecoat-application-name` when traffic matches and access logging is enabled.

Syntax

`application.name(`*`name`*`)`

Layer and Transaction Notes

- Layers: `<Cache>`, `<Proxy>`

Example

Set a custom name for applications associated with the specified domain path.

```
<Proxy>
  url.domain=resumes.com/jobs application.name("resumes.com_jobs")
```

attack_detection.failure_weight()

Overrides the default value of 1 for the failure weight of the defined HTTP response code. Each failed request can have a value of 0 - 500, depending on the nature of the failed request.

This action works in conjunction with the failure limits defined in the `#(config)attack-detection` CLI.

Syntax

```
attack_detection.failure_weight(N)
```

where:

- *N*: Set the failure weight value for the specified HTTP response code per failed request event. If set to 0, the response code is not counted as a failure. The default value is 1.

Layer and Transaction Notes

- Layers: `<Proxy>`
- Transactions: HTTP proxy, except FTP over HTTP transactions

Example

Set a failure weight value of 5 for HTTP response code 508.

```
<Proxy>
  http.response.code=508 attack_detection.failure_weight(5)
```

See Also

- Conditions: "http.response.code=" on page 176

authenticate()

Authenticate the user in the specified realm, or disable authentication for this transaction.

When authentication is dependent on any condition that is not part of the client's identity, then some transactions from the client are authenticated and some are not. However the browser offers some credential types pro-actively. The default behavior of the appliance is to forward any proxy credentials that it does not consume.

To prevent forwarding of proxy credentials in situations where there is no upstream proxy authentication, use the no_upstream_authentication parameter.

Syntax

```
authenticate(realm_name)
authenticate(no[,upstream_authentication|no_upstream_authentication])
```

where:

- *realm_name*: Authenticate with the specified realm.
- no: Do not authenticate.
- upstream_authentication: Offered proxy credentials should be passed upstream.
- no_upstream_authentication: Offered proxy credentials should not be passed upstream.

Layer and Transaction Notes

- Layers: <Admin>, <Proxy>
- Transactions: all proxies, administrator

Example

All traffic to a.com is authenticated. All traffic to b.com is authenticated by an upstream proxy. All other traffic is unauthenticated, and proxy credentials are not forwarded.

```
<Proxy>
  url.domain=//a.com/ authenticate(localr)
  url.domain=//b.com/ authenticate(no)
  authenticate(no, no_upstream_authentication)
```

See Also

- Conditions: "authenticated=" on page 81, "exception.id=" on page 137, "group=" on page 140, "has_attribute.name=" on page 142, "http.transparent_authentication=" on page 180, "realm=" on page 208, "user=" on page 312, "user.domain="

on page 316

- Properties: "authenticate.force()" on page 354, "authenticate.mode()" on page 360, "authenticate.use_url_cookie()" on page 371, "check_authorization()" on page 376, "socks.authenticate()" on page 554, "socks.authenticate.force()" on page 556

authenticate.authorization_refresh_time()

Specify the number of seconds that authorization data can be cached.

After the authorization data for a user (groups and attributes) is determined, that data is cached on the Proxy. This property sets the number of seconds that this cached data can be trusted. After that time, the data must be refreshed.

This property overrides the equivalent setting in the realm. If this property does not exist, then the realm setting apply.

Syntax

`authenticate.authorization_refresh_time(`*`seconds`*`)`

where:

- *seconds*: Number of seconds that authorization data can be cached and reused. After that time expires, the authorization data must be refreshed from the authorization server. Default value is 1.

Layer and Transaction Notes

- Layers: `<Admin>`, `<Proxy>`
- Transactions: all proxies, administrator

Example

Set the authorization refresh time to one hour.

```
<Proxy>
  authenticate(myrealm) authenticate.authorization_refresh_time(3600)
```

authenticate.charset()

Specify the character encoding used by HTTP basic authentication.

The HTTP Basic authentication protocol sends the username and password to the proxy or origin server as an array of bytes. The character encoding is arbitrarily chosen by the client. Within the HTTP protocol, there is no way for the client to tell the upstream device which encoding is used.

If the username or password contains non-ASCII characters, then the appliance needs to know what this character encoding is. Because there is no way for the proxy to determine this from the HTTP request, it must be specified in policy, using this property.

If the HTTP Basic credentials are not encoded as specified by the authenticate.charset property, then the HTTP request is terminated by an invalid_credentials exception. Therefore, if authenticate.charset is set to its default value of ascii, and the username or password contain non-ascii characters, then the request is terminated.

You must configure authenticate.charset to use non-ascii credentials using the HTTP Basic authentication protocol. An alternative to configuring this property is to use a different client-side authentication protocol, such as IWA, or forms-based authentication.

Syntax

`authenticate.charset(`*`character_set`*`)`

where:

- *`character_set`*: MIME character set name. Any of the standard charset names for encodings commonly supported by web browsers can be used. The default value is `ascii`. Refer to http://www.iana.org/assignments/character-sets for a list of standard charset names.

> **Tip:** In Microsoft Windows, you can use the `chcp` command in the Windows command line to find out your active code page. Once you know the code page number n, you can use `windows-n` as the charset name.

Layer and Transaction Notes

- Layers: `<Admin>`, `<Proxy>`
- Transactions: all proxies

Example

Set the authentication character encoding to windows-936, which is the extended ASCII encoding used by Microsoft Windows in North America.

```
<Proxy>
  authenticate(myrealm) authenticate.charset(windows-936)
```

authenticate.credential_refresh_time()

Specify the number of seconds that passwords can be cached.

When a realm uses Basic authentication, the password is cached by the User Management framework. This cached password can be trusted for the given number of seconds. After that time expires, the password must be verified with the authentication server.

This property overrides the equivalent setting in the realm. If this property does not exist, then the realm setting applies.

Syntax

`authenticate.credential_refresh_time(`*`seconds`*`)`

where:

- *seconds*: Number of seconds that passwords can be cached and reused. After that time expires, the password must be verified with the authentication server. The default value is 1.

Layer and Transaction Notes

- Layers: `<Admin>`, `<Proxy>`
- Transactions: all proxies, administrative

Example

Set the credential refresh time to one hour.

```
<Proxy>
  authenticate(myrealm) authenticate.credential_refresh_time(3600)
```

authenticate.force()

Control the relation between authentication and denial.

Syntax

```
authenticate.force(yes|no)
```

where:

- yes: Force users to authenticate (user IDs will be access logged), even if requests will be denied. When this property is set to yes, "authenticate()" on page 348 overrides "deny" on page 392 and "exception()" on page 402. It does not override "force_deny()" on page 410 or "force_exception()" on page 412.
- no (Default behavior): Use for early denials; do not force authentication. When this property is set to no, "deny" on page 392 and "exception()" on page 402 override "authenticate()" on page 348.

Layer and Transaction Notes

- Layers: <Admin>, <Proxy>
- Transactions: Does not apply to cache transactions

See Also

- Conditions: "authenticated=" on page 81, "group=" on page 140, "has_attribute.name=" on page 142, "http.transparent_authentication=" on page 180, "realm=" on page 208, "user=" on page 312, "user.domain=" on page 316
- Properties: "authenticate()" on page 348, "check_authorization()" on page 376, "socks.authenticate()" on page 554, "socks.authenticate.force()" on page 556

authenticate.force_307_redirect()

Force authentication redirects to use an HTTP 307 response code.

This property only affects authentication redirect modes—origin-cookie-redirect, origin-ip-redirect, form-cookie-redirect, form-ip-redirect. By default, authentication modes which redirect the browser use an HTTP 307 response code for Internet Explorer and an HTTP 302 response code for all other browsers. If an HTTP POST or PUT request requires authentication, a 307 redirect properly preserves the POST/PUT data, while a 302 redirect will convert the request to a GET and lose the POST/PUT data. The default behavior is to always preserve the POST/PUT data even at the cost of the authentication mode. To avoid losing the POST/PUT data, browsers which would use a 302 redirect are downgraded to a non-redirect authentication mode.

If the POST/PUT contains a multi-part mime type, Internet Explorer does not correctly preserve the data with a 307 redirect. The default behavior is to downgrade all multi-part POST/PUT requests to a non-redirect authentication mode.

Downgrading the authentication mode preserves the data, but means that the user is not authenticated using the virtual URL. This can be an issue if credentials need to be secured by using an SSL virtual URL. This property can be used to override the default behavior and force the use of 307 redirects. This will ensure that the user is authenticated using the virtual URL.

Most modern browsers support 307 redirects, but browsers other than Internet Explorer obey the RFC and display a pop-up asking the user if they want to repost the data. This pop-up can be repeated numerous times during authentication.

Syntax

```
authenticate.force_307_redirect(yes|no)
```

The default value is no.

Layer and Transaction Notes

- Layers: <Proxy>
- Transactions: HTTP proxy

Example

This example enables 307 redirects for FireFox. All other browsers will see the default behavior. Note that .User-Agent condition is a regex and can be used to match any particular browser.

```
<Proxy>
  .User-Agent="Firefox" authenticate.force_307_redirect(yes)
```

authenticate.form()

When forms-based authentication is in use, this property selects the form used to challenge the user.

Syntax

`authenticate.form(`*`authentication_form`*`)`

Layer and Transaction Notes

- Layers: <Proxy>
- Transactions: HTTP proxy

Example

All traffic from subnet Resubmit must use the authentication form “Reform.” All traffic from subunit ENG_subnet must use the authentication form “ENG_form.” All other traffic uses the default authentication form.

```
define subnet HR_subnet
  10.10.0.0/16
end

define subnet ENG_subnet
  10.9.0.0/16
end

<Proxy>
  authenticate(myrealm) authenticate.mode(form-cookie-redirect)

<Proxy>
  ; 1
  client.address=HR_subnet authenticate.form(HR_form)
  ; 2
  client.address=ENG_subnet authenticate.form(ENG_form)
  ; 3 -- no modification to 'authenticate.form' selects the default form
```

authenticate.forward_credentials()

Specifies whether to forward Authorization and Proxy-Authorizaton headers to the upstream OCS. You can use this property instead of the CLI command `#(config)security force-credential-forwarding`, which enables or disables the function globally.

In some situations–especially multi-tenant deployments–where the global CLI setting is not appropriate, Symantec recommends using this property.

For better security, use the `#(config)security force-credential-forwarding disable` command to disable sending all Authorization and Proxy-Authorizaton headers upstream, and use this property in policy to specify exceptions to the global rule.

Syntax

```
authenticate.forward_credentials(yes|no)
```

The `#(config)security force-credential-forwarding` command sets the default for this property.

Layer and Transaction Notes

- Layers: `<Proxy>`
- Transactions: HTTP proxy

Example

Forward credentials to myhost.com.

```
<Proxy>
  url.domain=//myhost.com/ authenticate.forward_credentials(yes)
```

See Also

- Properties: "authenticate.forward_credentials.log()" on the next page

authenticate.forward_credentials.log()

Specifies whether to log when Authorization and Proxy-Authorizaton headers are forwarded to an upstream OCS. When enabled, the ProxySG appliance logs instances of when it forwards Authorization and Proxy-Authorizaton headers in the event log.

You can use this property in conjunction with "authenticate.forward_credentials()" on the previous page and instead of the CLI command `#(config)security force-credential-forwarding-logging`, which enables or disables the function globally.

Syntax

```
authenticate.forward_credentials.log(yes|no)
```

The `#(config)security force-credential-forwarding-logging` command sets the default for this property.

Layer and Transaction Notes

- Layers: `<Proxy>`
- Transactions: HTTP proxy

Example

Forward credentials to myhost.com and log the action.

```
<Proxy>
  url.domain=//myhost.com/ authenticate.forward_credentials(yes) authenticate.forward_credentials.log
(yes)
```

See Also

- Properties: "authenticate.forward_credentials()" on the previous page

authenticate.guest()

Specify to authenticate as a guest user.

This property can be used to support authenticating guest users. If a transaction matches both a guest and regular authentication request it will first attempt regular authentication. If regular authentication fails on a tolerated error it then falls back to guest authentication.

Syntax

```
authenticate.guest(username[,surrogate-refresh-time[,realm]])
```

where:

- `username`: Substitution string to evaluate to determine the guest username, for example "`guest_$(client.address)`"
- `surrogate-refresh-time`: Refresh time for the guest user's surrogate credential. If no refresh time is specified (or "0" is specified) then the surrogate refresh time specified in the authentication realm is used.
- `realm`: Authentication realm to authenticate the user in. The user does not actually have to exist on the authentication server. If no realm is specified then the realm used in a previous authentication attempt in the transaction is used. If the transaction has not attempted a regular authentication and no realm is specified the user receives an authentication exception.

Layer and Transaction Notes

- Layers: `<Proxy>`
- Transactions: all proxies

Example 1

Log users in as guest automatically.

```
<Proxy>
 authenticate.guest(realm,guest_user) allow
```

Example 2

Attempt authentication of a user first and then have the user explicitly select to login as guest if authentication fails, modify the authentication_failed exception to included a link that has been designated as the guest login URL for that realm and then use the following policy:

```
<Proxy>
 url=virtual_URL authenticate.guest("guest_$(client.address)", 0, realm)
 authenticate(realm) allow
```

authenticate.mode()

Using the `authenticate.mode( )` property selects a combination of challenge type and surrogate credentials.

Challenge type is what kind of challenge (proxy, origin or origin-redirect) is issued.

Surrogate credentials are credentials accepted in place of the user's real credentials. They are used for a variety of reasons. Symantec supports three kinds of surrogate credentials:

- IP surrogate credentials authenticate the user based on the IP address of the client. After any client has been successfully authenticated, all future requests from that IP address are assumed to be from the same user.
- Cookie surrogate credentials use a cookie constructed by the appliance as a surrogate. The cookie contains information about the user, so multiple users from the same IP address can be distinguished. The cookie contains a temporary password to authenticate the cookie; this password expires when the credential cache entry expires.
- Connection surrogate credentials use the TCP/IP connection to authenticate the user. After authentication succeeds, the connection is marked authenticated and all future requests on that connection are considered to be from the same user.

The connection's authentication information includes the realm in which it was authenticated. The surrogate credentials are accepted only if the current transaction's realm matches the realm in which the session was authenticated.

> **Note:** Symantec recommends that you use the auto authentication mode when the appliance authenticates native FTP traffic from an FTP client, such as WS_Ftp.

Syntax

`authenticate.mode(`*`mode_type`*`)`

where:

- *`mode_type`*: One of the following, shown followed by the implied challenge type and surrogate credential:
  - auto: The default; the mode is automatically selected, based on the request. Chooses among proxy, origin-IP, and origin-IP-redirect, depending on the kind of connection (explicit or transparent) and the transparent authentication cookie configuration. For streaming transactions, authenticate.mode(auto) uses origin mode.
  - form-cookie: A form is presented to collect the user's credentials. The cookies are set on the OCS domain only, and the user is presented with the form for each new domain. This mode is most useful in reverse proxy scenarios where there are a limited number of domains.
  - form-cookie-redirect: A form is presented to collect the user's credentials. The user is redirected to the authentication virtual URL before the form is presented. The authentication cookie is set on both the virtual URL and the OCS domain. The user is only challenged when the credential cache entry expires.

- form-IP: A form is presented to collect the user's credentials. The form is presented whenever the user's credential cache entry expires.
- form-IP-redirect: This is similar to form-ip except that the user is redirected to the authentication virtual URL before the form is presented.
- origin: The appliance acts like an OCS and issues OCS challenges. The authenticated connection serves as the surrogate credential.
- origin-cookie: The appliance acts like an origin server and issues origin server challenges. A cookie is used as the surrogate credential. Origin-cookie is used in forward proxies to support pass-through authentication more securely than origin-ip if the client understands cookies. Only the HTTP and HTTPS protocols support cookies; other protocols are automatically downgraded to origin-ip. This mode could also be used in reverse proxy situations if impersonation is not possible and the origin server requires authentication.
- origin-cookie-redirect: The client is redirected to a virtual URL to be authenticated, and cookies are used as the surrogate credential. Note that the appliance does not support origin-redirects with the CONNECT method.
- origin-IP: The appliance acts like an OCS and issues OCS challenges. The client IP address is used as a surrogate credential. Origin-IP is used to support NTLM authentication to the upstream device when the client cannot handle cookie credentials. This mode is primarily used for automatic downgrading, but it can be selected for specific situations.
- origin-IP-redirect: The client is redirected to a virtual URL to be authenticated, and the client IP address is used as a surrogate credential. The appliance does not support origin-redirects with the CONNECT method.
- proxy: The appliance uses an explicit proxy challenge. No surrogate credentials are used. This is the typical mode for an authenticating explicit proxy. In some situations proxy challenges will not work; origin challenges are then issued.
- proxy-IP: The appliance uses the client IP address as a surrogate credential; thus, if the IP address is in the credential cache, the appliance passes the authentication request to upstream proxies or the OCS. If the IP address is not in the credential cache, the appliance responds with error 530 User access denied. Proxy authentication must be performed through another protocol first. The user must authenticate with another protocol (such as HTTP) on the same server, and then log into the FTP server with the cached credential. The proxy-ip mode is not supported with the Raptor login syntax when using an explicit proxy.
- SG2: The mode is selected automatically, based on the request.

Note: Modes that use an IP surrogate credential are insecure: After a user has authenticated from an IP address, all further requests from that IP address are treated as from that user. If the client is behind a NAT, or on a multi-user system, this can present a serious security problem.

Layer and Transaction Notes

- Layers: `<Proxy>` , (in 6.7.5.3 and later) `<Admin>`
- Transactions: all proxies

Example

Allow OPTIONS and POST methods without authentication. For all other requests, authenticate with the specified realm and using the origin-cookie-redirect challenge.

```
<Proxy>
  allow http.method=OPTIONS||POST
  authenticate(iwa_realm) authenticate.mode(origin-cookie-redirect)
```

authenticate.new_pin_form()

When Forms-Based authentication is in use, this selects the form to prompt user to enter a new PIN.

Syntax

`authenticate.new_pin_form(`*new_pin_form_name*`)`

Layer and Transaction Notes

- Layers: `<Proxy>`
- Transactions: HTTP proxy

Example

All traffic from subnet HR_subnet must use the new-pin form 'HR_new_pin_form' All traffic from subnet ENG_subnet must use the new-pin form ENG_new_pin_form All other traffic uses the default authentication form.

```
define subnet HR_subnet
  10.10.0.0/16
end

define subnet ENG_subnet
  10.9.0.0/16
end

<Proxy>
  authenticate(myrealm) authenticate.mode(form-cookie-redirect)

<Proxy>
  ; 1
  client.address=HR_subnet authenticate.new_pin_form(HR_new_pin_form)
  ; 2
  client.address=ENG_subnet authenticate.new_pin_form(ENG_new_pin_form)
  ; 3 -- no modification to 'authenticate.new_pin_form' selects the default form
```

authenticate.query_form()

When Forms-Based authentication is in use, this selects the form to display to the user when a yes/no questions needs to be answered.

Syntax

`authenticate.query_form(`*query_form_name*`)`

Layer and Transaction Notes

- Layers: <Proxy>
- Transactions: HTTP proxy

Example

All traffic from subnet HR_subnet must use the query form HR_query_form All traffic from subnet ENG_subnet must use the query form ENG_query_form All other traffic uses the default authentication form.

```
define subnet HR_subnet
  10.10.0.0/16
end

define subnet ENG_subnet
  10.9.0.0/16
end

<Proxy>
  authenticate(myrealm) authenticate.mode(form-cookie-redirect)

<Proxy>
  ; 1
  client.address=HR_subnet authenticate.query_form(HR_query_form)
  ; 2
  client.address=ENG_subnet authenticate.query_form(ENG_query_form)
  ; 3 -- no modification to 'authenticate.query_form' selects the default form
```

authenticate.redirect_stored_requests()

Determines whether requests stored during forms-based authentication can be redirected if the upstream host issues a redirecting response.

Syntax

```
authenticate.redirect_stored_requests(yes|no)
```

Layer and Transaction Notes

- Layers: <Proxy>
- Transactions: HTTP proxy

Example

Stored requests can be redirected.

```
<Proxy>
  authenticate.redirect_stored_requests(yes)
```

authenticate.surrogate_refresh_time()

Specify the number of seconds that surrogates can be trusted.

This property is used to control the number of seconds that a surrogate credential is trusted. A surrogate can be a cookie, an IP address or a previously authenticated connection. After this time expires, the surrogate is no longer trusted and must be refreshed by re-verifying the real credentials.

This property overrides the equivalent setting in the realm. If this property does not exist, then the realm setting will apply.

Syntax

`authenticate.surrogate_refresh_time(`*`seconds`*`)`

where:

- *`seconds`*: Number of seconds that surrogate credentials can be trusted (i.e. IP surrogate, cookie surrogate, connection surrogate). After that time expires, the real credentials must be verified. The default value is 1.

Layer and Transaction Notes

- Layers: `<Admin>`, `<Proxy>`
- Transactions: all proxies, administrator

Example

Set the surrogate refresh time to one hour.

```
<Proxy>
  authenticate(myrealm) authenticate.surrogate_refresh_time(3600)
```

See Also

authenticate.tolerate_error()

Specify to allow certain errors during user authentication.

This property can be used to support attempting authenticating but allowing the transaction to proceed if authentication fails for the specified error.

> **Caution:** Tolerating the error group all results in all authentication errors, including need_ credentials, to be tolerated. If specified then users will never be challenged which is often not the desired behavior. Use the error group "all" carefully.

Syntax

```
authenticate.tolerate_error.error(yes|no)
authenticate.tolerate_error[error,...](yes|no)
authenticate.tolerate_error(error,...)
```

where:

- yes: Specified error is permitted and the transaction should proceed unauthenticated.
- no: Specified error is not permitted and the transaction should terminate.
- *error*: Specified error(s).

> **Note:** `authenticate.tolerate_error(error,...)` is equivalent to `authenticate.tolerate_error[error,...](yes)`

Layer and Transaction Notes

- Layers: <Proxy>
- Transactions: all proxies

Example

Redirect a user to a password change page after a password expiry.

```
<Proxy>
  authenticate(realm) authenticate.tolerate_error(expired_credentials)

<Proxy>
  user.authentication_error=(expired_credentials)  action.redirect_to_password_change_page
```

```
define action redirect_to_password_change_page
  redirect(302, '.*', 'http://ourcompany.com/password_change');
end
```

authenticate.transaction()

Used in proxy deployments that forward authenticated user information from one proxy to another. This property (installed in policy on the proxy closest to the Origin Content Server or OCS) validates user and group credentials on a per-transaction basis, as opposed to the default behavior that associates authenticated details with each connection.

Syntax

```
authenticate.transaction(yes|no)
```

Layer and Transaction Notes

- Policy is installed on the parent proxy (the proxy closest to the requested website or OCS).
- Used in proxy chaining configurations where users are authenticated on one proxy and forwarded with their requests to another proxy which services the Internet content request. The forwarded or Parent proxy retrieves the authentication information using a policy substitution realm. For details, refer to TECH242654:

  http://www.symantec.com/docs/TECH242654
- Authentication schemes that use IP surrogates, (such as Proxy-IP and Origin-IP-Redirect) are not compatible with this configuration. Instead, Symantec recommends using either Proxy mode authentication for explicit deployments or Origin-Cookie-Redirect for Transparent deployments.

Example

A proxy administrator has received complaints that some users are being tracked as other users in their proxy forwarding deployment. Checking further, he finds that the child proxy is successfully authenticating all users and appropriately creating and populating the authenticated user and group information in forwarded HTTP headers to the parent proxy. In turn, the parent proxy is pulling the authenticated user and group details from HTTP headers as configured. The problem is that the parent proxy manages authentication on a per-session basis. A TCP session can contain several requests, all bound by the same source and destination IP addresses and TCP ports.

To remedy this situation, the administrator can force the parent proxy to handle authentication details on a per-transaction basis, rather than the default of per-session. This ensures that the user name for each request is extracted, preventing authentication and transaction logging inconsistency.

```
; Child proxy policy
<Proxy>
  authenticate(IWARealm) authenticate.mode(origin-cookie-redirect) authenticate.force(yes)

<Proxy>
  authenticated=yes action.Auth_Forward(yes)

define action Auth_Forward
  set(request.x_header.BC_Auth_User, "$(user:encode_base64)" )
```

```
  set(request.x_header.BC_Auth_Groups, "$(groups:encode_base64)" )
end

<Forward>
  forward("Outbound_proxy") forward.fail_open(no)
```

The element "Outbound_Proxy" above refers to an HTTP proxy forwarding host, set via the Management Console in **Forwarding > Forwarding Hosts**.

```
; Parent proxy policy
<Proxy>
  ALLOW authenticate(proxy_forward) authenticate.transaction(yes)

<Proxy>
  action.ControlRequestHeader1(yes)

define action ControlRequestHeader1
  delete(request.x_header.username)
  delete(request.x_header.BC_Auth_Groups)
end action ControlRequestHeader1
```

The preceding policy on the parent proxy also makes use of a policy substitution realm to pull user and group information from HTTP headers in the forwarded transaction. The CPL configuration for that policy substitution realm is below.

```
security policy-substitution create-realm proxy_forward
security policy-substitution edit-realm proxy_forward ;mode
identification username "$(request.x_header.BC_Auth_User:decode_base64)"
identification full-username "$(request.x_header.BC_Auth_User:decode_base64)"
```

authenticate.use_url_cookie()

Authenticate users who have third-party cookies explicitly disabled.

Syntax

```
authenticate.use_url_cookie(yes|no)
```

The default value is no. With a value of yes, if there is a problem loading the page (you get an error page or you cancel an authentication challenge), the cfauth cookie is displayed. You can also see the cookie in packet traces, but not in the browser URL window or history under normal operation.

Layer and Transaction Notes

- Layers: <Proxy>
- Transactions: HTTP proxy

Example

For requests from the specified subnet, do not authenticate users who have third-party cookies explicitly disabled.

```
<Proxy>
  client.address=HR_subnet authenticate.use_url_cookie(no)
```

See Also

- Properties: "authenticate.mode()" on page 360

authorize.add_group()

Add default group(s) to an authenticated user.

This property can be used to specify a single group or list of groups to add to an authenticated user. This property can only be used if the user is successfully authenticated.

Syntax

`authorize.add_group(group,...)`

Layer and Transaction Notes

- Layers: <Admin>, <Proxy>
- Transactions: all proxies

Example

Add a user to a default group if authentication succeeded but authorization failed due to a communication error with the authorization server.

```
<Proxy>
  authenticate(realm) authorize.tolerate_error(communication_error)

<Proxy>
  user.authorization_error=(communication_error) authorize.add_group(default_group)
```

bypass_cache()

Determines whether the cache is bypassed for a request. If set to yes, the cache is not queried and the response is not stored in the cache. Set to no to specify the default behavior, which is to follow standard caching behavior.

While static and dynamic bypass lists allow traffic to bypass the cache based on the destination IP address, this property is intended to allow a bypass based on the properties of the client; for example, you might use it to allow specific users or user groups to bypass the cache.

If this property is set to yes, the Flash video is played directly from the server even if the content is cached. If set to no (the default), cached portions of the video play from the cache and uncached portions play from the OCS.

Traffic is enforced on a per-stream basis and not the entire application.

Syntax

```
bypass_cache(yes|no)
```

The default behavior is no.

Layer and Transaction Notes

- Layers: <Proxy>
- Transactions: HTTP, HTTPS, FTP over HTTP, and transparent FTP transactions

Example

Bypass the cache for requests from this client IP address.

```
<Proxy>
  client.address=10.25.198.0 bypass_cache(yes)
```

See Also

- Properties: "advertisement()" on page 343, "always_verify()" on page 345, "cache()" on the next page, "cookie_sensitive()" on page 390, "direct()" on page 396, "dynamic_bypass()" on page 401, "force_cache()" on page 407, "pipeline()" on page 500, "refresh()" on page 503, "ttl()" on page 584, "ua_sensitive()" on page 585

cache()

Controls HTTP and FTP caching behavior. A number of CPL properties affect caching behavior.

- If"bypass_cache()" on the previous page is set to yes, then the cache is not accessed and the value of cache() is irrelevant.
- If cache(yes) is set, the "force_cache()" on page 407 property setting modifies the definition of what is considered a cacheable response.
- If this property is set to yes (the default), VOD content is cached. If set to no and the file is fully cached, the video is played from the cache. If set to no and the file is not cached or is partially cached, the video is played in pass-through mode.
- The properties cookie_sensitive(yes) and ua_sensitive(yes) have the same effect on caching as cache(no).

Other CPL properties that affect caching behavior are listed in the **See Also** section below. Remember that any conflict between their settings is resolved by CPL evaluation logic, which uses the property value that was last set when evaluation ends.

Syntax

```
cache(yes|no)
```

where:

- yes: (Default behavior) Cache responses from the origin server if they are cacheable.
- no: Do not store the response in the cache, and delete any object that was previously cached for this URL.

Layer and Transaction Notes

- Layers: <Cache>
- Transactions: all proxies

Example 1

Prevent objects at this URL from being added to the cache.

```
url=http://www.example.com/docs cache(no)
```

Example 2

Use the property in an exception to broader no-cache policy.

```
define url.domain condition non_cached_sites
  http://example1.com
```

```
  http://example2.com
end

<Cache>
  condition=non_cached_sites cache(no)

<Cache>
  url.extension=(gif, jpg) cache(yes) ; OK to cache these file types regardless.
```

See Also

- Properties: "advertisement()" on page 343, "always_verify()" on page 345, "bypass_cache()" on page 373, "cookie_sensitive()" on page 390, "direct()" on page 396, "dynamic_bypass()" on page 401, "force_cache()" on page 407, "pipeline()" on page 500, "refresh()" on page 503, "ttl()" on page 584, "ua_sensitive()" on page 585

check_authorization()

In connection with CAD (Caching Authenticated Data) and CPAD (Caching Proxy-Authenticated Data) support, `check_authorization( )` is used when you know that the upstream device sometimes (not always or never) requires the user to authenticate and be authorized for this object.

Setting the value to yes results in a GIMS (Get If Modified Since) to check authorization upstream, and the addition of a "Cache-Control: must-revalidate" header to the downstream response.

Syntax

```
check_authorization(yes|no)
```

The default behavior is no.

Layer and Transaction Notes

- Layers: `<Cache>`, `<Proxy>`
- Transactions: HTTP and RTSP proxy

See Also

- Conditions: "authenticated=" on page 81, "group=" on page 140, "has_attribute.name=" on page 142, "http.transparent_authentication=" on page 180, "realm=" on page 208, "user=" on page 312, "user.domain=" on page 316
- Properties: "authenticate()" on page 348, "authenticate.force()" on page 354

client.address.login.log_out_other()

Log out the any logins at this IP address other than the current login. This property is used to log out any other logins at the current IP address other than the current login of the transaction. Other users must re-authenticate at the this IP address before future transactions at this IP address can proceed.

Syntax

```
client.address.login.log_out_other(yes|no)
```

The default behavior is yes.

Layer and Transaction Notes

- Layers: <Admin>, <Proxy>
- Transactions: all proxies, administrator

Example

Log out the other logins of there is more than one login at this IP address.

```
<Proxy>
  client.address.login.count=2.. client.address.login.log_out_other(yes)
```

client.certificate.validate()

Determines whether client X.509 certificates are verified during the establishment of SSL connections.

Syntax

```
client.certificate.validate(yes|no)
```

The default value is taken from the configuration of the HTTPS service accepting the connection.

> **Note:** For best security, Symantec recommends that you do not disable certificate validation. If you must do so, disable it only for specific, trusted URLs, for example, using the "url=" on page 298 condition. Including `client.certificate.validate(no)` in policy disables all certificate validation for the affected transactions, including checks for the validity of the certificate (such as trust chain and validity date range), as well as checks on the well-formedness of the certificate (such as valid algorithm identifiers and extension fields).

Layer and Transaction Notes

- Layers: <SSL>
- Transactions: HTTPS forward and reverse proxy

Example

Disable certificate validation for the specified trusted domain.

```
<SSL>
  url.domain="example.com" client.certificate.validate(no)
```

See Also

- Properties: "server.certificate.validate()" on page 536

client.certificate.validate.ccl()

Validate the client certificate using the CA certificate list (CCL) certificate ID when "client.certificate.validate()" on the previous page is set to yes.

Syntax

`client.certificate.validate.ccl(`*`ccl_id`*`)`

When this property is not used, the certificate store for the default CCL is used.

Layer and Transaction Notes

- Layers: `<SSL-Intercept>`
- Transactions: HTTPS forward and reverse proxy transactions

Example

Validate using the specified CCL

```
<SSL-Intercept>
  url.domain="example.com" client.certificate.validate.ccl(testccl)
```

See Also

- Properties: "client.certificate.validate()" on the previous page, "server.certificate.validate.ccl()" on page 538

client.certificate.validate.check_revocation()

Determines whether client X.509 certificates will be checked for revocation.

Syntax

```
client.certificate.validate.check_revocation(auto|ocsp|local|no)
```

where:

- auto: (Default behavior) The certificate is checked through OCSP if available; otherwise, it is checked against locally installed revocation list.
- oscp: Check the certificate through OCSP.
- local: Check the certificate against the locally installed revocation list.
- no: The certificate is not checked for revocation.

Layer and Transaction Notes

- Layers: <SSL>
- Transactions: HTTPS forward and reverse proxy transactions

Example

Verify SSL client certificates and check against the OCSP.

```
<SSL>
  client.certificate.validate(yes) client.certificate.validate.check_revocation(oscp)
```

client.connection.dscp()

Controls client side outbound QoS/DSCP value.

Syntax

`client.connection.dscp(`*`dscp_value`*`)`

where:

`dscp_value`: One of the following:

- decimal value between 0 and 63
- `preserve`: (default behavior) track the incoming DSCP value on the primary server connection and use that as the value when sending packets on the client connections.
- `echo`: outbound packet's DSCP value will use the same value as the inbound packet's DSCP value.
- a class: `af11 | af12 | af13 | af21 | af22 | af23 | af31 | af32 | af33 | af41 | af42 | af43 | cs1 | cs2 | cs3 | cs4 | cs5 | cs6 | cs7 | ef`

Layer and Transaction Notes

- Layers: `<DNS-Proxy>`, `<Proxy>`
- Transactions: applies to all transactions; applies to HTTP/2 streams

Example

The first QoS policy rule sets the client outbound QoS/DSCP value to echo, and the second QoS policy rule sets the client outbound QoS/DSCP value to 50.

```
<Proxy>
  client.connection.dscp(echo)

<Proxy>
  client.connection.dscp(50)
```

client.connection.encrypted_tap()

Enables or disables encrypted tap of client-side traffic. When enabled, sends tapped traffic to the specified interface. Tapped data is presented in a TCP-like format which can be easily understood by common network traffic analysis tools like Wireshark and common network intrusion detection systems such as Snort.

Syntax

`client.connection.encrypted_tap(no|`*`interface`*`)`

where:

- no: Disable encrypted tap of client-side traffic.
- *interface*: Specify the interface for tapped encrypted SSL content on the client side. The form is *adapter:interface*.

Layer and Transaction Notes

- Layers: <SSL>
- Transactions: applies only to client connections

Example

Enable encrypted tap and send tapped traffic to the specified interface.

```
<SSL-Intercept>
  ssl.forward_proxy(stunnel)

<SSL>
  client.connection.encrypted_tap(1:0)

<SSL>
  server.certificate.validate(no)
```

See Also

- Properties: "server.connection.encrypted_tap()" on page 546
- Appendix D: "CPL Substitutions"

client.connection.tap()

Enables or disables tap of client-side HTTP, FTP, and TCP traffic. When enabled, sends tapped traffic to the specified interface. Tapped data is presented in a TCP-like format which can be easily understood by common network traffic analysis tools like Wireshark and common network intrusion detection systems such as Snort.

Syntax

`client.connection.tap(no|interface)`

where:

- no: Disable tap of client-side traffic.
- *interface*: Specify the interface for tapped content on the client side. The form is *adapter*:*interface*.

Layer and Transaction Notes

- Layers: <Proxy>
- Transactions: applies only to client connections

Example

Send tapped traffic to the specified interface.

```
<Proxy>
  client.connection.tap(1:0)
```

See Also

- Properties: "server.connection.encrypted_tap()" on page 546, "server.connection.tap()" on page 548
- Appendix D: "CPL Substitutions"

client.effective_address.connection()

Sets the IP address that the "client.effective_address=" on page 101 condition evaluates against, or resets the effective client IP address to the client address.

You can set the effective client IP address using HTTP request header fields; to handle cases where the header field is not present or does not contain a valid IP address, you can also specify explicit IP addresses. Policy uses the parameters in the specified order to look up the effective client IP address. This property has no effect if a valid IP address cannot be determined from the specified parameters.

> **Note:** Use this property only when the IP address can be determined at the time of connection. If the property is set to a value that is determined later during the transaction, the following policy compile error occurs:
>
> "Error: The field is not permitted since its value cannot be resolved at the time the substitution is performed."
>
> In this case, use "client.effective_address.request()" on page 386 instead.

> **Tip:** Attack detection policy uses the effective client address value set in this property. If policy includes both this property and "client.effective_address.request()" on page 386, the following warning occurs when installing policy:
>
> "Warning: The client effective address used for attack detection may be overridden for policy evaluation."
>
> This means that, due to policy rule evaluation, the effective client address might be overwritten by "client.effective_address.request()" on page 386. As a result, attack detection policy based on the value set by `client.effective_address.connection()` might not match. If you receive this message, you can perform a policy trace to help ensure that policy behaves as intended.

Syntax

`client.effective_address.connection(`*`HTTP_header`*`,...|`*`IP_address`*`,..|default)`

where:

- *HTTP_header*: A valid substitution string. If a valid IP address can't be determined from the first header, the next one in the list (if one exists) is attempted.

- *IP_address*: The IP address as a literal text string. If the first string is not a valid IP address, the next one in the list (if one exists) is attempted.
- default: Resets "client.effective_address=" on page 101 to the "client.address=" on page 87.

Layer and Transaction Notes

- Layers: <Admin>, <Cache>, <DNS-Proxy>, <Exception>, <Forward>, <Proxy> , <SSL>, <SSL-Intercept>
- Transactions: all

Example

For requests through the specified interface, set the effective address to 10.12.23.34.

```
<Proxy>
  client.interface=0.0.345 client.effective_address(10.12.23.34)
```

See Also

- Conditions: "client.address=" on page 87, "client.effective_address.country=" on page 102, "client.effective_address.is_overridden=" on page 105, "client.protocol=" on page 111
- Properties: "client.effective_address.request()" on the next page
- *ProxySG Log Fields and CPL Substitutions Reference*: https://techdocs.broadcom.com/content/broadcom/techdocs/us/en/symantec-security-software/web-and-network-security/proxysg/7-2/proxysg-log-fields-and-substitutions.html

client.effective_address.request()

Sets the IP address that the "client.effective_address=" on page 101 condition evaluates against, or resets the effective client IP address to the client address.

You can set the effective client IP address using HTTP request header fields; to handle cases where the header field is not present or does not contain a valid IP address, you can also specify explicit IP addresses. Policy uses the parameters in the specified order to look up the effective client IP address. This property has no effect if a valid IP address cannot be determined from the specified parameters.

> **Note:** Unlike "client.effective_address.connection()" on page 384, you can set this property to a value that is determined later during the transaction.

> **Tip:** Attack detection policy uses the effective client address value set in "client.effective_address.connection()" on page 384. If policy includes both this property and "client.effective_address.connection()" on page 384, the following warning occurs when installing policy:
>
> "Warning: The client effective address used for attack detection may be overridden for policy evaluation."
>
> This means that, due to policy rule evaluation, this property might overwrite the effective client address set by "client.effective_address.connection()" on page 384. As a result, attack detection policy based on the value set by "client.effective_address.connection()" on page 384 might not match. If you receive this message, you can perform a policy trace to help ensure that policy behaves as intended.

Syntax

`client.effective_address.request(`*`HTTP_header`*`,...|`*`IP_address`*`,..|default)`

where:

- *`HTTP_header`*: A valid substitution string. If a valid IP address can't be determined from the first header, the next one in the list (if one exists) is attempted.
- *`IP_address`*: The IP address as a literal text string. If the first string is not a valid IP address, the next one in the list (if one exists) is attempted.
- `default`: Resets "client.effective_address=" on page 101 to the "client.address=" on page 87.

Layer and Transaction Notes

- Layers: <Admin>, <Cache>, <DNS-Proxy>, <Exception>, <Forward>, <Proxy> , <SSL>, <SSL-Intercept>
- Transactions: all

Example

The following example uses request headers to obtain the client IP address.

```
<Proxy>
  ; Get X-Forwarded-For from all transactions with headers if valid
  ; Otherwise, use X-Client-IP if present and valid
  ; If neither are valid, use client.address
  client.effective_address.request("$(request.header.x-forwarded-for)", "$(request.x_header.x-client-
ip)")

<Proxy>
  client.effective_address=192.0.2.0 deny
```

Use Case: Use the client IP address from HTTPS transactions

In this example, the organization has deployed an additional load balancer, which can decrypt SSL transactions. The organization wants the HTTPS traffic through this load balancer to use the client IP address that the load balancer inserted.

To use the value in the XFFheader from HTTPS transactions, the administrator installs the following policy:

```
<Proxy>
  client.protocol=https client.effective_address.request("$(request.header.x-forwarded-for)")
```

Use Case: Use the client IP addresses from all HTTP and HTTPS transactions that have the headers

In this example, an organization wants to use XFF header values from all HTTP and HTTPS connections that have the header. The administrator installs the following policy:

```
client.effective_address.request("$(request.header.x-forwarded-for)")
```

See Also

- Conditions: "client.address=" on page 87, "client.effective_address.country=" on page 102, "client.effective_address.is_overridden=" on page 105, "client.protocol=" on page 111
- Properties: "client.effective_address.connection()" on page 384

- *ProxySG Log Fields and CPL Substitutions Reference*: https://techdocs.broadcom.com/content/broadcom/techdocs/us/en/symantec-security-software/web-and-network-security/proxysg/7-2/proxysg-log-fields-and-substitutions.html

client.ip_reputation.category()

Configure an IP reputation category and confidence level.

Syntax

`client.ip_reputation.`*`category1`*`[,`*`category2`*`,...](value,none)`

where:

- *`category`*: IP reputation category, including user-defined categories

  Symantec recommends using the form user_defined.category for user-defined categories.

- `value`: Confidence level for the specified reputation category or categories
- `none`: Any database entries for the specified reputation category or categories are suppressed and not access-logged

Layer and Transaction Notes

- Layers: `<Proxy>`
- Transactions: current HTTP proxy transaction; unavailable if the transaction is not associated with a client

Example

Create a custom IP reputation category for multiple known sources of attacks.

```
; defining subnets and IP ranges determined to be attack sources
define subnet attack_group_1
  192.2.0.2/24
  19.51.100.0/24
  203.0.113.0-203.113.0.198
end

; introduce user-defined reputation 'user_defined.attack'
; for subnet attack_group_1 with confidence level 6
<Proxy>
  client.address=attack_group_1 client.ip_reputation[user_defined.attack](6)
```

The category `user_defined.attack` applies to the current transaction only.

See Also

- Condition: "client.[effective_]address.ip_reputation=" on page 104
- *SGOS Administration Guide*, “Control Traffic Based on Client IP Reputation”

cookie_sensitive()

Used to modify caching behavior by declaring that the object served by the request varies based on cookie values. Set to yes to specify this behavior, or set to no for the default behavior, which caches based on HTTP headers.

Using `cookie_sensitive(yes)` has the same effect as setting "cache()" on page 374 to no.

There are a number of CPL properties that affect caching behavior, as listed in the **See Also** section below. Remember that any conflict between their settings is resolved by CPL evaluation logic, which uses the property value that was last set when evaluation ends.

Syntax

```
cookie_sensitive(yes|no)
```

The default value is no.

Layer and Transaction Notes

- Layers: `<Cache>`
- Transactions: HTTP proxy transactions, except FTP over HTTP transactions

See Also

- Properties: "advertisement()" on page 343, "always_verify()" on page 345, "bypass_cache()" on page 373, "direct()" on page 396, "force_cache()" on page 407, "pipeline()" on page 500, "refresh()" on page 503, "ttl()" on page 584, "ua_sensitive()" on page 585

delete_on_abandonment()

Specifies whether an object should be cached if client connection is interrupted before fetching of the object is completed. This property is ignored for Flash VOD caching.

Syntax

```
delete_on_abandonment(yes|no|auto)
```

where:

- yes: If all clients who may be simultaneously requesting a particular object close their connections before the object is delivered, the object fetch from the OCS is abandoned, and any prior instance of the object is deleted from the cache.
- no: If all clients who may be simultaneously requesting a particular object close their connections before the object is delivered, the object fetch from the origin server is completed and an instance of the object is copied to the cache.
- auto: (Default behavior) Behaves like yes if the connection is over an ADN or bandwidth-gain is enabled, and it behaves like no otherwise.

Layer and Transaction Notes

- Layers: <Cache>
- Transactions: all proxies; applies to HTTP/2 streams

See Also

- Properties:"advertisement()" on page 343, "always_verify()" on page 345, "bypass_cache()" on page 373, "cookie_sensitive()" on the previous page, "direct()" on page 396,"dynamic_bypass()" on page 401, "force_cache()" on page 407, "pipeline()" on page 500, "refresh()" on page 503, "ttl()" on page 584, "ua_sensitive()" on page 585

deny

Denies service.

Denial can be overridden by an "allow" on page 344 or "exception()" on page 402 in a later rule. To deny service in a way that cannot be overridden by a subsequent allow, use "force_deny()" on page 410 or "force_exception()" on page 412.

The relation between "authenticate()" on page 348 and "deny" above is controlled by the "authenticate.force()" on page 354 property. By default, "deny" above overrides "authenticate()" on page 348. Recall that this means that a transaction can be denied before authentication occurs, resulting in no user identification available for logging.

Similarly, the relation between "socks.authenticate()" on page 554 and "deny" above is controlled by the "socks.authenticate.force()" on page 556 property. By default, "deny" above overrides "socks.authenticate()" on page 554.

Syntax

```
deny
deny(details)
```

where:

- *details*: String defining a message to be displayed to the user. The details string can contain CPL substitution variables. Specifying details is equivalent to `exception(policy_denied, details)`. Test the exception being returned in an `<Exception>` layer using "exception.id=" on page 137.

Discussion

For HTTP, a `policy_denied` exception results in a 403 Forbidden response. This is appropriate when the denial does not depend on the user identity. When the denial does depend on user identity, use "deny.unauthorized()" on page 394 instead to give the user an opportunity to retry the request with different credentials.

Layer and Transaction Notes

- Layers: `<Admin>`, `<Cache>`, `<Proxy>` , `<SSL>`
- Transactions: all; applies to HTTP/2 streams, but not the underlying HTTP/2 connection.

Example

Deny requests for the address.

```
deny url.address=10.25.100.100
```

See Also

- Conditions: "exception.id=" on page 137
- Properties: "allow" on page 344, "authenticate.force()" on page 354, "deny.unauthorized()" on the next page, "force_deny()" on page 410, "never_refresh_before_expiry()" on page 495, "never_serve_after_expiry()" on page 496, "OK" on page 499, "remove_IMS_from_GET()" on page 504, "remove_PNC_from_GET()" on page 505, "remove_reload_from_IE_GET()" on page 506, "socks.authenticate()" on page 554, "socks.authenticate.force()" on page 556, "exception.format()" on page 405
- *ProxySG Access Log Fields and CPL Substitutions Reference*: https://techdocs.broadcom.com/content/broadcom/techdocs/us/en/symantec-security-software/web-and-network-security/proxysg/7-2/proxysg-log-fields-and-substitutions.html

deny.unauthorized()

This property instructs the appliance to issue a challenge (401 Unauthorized or 407 Proxy authorization required). This indicates to the client that the resource cannot be accessed with their current identity, but might be accessible using a different identity. The browsers typically respond by bringing up a dialog so the user can change their identity. (The details string appears in the challenge page so that if the user cancels, there is some additional help information provided).

Typically, use "deny" on page 392 if the policy rule forbids everyone access, but use `deny.unauthorized()` if the policy rule forbids only certain people.

Syntax

```
deny.unauthorized
deny.unauthorized(details)
```

where:

- *details*: String defining a message to be displayed to the user. The details string can contain CPL substitution variables.

If current policy contains rules that use the "authenticate()" on page 348 or "authenticate.force()" on page 354 properties, the `deny.unauthorized()` property is equivalent to exception(authorization_failed). If policy does not contain any rules that require authentication, `deny.unauthorized()` is equivalent to exception(policy_denied).

The identity of the exception being returned can be tested in an `<Exception>` layer using exception.id=.

Layer and Transaction Notes

- Layers: `<Proxy>`
- Transactions: HTTP transactions; for other protocols, the property is the equivalent to "deny" on page 392

See Also

- Conditions: "exception.id=" on page 137
- Properties: "deny" on page 392, "exception()" on page 402, "force_deny()" on page 410, "force_exception()" on page 412, "exception.format()" on page 405
- *ProxySG Log Fields and CPL Substitutions Reference*: https://techdocs.broadcom.com/content/broadcom/techdocs/us/en/symantec-security-software/web-and-network-security/proxysg/7-2/proxysg-log-fields-and-substitutions.html

detect_protocol()

Determines whether to invoke protocol recognition, and which protocols should be recognized. When one of the specified protocols is detected, the connection will be handled by the appropriate application proxy.

Syntax

```
detect_protocol.protocol(yes|no)
detect_protocol(all|none|protocol,..)
detect_protocol[protocol,..](yes|no)
```

where:

- *protocol*: One of the following: bittorrent, edonkey, fasttrack, gnutella, epmapper, http, https, sips, ssl

The default value is `all`.

Layer and Transaction Notes

- Layers: `<Proxy>` , `<SSL-Intercept>`
- Transactions: SOCKS, HTTP, TCP Tunnel, and SSL Intercept transactions

Example

Detect gnutella protocol.

```
<Proxy>
  detect_protocol(gnutella)
```

See Also

- Properties:"force_protocol()" on page 414

direct()

Used to prevent requests from being forwarded to a parent proxy or SOCKS server, when the appliance is configured to forward requests.

When set to `yes`, `<Forward>` layer policy is not evaluated for the transaction.

Syntax

```
direct(yes|no)
```

The default value is `no`, which allows request forwarding.

Layer and Transaction Notes

- Layers: `<Cache>`, `<Proxy>`
- Transactions: Does not apply to FTP over HTTP or transparent FTP transactions

Example

Do not forward requests from myhost.

```
<Proxy>
  url.host="myhost" direct(yes)
```

See Also

- Properties: "bypass_cache()" on page 373, "dynamic_bypass()" on page 401, "force_cache()" on page 407, "forward()" on page 415, "reflect_ip()" on page 502

dns.respond()

Terminates a proxied DNS query with the given response.

Syntax

`dns.respond.a(`*response*`|`*range*`)`

where:

- *response*: One of noerror, formerr, servfail, nxdomain, notimp, refused, yxdomain, yxrrset, nxrrset, notauth, notzone
- *range*: Numeric range from 0 to 15.

Layer and Transaction Notes

- Layers: `<DNS-Proxy>`
- Transactions: DNS proxy

Example

DNS queries using QTYPEs other than “PTR” or “A” are considered “not implemented.” Any DNS query for a host ending in example.com is refused.

```
<DNS-proxy>
  dns.request.type=!(A||PTR) dns.respond(notimp)
```

```
<DNS-proxy>
  dns.request.name=.example.com dns.respond(refused)
```

dns.respond.a()

Terminates a proxied DNS query of type 'A' with the given response.

Syntax

```
dns.respond.a(ip_address[,ip_address]*[,ttl])
dns.respond.a(hostname[,ip_address]*[,ttl])
dns.respond.a([hostname,]vip[,ttl])
```

Layer and Transaction Notes

- Layers: <DNS-Proxy>
- Transactions: DNS proxy

Example

DNS queries for host1.example.com are resolved to 10.10.10.1 with a TTL of 7200 seconds. DNS queries for host2.example.com are resolved to 10.10.10.2 with a CNAME of myhost.example.com and a TTL of 9600 seconds.

```
<DNS-Proxy>
  dns.request.name=host1.example.com dns.respond.a(10.10.10.1, 7200)
  dns.request.name=host2.example.com dns.respond.a(myhost.example.com, 10.10.10.2, 9600)
```

dns.respond.aaaa()

Generates type AAAA RRs (one RR per IPv6 address) in the answer section of the response. You can also replace [*ip_address*]* with theVIP keyword. The VIP keyword creates a type AAAA RR with the appliance 's interface IP.

Syntax

```
dns.respond.aaaa(ip_address[,ip_address]*[,ttl])
dns.respond.aaaa(hostname[,ip_address]*[,ttl])
dns.respond.aaaa([hostname,]vip[,ttl])
```

Notes

- You cannot mix `vip` with *ip_address*. For example, `dns.respond.a(vip, 10.1.1.1)` is not allowed.
- The maximum number of *ip_address* is 35.
- If *hostname* is present, type CNAME RR will be inserted in the response. All *ip_address* type AAAA RR will reference *hostname* instead of the qname in the question section. If there is no *ip_address* following *hostname*, only type CNAME RR is generated in the response
- Default *ttl* value is 3600.

Layer and Transaction Notes

- Layers: `<DNS-Proxy>`
- Transactions: DNS proxy

Example 1

A DNS response with an AAAA RR (2001::1) is sent to the requestor if the DNS request type is AAAA:

```
<DNS-Proxy>
  dns.request.type=AAAA dns.respond.aaaa(2001::1)
```

Example 2

If the DNS response from server contains an AAAA RR equaling 2001::1, it will send a DNS response to client with an AAAA RR (2001::2).

```
<DNS-Proxy>
   dns.response.aaaa=2001::1 dns.respond.aaaa(2001::2)
```

> **Note:** The DNS Proxy caches IPv6 AAAA records.

dns.respond.ptr()

Terminates a proxied DNS query of type “PTR” with the given response.

Syntax

`dns.respond.ptr(`*`hostname`*`[,`*`ttl`*`])`

Layer and Transaction Notes

- Layers: `<DNS-Proxy>`
- Transactions: DNS proxy

Example

Reverse DNS queries for 10.10.10.1 are resolved to host1.example.com with a TTL of 7200 seconds. Reverse DNS queries for 10.10.10.2 are resolved to host2.example.com with the default TTL.

```
<DNS-Proxy>
  dns.request.address=10.10.10.1 dns.respond.ptr(host1.example.com,7200)
  dns.request.address=10.10.10.2 dns.respond.ptr(host2.example.com)
```

dynamic_bypass()

Used to indicate that a particular transparent request is not to be handled by the proxy, but instead be subjected to ProxySG appliance dynamic bypass methodology.

The `dynamic_bypass(yes)` property takes precedence over "authenticate()" on page 348; however, a committed denial takes precedence over `dynamic_bypass(yes)`.

Syntax

```
dynamic_bypass(yes|no)
```

The default behavior is `no`.

Layer and Transaction Notes

- Layers: `<Proxy>`
- Transactions: Applies to transparent HTTP transactions only, including HTTP/2 streams

See Also

- Properties: "advertisement()" on page 343, "always_verify()" on page 345,"bypass_cache()" on page 373 "cache()" on page 374, "cookie_sensitive()" on page 390, "delete_on_abandonment()" on page 391,"direct()" on page 396,"force_cache()" on page 407,"pipeline()" on page 500, "refresh()" on page 503, "ttl()" on page 584, "ua_sensitive()" on page 585

exception()

Selects a built-in or user-defined response to be returned to the user. This property is overridden by "allow" on page 344 or "deny" on page 392. To set an exception that cannot be overridden by allow, use "force_exception()" on page 412. The identity of the exception being returned can be tested in an <Exception> layer using "exception.id=" on page 137.

> **Note:** When the exception response selected would have a Content-Length of 512 or fewer bytes, Internet Explorer may substitute “friendly” error messages. To prevent this behavior use exception.autopad(yes).

Syntax

`exception(`*`exception_id, details, string_name`*`)`

where:

- `exception_id`: Name of either a built-in exception or a custom exception of the form `user_defined.`*`exception_id`* that refers to a user-defined exception page.
- *`details`*: Text string that is substituted for `$(exception.details)` within the selected exception.
- *`string_name`*: String name, as defined by define string, that is substituted for `$(exception.format)`.

Layer and Transaction Notes

- Layers: <Admin>, <Cache>, <Proxy> , <SSL>
- Transactions: all

Example

For forbidden sites, display the specified user-defined exception.

```
<Proxy>
  condition=forbidden_sites exception(content_filter_denied,"attempt to access forbidden site," my_
blocked_content_format)

define string my_blocked_content_format
  ><html>
  ><head>
  ><title>Notice<\title>
  ><\head>
  ><body>
  >Access Blocked
  >Reason $(exception.details)
```

```
  ><\body>
  ><\html>
end

define condition forbidden_sites
  url.domain=//badcompany.com/...; additional sites omitted
end
```

See Also

- Conditions: "exception.id=" on page 137
- Properties: "allow" on page 344, "deny" on page 392, "deny.unauthorized()" on page 394, "exception.autopad()" on the next page, "force_deny()" on page 410, "force_exception()" on page 412,"exception.format()" on page 405
- Definitions: "define string" on page 654
- *ProxySG Log Fields and CPL Substitutions Reference*: https://techdocs.broadcom.com/content/broadcom/techdocs/us/en/symantec-security-software/web-and-network-security/proxysg/7-2/proxysg-log-fields-and-substitutions.html

exception.autopad()

Pad an HTTP exception response by including trailing white space in the response body so that Content-Length is at least 513 characters.

A setting of yes is used to prevent Internet Explorer from substituting friendly error messages in place of the exception response being returned, when the exception as configured would have a Content-Length of less than 512 characters.

Syntax

```
exception.autopad(yes|no)
```

The default behavior is `yes`.

Layer and Transaction Notes

- Layers: `<Exception>`
- Transactions: HTTP proxy

See Also

- Conditions: "exception.id=" on page 137
- Properties: "exception()" on page 402, "exception.format()" on the facing page, "force_exception()" on page 412

exception.format()

Selects the format to use when preparing exceptions to be displayed to users.

By default, the exception format is specified by the configuration in the exceptions file on the local appliance. This property is used to override the configured format.

> **Note:** This property is used in Common Policy (cloud and on-premises ProxySG hybrid) deployments to render the exceptions generated by the proxy in the same format as is used by the cloud.

Syntax

`exception.format(`*`string_name`*`|default)`

where:

- *`string_name`*: Name of the CPL string defined using "define string" on page 654.
- `default`: (Default behavior) Use the standard exception format specified in the ProxySG appliance configuration.

Layer and Transaction Notes

- Layers: `<Admin>`, `<Cache>`, `<Proxy>` , `<SSL>`
- Transactions: all

Example 1

Override the configured exception format.

```
; conditions can be added to the following rule as necessary
; exception.format(my_format)

define string my_format
  ><html>
  ><head>
  ><title>$(exception.summary)
  ><\head>
  ><body>
  >Access Blocked
  >Reason $(exception.details)
  ><\body>
  ><\html>
end
```

Example 2

The administrator of an appliance configured for a Common Proxy deployment wants to use the configured exception pages rather than the format used by exceptions served from the cloud. She includes the following in the local policy file:

```
<Proxy>
  exception.format(default)
```

See Also

- Properties: "deny" on page 392, "exception()" on page 402, "exception.autopad()" on page 404, "force_deny()" on page 410, "force_exception()" on page 412
- Definitions: "define string" on page 654

force_cache()

Specify one or more reasons to force caching of HTTP responses that are otherwise considered uncacheable. The value of the force_cache( ) property is ignored for HTTP unless all of the following property settings are in effect:

- bypass_cache(no)
- cache(yes)
- cookie_sensitive(no)
- ua_sensitive(no)

For streaming proxies, the value of the `force_cache( )` property is ignored unless bypass_cache(no) is set.

This property is typically used in conjunction with conditions; see **Example 3** below.

Syntax

```
force_cache.reason(yes|no)
force_cache[reason1,...](yes|no)
```

where:

- *reason*: Supported reason, listed in the table below.
- yes: Force caching for the specified reason(s).
- no: Disable forced caching for the specified reason(s).

```
force_cache(reason1,...|all|no)
```

where:

- reason: Supported reason, listed in the table below. resets previous settings and forces caching for the specified reason(s).
- all: Force caching for all supported reasons.
- no: (Default behavior for HTTP and streaming protocols) Disable forced caching.

> **Note:** The HTTP proxy supports all nine reasons, whereas streaming proxies have limited support.

Reason	Description
expired	The HTTP response includes the Expires header and its value is in the past. Not supported on streaming proxies.
missing-http-version	The first line of the HTTP response does not include HTTP/ at the beginning, so the appliance does not know the protocol/version. Not supported on streaming proxies.
personal-pages	For advanced users or Support only. The appliance looks for a non-304, non-image type N response first, and then checks to see if it has either a query string or a Cookie header in the request. If either a query string or Cookie request header is present, the appliance makes it non-cacheable, but force_cache(personal-pages) can override it. Not supported on streaming proxies.
private	The HTTP response includes the Cache-Control: private response header/meta tag. Supported on Windows Media over RTSP or HTTP, but not on RealMedia over RTSP or HTTP.
response-no-cache	The HTTP response includes both Cache-Control: no-cache and Pragma: no-cache response header/meta tag. Supported on Windows Media over RTSP or HTTP and RealMedia over RTSP or HTTP.
response-no-store	The HTTP response includes the Cache-Control: no-store response header/meta tag. Supported on Windows Media over RTSP or HTTP, but not on RealMedia over RTSP or HTTP.
set-cookie	The HTTP response includes both Set-Cookie and Set-Cookie2 response headers. Not supported on streaming proxies.
unknown-transfer-encoding	The HTTP response includes the Transfer-Encoding response header but its value is unknown. Not supported on streaming proxies.
vary	The HTTP response includes the vary response header. Not supported on streaming proxies.

Layer and Transaction Notes

- Layers: `<Cache>`
- Transactions: proxy transactions, which execute both `<Cache>` and `<Proxy>` layers

The HTTP proxy supports all nine reasons, whereas streaming proxies have limited support.

Example 1

Force caching for set-cookie reason for all responses.

```
<Cache>
  force_cache.set-cookie(yes)
```

Example 2

Force caching for both expired and private reasons for all responses.

```
<Cache>
  force_cache[expired, private](yes)
```

Example 3

This example incorporates conditions—unique conditions produce different actions—so that the cache is forced for different reasons. Both force_cache reasons are still in effect if both conditions are met.

```
; force caching for expired & private reasons but under different conditions
define condition is_mp3
  response.header.Content-Type="application/mp3"
end

define condition is_somesite
  url.domain=somesite.com
end

<Cache>
  condition=is_mp3 force_cache.expired(yes)

<Cache>
  condition=is_somesite force_cache.private(yes)
```

Example 4

Reset the reason to set-cookie only. In next layer, set reason to private only.

```
<Cache>
  force_cache(set-cookie)


<Cache>
  force_cache(private)
```

> **Note:** At the end of policy evaluation, only the private reason is in effect (it overrides any previous `force_cache` rules). If you intend to use rules to trigger caching for different reasons, keep in mind that policy is evaluated from top to bottom, with later rules taking precedence over earlier rules.

See Also

- Properties: "advertisement()" on page 343, "always_verify()" on page 345,"bypass_cache()" on page 373, "cache()" on page 374, "cookie_sensitive()" on page 390, "dynamic_bypass()" on page 401, "pipeline()" on page 500, "refresh()" on page 503, "ttl()" on page 584, "ua_sensitive()" on page 585

force_deny()

This property is similar to "deny" on page 392 with the following exceptions:

- It cannot be overridden by "allow" on page 344 or "authenticate.force()" on page 354
- Overrides any pending termination (that is, if a "deny" on page 392 has already been matched, and a `force_deny()` or"force_exception()" on page 412 is subsequently matched, the latter commits.
- Commits immediately (that is, the first one matched applies).

The force_deny( ) property is equivalent to force_exception(policy_denied).

Syntax

```
force_deny
force_deny(details)
```

where:

- *details*: Text string that will be substituted for `$(exception.details)` within the policy_denied exception. The details string may also contain CPL substitution patterns.

Layer and Transaction Notes

- Layers: `<Admin>`, `<Cache>`, `<Proxy>` , `<SSL>`
- Transactions: all; applies to HTTP/2 streams, but not the underlying HTTP/2 connection

Example

Block Facebook videos.

```
<Proxy>
  url.regex="fbcdn\-photos\-[a-z]\-a\.akamaihd\.net" allow
  condition=facebook-referer condition=FB-video force_deny

define condition facebook-referer
  request.header.Referer="facebook.com"
  request.header.Referer="(.*)\.akamaihd\.net"
  request.header.Referer="www.youtube.com/embed/"
end condition facebook-referer
```

See Also

- Conditions: "exception.id=" on page 137
- Properties: "deny" on page 392, "force_exception()" on page 412

- *ProxySG Log Fields and CPL Substitutions Reference*: https://techdocs.broadcom.com/content/broadcom/techdocs/us/en/symantec-security-software/web-and-network-security/proxysg/7-2/proxysg-log-fields-and-substitutions.html

force_exception()

This property is similar to exception() with the following exceptions:

- Cannot be overridden by "allow" on page 344 or by "authenticate.force()" on page 354.
- Overrides any pending termination (that is, if "deny" on page 392 has already been matched, and "force_deny()" on page 410 or "force_exception()" above is subsequently matched, the latter commits.
- Commits immediately (that is, the first one matched applies).

Syntax

`force_exception(`*`exception_id, details, string_name`*`)`

where:

- *`exception_id`*: Either the name of a built-in exception or a name of the form `user_defined.`*`exception_id`* that refers to a user-defined exception page.
- *`details`*: Text string that is substituted for `$(exception.details)` within the selected exception.
- *`string_name`*: String name, as defined by define string, that is substituted for `$(exception.format)`. The named string overrides the format field of the exception. The string can contain substitutions.

Layer and Transaction Notes

- Layers: `<Admin>`, `<Cache>`, `<Proxy>` , `<SSL>`
- Transactions: all

Example

Return a specified exception based on source subnet (URL path).

```
<Proxy>
  client.address=10.10.10.0/24 url.path.exact="/int_dept" force_exception(user-defined.errorA)
  client.address=10.20.20.0/24 url.path.exact="/int_dept" force_exception(user-defined.errorB)
```

See Also

- Conditions: "exception.id=" on page 137
- Properties: "deny" on page 392, "exception()" on page 402, "exception.autopad()" on page 404, "force_deny()" on page 410, "exception.format()" on page 405

- *ProxySG Log Fields and CPL Substitutions Reference*: https://techdocs.broadcom.com/content/broadcom/techdocs/us/en/symantec-security-software/web-and-network-security/proxysg/7-2/proxysg-log-fields-and-substitutions.html

force_protocol()

Specifies that the client connection should be treated as a particular protocol type. The connection is handled by the appropriate application proxy.

Syntax

```
force_protocol(no|ssl|http|https|bittorrent|edonkey|gnutella|epmapper)
```

The default value is no.

Layer and Transaction Notes

- Layers: <Proxy>, <SSL-Intercept>
- Transactions: SOCKS, HTTP, TCP Tunnel, and SSL Intercept transactions

Example

Handle client connections with gnutella proxy.

```
<Proxy>
  force_protocol(gnutella)
```

See Also

- Properties: "detect_protocol()" on page 395

forward()

Determines forwarding behavior.

There is a global configuration setting (in the Management Console, select **Configuration > Forwarding > Sequence**) for the default forwarding failover sequence. The `forward( )` property is used to override the default forwarding failover sequence with a specific list of host and/or group aliases. The list of aliases might contain the special token default, which expands to include the default forward failover sequence defined in configuration.

Duplication is allowed in the specified alias list only in the case where a host or group named in the default failover sequence is also named explicitly in the *alias_list*.

In addition, there is a global configuration setting (in the Management Console, select **Configuration > Forwarding > Failure Mode**) for the default forward failure mode. The"forward.fail_open()" on the next page property overrides the configured default.

Syntax

`forward(`*alias_list*`|no)`

where:

- *alias_list*: Forward this request through the specified alias list, which might refer to both forward hosts and groups. The appliance attempts to forward this request through the specified hosts or groups, in the order specified by the list. It proceeds to the next alias as necessary when the current host or group is down, as determined by health checks.
- no: (Default behavior) Do not forward this request through a forwarding host. A SOCKS gateway or ICP host may still be used, depending on those properties. If neither are set, the request is sent directly to the origin server. No overrides the default sequence defined in configuration.

Layer and Transaction Notes

- Layers: <Forward>
- Transactions: all transactions except administrator, instant messaging, and SOCKS

Example

See the example for "has_client=" on page 145.

See Also

- Properties: "direct()" on page 396, "dynamic_bypass()" on page 401, "reflect_ip()" on page 502, "refresh()" on page 503, "socks_gateway()" on page 557, "socks_gateway.fail_open()" on page 558, "streaming.transport()" on page 569

forward.fail_open()

Controls whether the appliance terminates or continues to process the request if the specified forwarding host or any designated backup or default cannot be contacted.

There is a global configuration setting (in the Management Console, select **Configuration > Forwarding > Failure Mode**) for the default forward failure mode. The `forward.fail_open( )` property overrides the configured default.

Syntax

```
forward.fail_open(yes|no)
```

where:

- `yes`: Continue to process the request if the specified forwarding host or any designated backup or default cannot be contacted. This may result in the request being sent through a SOCKS gateway or ICP, or may result in the request going directly to the origin server.
- no: (Default behavior) Terminate the request if the specified forwarding host or any designated backup or default cannot be contacted.

Layer and Transaction Notes

- Layers: `<Forward>`
- Transactions: all transactions except administrator, instant messaging, and SOCKS

Example

Force forwarding of clientless connections.

```
<Forward>
  ; catch all clientless transactions and forward them
  has_client=no forward.fail_open(no) forward(forwarding_host-name)
```

See Also

- Properties: "bypass_cache()" on page 373, "dynamic_bypass()" on page 401, "forward()" on the previous page, "reflect_ip()" on page 502 "socks_gateway()" on page 557, "socks_gateway.fail_open()" on page 558

ftp.match_client_data_ip()

Sets whether to make a data connection to the client with the control connection's IP address or the local physical IP address.

Syntax

```
ftp.match_client_data_ip(yes|no)
```

where:

- yes: Make the data connection using the control connection's IP address.
- no: Make the data connection using the local physical IP address.

Layer and Transaction Notes

- Layers: <Proxy>
- Transactions: FTP proxy transactions and FTPS implicit and explicit proxy transactions

Example

Make the data connection using the control connection's IP address.

```
<Proxy>
  ftp.match_client_data_ip(yes)
```

ftp.match_server_data_ip()

Sets whether to make a data connection to the server with the control connection's IP address or the local physical IP address.

Syntax

```
ftp.match_server_data_ip(yes|no)
```

where:

- yes: Make the data connection using the control connection's IP address.
- no: Make the data connection using the local physical IP address.

Layer and Transaction Notes

- Layers:<Proxy>
- Transactions: FTP proxy transactions and implicit and explicit FTPS proxy transactions.

Example

Make the data connection using the control connection's IP address.

```
<Proxy>
  ftp.match_server_data_ip(yes)
```

ftp.server_connection()

Determines when the control connection to the server is established. If set to deferred, the proxy defers establishing the control connection to the server.

Syntax

```
ftp.server_connection(deferred|immediate)
```

The default value is `immediate`.

> **Note:** This property does not apply to implicit FTPS; for implicit FTPS, control connection to the server is always established immediately. For explicit FTPS, the server connection might be deferred until the AUTH TLS command is received.

Layer and Transaction Notes

- Layer: `<Cache>`, `<Proxy>`
- Transactions: FTP proxy transactions and explicit FTPS proxy transactions.

See Also

- Properties:"ftp.server_data()" on the next page, "ftp.transport()" on page 421

ftp.server_data()

Determines the type of data connection to be used with this FTP(S) transaction.

Syntax

```
ftp.server_data(auto|active|passive)
```

where:

- `auto`: First attempt a passive (PASV for IPv4, EPSV for IPv6) data connection. If this fails, switch to active (PORT for IPv4, EPRT for IPv6).
- `active`: Use an active data connection.
- `passive`: Use a passive data connection. Note that passive data connections are not allowed by some firewalls.

Layer and Transaction Notes

- Layers: `<Forward>`
- Transactions: FTP proxy transactions and implicit and explicit FTPS proxy transactions

See Also

- Properties: "ftp.server_connection()" on the previous page, "ftp.transport()" on the facing page

ftp.transport()

Determines the upstream transport mechanism.

This setting is not definitive. It depends on the capabilities of the selected forwarding host.

Syntax

```
ftp_transport(auto|ftp|http)
```

where:

- `auto`: (Default behavior) Use the default transport for the upstream connection, as determined by the originating transport and the capabilities of any selected forwarding host.
- `ftp`: Use FTP as the upstream transport mechanism.
- `http`: Use HTTP as the upstream transport mechanism.

Layer and Transaction Notes

- Layers: `<Forward>`
- Transactions: WebFTP transactions where the client uses the HTTP protocol to request a URL with an ftp: schema.

See Also

- Properties: "ftp.server_connection()" on page 419, "ftp.server_data()" on the previous page

http.allow_compression()

Determines whether the HTTP proxy is allowed to compress data in transit.

Syntax

```
http.allow_compression(yes|no)
```

The default value is no.

Layer and Transaction Notes

- Layers: <Cache>, <Proxy>
- Transactions: HTTP transactions (proxy, refresh, pipeline)

Example

Compress HTTP transaction data.

```
<Proxy>
  http.allow_compression(yes)
```

See Also

- Properties:"http.allow_decompression()" on the facing page

http.allow_decompression()

Determines whether the HTTP proxy is allowed to decompress data in transit.

Syntax

```
http.allow_decompression(yes|no)
```

The default value is no.

Layer and Transaction Notes

- Layers: <Cache>, <Proxy>
- Transactions: HTTP transactions (proxy, refresh, pipeline)

Example

Decompress HTTP transaction data.

```
<Proxy>
  http.allow_decompression(yes)
```

See Also

- Properties: "http.allow_compression()" on the previous page

http.client.allow_encoding()

Determines which encodings are allowed in the response sent to the client.

Syntax

```
http.client.allow_encoding.encoding(yes|no)
http.client.allow_encoding[encoding,..](yes|no)
http.client.allow_encoding(encoding,..|client,..)
```

where:

- *encoding*: One of `br`, `deflate`, or `gzip`.
- *client*: (Default behavior) Specified client(s) that will be replaced by the encoding(s) specified in client request.

Layer and Transaction Notes

- Layers: `<Admin>`, `<Cache>`, `<DNS-Proxy>`, `<Exception>`, `<Forward>`, `<Proxy>` , `<SSL>`, `<SSL-Intercept>`
- Transactions: HTTP transactions (proxy, refresh, pipeline)

Example

Allow gzip encoding in server responses.

```
<Proxy>
  http.client.allow_encoding(gzip)
```

See Also

- Properties: "http.server.accept_encoding()" on page 480

http.client.persistence()

Controls persistence of individual client connections to the HTTP client.

Syntax

```
http.client.persistence(yes|no|preserve)
```

where:

- yes: (Default behavior) Persist connection to the HTTP client.
- no: After the current transaction is complete, the client connection is dropped.
- preserve: Reflect the server's persistence to the client connection.

Layer and Transaction Notes

- Layers: <Exception>, <Proxy>
- Transactions: HTTP proxy

Example

Disable persistent connections for clients from a specified subnet going to a particular host and retrieving a particular content type in the response.

```
<Proxy>
  client.address=10.10.167.0/8 url.host=my_host.my_business.com response.header.Content-
Type="text/html" http.client.persistence(no)
```

See Also

- Properties: "http.server.persistence()" on page 485

http.client.recv.timeout()

Sets the socket timeout for receiving bytes from the client.

Syntax

```
http.client.recv.timeout(auto|recv-timeout)
```

where:

- *recv-timeout*: Amount of time (in seconds) to wait to receive data from the client before timing out. Default is 120.

Layer and Transaction Notes

- Layers: <Proxy>
- Transactions: HTTP proxy

Example

Requests from HR_subnet get a receive timeout of 200 seconds. Any request heading to a host that ends in example.com gets a receive timeout of 20 seconds. All refresh traffic except that for example.com hosts gets a receive timeout of 300 seconds.

```
define subnet HR_subnet
  10.10.0.0/16
end

<Proxy>
  client.address=HR_subnet http.client.recv.timeout(200)

<Forward>
  server_url.domain=example.com http.server.recv.timeout(20) http.refresh.recv.timeout(auto)
  http.refresh.recv.timeout(300)
```

http.compression_level()

Determines the compression level used by HTTP proxy when "http.allow_compression()" on page 422 is true.

Syntax

```
http.compression_level(low|medium|high)
```

The default value is `low`.

Layer and Transaction Notes

- Layers: `<Cache>`, `<Proxy>`
- Transactions: HTTP transactions (proxy, refresh, pipeline)

Example

Set compression level to medium for HTTP proxy transactions.

```
<Proxy>
  http.compression_level(medium)
```

See Also

- Properties: "http.allow_compression()" on page 422

http.csrf.authentication_link()

Used in WAF deployments to specify the authentication details to be included in a Cross-Site Request Forgery (CSRF) token and is used in conjunction with http.csrf.token.insert and http.csrf.detection.

The information contained in a CSRF authentication token is encrypted and signed with a cryptographically-secure signature. An anti-CSRF token cannot be deciphered by an entity other than the appliance that generated it, or another appliance that has been configured with the same private key.

Anti-CSRF tokens are used to accurately identify subsequent POST requests as genuine, thereby preventing malicious actors from initiating unwanted actions on behalf of an authenticated user.

Syntax

```
http.csrf.authentication_link(user_id)
http.csrf.authentication_link(client_ip)
http.csrf.authentication_link(user_id,client_ip)
```

where:

- *user_id*: (Default behavior) Authenticated user's username.
- *client_ip*: Authenticated user's IP address.
- *user_id,client_ip*: The token will attempt to first use the *user_id* as the identifier. If one is not present, the *client_ip* is used to identify the user in the authentication token.

> **Note:** When using this property with *client_ip*, make sure that the proxy generating the token can see the client's actual IP address. Proxy deployments that use NAT, load-balancing, and proxy chain configurations can obscure a client's original IP address.

> **Note:** If client source IP addresses are obfuscated, but requests that reach the proxy include an `X-Forwarded-For` header from the child proxy in a proxy chain, you can use "client.effective_address.request()" on page 386 to force the proxy to use the appropriate client IP.

Layer and Transaction Notes

- Layers: `<Proxy>`
- Transactions: HTTP proxy

Example

Insert and validate a CSRF authentication token with a lifespan of 1200 seconds that attempts to use the userid to identify the user. If that is not available, the IP address is used. If a valid token is not found, block requests that fail CSRF validation:

```
<Proxy>
  authenticate(WindowsIWArealm) authenticate.force(yes) authenticate.mode(auto)

<Proxy>
  http.csrf.token.insert(1200) http.csrf.authentication_link(user_id,client_ip)

<Proxy>
  http.csrf.detection(block)
```

See Also

- Properties: "http.csrf.detection()" on the next page, "http.csrf.token.insert()" on page 432, "http.csrf.token.name()" on page 433
- *SGOS Web Application Firewall (WAF) Solutions Guide*

http.csrf.detection()

Used in WAF deployments to validate a Cross-Site Request Forgery (CSRF) token in client HTTP POST requests. The specific type of validation information is determined in "http.csrf.authentication_link()" on page 428

When used in policy, only form-type HTTP POST requests (either URL-encoded or multipart-form) will be examined for CSRF tokens. CSRF tokens are defined with "http.csrf.token.insert()" on page 432

Syntax

```
http.csrf.detection(block|monitor|ignore)
```

where:

- `block`: Proxy will validate CSRF tokens wherever possible. If a valid token is not found, the request is denied and reported in the `x-bluecoat-waf-block-details` and `x-bluecoat-waf-attack-family` access log fields.
- `monitor`: Pproxy will validate CSRF tokens wherever possible. If a valid token is not found, it will be reported in the `x-bluecoat-waf-monitor-details` and `x-bluecoat-waf-attack-family` access log fields.
- `ignore`: While CSRF tokens may or may not be present, no action will be taken. This has the same impact as not having this property in policy and is used for false positive mitigation.

Layer and Transaction Notes

- Layers: `<Proxy>`
- Transactions: HTTP proxy

Example

Using a CSRF authentication token with a lifespan of 1200 seconds that uses the userid to identify the client in their next HTTP POST request, block requests that fail CSRF validation:

```
<Proxy>
  authenticate(WindowsIWArealm) authenticate.force(yes) authenticate.mode(auto)

<Proxy>
   http.csrf.token.insert(1200) http.csrf.authentication_link(user_id)

<Proxy>
  http.csrf.detection(block)
```

See Also

- Properties: "http.csrf.authentication_link()" on page 428, "http.csrf.token.insert()" on the next page, "http.csrf.token.name()" on page 433
- *SGOS Web Application Firewall (WAF) Solutions Guide*

http.csrf.token.insert()

This property is used in WAF deployments as a method to prevent Cross-Site Request Forgery (CSRF) attacks on WAF-protected web application servers. CSRF attacks are performed when a malicious site poses as a user who has recently authenticated on a web server, in order to act maliciously on that user's behalf.

The anti-CSRF token inserted with this property includes data that the proxy uses to identify the user after they have authenticated against the web server. After the initial authentication exchange, a users' subsequent HTTP POST request includes a token that identifies them. The proxy validates the token. If it is valid the user is permitted to access the web server. If the token is invalid or missing, the request may be blocked or logged, depending on the configuration of "http.csrf.detection()" on page 430 in policy.

Previous releases of SGOS include CSRF protection in the form of referer header checking and CAPTCHAs. These methods are only suitable in limited cases.

Syntax

`http.csrf.token.insert(yes|no|`*`no`*`)`

where:

- `yes`: Proxy adds content to the response from the WAF-protected Web Application server, and an anti-CSRF token is inserted in subsequent client POST requests.
- `no`: Proxy will not add content to the response from the WAF-protected web application server and no anti-CSRF token will be inserted in subsequent requests.
- *n*: Time, in seconds, for which the token is valid. The time value takes the place of `(yes)` in policy. If no time value is specified, the anti-CSRF token is valid for 900 seconds.

Layer and Transaction Notes

- Layers: `<Proxy>`
- Transactions: HTTP proxy

See Also

- Properties: "http.csrf.authentication_link()" on page 428, "http.csrf.detection()" on page 430, "http.csrf.token.name()" on the facing page
- *SGOS Web Application Firewall (WAF) Solutions Guide*

http.csrf.token.name()

Use this property in conjunction with other CSRF-related policy in WAF deployments. The anti-CSRF token that is inserted using "http.csrf.token.insert()" on the previous page has a default name of CSRF-Token; thus, by specifying a custom token for legitimate requests, you can make it difficult for malicious users to determine the name of the token and identify the WAF solution in your deployment.

When a custom anti-CSRF token is set in CPL, you can view the source of a browser form to verify that it shows `var antiCsrfTokenName = "<custom_token_name>"`.

A suspected CSRF attack is written to access logs when the expected token name—whether that is the default CSRF-Token or the custom name—is not present. Make sure that the access log format includes the `x-bluecoat-waf-monitor-details` and `x-bluecoat-waf-attack-family` fields.

Syntax

`http.csrf.token.name(string)`

where:

- *string*: Custom token name.

Layer and Transaction Notes

- Layers: <Proxy>
- Transactions: HTTP POST transactions

Example

Insert an anti-CSRF token with the specified properties.

```
; for the given form, insert an anti-CSRF token valid for 10800 seconds
; validate CSRF tokens and log "token missing" if no tokens are detected
<Proxy>
  url.domain=bankform.com http.csrf.token.insert(10800) http.csrf.detection(monitor)

;include the client IP from initial auth with the token details
<Proxy>
  http.csrf.authentication_link(client_ip)

; use the specified custom name for bankform.com
<Proxy>
  http.csrf.token.name(ccsrf-46536-bf-va78)
```

See Also

- Properties: "http.csrf.authentication_link()" on page 428, "http.csrf.detection()" on page 430, "http.csrf.token.insert()" on page 432
- *Web Application Firewall (WAF) Solutions Guide*

http.dns_handoff()

Enables or disables DNS-over-HTTPS (DoH) handoff of DNS requests to DoH servers.

Syntax

```
http.dns_handoff(yes|no)
```

where:

- yes: Request is handed off to DoH forward proxy.
- no: Proxy intercepts and decrypts the request.

Layer and Transaction Notes

- Layers: <Proxy>
- Transactions: HTTP proxy

Example

When the a DNS-proxied request has a Threat Risk Level of 7 or higher, the request is refused. For other DNS traffic, hand off to DoH servers.

```
<DNS-proxy>
  dns.request.threat_risk.level=7.. dns.respond(refused)

<Proxy>
  http.dns_handoff(yes)
```

http.force_ntlm_for_server_auth()

This is a fine-grained control of the global configuration that can be set or unset through CLI commands `#(config)`**`http force-ntlm`** or `#(config)`**`http no force-ntlm`**. These commands are used to work around the Microsoft limitation that Internet Explorer does not allow origin content server (OCS) NTLM authentication through a appliance when explicitly proxied.

To correct this problem, Symantec added a `Proxy-Support: Session-based-authentication` header that is sent by default when the appliance receives a 401 authentication challenge when the client connection is an explicit proxy connection.

For older browsers or if both the appliance and the OCS do NTLM authentication, the Proxy-Support header might not work.

In this case, you can disable the header and instead use the CLI command `#(config)`**`http force-ntlm`** or the `http.force_ntlm_for_server_auth( )` property, which converts the 401-type server authentication challenge to a 407-type proxy authentication challenge, supported by Internet Explorer. The appliance also converts the resulting Proxy-Authentication headers in client requests to standard standard server authorization headers, which allows an origin server NTLM authentication challenge to pass through when Internet Explorer is explicitly proxied through the appliance.

Syntax

```
http.force_ntlm_for_server_auth(yes|no)
```

where:

- `yes`: Allows Internet Explorer clients explicitly proxied through an appliance to participate in NTLM authentication.
- `no`: Proxy-Support: Session-based-authentication header is used to respond to 401 authentication-type challenges.

This property overrides the default specified in configuration.

Layer and Transaction Notes

- Layers: `<Proxy>`
- Transactions: HTTP proxy

Example

All clients from the HR_subnet have force-ntlm turned off. Requests for hosts in the example.com domain have force-ntlm turned on. Requests for all other hosts have force-ntlm turned off.

```
define subnet HR_subnet
  10.10.0.0/16
end

<Proxy>
  client.address=HR_subnet http.force_ntlm_for_server_auth(no)
  url.domain=example.com http.force_ntlm_for_server_auth(yes)
  http.force_ntlm_for_server_auth(no)
```

http.refresh.recv.timeout()

Sets the socket timeout for receiving bytes from the upstream host when performing a refresh.

Syntax

`http.refresh.recv.timeout(auto|`*`recv-timeout`*`)`

where:

- *`recv-timeout`*: Amount of time (in seconds) that the host waits to receive data before timing out. Default is 90.

Layer and Transaction Notes

- Layers: <Cache>, <Forward>
- Transactions: HTTP refresh transactions

Example

Requests from HR_subnet get a receive timeout of 200 seconds. Any request heading to a host that ends in example.com gets a receive timeout of 20 seconds. All refresh traffic except that for example.com hosts gets a receive timeout of 300 seconds.

```
define subnet HR_subnet
  10.10.0.0/16
end

<Proxy>
  client.address=HR_subnet http.client.recv.timeout(200)

<Forward>
  server_url.domain=example.com http.server.recv.timeout(20) http.refresh.recv.timeout(auto)
  http.refresh.recv.timeout(300)
```

http.request.apparent_data_type.allow()

Used to permit HTTP POST transactions based on their Apparent Data Type (ADT). This action supports ADT detection for multipart data, form data and single files. If you select allow, only those data types defined in the object are permitted; all other types are denied.

> **Note:** Recommended for Reverse Proxy deployments, where files will be uploaded through the appliance to a back-end web server.

Syntax

```
http.request.apparent_data_type.allow(data_type,...)
http.request.apparent_data_type.allow[data_type,...](yes|no)
http.request.apparent_data_type.allow.data_type(yes|no)
```

where:

- *data_type*: One of the data types in the following table; specify using either the label or the file extension:

*Added in 7.2.4.1.

Label	Description	Common Extensions
7ZIP*	7-Zip archive	.7z
ACE*	ACE archive	.ace
ARJ*	ARJ archive	.arj
BMP	BMP image	.bmp
BZ2	BZip2 archive	.bz2, .tbz2
CAB	MS Cab archive	.cab
COMPRESS*	compress compressed file	.Z (different from .z)
CPIO*	cpio archive	.cpio
DAA*	Direct Access Archive	.daa
EGG*	.EGG archive	.egg
EML*	raw email	.eml, .mht, .mhtml
EXE	MS application	.exe
FLASH	Adobe Flash	.swf, flv
GIF	GIF image	.gif
GZIP	GZIP compressed file	.gz
HTML	HTML file	.html

Label	Description	Common Extensions
ICC	ICC profile	.icc
JPG	JPEG image	.jpg
LHA*	LHA archive	.lha, .lzh
LZIP*	Lzip compressed file	.lz
MACH-O*	macOS application or library	
MSDOC	Microsoft document	.doc, .docx, .xls, .xlsx, .ppt, .pptx, .msi, .vba
MRAR	multi-part RAR	.partX.rar
MZIP	multi-part ZIP	.zip.xxx
PDF	Portable Document Format file	.pdf
PNG	PNG image	.png
RAR	RAR archive	.rar
RTF	Rich text document	.rtf
TAR	TAR archive	.tar
TIF	TIFF image	.tif
TNEF*	file encoded in Microsoft Transport-Neutral Encapsulation Format	.dat, .tnef
TTF	True-Type font	.ttf
TXT	plain text	.txt
UUE*	file encoded with uuencode or xxencode	.uu, .uue, .xx, .xxe
XAR*	Extensible Archive Format	.mpkg, .pkg, .xar
XML	XML file	.xml
XZ*		.xz
ZIP	ZIP archive	.zip

Layer and Transaction Notes

- Layers: `<Proxy>`
- Transactions: HTTP POST requests.
- Does not involve ICAP services.

Example

A proxy administrator for a community website would like to allow Internet-based users to upload images to the website, but only in JPG format. Some uploads occur in multiple parts. The following policy is able to detect the apparent data type for those uploads as well.

```
<Proxy>
  http.request.apparent_data_type.allow(JPG)
```

See Also

- Conditions: "http.request.apparent_data_type=" on page 158, "http.response.apparent_data_type=" on page 174, "request.icap.apparent_data_type=" on page 235, "response.icap.apparent_data_type=" on page 249
- Properties: "http.request.apparent_data_type.deny()" on the facing page

http.request.apparent_data_type.deny()

Used to restrict HTTP POST transactions based on their Apparent Data Type. This action supports Apparent Data Type detection for multipart data, form data and single files. If you select deny, only those data types defined in the object are denied; all other types are allowed.

> **Note:** Recommended for Reverse Proxy deployments, where files will be uploaded through the appliance to a back-end web server.

Syntax

```
http.request.apparent_data_type.deny(data_type,...)
http.request.apparent_data_type.deny[data_type,...](yes|no)
http.request.apparent_data_type.deny.data_type(yes|no)
```

where:

- *data_type*: One of the data types in the following table; specify using either the label or the file extension:

*Added in 7.2.4.1.

Label	Description	Common Extensions
7ZIP*	7-Zip archive	.7z
ACE*	ACE archive	.ace
ARJ*	ARJ archive	.arj
BMP	BMP image	.bmp
BZ2	BZip2 archive	.bz2, .tbz2
CAB	MS Cab archive	.cab
COMPRESS*	compress compressed file	.Z (different from .z)
CPIO*	cpio archive	.cpio
DAA*	Direct Access Archive	.daa
EGG*	.EGG archive	.egg
EML*	raw email	.eml, .mht, .mhtml
EXE	MS application	.exe
FLASH	Adobe Flash	.swf, flv
GIF	GIF image	.gif
GZIP	GZIP compressed file	.gz
HTML	HTML file	.html

Label	Description	Common Extensions
ICC	ICC profile	.icc
JPG	JPEG image	.jpg
LHA*	LHA archive	.lha, .lzh
LZIP*	Lzip compressed file	.lz
MACH-O*	macOS application or library	
MSDOC	Microsoft document	.doc, .docx, .xls, .xlsx, .ppt, .pptx, .msi, .vba
MRAR	multi-part RAR	.partX.rar
MZIP	multi-part ZIP	.zip.xxx
PDF	Portable Document Format file	.pdf
PNG	PNG image	.png
RAR	RAR archive	.rar
RTF	Rich text document	.rtf
TAR	TAR archive	.tar
TIF	TIFF image	.tif
TNEF*	file encoded in Microsoft Transport-Neutral Encapsulation Format	.dat, .tnef
TTF	True-Type font	.ttf
TXT	plain text	.txt
UUE*	file encoded with uuencode or xxencode	.uu, .uue, .xx, .xxe
XAR*	Extensible Archive Format	.mpkg, .pkg, .xar
XML	XML file	.xml
XZ*		.xz
ZIP	ZIP archive	.zip

Layer and Transaction Notes

- Layers: `<Proxy>`
- Transactions: HTTP POST requests.
- Does not involve ICAP services.

Example

A reverse proxy administrator allows files to be uploaded to personal storage space on a web server. Because of security concerns with executable files, she wants to reject some types of data.

```
<Proxy>
  http.request.apparent_data_type.deny(EXE,Flash,CAB)
```

See Also

- Conditions: "http.request.apparent_data_type=" on page 158, "http.response.apparent_data_type=" on page 174, "request.icap.apparent_data_type=" on page 235, "response.icap.apparent_data_type=" on page 249
- Properties: "http.request.apparent_data_type.allow()" on page 438

http.request.body.data_type()

Allows an administrator to configure the appliance to either evaluate and trust the content-type header, or treat the request as if a specific content-type was specified, regardless of the actual value in the request.

Syntax

```
http.request.body.data_type(json|webform|xml|multipartform|none|auto)
```

where:

- `auto`: (Default behavior) Evaluate content-type, and resolve to one of the following:
  - multipartform
  - url encoded
  - json
  - xml
  - other (none)
- `json`: Act as if body content-type is JSON.
- `multipartform`: Act as if multipart-form content-type is specified.
- `none`: Act as if no content-type is present and the body data is not considered as json, webform, xml, or multipartform. As a result, the following gestures do not detect invalid request data:
  - "http.request.detection.xml.invalid()" on page 458
  - http.request.detection.other.invalid_json()
  - http.request.detection.other.invalid_form_data()
- `webform`: Act as if body content-type is URL-encoded.
- `xml`: Act as if body content-type is XML.

Layer and Transaction Notes

- Layers: `<Proxy>`
- Transactions: HTTP proxy

Example

Act as if body content-type is XML.

```
<Proxy>
  http.request.body.data_type(xml)
```

http.request.body.inspection_size()

> **Note:** This property requires another WAF policy gesture present in order for it to change the maximum number of bytes for HTTP requests. The default maximum is 8192 bytes.

Specify the maximum number of bytes of an HTTP request body that Web Application Firewall (WAF) content nature detection engines or policy can scan per transaction. The maximum is 65536 bytes. If the request body is chunk-encoded or compressed and the request data is a type for which WAF engines are enabled, the appliance unchunks and decompresses the data. For example, if the request body is chunked and includes SQL statements, and the SQL injection engine is enabled, the appliance unchunks the data. The `http.request.body.inspection_size()` property considers the size of the unchunked data.

While WAF engines scan the HTTP request body, contact to the origin content server (OCS) does not occur until the engines scan the specified amount of request body bytes or reach the end of the HTTP request body. Because the OCS connection could be delayed, this could lead to increased latency of the transaction.

If the "http.request.data=" on page 167 condition is already specified in policy, the WAF engines scan up to the number of bytes specified in the condition and you do not have to use the `http.request.body.inspection_size()` property. If policy includes both "http.request.data=" on page 167 and `http.request.body.inspection_size()`, WAF engines use the greatest value specified for scanning.

For information on WAF engines, refer to the *Web Application Firewall Solutions Guide*.

Syntax

`http.request.body.inspection_size(N)`

where:

- *N*: Maximum number of bytes in the HTTP request body to scan. The maximum is 65536 bytes.

Layer and Transaction Notes

- Layers: `<Cache>`, `<Proxy>`
- Transactions: HTTP requests that have a request body

Example

WAF engines will scan up to 50000 bytes of an HTTP request body and block any requests larger than this value.

```
<Proxy>
  http.request.body.inspection_size(50000) http.request.detection.other.threshold_exceeded(block)
```

See Also

- Conditions: "http.request.data=" on page 167, "http.request.body.max_size_exceeded=" on page 165

http.request.body.max_size()

Allows you to deny HTTP requests that include a body content size that exceeds a specified number of bytes.

Syntax

`http.request.body.max_size(N)`

where:

- *N*: Maximum number of bytes in the HTTP request body.

Layer and Transaction Notes

- Layers: `<Cache>`, `<Proxy>`
- Transactions: HTTP requests that include body content.
- Deny occurs after the HTTP request body received by the appliance.
- When the connection is terminated using this property, the `silent_denied` exception is called. The user receives no feedback from the appliance.

Example

Terminate any HTTP request from the client at IP address 1.2.3.4 if the body exceeds 10 MB.

```
<Proxy>
  client.address=1.2.3.4 http.request.body.max_size(10485760)
```

http.request.detection.bypass_cache_hit()

To improve performance, you can optimize Web Application Firewall (WAF) scanning for requests whose response is served from the object cache on the appliance.

If policy includes this property, the x-bluecoat-waf-scan-info field in the bcreporterwarp_v1 access log format indicates if WAF processing is intentionally skipped due to cache hit optimization being bypassed:

- If WAF engines scan a transaction, the field reports WAF_SCANNED.
- If WAF evaluation does not occur due to the presence of the `http.request.detection.bypass_cache_hit(yes)` property or the absence of WAF policy, the field reports WAF_SCAN_BYPASSED.
- If no WAF policy is present, the field reports WAF_DISABLED.

The WAF is designed to protect backend web application servers in a reverse proxy deployment. If the object cache can serve the response for a given request, the request is not required to make the round trip between the appliance and origin content server (OCS).

In reverse proxy deployments with a high cache-hit ratio, enabling this property can have a significant positive impact on performance.

For information on WAF engines, refer to the *Web Application Firewall Solutions Guide*.

Important Note

The `http.request.detection.bypass_cache_hit()` property allows you to optimize for either performance or security; thus, it is important to understand the trade-offs and implications of using this feature. A misconfigured web application server, which allows caching via the cache-control response header on dynamic responses, causes the appliance to store responses in the object cache. If a protected web application returns cacheable dynamic responses while this property is enabled, the appliance could serve malicious responses from cache. To address this potential issue, do one of the following:

- Correct the cache directives that the web application server served.
- Use the "rewrite()" on page 615 action in a "define action" on page 628 block to rewrite the cache-control header, as follows:
  `rewrite(response.header.cache-control, "regex_pattern", "string")`
- Use the bypass_cache(no) property to bypass caching for specific requests or cache(no) to avoid storing specific responses in the cache.
- Clear the object cache with the #clear-cache object-cache CLI command. For maximum security, run this command before using `http.request.detection.bypass_cache_hit()` in an existing deployment, as well as whenever WAF policy or content from the Application Protection subscription is updated.

> **Note:** To determine when the Application Protection subscription was last updated (or, if Notify Only is enabled, a newer database is available for download), issue the `#(config application-protection)view` command and check the Last successful download details.

Refer to the *Command Line Interface Reference* for details on the `#clear-cache object-cache` and `#(config application-protection)view` commands.

Syntax

```
http.request.detection.bypass_cache_hit(yes|no)
```

where:

- *yes*: WAF engines do not scan requests that result in a cache hit.
- *no*: WAF engines scan all requests; this is the default behavior.

Layer and Transaction Notes

- Layers: <Admin>, <Cache>, <DNS-Proxy>, <Exception>, <Forward>, <Proxy> , <SSL>, <SSL-Intercept>
- Transactions: HTTP proxy

Example

Avoid WAF scanning for all requests to www.symantec.com that result in an object cache hit.

```
<Proxy>
  url.domain="www.symantec.com" http.request.detection.bypass_cache_hit(yes)
```

See Also

- Properties: "bypass_cache()" on page 373, "cache()" on page 374
- Actions: "rewrite()" on page 615

http.request.detection.constraint_set()

Block or monitor transactions that violate constraints defined in the specified "define constraint_set" on page 639. The constraints are applied to field attributes after they are normalized.

When a transaction violates constraints, it is blocked or monitored and the following messages are logged:

- the `x-bluecoat-waf-attack-family` field shows `Constraint Violation`
- the `x-bluecoat-waf-block-details` or `x-bluecoat-waf-monitor-details` field shows details with the following syntax:

  `"{""detection"":""constraint"",""part"":""{name|query_arg_name|query_arg|arg_name|arg|cookie_name|cookie|post_arg_name|post_arg|header_name|header|path}"",""line"":""`*`constraint_set_defn_cpl_line`*`"",""data"":""`*`matched_data`*`""}"`

These fields are included in the bcreporterwarp_v1 log format.

Syntax

`http.request.detection.constraint_set.`*`constraint_id`*`(block|monitor)`

where:

- *`constraint_id`*: Name of constraint set specified in a "define constraint_set" on page 639 definition block.

Layer and Transaction Notes

- Layers: `<Proxy>`
- Transactions: HTTP proxy

Example

See "define constraint_set" on page 639 for an example of usage.

See Also

- Definitions: "define constraint_set" on page 639
- *ProxySG Access Log Fields and CPL Substitutions Reference*: https://techdocs.broadcom.com/content/broadcom/techdocs/us/en/symantec-security-software/web-and-network-security/proxysg/7-2/proxysg-log-fields-and-substitutions.html

http.request.detection.exception()

Specify a built-in or user-defined exception message to return to the user when a Web Application Firewall (WAF) engine or property blocks a request. If this property is not specified, the default message for a blocked request is invalid_request.

A WAF exception is served whenever a WAF engine or property issues a Block action. See"define application_protection_set" on page 632 for details.

> **Note:** The "exception()" on page 402 property does not override this property for WAF block exceptions and this property does not override "exception()" on page 402 for standard exceptions.

For information on WAF engines, refer to the *Web Application Firewall Solutions Guide*.

Syntax

`http.request.detection.exception(`*`exception_id`*`,`*`details`*`,`*`format_string`*`)`

where:

- *`exception_id`*: One of the following:
  - Built-in exception such as `silent_denied`
  - User-defined exception in the form `user_defined.`*`exception_id`*
- *`details`*: Text string, enclosed within quotation marks, for the exception message. The message is substituted for `$(exception.details)` when the exception occurs.
- *`format_string`*: Text defined with define string and substituted for `$(exception.format)`. The named string overrides the format field of the exception and can include substitutions.

Layer and Transaction Notes

- Layers: `<Proxy>`
- Transactions: client-initiated HTTP transactions

Example

Block requests for the XSS engine. Return a gateway error page when XSS engine is detected, and return a policy_denied page when HTTP error code 400 is detected.

```
define application_protection_set EngConf
  engine=xss
end
```

```
define string ErrorMessage
; define an HTML page for the error message
><html>
><head>
><title>Gateway Error</title>><meta http-equiv=refresh content="10;$(url)">
></head>
><body>
>Cannot complete your request.
></body>
></html>
end

<Proxy>
  http.request.detection.EngConf(block)

<Proxy>
  http.request.detection.exception(gateway_error, "Gateway Error", ErrorMessage)

<Proxy>
  http.response.code=400 exception(policy_denied)
```

See Also

- Definitions: "define application_protection_set" on page 632, "define string" on page 654

http.request.detection.other()

Enables and defines settings for request validation in HTTP requests. The set of validation occurs after the URI path and all names and values are normalized in the query string, cookie, and body in JSON, URL-encoded and Multipart-Form-encoded formats.

The engines parse SOAP and Multipart content.

Syntax

`http.request.detection.other.[attribute](block|monitor|ignore)`

where:

- *attribute*: One of the following:
  - `null_byte` - Detects content that contains null bytes.
  - `invalid_form_data` - Detects invalid multipart/form-data evasion techniques.
  - `invalid_json` - Detects invalid JSON data in the request body.
  - `parameter_pollution` - Detects multiple instances of parameters with the same name.
  - `parameter_pollution_separator("<char>")` - When multiple instances of parameters with the same name exist, the values are concatenated and separated by the specified character. For example, when you specify a comma (",") as the separator, field=1&field=2 are concatenated to field=1,2 before WAF engines analyze the values.

    > **Note:** To preserve WAF engine effectiveness in detecting threats:
    > - Use this attribute only in environments where the backend system supports parameter concatenation.
    > - Specify a useful separator character; typically, backend systems (such as ASP) support the comma as a string separator.

  - `multiple_encoding` - Detects request data encoded more times than what is specified in "define application_protection_set" on page 632.
  - `invalid_encoding` - Detects invalid UTF-8 encoding in full and half Unicode. Applies only to normalizations specified in the "http.request.normalization.default()" on page 470 property.
  - `multiple_header` - Detects headers declared more than once, with the same name, in a request. Does not consider any legitimate, known headers that are allowed to appear multiple times.
  - `threshold_exceeded` - Triggers when the "http.request.body.inspection_size()" on page 446 limit is reached.

- `block`: Denies the request and logs the action.
- `monitor`: Allows the request and logs the action.
- `ignore`: Allows the request and does not log the action.

> **Note:** If the body size of the JSON or multipart/form-data HTTP request is greater than the maximum number of bytes for HTTP request bodies, the ProxySG appliance might report the request as invalid. The ProxySG appliance might report valid requests as invalid because the appliance removes the data elements (closing tags, matching patterns, and/or necessary syntax) that exceed the maximum number of bytes for HTTP request bodies. The default maximum is 8192 bytes. To increase the maximum number of bytes for JSON or multipart/form-data HTTP request bodies, see the documentation for the "http.request.data=" on page 167 or "http.request.body.inspection_size()" on page 446 properties. The latter property requires another WAF policy gesture present in order for it to change the maximum number of bytes for HTTP requests.

Layer and Transaction Notes

- Layers: `<Proxy>`
- Transactions: HTTP proxy

Example

Define the settings for request validation in HTTP requests.

```
;; Conditions
;;===========
define application_protection_set AllEnginesAllParts
  engine=xss
  engine=injection.sql
  engine=injection.code
  engine=injection.command
  engine=injection.html
  engine=reference.directory_traversal
  engine=blacklist
  engine=analytics_filter
end


;; Normalization
;;=============
<Proxy>
  http.request.normalization.default(auto)
```

```
;; Evasion Detection
;;==================
<Proxy>
  http.request.body.data_type(auto) http.request.detection.other.null_byte(monitor) \
    http.request.detection.other.invalid_encoding(monitor) http.request.detection.other.invalid_form_
data(monitor) \
      http.request.detection.other.invalid_json(monitor) http.request.detection.other.parameter_
pollution(monitor) \
        http.request.detection.other.multiple_encoding(monitor) http.request.detection.other.multiple_
header(monitor) \
          http.request.body.inspection_size(65536) http.request.detection.other.threshold_exceeded
(monitor)

;; WAF Engines
;;============
<Proxy>
  http.request.detection.AllEnginesAllParts(monitor)
```

See Also

- Definitions: "define application_protection_set" on page 632

http.request.detection.xml.cdata()

Controls the expansion and evaluation of <![CDATA[]> structures in the request XML body.

Syntax

```
http.request.detection.xml.cdata(yes|no)
```

where:

- yes: Consider <![CDATA[]> as markup, and expand the section prior to scanning the request XML body; evaluation can trigger Blacklist rule detection.
- no: (Default behavior) Do not consider <![CDATA[]> as markup, and do not expand the section prior to scanning the request XML body; evaluation cannot trigger Blacklist rule detection.

Layer and Transaction Notes

- Layers: <Proxy>

Example

Expand and scan CDATA in the XML body.

```
<Proxy>
  http.request.detection.xml.cdata(yes)
```

http.request.detection.xml.invalid()

When the XML body cannot be processed because it is not well-formed, take the specified action.

> **Note:** If the body size of the XML HTTP request is greater than the maximum number of bytes for HTTP request bodies, the ProxySG appliance might report the request as invalid. The ProxySG appliance might report valid requests as invalid because the appliance removes the data elements (closing tags, matching patterns, and/or necessary syntax) that exceed the maximum number of bytes for HTTP request bodies. The default maximum is 8192 bytes. To increase the maximum number of bytes for XML HTTP request bodies, see the documentation for "http.request.data=" on page 167 or "http.request.body.inspection_size()" on page 446. The latter property requires another WAF policy gesture present in order for it to change the maximum number of bytes for HTTP requests.

Syntax

```
http.request.detection.xml.invalid(block|monitor|ignore)
```

where:

- `block`: Denies the request and logs the action.
- `monitor`: Allows the request and logs the action.
- `ignore`: Allows the request and does not log the action; this is the default.

Layer and Transaction Notes

- Layers: `<Proxy>`

Example

When the XML body is not well-formed, allow and log the request

```
<Proxy>
  http.request.detection.xml.invalid(monitor)
```

http.request.detection.xml.node_depth()

Controls the maximum allowed number of nested nodes in the request XML body.

Syntax

`http.request.detection.xml.node_depth(`*`integer`*`)`

where:

- *`integer`*: Value from 1 to 99. The default value is 99.

Layer and Transaction Notes

- Layers: `<Proxy>`

Example

The maximum number of nested nodes allowed is 10.

```
<Proxy>
  http.request.detection.xml.node_depth(10)
```

http.request.detection.xml.schema.*schema_name*()

Validates the XML request body against the schema defined in the `define xml_`*schema-type*`_schema` block. If validation fails, take the specified action.

Syntax

`http.request.detection.xml.schema.`*schema_name*`(block|monitor|ignore)`

where:

- *schema_name*: Name of the define xml_schema-type_schema definition block.
- `block`: Denies the request and logs the action.
- `monitor`: Allows the request and logs the action.
- `ignore`: Allows the request and does not log the action.

Layer and Transaction Notes

- Layers: `<Proxy>`

Example

Allow the request and log the action if validation fails.

```
define xml_dtd_schema validate_body
  <!ELEMENT note (to,from,heading,body)>
  <!ELEMENT to (#PCDATA)>
  <!ELEMENT from (#PCDATA)>
  <!ELEMENT heading (#PCDATA)>
  <!ELEMENT body (#PCDATA)>
  <!ELEMENT body (#PCDATA)>
  <!ENTITY x "ecks">
end

<Proxy>
  http.request.detection.xml.schema.validate_body(monitor)
```

See Also

- Definitions: "define xml_schema-type_schema" on page 666

http.request.detection.xml.xinclude()

If the request XML body contains XInclude nodes, take the specified action.

Syntax

```
http.request.detection.xml.xinclude(block|monitor|ignore)
```

where:

- `block`: Denies the request and logs the action.
- `monitor`: Allows the request and logs the action.
- `ignore`: Allows the request and does not log the action.

Layer and Transaction Notes

- Layers: `<Proxy>`

Example

Block the request if the XML request contains XInclude nodes

```
<Proxy>
  http.request.detection.xml.xinclude(block)
```

http.request.detection.xml.xpath_validation()

Performs an XPath search on the request XML body. If the node is found, the appliance performs a regex search on the content. If the content matches the regular expression, block or monitor the request according to the action value.

Syntax

`http.request.detection.xml.xpath_validation(`*`xpath_expression`*`,`*`regular_expression`*`, (block|monitor|ignore))`

where:

- *`xpath_expression`*: Specify an XPath expression to search for a specific node.
- *`regular_expression`*: Specify a regular expression to search for a specific string.
- `block`: Denies the request and logs the action.
- `monitor`: Allows the request and logs the action.
- `ignore`: Allows the request and does not log the action.

Layer and Transaction Notes

- Layers: `<Proxy>`

Example

When the node and expression match, block the request and log the action.

```
<Proxy>
  http.request.detection.xml.xpath_validation("//forbidden", ".*", block)
```

http.request.detection.xml.xxe()

If the request XML body contains an XML External Entity (XXE) pattern (<!ENTITY...SYSTEM>), take the specified action.

Syntax

```
http.request.detection.xml.xxe(block|monitor|ignore)
```

where:

- `block`: Denies the request and logs the action.
- `monitor`: Allows the request and logs the action.
- `ignore`: Allows the request and does not log the action.

Layer and Transaction Notes

- Layers: `<Proxy>`

Example

Block the request if WAF engines detect an XXE injection attack.

```
<Proxy>
  http.request.detection.xml.xxe(block)
```

http.request.log_details()

Outputs the contents of the header, body, or both from an HTTP request to access log fields `x-bluecoat-request-details-header` and `x-bluecoat-request-details-body`.

In a WAF deployment, administrators can use this information to validate detections, in an effort to rule out false-positives.

In a standard forward proxy deployment, administrators can use this information to log the data users are transmitting to the Internet.

Syntax

`http.request.log_details[`*`header`*`,`*`body`*`](yes|no)`

where:

- *header*: Outputs the HTTP headers from user requests to `x-bluecoat-request-details-header`.
- *body*: Outputs the HTTP body from user requests to `x-bluecoat-request-details-body`. By default, only the first 8 kB of body contents are logged. You can increase this with either of the following policy gestures:
  - "http.request.data=" on page 167
  - "http.request.body.inspection_size()" on page 446
- `yes`: Enables the access log fields, `x-bluecoat-request-details-header` and `x-bluecoat-request-details-body` in the bcreporterwarp_v1 access log format, and outputs the contents of request body, header, or both.
- `no`: (Default behavior) The bcreporterwarp_v1 access log format does not include the additional access log fields.

Layer and Transaction Notes

- Layers: `<Proxy>`

Example

Always log header contents, and conditionally log the full body when a WAF detection occurs:

```
<Proxy>
  http.request.normalization.default(auto)

<Proxy>
  http.request.detection.other.null_byte(monitor)

<Proxy>
  http.request.log_details[header](yes)

<Proxy>
```

```
  http.request.detection.result.validation=(monitor||block) http.request.log_details[body](yes)
  http.request.detection.result.application_protection_set=(monitor||block) http.request.log_details
[body](yes)
```

See Also

- Conditions: "http.request.data=" on page 167, "http.request.detection.result.application_protection_set=" on page 169, "http.request.detection.result.validation=" on page 170
- Properties: "http.request.body.inspection_size()" on page 446

http.request.log.mask_by_name[*regex_pattern*]()

Where header or body request logging is used, this property can be used to mask specific query, header, or body strings from the access logs; replacing key values with a string of five asterisks (*****).

Administrators can use this property to ensure that sensitive or personally identifiable data is not leaked through the access logs when "http.request.log_details()" on page 464 is used.

Syntax

```
http.request.log.mask_by_name(yes|no)
http.request.log.mask_by_name.["regex_pattern",....](yes|no)
```

where:

- yes: Enables masking of the value associated with the regex key pattern in access log fields, x-bluecoat-request-details-header and x-bluecoat-request-details-body in the bcreporterwarp_v1 access log format.
- no: (Default behavior) Data in the bcreporterwarp_v1 access log format does not include any masked information. This impacts the policy in which (no) is used, as well as any other http.request.log.mask_by_name policy.
- *regex_pattern*: One or more regular expression strings, used to match the names of specific key strings in HTTP content where the values for those strings contain sensitive or personally-identifiable information.

> **Note:** Square brackets [ ] are optional if specifying one regex string, but are required when specifying multiple strings in a comma-separated list. The period . is optional when specifying multiple strings within square brackets.

Layer and Transaction Notes

- Layers: <Proxy>

Example 1

Write all body contents to the access log, but mask values for email addresses and passwords. Before implementing this policy, the x-bluecoat-request-details-body log shows the following:

```
GET /?query=value HTTP/1.1 Host: 10.167.0.116 X-WAF-Header-1: '';!--\"\"<XSS>=*{()}
Cookie: name=user; pa%73%73word=pass1; email=user@email.com
```

After implementing this policy, the x-bluecoat-request-details-body log appears as follows:

```
"GET /?query=value HTTP/1.1\r\nHost: 10.167.0.116\r\nX-WAF-Header-1: ‘';!--\\""\\""<XSS>=*
{()}\r\nCookie: name=user; pa%73%73word=*****; email=*****\r\n"
```

```
<Proxy>
  http.request.log_details[body](yes)

<Proxy>
  http.request.log.mask_by_name["password", "email"](yes)
```

Example 2

Write header and body contents to the access log, but mask values users enter in a web form about their physical attributes.

```
<Proxy>
  http.request.log_details[header,body](yes)

<Proxy>
  http.request.log.mask_by_name["eye_color","hair_color","height"](yes)
```

See Also

- Conditions: "http.request.data=" on page 167,
- Properties: "http.request.log_details()" on page 464, "http.request.log.mask_by_value[regex_pattern]()" on the next page

http.request.log.mask_by_value[*regex_pattern*]()

Where header or body request logging is used, this CPL property can be used to mask specific query, header, cookie, or body strings from the access logs; replacing sensitive information with a string of five asterisks (*****).

Administrators can use this property to ensure that sensitive or personally identifiable data is not leaked through the access logs when http.request.log_details is used.

Unlike http.request.log.mask_by_name, this property looks for strings of text rather than key values, and those strings are masked.

Syntax

```
http.request.log.mask_by_value(yes|no)
http.request.log.mask_by_value.["regex_pattern",..](yes|no)
```

where:

- yes: Enables the obfuscation of the defined pattern in access log fields, x-bluecoat-request-details-header and x-bluecoat-request-details-body in the bcreporterwarp_v1 access log format.
- no: (Default behavior) Data in the bcreporterwarp_v1 access log format does not include any obfuscated information. This impacts the policy in which (no) appears, as well as any other http.request.log.mask_by_name policy.
- *regex_pattern*: Regular expression value, used to match specific strings in HTTP content that contain sensitive or personally-identifiable information.

  Some common regular expression strings to use in policy with this property:

  - U.S. Passport Cards: “C0[0-9]{7}”
  - U.S Passport Number: “[23][0-9]{8}”
  - US Social Security Number: “[0-9]{3}[-]?[0-9]{2}[-]?[0-9]{4}”
  - Canadian Social Insurance Number: “[1-79]\d{2}[-]?\d{3}[-]?\d{3}”

> **Note:** Square brackets [ ] are optional if specifying one regex string, but are required when specifying multiple strings in a comma-separated list. The period . is optional when specifying multiple strings within square brackets.

Layer and Transaction Notes

- Layers: <Proxy>

Example

Always log header contents, and conditionally log the full body when a WAF detection occurs, but do not log Social Security Numbers present in body content. Before implementing this policy, the x-bluecoat-request-details-body log shows the following:

"SSN=111 22 3333"

After implementing this policy, the x-bluecoat-request-details-body log appears as follows:

"SSN=*****"

```
<Proxy>
  http.request.normalization.default(auto)

<Proxy>
  http.request.detection.other.null_byte(monitor)

<Proxy>
  http.request.log_details[header](yes)

<Proxy>
  http.request.detection.result.validation=(monitor||block) http.request.log_details[body](yes)
  http.request.detection.result.application_protection_set=(monitor||block) http.request.log_details
[body](yes)

<Proxy>
  http.request.log.mask_by_value["[0-9]{3}[ -]?[0-9]{2}[ -]?[0-9]{4}"](yes)
```

See Also

- Conditions: "http.request.data=" on page 167,
- Properties: "http.request.log_details()" on page 464, "http.request.log.mask_by_name[regex_pattern]()" on page 466

http.request.normalization.default()

Per transaction, normalize only the value of specified attributes(s) using the specified normalization function(s).

Syntax

```
http.request.normalization.default("function:(attribute,..)")
```

where:

- auto: Provides the recommended normalization settings. See "Recommended Usage" on page 472 for more information.
- function: Accepts one or more of the following:
  - urlDecode - Decode URL-encoded strings.
  - htmlEntityDecode - Decode characters encoded as HTML entities.
  - urlDecodeUni - Same as urlDecode, with support for Microsoft-specific %u encoding.
  - cmdLine - Delete or replace characters used as Windows and Linux command line escape characters.
  - cssDecode - Decode characters encoded using CSS 2.x escape characters.
  - jsDecode - Decode JavaScript escape sequences.
  - compressWhiteSpace - Convert whitespace characters (0x20, \f, \t, \n, \r, \v, 0xa0) to spaces (ASCII 0x20) and compress multiple consecutive space characters into one.
  - lowercase - Convert all characters to lowercase. You cannot use this with the .case_sensitive modifier.
  - normalizePath - Remove multiple slashes, directory self-references, and directory back-references (except when at the beginning of the input).
  - normalizePathWin - Same as normalizePath, but first convert backslash characters to forward slashes.
  - removeWhitespace - Delete all whitespace characters.
  - removeNulls - Remove all null bytes.
  - replaceComments - Replace each occurrence of a C-style comment
  - (/* ... */) with a single space (multiple consecutive occurrences of zre not compressed).
  - utf8toUnicode - Convert all UTF-8 characters sequences to Unicode. This normalization function is not applied on POST body data if the content-type is unknown or not specified.

- base64Decode - Decode partial and complete Base64-encoded strings.
- trimDecode - Delete whitespace prefixes and suffixes.

- attribute: One or more of the supported values/names:

Name	Value
`name` - All argument names found in the URL query string, post body (URL-encoded and multipart-form encoded formats), or cookie.	`value` - All named and unnamed argument values found in URL query string, post body (URL-encoded and multipart-form encoded formats), or cookie.
`query_arg_name` - All argument names found in the URL query string.	`query_arg` - All named and unnamed argument values found in the URL query string.
`arg_name` - All argument names found in both the URL query string and the post body (URL-encoded and multipart-form encoded formats).	`arg` - All named and unnamed argument values found in both the URL query string and the post body (URL-encoded and multipart-form encoded formats).
`cookie_name` - All argument names found in all Cookie and Cookie2 headers.	`cookie` - All named and unnamed argument values found in all Cookie and Cookie2 headers.
`post_arg_name` - All argument names found in the post body (URL-encoded and Multipart-form encoded formats)	`post_arg` - All named and unnamed argument values found in the post body (URL-encoded and Multipart-form encoded formats).
`header_name` - All header names.	`header` - All header values.
`path` - Path of the URL. This attribute does not have `name=value` format.	

Layer and Transaction Notes

- Layers: `<Proxy>`

Example

Scan for the specified patterns in objects and deny requests with scores equal to or greater than 20.

```
; Perform regex scan for case-insensitive pattern "bad" in any name or value in cookie values and
cookie names
<Proxy>
  http.request[cookie_name,cookie].regex="bad"

; Perform regex scan for "bad"in any name or value within any parameter
; a URL encoded query string will match this rule "http://mydomain.com/path?%42%41%44=value"
<Proxy>
  http.request[name,value].regex="bad"

; Reject HTTP requests with scores equal to or greater than 20
<Proxy>
  http.request[arg].count=20.. deny
```

Recommended Usage

Symantec recommends that you specify this property as follows:

```
<Proxy>
  http.request.normalization.default(auto)
```

The (auto) option expands to the following normalization setting:

```
http.request.normalization.default("urlDecode:(path),urlDecode:jsDecode:htmlEntityDecode:trimDecode:
(header_name,header,cookie_
name,cookie),urlDecode:urlDecode:jsDecode:htmlEntityDecode:utf8toUnicode:trimDecode:(arg_name,arg)")
```

See Also

- Condition: "http.request.data=" on page 167
- Properties: "http.request.detection.other()" on page 454
- Definitions: "define application_protection_set" on page 632

http.request.normalization.unicodecodepage()

Specify which code page to use when decoding characters during urlDecodeUni normalization.

Syntax

`http.request.normalization.unicodecodepage(`*codepage*`)`

where:

- *codepage*: Supported code page:
  - 1250 (ANSI - Central Europe)
  - 1251 (ANSI - Cyrillic)
  - 1252 (ANSI - Latin I)
  - 1253 (ANSI - Greek)
  - 1254 (ANSI - Turkish)
  - 1255 (ANSI - Hebrew)
  - 1256 (ANSI - Arabic)
  - 1257 (ANSI - Baltic)
  - 1258 (ANSI/OEM - Viet Nam)
  - 20127 (US-ASCII)
  - 20261 (T.61)
  - 20866 (Russian - KOI8)
  - 28591 (ISO 8859-1 Latin I)
  - 28592 (ISO 8859-2 Central Europe)
  - 28605 (ISO 8859-15 Latin 9)
  - 37 (IBM EBCDIC - U.S./Canada)
  - 437 (OEM - United States)
  - 500 (IBM EBCDIC - International)
  - 850 (OEM - Multilingual Latin I)
  - 860 (OEM - Portuguese)

- 861 (OEM - Icelandic)
- 863 (OEM - Canadian French)
- 865 (OEM - Nordic)
- 874 (ANSI/OEM - Thai)
- 932 (ANSI/OEM - Japanese Shift-JIS)
- 936 (ANSI/OEM - Simplified Chinese GBK)
- 949 (ANSI/OEM - Korean)
- 950 (ANSI/OEM - Traditional Chinese Big5)

Layer and Transaction Notes

- Layers: `<Proxy>`
- Transactions: HTTP proxy

Example

Use the Simplified Chinese code page when decoding during normalization.

```
<Proxy>
  http.request.normalization.unicodecodepage(936)
```

http.request.version()

Sets the version of the HTTP protocol to be used in the request to the origin content server or upstream proxy.

Syntax

```
http.request.version(1.0|1.1|preserve)
```

where:

- *preserve*: HTTP server request version will be set to the same value found on the client side inbound HTTP version, if it exists.
  The default is taken from the CLI configuration setting http version, which can be set to either 1.0 or 1.1. Changing this value in the CLI changes the default for both this property and "http.response.version()" on page 479.

Layer and Transaction Notes

- Layers: `<Cache>`, `<Proxy>`
- Transactions: HTTP proxy; does not apply to HTTP/2 connections

Example

For the given domain, respond with HTTP 1.0 when receiving a HTTP/1.0 request.

```
<Proxy>
  url.domain=//example.com/ http.response.version(1.0) http.request.version(1.0)
```

See Also

- Conditions: "http.response.version=" on page 179
- Properties: "http.response.version()" on page 479

http.response.parse_meta_tag.Cache-Control()

Controls whether the 'Cache-Control' META Tag is parsed in an HTML response body.

Syntax

```
http.response.parse_meta_tag.Cache-Control(yes|no)
```

Layer and Transaction Notes

- Layers: <Cache>
- Transactions: HTTP proxy transactions, HTTP refresh transactions, and HTTP pipeline transactions.

Example

Parse the Cache-Control tag in response bodies.

```
<Proxy>
  http.response.parse_meta_tag.Cache-Control(yes)
```

http.response.parse_meta_tag.Expires()

Controls whether the 'Expires' META Tag is parsed in an HTML response body.

Syntax

```
http.response.parse_meta_tag.Expires(yes|no)
```

Layer and Transaction Notes

- Layers: <Cache>
- Transactions: HTTP proxy transactions, HTTP refresh transactions, and HTTP pipeline transactions

Example

Parse the Expires tag in response bodies.

```
<Proxy>
  http.response.parse_meta_tag.Expires(yes)
```

http.response.parse_meta_tag.Pragma-no-cache()

Controls whether the 'Pragma: no-cache' META Tag is parsed in an HTML response body.

Syntax

```
http.response.parse_meta_tag.pragma-no-cache(yes|no)
```

Layer and Transaction Notes

- Layers: <Cache>
- Transactions: HTTP proxy transactions, HTTP refresh transactions, and HTTP pipeline transactions

Example

Parse the Pragma:no-cache tag in response bodies.

```
<Proxy>
  http.response.parse_meta_tag.pragma-no-cache(yes)
```

http.response.version()

Sets the version of the HTTP protocol to be used in the response to the client's user agent.

Syntax

```
http.response.version(1.0|1.1|preserve)
```

where:

- *preserve*: HTTP client response version will be set to the same value found on the server side inbound HTTP version, if it exists. The default is taken from the CLI configuration setting http version, which can be set to either 1.0 or 1.1. Changing this value in the CLI changes the default for both this property and "http.request.version()" on page 475.

Layer and Transaction Notes

- Layers: `<Proxy>`
- Transactions: HTTP proxy; does not apply to HTTP/2 connections

Example

See the Example for "http.request.version()" on page 475.

See Also

- Conditions: "http.response.version=" on page 179
- Properties: "http.request.version()" on page 475

http.server.accept_encoding()

Determines which encodings are allowed in an upstream request.

Syntax

```
http.server.accept_encoding.encoding(yes|no)
http.server.accept_encoding[encoding.,..](yes|no)
http.server.accept_encoding(encoding,..|client,..)
http.server.accept_encoding(all)
```

where:

- *encoding*: One of `br`, `deflate`, `gzip`, or `identity`.
- *client*: (Default behavior) Specified client(s) that will be replaced by the encoding(s) specified in client request. For client-less transactions, the default with a valid compression license is `all`; otherwise it is `identity`.
- `all`: All encodings supported by the client or by the appliance.

Layer and Transaction Notes

- Layers: `<Cache>`, `<Proxy>`
- Transactions: HTTP transactions (proxy, refresh, pipeline)

Example

Determine the accepted encodings.

```
<Proxy>
; accept only the identity encoding
  condition=condition1 http.server.accept_encoding(identity)
; accept only what the client allows
  condition=condition2 http.server.accept_encoding(client)
; accept all encodings supported by either the client or the appliance
  http.server.accept_encoding(all)
```

See Also

- Properties: "http.client.allow_encoding()" on page 424, "http.server.accept_encoding.allow_unknown()" on the facing page

http.server.accept_encoding.allow_unknown()

Determines whether or not unknown encodings in the client's request are allowed.

Syntax

```
http.server.accept_encoding.allow_unknown(yes|no)
```

The default value with a valid compression license is no; otherwise, yes.

Layer and Transaction Notes

- Layers: <Cache>, <Proxy>
- Transactions: HTTP transactions (proxy, refresh, pipeline)

Example

Allow only encodings supported by the appliance.

```
<Proxy>
  http.server.accept_encoding(all) http.server.accept_encoding.allow_unknown(no)
```

See Also

- Properties: "http.server.accept_encoding()" on the previous page

http.server.connect_attempts()

Set the number of attempts to connect performed per-address when connecting to the upstream host.

Syntax

`http.server.connect_attempts(`*`number`*`)`

where:

- *number*: Number from 1 to 10.

Layer and Transaction Notes

- Layers: `<Forward>`
- Transactions: HTTP transactions (proxy, refresh, pipeline)

```
<Forward>
  http.server.connect_attempts(7)
```

http.server.connect_timeout()

Controls the IP connection timeout used when attempting to establish a server connection.

Syntax

`http.server.connect_timeout(default|N)`

where:

- `default`: Default connection timeout.
- *N*: Number from 10 to 120 that specifies the number of seconds to wait before the connection times out.

Layer and Transaction Notes

- Layers: `<Forward>`, `<Proxy>`, `<SSL-Intercept>`
- Transactions: HTTP and HTTPS transactions

Example

Specifies a timeout value of 20 seconds for connecting to myhost.com and sets the default timeout value for all other hosts.

```
<Proxy>
  url.host=myhost.com http.server.connect_timeout(20)
  http.server.connect_timeout(default)
```

See Also

- Properties: "http.server.connect_attempts()" on the previous page, "http.server.recv.timeout()" on the next page

http.server.recv.timeout()

Sets the socket timeout for receiving bytes from the upstream host.

Syntax

`http.server.recv.timeout(auto|`*`recv-timeout`*`)`

where:

- *`recv-timeout`*: Amount of time (in seconds) that the server waits to receive data before timing out. Default is 180.

Layer and Transaction Notes

- Layers: `<Forward>`, `<Proxy>`
- Transactions: HTTP transactions, HTTP pipeline transactions

Example

Requests from HR_subnet get a receive timeout of 200 seconds Any request heading to a host that ends in example.com gets a receive timeout of 20 seconds All refresh traffic except that for example.com hosts gets a receive timeout of 300 seconds

```
define subnet HR_subnet
  10.10.0.0/16
end

<Proxy>
  client.address=HR_subnet http.client.recv.timeout(200)

<Forward>
  server_url.domain=example.com http.server.recv.timeout(20) http.refresh.recv.timeout(auto)
  http.refresh.recv.timeout(300)
```

http.server.persistence()

Controls persistence of the connection to the HTTP server.

Syntax

```
http.server.persistence(yes|no|preserve)
```

where:

- yes: (Default behavior) Persist connection to the server. The default value is taken from HTTP configuration, which is yes by default.
- no: After the current transaction is complete, the server connection is dropped.
- preserve: Reflect the client's persistence to the server connection.

Layer and Transaction Notes

- Layers: <Cache>, <Proxy>
- Transactions: HTTP transactions (proxy, refresh, pipeline)

Example

This property allows control of the persistence of individual server connections based on any request based conditions available in the <Cache> layer, or any request or client based conditions in a <Proxy> layer. The following example shows the property used to disable persistent connections to a specific host.

```
<Proxy>
  server_url.host=my_host.my_business.com http.server.persistence(no)
```

See Also

- Properties: "http.client.persistence()" on page 425

http2.client.accept()

This property determines whether the proxy accepts HTTP/2 on the client-side connection.

Use of this property is optional; the appliance supports HTTP/2 with the specified default behavior without the need for additional configuration or policy.

If the proxy is configured to not accept HTTP/2, requests proceed over HTTP/1.1.

Syntax

```
http2.client.accept(yes|no)
```

where:

- yes: (Default behavior) Server accepts HTTP/2 requests from the client.
- no: Server accepts HTTP/2 requests from the client as HTTP/1.1 on the connection.

Layer and Transaction Notes

- Layers: <Proxy> , <SSL>
- Transactions: HTTP and SSL-terminated HTTPS transactions

http2.client.max_concurrent_streams()

Each request on an HTTP/2 connection is handled on a different stream, but you can specify the number of concurrent streams. This property determines the maximum number of concurrent HTTP/2 streams that the client may initiate on the current client connection.

Use of this property is optional; the appliance supports HTTP/2 with the specified default behavior without the need for additional configuration or policy.

Syntax

`http2.client.max_concurrent_streams(`*`streams`*`)`

where:

- *`streams`*: Maximum number of streams. Accepted values are from 1 to 150. The default is 15.

Layer and Transaction Notes

- Layers: `<Proxy>`
- Transactions: HTTP and SSL-terminated HTTPS transactions

http2.server.request()

This property determines whether the proxy requests HTTP/2 on the server-side connection.

Use of this property is optional; the appliance supports HTTP/2 with the specified default behavior without the need for additional configuration or policy.

Syntax

```
http2.server.request(yes|no|preserve)
```

where:

- yes: Proxy requests HTTP/2 on server-side connection.
- no: Proxy does not request HTTP/2 on server-side connections
- preserve: (Default behavior) Proxy requests HTTP/2 on the server side only if HTTP/2 is requested on the client-side connection.

Layer and Transaction Notes

- Layers: <Forward>, <Proxy> , <SSL>
- Transactions: HTTP and SSL-terminated HTTPS transactions

Example

Accept HTTP/2 requests from the client, but request HTTP/2 from servers only to match the client-side setting.

```
<Proxy>
  http2.client.accept(yes) http2.server.request(preserve)
```

im.tunnel()

(Deprecated) Determines whether IM traffic will be tunneled.

Syntax

```
im.tunnel(yes|no)
```

where:

- yes: Transaction associated with connection will be tunneled.
- no: (Default behavior) If IM client version is not supported, the connection is blocked. If IM client version is supported, provide full policy support.

Layer and Transaction Notes

- Layers: <Proxy>
- Transactions: IM proxies

integrate_new_hosts()

Determines whether to add new host addresses encountered during DNS resolution of forwarding hosts to health checks and load balancing.

Syntax

```
integrate_new_hosts(yes|no)
```

The default is no.

Layer and Transaction Notes

- Layers: <Forward>
- Transactions: all except SOCKS and administrator transactions

See Also

- Properties: "forward()" on page 415

log.rewrite.field_id()

Controls rewrites of a specific log field in one or more access logs. Individual access logs are referenced by the name given in configuration. Configuration also determines the format of each log.

Syntax

```
log.rewrite.field_id("substitution"|no)
log.rewrite.field_id[access_log,..]("substitution"|no)
```

where:

- *field_id*: Specifies the log field to rewrite. Some field ids have embedded parentheses, for example cs(User-agent). These field-ids must be enclosed in quotes. There are two choices for quoting, either of which are accepted by the CPL compiler:
  - `log.rewrite."cs(User-agent)"(...)`
  - `"log.rewrite.cs(User-agent)(...)"`

    Either single or double quotes may be used.
- *substitution*: Quoted string containing replacement text for the field. The substitution string can contain CPL substitution variables.
- no: Cancels any previous substitution for this log field.

Discussion

Each of the syntax variants has a different role in specifying the rewrites for the access log fields used to record the transaction:

- `log.rewrite.field_id( )` specifies a rewrite of the field_id field in all access logs selected for this transaction.
- `log.rewrite.field_id[access_log,..]( )` specifies a rewrite of the field_id field in all access logs named in log_name_list. The field_id field in any logs not named in the list is unaffected.

Layer and Transaction Notes

- Layers: `<Cache>`, `<Exception>`, `<Forward>`, `<Proxy>`
- Transactions: all proxies

See Also

- Properties: "access_log()" on page 333, "log.suppress.field_id( )" on the facing page
- *ProxySG Log Fields and CPL Substitutions Reference*: https://techdocs.broadcom.com/content/broadcom/techdocs/us/en/symantec-security-software/web-and-network-security/proxysg/7-2/proxysg-log-fields-and-substitutions.html

log.suppress.*field_id*( )

Controls suppression of the specified field-id in one or more access logs. Individual access logs are referenced by the name given in configuration. Configuration also determines the format of the each log.

Syntax

```
log.suppress.field_id(yes|no)
log.suppress.field_id[access_log,..](yes|no)
```

where:

- *field_id*: Specifies the log field to suppress. Some field-ids have embedded parentheses, for example cs(User-agent). These field-ids must be enclosed in quotes. There are two choices for quoting, either of which are accepted by the CPL compiler:
  - `log.suppress."cs(User-agent)"(yes|no)`
  - `"log.suppress.cs(User-agent)(yes|no)"`

    Either single or double quotes may be used.
- yes: Suppresses the specified field_id.
- no: Turns suppression off for the specified field_id.

Discussion

Each of the syntax variants has a different role in suppressing the access log fields used to record the transaction:

- `log.suppress.`*field_id*`( )` controls suppression of the field_id field in all access logs selected for this transaction.
- `log.suppress.`*field_id*`[log_name_list]( )` controls suppression of the field_id field in all access logs named in log_name_list. The field_id field in any logs not named in the list is unaffected.

Layer and Transaction Notes

- Layers: `<Cache>`, `<Exception>`, `<Forward>`, `<Proxy>`
- Transactions: all proxies

See Also

- Properties: "access_log()" on page 333, "log.rewrite.field_id()" on page 491

max_bitrate()

Enforces upper limits on the instantaneous bandwidth of the current streaming transaction. This policy is enforced during initial connection setup. If the client requests a higher bit rate than allowed by policy, the request is denied.

> **Note:** Under certain network conditions, a client may receive a stream that temporarily exceeds the specified bit rate.

Syntax

```
max_bitrate(bitrate|no)
```

where:

- `bitrate`: Maximum bit rate allowed. Specify using an integer, in bits, kilobits (1000x), or megabits (1,000,000x), as follows: *integer* | *integer*`k` | *integer*`m`
- no: (Default behavior) Allows any bitrate.

Layer and Transaction Notes

- Layers: `<Cache>`, `<Proxy>`
- Transactions: streaming transactions

Example

Client bit rate for streaming media cannot exceed 56 kilobits.

```
<Proxy>
  max_bitrate(56k)
```

See Also

- Conditions: "bitrate=" on page 83, "live=" on page 192, "streaming.content=" on page 284

never_refresh_before_expiry()

This property is similar to the CLI command `#(config)`**`http strict-expiration refresh`** except that it provides per-transaction control to allow overriding the global default set by the command.

Syntax

```
never_refresh_before_expiry(yes|no)
```

The default value is taken from configuration.

Layer and Transaction Notes

- Layers: `<Cache>`
- Transactions: all proxies

See Also

- Properties: "never_serve_after_expiry()" on the next page, "remove_IMS_from_GET()" on page 504, "remove_PNC_from_GET()" on page 505, "remove_reload_from_IE_GET()" on page 506
- *Command Line Interface Reference*

never_serve_after_expiry()

This property is similar to the CLI command `#(config)`**`http strict-expiration serve`** except that it provides per transaction control to allow overriding the box-wide default set by the command.

Syntax

```
never_serve_after_expiry(yes|no)
```

The default value is taken from configuration.

Layer and Transaction Notes

- Layers: `<Cache>`

See Also

- Properties: "always_verify()" on page 345, "never_refresh_before_expiry()" on the previous page
- *Command Line Interface Reference*

notify_email.recipients()

Specifies an e-mail recipient list to be used when a notify_email action object is triggered in policy.

When notify_email.recipients() is used in the same policy set as notify_email, this gesture overrides the defined list of recipients and allows administrators to specify a list of addresses in policy to be notified when a user matches policy with this object attached.

Syntax

`notify_email.recipients(email_address,..)`

where:

- *email_address*: Addresses to which the appliance will send email when a notify_email object is matched in policy. The email "From" and SMTP server address must be set in **Maintenance > Event Logging > Mail**. In the event that multiple conditions match the same notify_email.recipients() object, only the last rule matched will apply.

Layer and Transaction Notes

- Layers: <Admin>, <Cache>, <Diagnostic>, <DNS-Proxy>, <Exception>, <Forward>, <Proxy> , <SSL>, <SSL-Intercept>

Example

Insert CPL usage example here, preferably with introduction text. If more than one example, number them Example 1, Example 2, etc.

```
<Proxy>
  condition=department1_subnet notify_email.recipients(admin1@department1.abc.com,
admin2@department1.abc.com)
  condition=department2_subnet notify_email.recipients(admin1@department2.abc.com,
admin3@department1.abc.com)

<Proxy>
  condition=restricted_sites action.notifyAdmin(yes)

define action notifyAdmin
  notify_email("RESTRICTED ACCESS DETECTED","A user has accessed a domain in a restricted
policy.$(CRLF)$(CRLF)The user at $(client.address), logged in as $(user) attempted to access
$(url.host)$(url.path), which is in the following URL categories: $(cs-categories).")
end define
```

See Also

- Actions: "notify_email()" on page 609, "notify_snmp()" on page 610
- *ProxySG Log Fields and CPL Substitutions Reference*: https://techdocs.broadcom.com/content/broadcom/techdocs/us/en/symantec-security-software/web-and-network-security/proxysg/7-2/proxysg-log-fields-and-substitutions.html

OK

Specifies no action on a policy rule. If the rule matches, any subsequent rules in the layer are not executed and preceding policy decisions are preserved. Consider the following example:

```
group=special OK
 url.domain=restricted.com deny
```

If a transaction matches the group= condition, the rest of the layer is skipped and any previous decisions are not overruled.

> **Tip:** Using OK achieves the same result as writing conditions without explicit actions. For example, the following rules have the same effect:
>
> ```
> group=special OK
>  group=special
> ```
>
> Although the second rule is syntactically correct, using OK makes the intent of the rule clearer.

Syntax

```
condition OK
```

Layer and Transaction Notes

- Layers: <Admin>, <Cache>, <Diagnostic>, <DNS-Proxy>, <Exception>, <Forward>, <Proxy> , <SSL>, <SSL-Intercept>, <Tenant>
- Transactions: all

Example

If the effective client address is not in one of the specified geolocations, deny the transaction and display the specified message.

```
<Proxy>
  client.effective_address.country=(US, CA) OK
  deny("Restricted location: $(x-cs-client-effective-ip-country)")
```

See Also

- Properties: "allow" on page 344, "deny" on page 392

pipeline()

Determines whether an object embedded within an HTML container object is pipelined. Set to yes to force pipelining, or set to no to prevent the embedded object from being pipelined. This property affects processing of the individual URLs embedded within a container object. It does not prevent parsing of the container object itself.

If this property is used with a URL access condition, such as url.host=, each embedded object on a page is evaluated against that policy rule to determine pipelining behavior. For example, a rule that disallows pipelining for a particular host would still allow pipelining for images on the host's pages that come from other hosts.

> **Note:** Pipelining might cause issues for upstream devices that are low in TCP resources. The best solution is to remove the bottleneck. A temporary solution might include fine-tuning the device and disabling pipelining.

Syntax

```
pipeline(yes|no)
```

The default value is yes.

Layer and Transaction Notes

- Layers: <Cache>
- Transactions: HTTP proxy

reference_id()

Set a policy ID for a rule. The ID will be visible in all policy traces and access logs associated with requests matching the rule.

To view the ID in access logs, include the `x-bluecoat-reference-id` field in the access log format.

Syntax

`reference_id(`*`policy_ID`*`)`

Layer and Transaction Notes

- Layers: `<Admin>`, `<Cache>`, `<Proxy>` , `<SSL>`, `<SSL-Intercept>`

Example

Set a policy ID for a rule denying access to sites matching the specified regex.

```
<Proxy>
  url.regex="youtube" Deny reference_id("youtube_deny")
```

reflect_ip()

Determines how the client IP address is presented to the origin server for proxied requests.

> **Note:** The appliance must be in the routing path for the reflect_ip property to work properly.

Syntax

`reflect_ip(auto|no|client|vip|`*`ip_address`*`)`

where:

- `auto`: Might reflect the client IP address, based on a config setting.
- `no`: Appliance's IP address is used to originate upstream connections.
- `client`: Client's IP address is used in initiating upstream connections.
- `vip`: Appliance's VIP on which the client request arrived is used to originate upstream traffic.
- *`ip_address`*: Specific IP address, which must be an address (either physical or virtual) belonging to the appliance. If not, at runtime this is converted to auto.

Layer and Transaction Notes

- Layers: `<DNS-Proxy>`, `<Forward>`, `<Proxy>`
- Transactions: proxy and DNS transactions

Example

For requests from a specific client, use the virtual IP address.

```
<Proxy>
  client.address=10.1.198.0 reflect_ip(vip)
```

See Also

- Properties: "forward()" on page 415

refresh()

Controls refreshing of requested objects. Set to no to prevent refreshing of the object if it is cached. Set to yes to allow the cache to behave normally.

Syntax

```
refresh(yes|no)
```

The default value is yes.

Layer and Transaction Notes

- Layers: <Cache>

See Also

- Properties: "advertisement()" on page 343, "always_verify()" on page 345, "bypass_cache()" on page 373, "cache()" on page 374, "cookie_sensitive()" on page 390, "direct()" on page 396, "force_cache()" on page 407, "never_refresh_before_expiry()" on page 495, "never_serve_after_expiry()" on page 496, "ttl()" on page 584, "ua_sensitive()" on page 585

remove_IMS_from_GET()

This property is similar to the CLI command `#(config)` **`http substitute if-modified-since`** except that it provides per transaction control to allow overriding the global default set by the command.

Syntax

```
remove_IMS_from_GET(yes|no)
```

The default value is taken from configuration.

Layer and Transaction Notes

- Layers: `<Cache>`
- Transactions: HTTP proxy

See Also

- Properties: "never_refresh_before_expiry()" on page 495, "never_serve_after_expiry()" on page 496, "remove_PNC_from_GET()" on the facing page, "remove_reload_from_IE_GET()" on page 506
- *Command Line Interface Reference*

remove_PNC_from_GET()

This property is similar to the CLI command `#(config)` **`http substitute pragma-no-cache`** except that it provides per transaction control to allow overriding the global default set by the command.

Syntax

```
remove_PNC_from_GET(yes|no)
```

The default value is taken from configuration.

Layer and Transaction Notes

- Layers: `<Cache>`
- Transactions: HTTP proxy

See Also

- Properties:"never_refresh_before_expiry()" on page 495, "never_serve_after_expiry()" on page 496, "remove_IMS_from_GET()" on the previous page, "remove_reload_from_IE_GET()" on the next page
- *Command Line Interface Reference*

remove_reload_from_IE_GET()

The remove_reload_from_IE_GET( ) property is similar to the CLI command `#(config) http substitute ie-reload` except that it provides per transaction control to override the global default set by the command.

Syntax

```
remove_reload_from_IE_GET(yes|no)
```

The default value is taken from configuration.

Layer and Transaction Notes

- Layers: `<Cache>`
- Transactions: HTTP proxy

See Also

- Properties: "never_refresh_before_expiry()" on page 495, "never_serve_after_expiry()" on page 496, "remove_IMS_from_GET()" on page 504, "remove_PNC_from_GET()" on the previous page
- *Command Line Interface Reference*

request.icap_mirror()

Releases content that is to be uploaded from the client to the server while a configured ICAP external service simultaneously scans the content. This property prevents errors from occurring on the client when the ICAP scan takes longer than the timeout of the protocol handling the request.

Syntax

```
request.icap_mirror (yes|no)
```

Layer and Transaction Notes

- Layers: <Proxy>
- Transactions: FTP, HTTP, and HTTPS transactions
- Requires that a "request.icap_service()" on page 509 rule exists in the <Proxy> and <Cache> layers.
- Can use with "request.icap_mirror()" above in the same transaction.
- If the server sends a response to the client before ICAP has finished scanning the content, the client receives the response from the server without waiting for the scan to complete.
- Supports persistent connections. For example, you can initiate a second file transfer on the same connection as the first completed file transfer, even if ICAP has not finished scanning the first transfer.
- For details on the ICAP scan results, refer to the access log. Entries are written to the access log when the ICAP scan completes.

Example

The following policy ensures that only video files that are larger than 10MB are simultaneously scanned by ICAP services and uploaded to youtube.com.

```
<Proxy>
; specifies that the content is an upload to youtube
  http.method=(POST,PUT) url.domain="youtube.com"

; specifies that only uploads larger than 10MB are
; to be simultaneously scanned and uploaded
  http.request.body.size=10485760.. request.icap_mirror(yes)

<Cache>
; specifies that uploads that fail the icap scan are closed
  http.method=(POST,PUT) request.icap_service(icap1,fail_closed)
```

See Also

- Properties: "request.icap_service()" on the facing page, "response.icap_service()" on page 522

request.icap_service()

Determines whether a request from a client should be processed by an external ICAP service before going out. Typical applications include regulatory compliance monitoring and intellectual property protection.

This property can specify a fail-over sequence of ICAP request modification services or service groups. The first healthy service in the sequence will be used to process the request. ICAP request processing can also be disabled using this property. Optionally, the failure mode can be specified to control behavior if none of the listed services is healthy.

Syntax

```
request.icap_service(service_name,.. [fail_open|fail_closed])
request.icap_service(no)
```

where:

- *service_name*: Configured ICAP service or service group that supports request modification.
- fail_open: (Default behavior) If none of the ICAP services listed is healthy, the request is sent out and a response delivered to the client (subject to other policies).
- fail_closed: If none of the ICAP services listed is healthy, the request is denied. This is the default and need not be specified to be in effect.
- no: Disables ICAP processing for this request.

Layer and Transaction Notes

- Layers: <Cache>, <Proxy>
- Transactions: HTTP proxy transactions, FTP proxy transactions; implicit and explicit FTPS proxy transactions

Example

All requests will be scanned except those going to internal servers. Requests coming from a special group of clients will be allowed out even if the request cannot be scanned.

```
; general rule - scan all requests
<Proxy>
; some users can get out if the scanners are down
  group=trusted_users request.icap_service(IP_service, IP_backup_service, fail_open)
  request.icap_service(IP_service, IP_backup_service) ; default is fail_closed

; exception - no need to scan requests to internal servers
<Proxy>
  condition=internal_servers request.icap_service(no)
```

See Also

- Properties:"response.icap_service()" on page 522

request.icap_service.secure_connection()

Determines whether an ICAP connection should be secure or not. This property can specify whether an ICAP connection should be secure or not for all ICAP services, or for some specific ICAP service, or for a list of ICAP services.

Syntax

```
request.icap_service.secure_connection(yes|no|auto)
request.icap_service.secure_connection.service_name(yes|no|auto)
request.icap_service.secure_connection.[service_name,..](yes|no|auto)
```

where:

- yes: ICAP service(s) use secure connection.
- no: ICAP service(s) use plain (nonsecure) connection.
- auto: (Default behavior) ICAP service(s) use global default connection setting.
- *service_name*: Configured ICAP service that supports request modification.

Layer and Transaction Notes

- Layers: <Cache>, <Proxy>
- Transactions: HTTP proxy transactions, FTP proxy transactions; implicit and explicit FTPS proxy transactions

Notes

- If an ICAP service does not support secure connection and the value is set to "yes", there will be an CPL compiler error. Likewise, if an ICAP service does not support plain connection and the value is set to "no", there will be an CPL compiler error.
- If it is a global set (a set without service names), there will be no compiler error even if some ICAP service does not support the specified connection type. On the other hand, runtime error may be generated as a result of that.

Example

All ICAP services use default global connection settings (if not specified otherwise), Except service icap1 must use secure connection, And service icap2, icap3 must use plain connection.

```
<Proxy>
  request.icap_service.secure_connection(auto)

<Proxy>
  request.icap_service.secure_connection.icap1(yes)
```

```
<Proxy>
  request.icap_service.secure_connection[icap2, icap3](no)
```

See Also

- Conditions: "request.icap.error_code=" on page 238, "response.icap.error_code=" on page 252
- Properties: "request.icap_service.secure_connection()" on the previous page

response.icap_feedback()

Controls the type of feedback given to both interactive and non-interactive clients while response scanning is in progress, and the delay before any feedback delivery starts. This controls the feedback delivered to both interactive and non-interactive clients. However since patience pages are served only to interactive clients, response.icap_feedback(patience_page, delay) is interpreted as: response.icap_feedback.interactive(patience_page, delay) response.icap_feedback.non_interactive(no) Administrators should be aware of the increased security risks associated with trickling.

Syntax

```
response.icap_feedback(patience_page[,patience_delay])
response.icap_feedback(trickle_start|trickle_end[,trickle_delay])
response.icap_feedback(no)
```

where:

- *patience_page*: Returns a patience page to an interactive client after patience_delay seconds if scanning has not completed within that time. No feedback is given to non_interactive clients. This option has good security characteristics, and a reasonable user experience for interactive clients.
- patience_delay: Number of seconds before a patience page is delivered to the client. The allowed range is 5 to 65535. The default is 5. Setting the delay to 0 turns trickling off.
- trickle_start: Begins delivering bytes to the client after trickle_delay seconds if scanning has not completed within that time. HTTP response headers are delivered at line speed. The response body is delivered to the client at the reduced (trickle) rate. The last 12K bytes of the response will be held until the scanning result is known..

  Trickled data may contain a threat, and although the end of the response is corrupted to render it unusable, some client applications may still be vulnerable. Since all the data is delivered to the client at a reduced rate, this is somewhat more secure than trickle_end, but the user will see very little intermediate progress.
- trickle_end: Begins delivering bytes at line speed to the client after trickle_delay seconds if scanning has not completed within that time. The last 16K bytes will be buffered by the appliance and trickling begins only when no more data is expected from the server. The last 12K bytes of the response will be held until the scanning result is known.

  Trickled data may contain a threat, and although the end of the response is corrupted to render it unusable, some client applications may still be vulnerable. Since only the last part of the data is delivered to the client at a reduced rate, this is somewhat less secure than trickle_start, but the user will see immediate initial progress.
- trickle_delay: Number of seconds before feedback to the client starts. The allowed range is 1 to 65535. The default is 5. Setting the delay to 0 turns trickling off.
- no: (Default behavior) Prevents any bytes from being delivered to the client until scanning completes. This option has good security characteristics, but the worst user experience.

Layer and Transaction Notes

- Layers: <Proxy>
- Transactions:HTTP proxy transactions, FTP proxy transactions

Example

Assume that automated tools used within the organization can be distinguished through a "robots" condition, defined elsewhere in the policy.

```
<Proxy>
; To prevent timeouts, robots get data trickled to them right away.
  condition=robots response.icap_feedback(trickle_start, 1)
; everyone else (presumably real users) get a patience page after 7 seconds
  response.icap_feedback(patience_page,7)
```

See Also

- Properties: "request.icap_service()" on page 509, response.icap_feedback.interactive( ), response.icap_feedback.non_interactive( ), response.icap_feedback.force_interactive( )

response.icap_feedback.force_interactive()

Modifies the logic used to determine whether or not this HTTP transaction represents an opportunity to interact with the user.

Different feedback can be returned to the client depending on whether or not the current transaction is judged to be interactive or non-interactive. For example, patience pages can only be delivered when the transaction is interactive. Logic built into the system makes the determination for each transaction.

This property is used to override portions of that logic. This property cannot be used to override the following reasons to consider the transaction non-interactive:

- the request method is not 'GET'
- the response is not '200 OK'
- the server provides an unsolicited content encoding (one not requested by the appliance)
- there is a 'Range' or 'If-Range' header in the request
- the Always check with source before serving object (or #(config caching) always-verify-source CLI) option is enabled

Syntax

```
response.icap_feedback.force_interactive(yes|no)
response.icap_feedback.force_interactive.reason(yes|no)
response.icap_feedback.force_interactive(reason,..)
response.icap_feedback.force_interactive[reason,..](yes|no)
```

where:

- reason: Takes one of the following values, corresponding to the overridable portions of the logic used to determine whether or not interaction with a user is possible.
  - user-agent - Overrides the decision to consider non-graphical browsers to be considered interactive. Any User-Agent header string beginning with mozilla or opera are considered graphical.
  - extension - Overrides the decision to consider requests for URLs with graphical file extensions or extensions indicating cascading styleheets, javascript, vbscript, vbx, or java applet or flash animation content as non-interactive.
  - content-type - Overrides the decision to consider as non-interactive content similar to that listed under extension, but based on the Content-Type header of the HTTP response.

The default behavior is no.

Layer and Transaction Notes

- Layers: `<Proxy>`
- Transactions: HTTP proxy

Example

The following example assumes that the organization has an interactive Mozilla-compatible browser that identifies itself with a custom User-Agent header. The policy overrides the default logic so that this user agent will be considered interactive. This form of the syntax is used so that this policy will not interfere with other policy making decisions about the content type or extension (for example whether a transaction requesting graphical content for example will be considered interactive.)

```
<Proxy>
  .User-Agent=brandedagent response.icap_feedback.force_interactive.user-agent(yes)
```

See Also

- Properties: "response.icap_feedback()" on page 513, "response.icap_feedback.non_interactive()" on page 519, "response.icap_service()" on page 522

response.icap_feedback.interactive()

Controls the type of feedback given to interactive clients while response scanning is in progress, and the delay before any feedback delivery starts. Administrators should be aware of the increased security risks associated with trickling.

Syntax

```
response.icap_feedback.interactive(patience_page[,patience_delay])
response.icap_feedback.interactive(trickle_start|trickle_end[,trickle_delay])
response.icap_feedback.interactive(no)
```

where:

- patience_page: Returns a patience page to an interactive client after patience_delay seconds if scanning has not completed within that time. No feedback is given to non_interactive clients.

  This option has good security characteristics, and a reasonable user experience for interactive clients.

- patience_delay: Number of seconds before a patience page is delivered to the client. The allowed range is 5 to 65535. The default is 5.

- trickle_start: Begins delivering bytes to the client after trickle_delay seconds if scanning has not completed within that time. HTTP response headers are delivered at line speed. The response body is delivered to the client at the reduced (trickle) rate. The last 12K bytes of the response will be held until the scanning result is known.

  Trickled data may contain a threat, and although the end of the response is corrupted to render it unusable, some client applications may still be vulnerable. Since all the data is delivered to the client at a reduced rate, this is somewhat more secure than trickle_end, but the user will see very little intermediate progress.

- trickle_end: Begins delivering bytes at line speed to the client after trickle_delay seconds if scanning has not completed within that time. The last 16K bytes will be buffered by the appliance and trickling begins only when no more data is expected from the server. The last 12K bytes of the response will be held until the scanning result is known.

  Trickled data may contain a threat, and although the end of the response is corrupted to render it unusable, some client applications may still be vulnerable. Since only the last part of the data is delivered to the client at a reduced rate, this is somewhat less secure than trickle_start, but the user will see immediate initial progress.

- trickle_delay: Number of seconds before feedback to the client starts. The allowed range is 0 to 65535. The default is 5. Setting the delay to 0 turns trickling off.

- no: (Default behavior) prevents any bytes from being delivered to the client until scanning completes. This option has good security characteristics, but the worst user experience.

Layer and Transaction Notes

- Layers: <Proxy>
- Transactions: HTTP proxy transactions, FTP proxy transactions

Example

The following sample policy serves patience pages to FTP clients if scanning has not completed within 3 seconds. Note that all FTP transactions are considered interactive. Interactive HTTP transactions will have the first portion of the data delivered at line speed, while the last part of the response will be trickled. No policy is specified for non-interactive clients.

```
<Proxy>
  client.protocol=ftp response.icap_feedback.interactive(patience_page,3)
  response.icap_feedback.interactive(trickle_end)
```

See Also

- Properties: "response.icap_feedback()" on page 513, response.icap_service( ), response.icap_feedback.non_ interactive( ), response.icap_feedback.force_interactive( )

response.icap_feedback.non_interactive()

Controls the type of feedback given to non-interactive clients while response scanning is in progress, and the delay before any feedback delivery starts. Administrators should be aware of the increased security risks associated with trickling.

Syntax

```
response.icap_feedback.non_interactive(trickle_start|trickle_end[,trickle_delay])
response.icap_feedback.non_interactive(no)
```

where:

- trickle_start: Begins delivering bytes to the client after trickle_delay seconds if scanning has not completed within that time. HTTP response headers are delivered at line speed. The response body is delivered to the client at the reduced (trickle) rate. The last 12K bytes of the response will be held until the scanning result is known.

  Trickled data may contain a threat, and although the end of the response is corrupted to render it unusable, some client applications may still be vulnerable. Since all the data is delivered to the client at a reduced rate, this is somewhat more secure than trickle_end, but the user will see very little intermediate progress.

- trickle_end: Begins delivering bytes at line speed to the client after trickle_delay seconds if scanning has not completed within that time. The last 16K bytes will be buffered by the appliance and trickling begins only when no more data is expected from the server. The last 12K bytes of the response will be held until the scanning result is known.

  Trickled data may contain a threat, and although the end of the response is corrupted to render it unusable, some client applications may still be vulnerable. Since only the last part of the data is delivered to the client at a reduced rate, this is somewhat less secure than trickle_start, but the user will see immediate initial progress.

- trickle_delay: Number of seconds before feedback to the client starts. The allowed range is 0 to 65535. The default is 5. Setting the delay to 0 turns trickling off.

- no: Prevents any bytes from being delivered to the client until scanning completes. This option has good security characteristics, but the worst user experience. This is the default value.

Layer and Transaction Notes

- Layers:<Proxy>
- Transactions: HTTP proxy

Example

Set distinct feedback policies for known robots based on whether or not they might execute code from a corrupted package. It is assumed here that automated tools within the organization would provide distinct credentials that associates them with a "robots" group, and with other groups that distinguish risk. No policy is specified for interactive clients.

```
<Proxy>
  authenticate(my_realm)
; the following applies only to members of the "robots" group

<Proxy> group=robots
; high risk of executing code from a corrupted package
; -> no feedback
  group=high_execution_risk response.icap_feedback.non_interactive(no)
; low risk of executing code from a corrupted package
; -> trickle from the start with default delay
  group=low_execution_risk response.icap_feedback.non_interactive(trickle_start)
; no risk of executing code from a corrupted package
; -> trickle at the end, begin serving data immediately
  group=no_execution_risk response.icap_feedback.non_interactive(trickle_end,0)
```

See Also

- Properties: "response.icap_feedback()" on page 513, "response.icap_feedback.interactive()" on page 517, "response.icap_feedback.force_interactive()" on page 515, response.icap_service( )

response.icap_mirror()

Serves requested content directly to a user while simultaneously scanning that content via a configured ICAP external service. This prevents issues with some types of streaming content that would suffer from the latency introduced by the ICAP scan, degrading the user experience as users wait for content to be completely scanned.

Syntax

```
response.icap_mirror(yes|no)
```

Layer and Transaction Notes

- Layers: <Proxy>
- Transactions: HTTP proxy
- Requires that a "response.icap_service()" on the next page rule exists in a <Cache> layer.
- Can use with "response.icap_mirror()" above in the same transaction.
- Data handled by this action is not cached.
- In cases where communication between the appliance and the ICAP server is impeded, (the server is unavailable or the configured transaction count reaches the specified limit) and if the ICAP response modification rule is configured to fail_open, data will be sent to the client without sending it to the ICAP server. If there are no communication issues with the ICAP server, fail_open will not affect the behavior of ICAP Mirroring.

Example

The following policy ensures that the Yahoo Finance stock ticker stream is scanned, but users receive no interruption in service as a result of that scan.

```
<Proxy>
  url.domain="finance.yahoo.com" response.icap_mirror(yes)

<Cache>
  response.icap_service(icap1,fail_closed)
```

See Also

- Properties: "request.icap_mirror()" on page 507, "response.icap_service()" on the next page

response.icap_service()

Determines whether a response to a client is first sent to an ICAP service before being given to the client. Depending on the ICAP service, the response may be allowed, denied, or altered. Typical applications include malware scanning.

This property can specify a fail-over sequence of ICAP response modification services or service groups. The first healthy service in the sequence will be used to process the response. ICAP response processing can also be disabled using this property. Optionally, the failure mode can be specified to control behavior if none of the listed services is healthy.

Syntax

```
response.icap_service(servicename_1,servicename_2, fail_open|fail_closed)
response.icap_service(no)
```

where:

- servicename: Configured ICAP service or service group that supports response modification.
- fail_open: If none of the ICAP services listed is healthy, the response is processed and delivered to the client (subject to other policies).
- fail_closed: (Default behavior) If none of the ICAP services listed is healthy, the transaction is denied. This is the default and need not be specified to be in effect.
- no: Disables ICAP processing for this response.

Layer and Transaction Notes

- Layers: `<Cache>`
- Transactions: HTTP transactions (proxy, refresh, pipeline), FTP proxy transactions; implicit and explicit FTPS proxy transactions

Example

All responses will be scanned except those going to internal servers. Responses coming from some business critical sites will be allowed even if the response cannot be scanned.

```
; general rule - scan all responses
<Cache>
; responses from some critical sites get through even if the scanners are down
  condition=critical_sites response.icap_service(VS_service, VS_backup_service, fail_open)
  response.icap_service(IP_service, IP_backup_service); default is fail_closed

; exception - no need to scan responses from internal servers
<Cache>
  condition=internal_servers response.icap_service(no)
```

See Also

- Properties: "request.icap_service()" on page 509

response.icap_service.force_rescan()

Forces ICAP to scan cached objects every time they are requested, even if the ICAP server ISTAG has not changed.

Syntax

```
response.icap_service.force_rescan(yes|no)
```

where:

- yes : ICAP service scans cached objects every time they are requested.
- no: (Default behavior) ICAP service rescans cached objects only when the ICAP server's ISTAG has changed since the last scan.

Layer and Transaction Notes

- Layers: <Proxy>
- Transactions: HTTP transactions (proxy, refresh, pipeline), FTP proxy transactions

Example

Rewrite ICAP headers, substituting 'test' headers with 'substitution' headers.

```
define action custom_ICAP_headers
  set(icap_respmod.request.x_header.test1, "substitution1")
  set(icap_respmod.request.x_header.test2, "substitution2")
end

; force ICAP service to rescan headers
<Proxy>
  response.icap_service.force_rescan(yes)
```

See Also

- Action: "set()" on page 619

response.icap_service.secure_connection()

Determines whether an ICAP connection should be secure or not. This property can specify whether an ICAP connection should be secure or not for all ICAP services, or for some specific ICAP service, or for a list of ICAP services.

Notes

If an ICAP service does not support secure connection and the value is set to "yes", there will be an CPL compiler error. Likewise, if an ICAP service does not support plain connection and the value is set to "no", there will be an CPL compiler error.

If it is a global set (a set without service names), there will be no compiler error even if some ICAP service does not support the specified connection type. On the other hand, runtime error may be generated as a result of that.

Syntax

```
response.icap_service.secure_connection(yes|no|auto)
response.icap_service.secure_connection.service_name(yes|no|auto)
response.icap_service.secure_connection.[service_name,..](yes|no|auto)
```

where:

- `yes`: ICAP service(s) use secure connection.
- `no`: ICAP service(s) use plain (nonsecure) connection.
- `auto`: (Default behavior) ICAP service(s) use global default connection setting.
- *`service_name`*: configured ICAP service that supports response modification.

Layer and Transaction Notes

- Layers: `<Cache>`
- Transactions: HTTP proxy transactions, FTP proxy transactions, and (in version 6.7.4) implicit and explicit FTPS proxy transactions

Example

All ICAP services use default global connection settings (if not specified otherwise), except service icap1 must use secure connection. Services icap2 and icap3 must use a plain connection.

```
<Cache>
  response.icap_service.secure_connection(auto)

<Cache>
  response.icap_service.secure_connection.icap1(yes)
```

```
<Cache>
  response.icap_service.secure_connection[icap2, icap3](no)
```

See Also

- Conditions: "request.icap.error_code=" on page 238, "response.icap.error_code=" on page 252
- Properties: "request.icap_service.secure_connection()" on page 511

response.raw_headers.max_count()

Limit the number of response headers allowed in an HTTP response.

The number of HTTP response headers will be limited to the given number. If this limit is exceeded, then the appliance will throw an "invalid_response" exception. A leading white space based header continuation will not be counted as a separate header. In other words, number here refers to headers, not response lines, and does not include response status line or end-of-header marker (blank line).

The default value is 1,000. The minimum value is 0, and the maximum value is 10,000.

Syntax

`response.raw_headers.max_count(`*`unsigned_integer`*`)`

Layer and Transaction Notes

- Layers: `<Cache>`, `<Proxy>`
- Transactions: HTTP proxy

Example

Limit the number of response headers to 2,500.

```
<Proxy>
  response.raw_headers.max_count(2500)
```

response.raw_headers.max_length()

Limit the amount of response header data allowed in an HTTP response.

The total number of bytes of HTTP response header data is restricted to the given number. If this limit is exceeded, then the appliance will throw an "invalid_response" exception.

The default value is 100,000. The minimum value is 0, and the maximum value is 1,000,000.

Syntax

`response.raw_headers.max_length(`*`unsigned_integer`*`)`

Layer and Transaction Notes

- Layers: `<Cache>`, `<Proxy>`
- Transactions: HTTP proxy

Example

Limit the length of response headers to 250,000 bytes..

```
<Proxy>
  response.raw_headers.max_length(250000)
```

response.raw_headers.tolerate()

Determines which deviations from the protocol specification will be tolerated when parsing the response headers.

Syntax

```
response.raw_headers.tolerate(none|continue|invalid_header)
```

where:

- none: (Default behavior) No deviations are tolerated.
- continue: The response header parsing should tolerate a continuation line (white space) prior to the start of the first header
- invalid_header: The response header parsing should tolerate invalid headers. For more information on how invalid headers can affect the ProxySG appliance's performance, refer to TECH245814:

  http://www.symantec.com/docs/TECH245814
- invalid_status: Response header parsing should tolerate invalid status.

Layer and Transaction Notes

- Layers: <Cache>, <Proxy>
- Transactions: HTTP transactions (proxy, refresh, pipeline), HTTPS transactions

Example

This example illustrates how the property is used to specify which errors to tolerate. The conditions used are assumed to be defined elsewhere).

```
<Proxy>
; Tolerate a continuation line prior to the first header,
;  but only from a specified list of legacy servers.
  condition=legacy_server response.raw_headers.tolerate(continue)

; This is actually the default, so it doesn't need to be specified
  response.raw_headers.tolerate(none)
```

See Also

- Properties: "response.raw_headers.max_count()" on page 527, "response.raw_headers.max_length()" on the previous page

risk_score.maximum()

Allows you to specify that the value used to determine when the appliance discontinues scanning requests immediately after the maximum allowable risk score is reached. Specifying a maximum helps decrease the load on appliance resources dedicated to processing web attacks.

Syntax

`risk_score.maximum(integer)`

where:

- *integer*: Maximum risk score value before the appliance discontinues scan requests. You can specify an integer between 0 and 2147483647. Specifying zero disables risk score capping. The default value is 40.

Layer and Transaction Notes

- Layers: `<Proxy>`
- Transactions: HTTP proxy

Example

Discontinue scan requests when two anomalies are detected. Assume a default risk-score of 10.

```
<Proxy>
  risk_score_maximum(20)
```

See Also

- Properties: "risk_score.other()" on the facing page

risk_score.other()

Allows you to specify the risk score-based trigger to set an action based on the cumulative risk score that a client reaches for a given transaction.

Syntax

`risk_score.other[.attribute](risk_score_value)`

where:

- *attribute*: One of the following attack types:
  - null_byte
  - invalid_form_data
  - parameter_pollution
  - multiple_encoding
  - invalid_encoding
  - multiple_header
  - threshold_exceeded
- *risk_score_value*: Custom risk score.

Layer and Transaction Notes

- Layers: <Proxy>
- Transactions: HTTP proxy

Example

Set the risk score for HPP attacks to 30.

```
<Proxy>
  risk_score.other.parameter_pollution(30)
```

See Also

- Properties: "risk_score.maximum()" on the previous page

server.authenticate.basic()

Determines how to authenticate to an upstream server using BASIC credentials.

This property controls sending BASIC credentials to an upstream server or proxy for an authenticated user. By default, no credentials will be sent upstream. If origin is selected, then BASIC credentials will be sent upstream using the HTTP Authorization header.

If proxy is selected and forwarding is configured, then credentials will be sent upstream using the HTTP Proxy-Authorization header. If the user authenticated to the appliance using BASIC credentials, then by default, those credentials will be forwarded upstream. If the user authenticated using NTLM, Kerberos, or a realm which does not use passwords, then by default the username will be forwarded along with an empty password. Optionally, the username and password sent upstream can be configured with substitution strings.

Syntax

```
server.authenticate.basic(no)
server.authenticate.basic(origin|proxy[,username_substitution[,password_substitution]])
```

where:

- no: (Default behavior) Suppresses sending basic credentials to the upstream server.
- origin: Sends basic credentials to an upstream server.
- proxy: Sends basic credentials to an upstream proxy server.
- *username_substitution*: Substitution string which will be sent as the username instead of using the authenticated username.
- *password_substitution*: Substitution string which will be sent as the password instead of using the authenticated password.

Layer and Transaction Notes

- Layers: <Proxy>
- Transactions: HTTP proxy

Example 1

Authenticate to an upstream server using the user's BASIC credentials:

```
<Proxy>
  url.host.exact="webmail.company.com" server.authenticate.basic(origin)
```

Example 2

Authenticate to an upstream server using the authenticated username prefixed with a domain and the authenticated password:

```
<Proxy>
  url.host.exact="images.company.com" server.authenticate.basic(origin, "domain\$(user.name)")
```

Example 3

Authenticate to an upstream proxy using a fixed username and password:

```
<Proxy>
  url.host.exact="proxy.company.com" server.authenticate.basic(proxy, "internaluser",
"internalpassword")
```

Example 4

Authenticate to an upstream server using the IP address of the client and an empty password:

```
<Proxy>
  url.host.exact="images.company.com" server.authenticate.basic(origin, "$(client.address)", ""
```

server.authenticate.constrained_delegation()

Determines how to authenticate to an upstream server using Kerberos Constrained Delegation (KCD).

This property controls sending Kerberos credentials to an upstream server or proxy for an authenticated user. By default, no credentials will be sent upstream. If origin is selected, KCD will be used to send credentials upstream using the HTTP Authorization header. If proxy is selected and forwarding is configured, then Kerberos constrained delegation will be used to send credentials upstream using the HTTP Proxy-Authorization header.

Syntax

```
server.authenticate.constrained_delegation(no)
server.authenticate.constrained_delegation(origin, iwa_realm)
server.authenticate.constrained_delegation(proxy, iwa_realm)
```

where:

- no: (Default behavior) Suppresses sending Kerberos credentials to the upstream server.
- origin: Sends Kerberos credentials to an upstream server.
- proxy: Sends Kerberos credentials to an upstream proxy server.
- *iwa_realm*: Name of a configured IWA realm that is used to acquire the Kerberos tickets.

Layer and Transaction Notes

- Layers: <Proxy>
- Transactions: HTTP proxy

Example 1

Authenticate to an upstream server with KCD.

```
<Proxy>
  url.host.exact="images.company.com" server.authenticate.constrained_delegation(origin, iwa_realm_1)
```

Example 2

Authenticate to an upstream server with KCD.

```
<Proxy>
  url.host.exact="proxy.company.com" server.authenticate.constrained_delegation(proxy, iwa_realm_2)
```

server.authenticate.constrained_delegation.spn()

Determines the Service Principal Name (SPN) to use with Kerberos Constrained Delegation. This property is used to override the default Service Principal Name for an upstream server or proxy. By default the SPN is:

`HTTP/hostname_of_upstream_device[:port_number]`

The port number need only be specified if it is not standard. If the SPN is different, it can be explicitly set using the following property. KCD will be used to send credentials upstream using the HTTP Proxy-Authorization header.

Syntax

`server.authenticate.constrained_delegation.spn(service_principal_name)`

where:

- *service_principal_name*: SPN to use for Kerberos constrained delegation.

Layer and Transaction Notes

- Layers:<Proxy>
- Transactions: HTTP proxy

Example

Set the service principal name to use when authenticating to an upstream server with Kerberos constrained delegation:

```
<Proxy>
  url.host.exact="images.company.com" server.authenticate.constrained_delegation.spn
("HTTP/sharedserver.company.com")
```

server.certificate.validate()

Determines whether server X.509 certificates will be verified during the establishment of SSL connections.

For HTTPS-Reverse-Proxy and SSL-Proxy (Forward Proxy) intercepted transactions, this property checks for the following certificate errors:

- Expiration
- Untrusted Issuer
- Revocation
- Hostname-Mismatch

For SSL-Proxy (Forward Proxy) Tunneled transactions, server.certificate.validate() checks for the following certificate errors:

- Expiration
- Untrusted Issuer
- Revocation

When SSL-Proxy is tunneling the HTTPS traffic, it cannot check for Hostname-Mismatch.

Syntax

```
server.certificate.validate(yes|no)
```

- For HTTPS forward proxy and SSL tunnel transactions, the default is `yes`.
- For a reverse proxy configured to use a forwarding host, the default comes from the forwarding host's ssl-verify-server setting. By default, it is set to no. To verify server certificates in this scenario, issue the CLI command:

  ```
  #(config forwarding host_alias)ssl-verify-server
  ```

> **Note:** For best security, Symantec recommends that you do not disable certificate validation. If you must do so, disable it only for specific, trusted URLs, for example, using the "url=" on page 298condition. Including `server.certificate.validate(no)` in policy disables all certificate validation for the affected transactions, including checks for the validity of the certificate (such as trust chain and validity date range), as well as checks on the well-formedness of the certificate (such as valid algorithm identifiers and extension fields).

Layer and Transaction Notes

- Layers: <SSL>
- Transactions: HTTPS forward and reverse proxy transactions, SSL tunnel transactions

Examples

Do not validate server X.509 certificates for the given domain.

```
<SSL>
  url.domain="example.com" server.certificate.validate(no)
```

Tunnel the request without decrypting or validating the server certificate on the appliance. The browser performs server certificate validation and displays an SSL certificate warning to the user.

```
<SSL-Intercept>
  ssl.forward_proxy(no) server.certificate.validate(no)
```

See Also

- Properties: "server.certificate.validate.ignore()" on page 540, "ssl.forward_proxy.preserve_untrusted()" on page 562

server.certificate.validate.ccl()

Specify the upstream server CCL certificate ID.

Syntax

```
gserver.certificate.validate.ccl(ccl_id)
```

The host can be either an IP address or a URL.

Layer and Transaction Notes

- Layers: <Forward>, <SSL-Intercept>
- Transactions: HTTPS forward and reverse proxy transactions, SSL tunnel transactions

Example

For all connections with an SNI matching "www.thecompany.com", the server SSL certificate is validated using the certificate store created for “testccl”.

```
<SSL-Intercept>
  url.host="www.thecompany.com" server.certificate.validate.ccl(testccl)
```

See Also

- Properties: "client.certificate.validate.ccl()" on page 379

server.certificate.validate.check_revocation()

Check SSL server certificates for revocation.

Syntax

```
server.certificate.validate.check_revocation(auto|ocsp|local|no)
```

where:

- auto: (Default behavior) Certificate will be checked through OCSP if available, otherwise it will be checked against locally installed revocation list.
- ocsp: Check the certificate through OCSP.
- local: Check the certificate against the locally installed revocation list.
- no: Certificate will not be checked for revocation.

Layer and Transaction Notes

- Layers: <SSL>
- Transactions: HTTPS forward and reverse proxy transactions, SSL tunnel transactions

Example

Verify SSL server certificates and check against the OCSP or local list.

```
<SSL>
  server.certificate.validate(yes) server.certificate.validate.check_revocation(auto)
```

server.certificate.validate.ignore()

Ignore errors during server certificate validation.

This property specifies which errors should be ignored while validating the server certificate during the setup of an SSL connection. For SSL-Proxy (Forward-Proxy) tunneled transactions, the policy to ignore the hostname_mismatch error does not apply.

Syntax

```
server.certificate.validate.ignore.flag(yes|no)
server.certificate.validate.ignore[flag,..](yes|no)
server.certificate.validate.ignore(all|none|flag,..)
```

where:

- flag: One of expiration, untrusted_issuer, or hostname_mismatch
- all: Ignore all errors during server certificate validation.
- none: (Default behavior) Do not ignore errors during server certificate validation.

Layer and Transaction Notes

- Layers: <SSL>
- Transactions: HTTPS forward and reverse proxy transactions and SSL tunnel transactions

Example

For maximum security, you should validate all server certificates. For sites that have bad certificates that you nevertheless must be able to access, you can create a white list that disables only that part of the validation process necessary to access the site.

```
<SSL>
  server_url.host=some-server.com
  server.certificate.validate.ignore.expiration(yes)
  server_url.host=blah.com
  server.certificate.validate.ignore.hostname_mismatch(yes)
  server_url.host=do.this.at.your.own.peril
  server.certificate.validate.ignore.untrusted_issuer(yes)
```

See Also

- Properties: "server.certificate.validate()" on page 536

server.connection.client_issuer_keyring()

Authenticating users in a typical reverse proxy deployment involves steps such as configuring a client certificate authentication realm in SGOS and providing authentication to origin content servers (OCSes) behind the proxy using Kerberos, or forwarding specific client certificate fields to the OCS using an HTTP header.

To facilitate choosing signing certificates for the client, you can emulate client certificates. When this feature is enabled:

- The appliance requests a certificate from the client.
- If the client returns a certificate, the appliance copies the certificate attributes to a new client certificate (so that it appears to originate from the client). Emulation does not occur if the client does not return a certificate.
- The appliance presents the certificate during the SSL/TLS handshake when an OCS requests a client certificate.

Syntax

`server.connection.client_issuer_keyring(no|keyring_id|hsm_keyring_id|hsm_keygroup_id)`

where:

- no: (Default behavior): Disable client certificate emulation.
- *keyring_id*: Use the specified keyring for client certificate emulation. This must be a valid keyring, specified on the appliance with a CA certificate.
- *hsm_keyring_id*: Use the specified HSM keyring for client certificate emulation.
- *hsm_keygroup_id*: Use the specified HSM keygroup for client certificate emulation.

Layer and Transaction Notes

- Layers: `<Proxy>` , `<SSL>`
- Transactions: SSL transactions for forward proxies and reverse proxies

Example 1

Request a certificate from the client during SSL transactions.

```
<SSL>
  client.certificate.require(yes)
```

Example 2

If the web application is Facebook, use the specified keyring. Otherwise, disable client certificate emulation.

```
<SSL>
  request.application.name=Facebook server.connection.client_issuer_keyring(keyfile_1)
  server.connection.client_issuer_keyring(no)
```

Example 3

If requests match the 'internal' condition, use the specified keyring. Otherwise, disable client certificate emulation.

```
define url.domain condition internal
  corporate.internal.org
  it.internal.org
  hr.internal.org
  admin.internal.org
end

<SSL>
  condition=internal server.connection.client_issuer_keyring(hsm_1)
  server.connection.client_issuer_keyring(no)
```

See Also

- *SGOS Administration Guide*, “Managing X.509 Certificates” chapter

server.connection.client_keyring()

Set the keyring or keylist to use for client certificate requests.

Syntax

```
server.connection.client_keyring(keyring)
server.connection.client_keyring(keylist,selector)
```

where:

- *keyring*: Specifies the keyring to use for client certificate requests.
- *keylist*: Specifies the keylist to use for client certificate requests. The selector value must also be specified.
- *selector*: Takes a substitution variable.

  All substitution variables are supported; however recommended substitution variables for the selector include $(user), $(group), and $(client.address).

> **Note:** The Selector value must match the set of extractor values that are displayed when you run the view command for a keylist. For example, if the Subject.CN in the certificate is set to represent a user name, use the Selector $(user), and select the Extractor value $(Subject.CN) for the policy to take effect. If the Extractor value was set to $(Subject.O), no match would be found and policy would not be enforced.

Layer and Transaction Notes

- Layers: <Proxy> , <SSL>

Example 1

Use the certificate from <keyring> as the client certificate for user <user> connecting to a specific website <url>.

```
url=<url> user=<user> server.connection.client_keyring(<keyring>)
```

Example 2

Use the certificate from <keyring> as the client certificate for user <user> connecting to any website that requires a client certificate.

```
user=<user> server.connection.client_keyring(<keyring>)
```

Example 3

Use the certificate from <keyring> as the client certificate for all users of group <group> connecting to a specific website <url>.

```
url=<url> group=<group> server.connection.client_keyring(<keyring>)
```

Example 4

Select a keyring or certificate from the keylist <keylist> whose extractor value is equal to the user of the connection, for a specific website <url>.

```
url=<url> server.connection.client_keyring(<keylist>, "$(user)")
```

Example 5

For connections to a website <url>, this will select a keyring or certificate from keylist <keylist> whose extractor value is equal to the group of the connection.

```
url=<url> server.connection.client_keyring(<keylist>, "$(group)")
```

server.connection.dscp()

Controls server-side outbound QoS/DSCP value.

Syntax

`server.connection.dscp(`*`dscp_value`*`)`

where:

`dscp_value`: One of the following:

- decimal value between 0 and 63
- `preserve`: (default behavior) track the incoming DSCP value on the primary server connection and use that as the value when sending packets on the client connections.
- `echo`: outbound packet's DSCP value will use the same value as the inbound packet's DSCP value.
- a class: `af11 | af12 | af13 | af21 | af22 | af23 | af31 | af32 | af33 | af41 | af42 | af43 | cs1 | cs2 | cs3 | cs4 | cs5 | cs6 | cs7 | ef`

Layer and Transaction Notes

- Layers: `<Cache>`, `<DNS-Proxy>`, `<Forward>`, `<Proxy>`
- Transactions: all, including HTTP/2 streams

Example

The first QoS policy rule sets the server outbound QoS/DSCP value to echo, and the second QoS policy rule sets the server outbound QoS/DSCP value to 50.

```
<Proxy>
  server.connection.dscp(echo)

<Proxy>
  server.connection.dscp(50)
```

server.connection.encrypted_tap()

Enables or disables encrypted tap of server-side traffic. Server-side encrypted tap is useful for tracing content that has changed substantially after proxy processing, as in the case with MAPI over HTTP proxy.

When enabled, tapped traffic is sent to the specified interface. Tapped data is presented in a TCP-like format which can be easily understood by common network traffic analysis tools like Wireshark and common network intrusion detection systems such as Snort.

> **Note:** Server-side tap uses 02:02:02:02:02:02 as the destination MAC address and 01:01:01:01:01:01 as the source MAC address.

> **Note:** Encrypted tap does not support server-side HTTP/2 traffic.

Syntax

`server.connection.encrypted_tap(no|`*`interface`*`)`

where:

- no: Disable encrypted tap of server-side traffic.
- *`interface`*: Specify the interface for tapped encrypted SSL content on the server side. The form is *adapter:interface*.

Layer and Transaction Notes

- Layers: `<SSL>`
- Transactions: applies only to server connections

Example

Send tapped traffic to the 0:0 interface.

```
<SSL>
  server.connection.encrypted_tap(0:0)
```

See Also

- Property: "client.connection.encrypted_tap()" on page 382

- *ProxySG Log Fields and CPL Substitutions Reference*: https://techdocs.broadcom.com/content/broadcom/techdocs/us/en/symantec-security-software/web-and-network-security/proxysg/7-2/proxysg-log-fields-and-substitutions.html

server.connection.tap()

Enables or disables tap of server-side HTTP, FTP, and TCP traffic. Server-side tap is useful for tracing content that has changed substantially after proxy processing.

When enabled, tapped traffic is sent to the specified interface. Tapped data is presented in a TCP-like format which can be easily understood by common network traffic analysis tools like Wireshark and common network intrusion detection systems such as Snort.

> **Note:** Server-side tap uses 02:02:02:02:02:02 as the destination MAC address and 01:01:01:01:01:01 as the source MAC address.

> **Note:** Encrypted tap does not support server-side HTTP/2 traffic.

Syntax

`server.connection.tap(no|`*`interface`*`)`

where:

- no: Disable tap of server-side traffic.
- *`interface`*: Specify the interface for tapped content on the server side. The form is *adapter*:*interface*.

Layer and Transaction Notes

- Layers: `<Proxy>`
- Transactions: applies only to server connections

Example

Send tapped traffic to the 0:0 interface.

```
<Proxy>
  server.connection.tap(0:0)
```

See Also

- Properties: "client.connection.encrypted_tap()" on page 382, "client.connection.tap()" on page 383
- *ProxySG Log Fields and CPL Substitutions Reference*: https://techdocs.broadcom.com/us/en/symantec-security-software/web-and-network-security/proxysg/7-2/proxysg-log-fields-and-substitutions.html

server_url.dns_lookup()

Sets the global policy for IP connection type preference.

Syntax

`server_url.dns_lookup(dns_lookup_value)`

where:

- *dns_lookup_value*: One of the following:
  - (Default behavior) IPv4-Only
  - IPv6-Only
  - Prefer-IPv4
  - Prefer-IPv6

Layer and Transaction Notes

- Layers: `<Forward>`, `<Proxy>`
- Transactions: transactions connecting upstream to a proxy or origin content server (but not DNS Proxy transactions)

Example

The DNS resolver will query for only the IPv6 AAAA record for the etrade.com domain. Such a policy rule can be used to set the IP preference for the outgoing connection if a destination node has both an IPv4 and IPv6 address.

```
<Proxy>
  url.domain=etrade.com server_url.dns_lookup(IPv6-Only)
```

shell.prompt()

Sets the prompt for a proxied shell transaction.

Syntax

`shell.prompt(`*`substitution_string`*`)`

Layer and Transaction Notes

- Layers: <Proxy>
- Transactions: shell (Telnet) proxy

Example

All requests from HR_subnet get the Shell prompt “client's address: Welcome to this appliance.” All requests from ENG_ subnet get the default Shell prompt. All other requests get no Shell prompt.

```
define subnet HR_subnet
  10.10.0.0/16
 end

define subnet ENG_subnet
  10.9.0.0/16
end

<Proxy>
  client.address=HR_subnet shell.prompt("$(client.address): Welcome to $(appliance.name)")
  client.address=ENG_subnet shell.prompt(default)
  shell.prompt(no)
```

See Also

- *ProxySG Log Fields and CPL Substitutions Reference*: https://techdocs.broadcom.com/content/broadcom/techdocs/us/en/symantec-security-software/web-and-network-security/proxysg/7-2/proxysg-log-fields-and-substitutions.html

shell.realm_banner()

Sets the realm banner for a proxied shell transaction.

Syntax

`shell.realm_banner(`*substitution_string*`)`

Layer and Transaction Notes

- Layers: <Proxy>
- Transactions: shell (Telnet) proxy

Example

All requests from HR_subnet get the Shell realm banner “client's address: Welcome to this appliance.” All requests from ENG_ subnet get the default Shell realm banner. All other requests get no Shell realm banner.

```
define subnet HR_subnet
  10.10.0.0/16
end

define subnet ENG_subnet
  10.9.0.0/16
end

<Proxy>
  client.address=HR_subnet shell.realm_banner("$(client.address): Welcome to $(appliance.name)")
  client.address=ENG_subnet shell.realm_banner(default)
  shell.realm_banner(no)
```

See Also

- *ProxySG Log Fields and CPL Substitutions Reference*: https://techdocs.broadcom.com/content/broadcom/techdocs/us/en/symantec-security-software/web-and-network-security/proxysg/7-2/proxysg-log-fields-and-substitutions.html

shell.welcome_banner()

Sets the welcome banner for a proxied shell transaction.

Syntax

`shell.welcome_banner(`*`substitution_string`*`)`

Layer and Transaction Notes

- Layers: <Proxy>
- Transactions: shell (Telnet) proxy

Example

All requests from HR_subnet get the Shell welcome banner “client's address: Welcome to this appliance.” All requests from ENG_subnet get the default Shell welcome banner. All other requests get no Shell welcome banner.

```
define subnet HR_subnet
  10.10.0.0/16
end

define subnet ENG_subnet
  10.9.0.0/16
end

<Proxy>
  client.address=HR_subnet shell.welcome_banner("$(client.address): Welcome to $(appliance.name)")
  client.address=ENG_subnet shell.welcome_banner(default)
  shell.welcome_banner(no)
```

socks.accelerate()

The socks.accelerate property controls the SOCKS proxy handoff to other protocol agents.

Syntax

```
socks.accelerate(no|auto|http|aol_im|msn_im|yahoo_im)
```

where:

- no: The SOCKS proxy does not hand off the transaction to another proxy agent, but tunnels the SOCKS transaction.
- auto: (Default behavior) The handoff is determined by the URL scheme.

Any other value forces the SOCKS proxy to hand off the transaction to the agent for the indicated protocol.

"socks.accelerated=" on page 278 can be used to test which agent was selected for handoff. "tunneled=" on page 297 can be used to test for unaccelerated (tunneled) SOCKS transactions.

After the handoff, the transaction is subject to policy as a proxy transaction for the appropriate protocol. Within that policy, "socks=" on page 277 can be used to test for transactions use SOCKS for client communication.

Layer and Transaction Notes

- Layers: <Proxy>
- Transactions: SOCKS

See Also

- Conditions: "socks=" on page 277, "socks.accelerated=" on page 278, "socks.method=" on page 279, "socks.version=" on page 280
- Properties: "socks.authenticate()" on the next page, "socks.authenticate.force()" on page 556,"socks_gateway()" on page 557

socks.authenticate()

The same realms can be used for SOCKS proxy authentication as can be used for regular proxy authentication. This form of authentication applies only to SOCKS transactions.

"authenticate()" on page 348 does not apply to SOCKS transactions. However, if an accelerated SOCKS transaction has already been authenticated in the same realm by the SOCKS proxy, no new authentication challenge is issued. If the realms identified in the socks.authenticate( ) and "authenticate()" on page 348 properties differ, however, a new challenge is issued by the proxy agent used to accelerate the SOCKS transaction.

> **Note:** There is no optional display name.

Following SOCKS proxy authentication, the standard "group=" on page 140, "realm=" on page 208, and "user=" on page 312 tests are available.

The relation between SOCKS authentication and denial is controlled through the socks.authenticate.force( ) property. The default setting no implies that denial overrides socks.authenticate( ), with the result that user names may not appear for denied requests if that denial could be determined without authentication. To ensure that user names appear in access logs, use socks.authenticate.force(yes).

This property depends exclusively on a limited number of conditions:

- "client.address=" on page 87
- "proxy.address=" on page 198
- "proxy.port=" on page 200
- "socks.version=" on page 280

Date and time triggers, while available, are not recommended.

Syntax

`socks.authenticate(`*`realm_name`*`)`

Layer and Transaction Notes

- Layers: `<Proxy>`
- Transactions: SOCKS

See Also

- Conditions: "socks=" on page 277, "socks.accelerated=" on page 278, "socks.method=" on page 279, "socks.version=" on page 280
- Properties: "authenticate()" on page 348, "socks.accelerate()" on page 553, "socks.authenticate.force()" on the next page, "socks_gateway()" on page 557

socks.authenticate.force()

This property controls the relation between SOCKS authentication and denial. This does not affect "authenticate()" on page 348.

Syntax

```
socks.authenticate.force(yes|no)
```

where:

- yes: Makes "socks.authenticate()" on page 554 higher priority than "deny" on page 392 and "exception()" on page 402. Use yes to ensure that user IDs are available for access logging, even of denied requests.
- no: (Default behavior) "deny" on page 392 and "exception()" on page 402 have a higher priority than "socks.authenticate()" on page 554. This setting allows early denial (based on proxy card, address or port, client address, or SOCKS version, for example). That is, the denial preempts any authentication requirement.

Layer and Transaction Notes

- Layers: <Proxy>
- Transactions: SOCKS

See Also

- Conditions: "socks.method=" on page 279, "socks.version=" on page 280
- Properties:"socks.accelerate()" on page 553 , "socks.authenticate()" on page 554, "socks_gateway()" on the facing page

socks_gateway()

Controls whether or not the request associated with the current transaction is sent through a SOCKS gateway.

There is a global configuration setting (**Configuration > Forwarding > SOCKS Gateways > Default Sequence**) for the default SOCKS gateway failover sequence. The `socks_gateway( )` property is used to override the default SOCKS gateway failover sequence with a specific list of SOCKS gateway aliases. The list of aliases might contain the special token default, which expands to include the default SOCKS gateway failover sequence defined in configuration.

Duplication is allowed in the specified alias list only in the case where a gateway named in the default failover sequence is also named explicitly in *alias_list*.

In addition, there is a global configuration setting (**Configuration > Forwarding > SOCKS Gateways > Global Defaults**) for the default SOCKS gateway failure mode. The "socks_gateway.fail_open()" on the next page property overrides the configured default.

Syntax

`socks_gateway(`*alias_list*`|no)`

where:

- *alias_list*: Send this request through the specified alias list. The appliance attempts to send this request through the specified gateways in the order specified by the list. It proceeds to the next gateway alias as necessary when the gateway is down, as determined by health checks.
- no: (Default behavior) Do not send this request through a SOCKS gateway. A forwarding host or ICP host may still be used, depending on those properties. If neither are set, the request is sent directly to the origin server. A setting of no overrides the default sequence defined in configuration.

Layer and Transaction Notes

- Layers: `<Forward>`
- Transactions: all, except administrator transactions

See Also

- Conditions: "socks.method=" on page 279, "socks.version=" on page 280
- Properties: "direct()" on page 396, "forward()" on page 415, "socks.accelerate()" on page 553, "socks.authenticate()" on page 554, "socks.authenticate.force()" on the previous page

socks_gateway.fail_open()

Controls whether the appliance terminates or continues to process the request if the specified SOCKS gateway or any designated backup or default cannot be contacted.

There is a global configuration setting (**Configuration > Forwarding > SOCKS Gateways > Global Defaults**) for the default SOCKS gateway failure mode. This property overrides the configured default.

Syntax

```
socks_gateway.fail_open(yes|no)
```

where:

- yes: Continue to process the request if the specified SOCKS gateway or any designated backup or default cannot be contacted. This may result in the request being forwarded through a forwarding host or ICP, or may result in the request going direct to the origin server.
- no: (Default behavior) Terminates the request if the specified SOCKS gateway or any designated backup or default cannot be contacted.

Layer and Transaction Notes

- Layers: `<Forward>`
- Transactions: all, except administrator transactions

See Also

- Conditions: "socks.method=" on page 279, "socks.version=" on page 280
- Properties: "direct()" on page 396, "forward()" on page 415, "socks.accelerate()" on page 553, "socks.authenticate()" on page 554, "socks.authenticate.force()" on page 556, "socks_gateway()" on the previous page

ssl.forward_proxy()

Determines whether SSL connections should be intercepted.

Syntax

```
ssl.forward_proxy(no|yes|stunnel|sips|https[,always|on_exception])
```

where:

- no: Tunnels the SSL connection without breaking encryption; no acceleration is provided.
- yes:Intercepts SSL traffic, and takes one of the following actions, depending on the actions in use:
  - If `force_protocol` is also used, the SSL proxy will hand the intercepted SSL traffic to its corresponding proxy; for example, if `force_protocol(https)` is used, all traffic is handed to the HTTPS forward proxy for further processing
  - If `detect_protocol` is used, but `force_protocol` is not, the SSL proxy will detect the HTTPS protocol after intercepting the SSL traffic, and hand it off as appropriate; if there is no appropriate proxy, the traffic will be tunneled using STunnel
  - If neither `detect_protocol` nor `force_protocol` is used, SSL traffic is intercepted, and tunneled using STunnel
- `stunnel`: Tunnels intercepted SSL traffic; if secure ADN is enabled, traffic is accelerated via byte caching.
- `sips`: Tunnels SIP (Session Initiation Protocol) traffic over SSL. detect_protocol(yes) must be set for this property to work.
- `https`: Intercepts SSL connections using the HTTPS Forward Proxy.
- `always`: Intercepts all SSL traffic that matches the intercept policy. This is the default behavior, so this argument is not required unless the policy is meant to override an earlier layer's `on_exception` action.
- `on_exception`: Only intercepts SSL traffic if policy execution results in an exception (typically a deny). Otherwise, the SSL traffic is tunneled.

Layer and Transaction Notes

- Layers: `<SSL-Intercept>`
- Transactions: SSL intercept

Example

Since interception is the default action for HTTPS traffic, the general usage model is to create exceptions for connections that need to be tunneled.

```
<ssl-intercept>
  server.certificate.hostname.category="Financial Services" ssl.forward_proxy(no)
```

ssl.forward_proxy.hostname()

Specify the hostname for the forged certificate that is used for SSL interception.

Syntax

`ssl.forward_proxy.hostname(`*`string`*`)`

The default value is "". The default behavior is to use the hostname from the original server certificate.

Layer and Transaction Notes

- Layers: `<SSL-Intercept>`
- Transactions: SSL intercept

Example

When the browser receives a server certificate signed by an unknown CA or with a hostname that does not match the URL hostname, it shows a security alert popup. This popup can be leveraged as an SSL level splashing mechanism. Various combinations of the ssl.forward_proxy.* properties can be used to force the Security Alert popup and provide additional information.

The security alert popup can be forced by a hostname mismatch or by using an unknown CA as follows:

```
<SSL-Intercept>
  ssl.forward_proxy.hostname("WE ARE WATCHING YOU") ssl.forward_proxy.issuer_keyring(new-private-ca) \
    ssl.forward_proxy.splash_text("This session is being monitored.") ssl.forward_proxy.splash_url
(http://example.com/ssl-intercept-policy.html)
```

ssl.forward_proxy.issuer_keyring()

Specify the CA keyring for signing the forged certificate that is used for SSL interception.

Syntax

`ssl.forward_proxy.issuer_keyring(auto|keyring_id|hsm_keyring())`

The default value is auto. The default behavior is to use the keyring specified in configuration by the SGOS#(config ssl) intercept CLI command.

Layer and Transaction Notes

- Layers: `<SSL-Intercept>`
- Transactions: SSL intercept

Example 1

When the browser receives a server certificate signed by an unknown CA or with a hostname that does not match the URL hostname, it shows a security alert popup. This popup can be leveraged as an SSL level splashing mechanism. Various combinations of the ssl.forward_proxy.* properties can be used to force the Security Alert popup and provide additional information.

The security alert popup can be forced by a hostname mismatch or by using an unknown CA as follows:

```
<SSL-Intercept>
  ssl.forward_proxy.hostname("WE ARE WATCHING YOU") ssl.forward_proxy.issuer_keyring(new-private-ca) \
    ssl.forward_proxy.splash_text("This session is being monitored.") ssl.forward_proxy.splash_url
(http://example.com/ssl-intercept-policy.html)
```

Example 2

Set the SSL Proxy to use the HSM keyring test-hsmkeyring1.

```
<SSL-Intercept>
  ssl.forward_proxy.issuer_keyring(test-hsmkeyring1)
```

ssl.forward_proxy.preserve_untrusted()

When an OCS presents a certificate to the appliance that is not signed by a trusted Certificate Authority (CA), the appliance can present the browser with an untrusted certificate that is signed by its untrusted issuer keyring. An SSL certificate warning message is displayed to the user, who can either ignore the warning and visit the website or cancel the request.

Syntax

```
ssl.forward_proxy.preserve_untrusted(auto|yes|no)
```

where:

- auto: (Default behavior) Uses the “preserve-untrusted” configuration setting on the appliance to determine whether untrusted certificate issuer should be preserved for a connection.
- yes: Preserve untrusted certificate issuer is enabled for the connection.
- no: Preserve untrusted certificate issuer is disabled for the connection.

Layer and Transaction Notes

- Layers: <SSL-Intercept>
- Transactions: SSL/HTTPS intercepted traffic

Example

Intercept the request without validating the server certificate on the appliance. Even if "server.certificate.validate()" on page 536 is set to yes, the appliance does not present an exception page to the user when a certificate or hostname error occurs. Instead, the browser performs server certificate validation and displays an SSL certificate warning to the user.

```
<SSL-Intercept>
  ssl.forward_proxy(yes) ssl.forward_proxy.preserve_untrusted(yes)
```

See Also

- Properties: "server.certificate.validate()" on page 536

ssl.forward_proxy.server_keyring()

Specify a static server certificate and keypair for use during SSL interception.

When an SSL connection is intercepted, the normal behavior is to dynamically generate a forged server certificate and keypair. The contents of this forged certificate are controlled by the .hostname, .splash_text, .splash_url and .issuer_keyring members of the ssl.forward_proxy family of properties. The ssl.forward_proxy.server_keyring property overrides this behavior, and allows you to specify a static certificate and keypair which will be used instead. It is normally only used for debugging.

The default value is no, which causes a forged certificate to be dynamically generated.

Syntax

`ssl.forward_proxy.server_keyring (no|`*`keyring_id`*`)`

where:

- no: (Default behavior) Dynamically generate a forged certificate
- *`keyring_id`*: Specify a static keyring ID.

Layer and Transaction Notes

- Layers: `<SSL-Intercept>`
- Transactions: SSL intercept

Example

Use the specified keyring for SSL interception.

```
<SSL-Intercept>
  ssl.forward_proxy.server_keyring(my_keyring)
```

ssl.forward_proxy.splash_text()

Specify informational text to be inserted into the forged certificate that is used for SSL interception.

Syntax

`ssl.forward_proxy.splash_text(`*`string`*`)`

The default value is "". The string argument is limited to 200 printable characters.

Layer and Transaction Notes

- Layers: `<SSL-Intercept>`
- Transactions: SSL intercept

Example

When the browser receives a server certificate signed by an unknown CA or with a hostname that does not match the URL hostname, it shows a security alert popup. This popup can be leveraged as an SSL level splashing mechanism. Various combinations of the ssl.forward_proxy.* properties can be used to force the Security Alert popup and provide additional information.

The security alert popup can be forced by a hostname mismatch or by using an unknown CA as follows:

```
<SSL-Intercept>
  ssl.forward_proxy.hostname("WE ARE WATCHING YOU") ssl.forward_proxy.issuer_keyring(new-private-ca) \
    ssl.forward_proxy.splash_text("This session is being monitored.") ssl.forward_proxy.splash_url
(http://example.com/ssl-intercept-policy.html)
```

ssl.forward_proxy.splash_url()

Specify an informational url to be inserted into the forged certificate that is used for SSL interception.

Syntax

`ssl.forward_proxy.splash_url(""|`*`url`*`)`

The default value is "".

Layer and Transaction Notes

- Layers: `<SSL-Intercept>`
- Transactions: SSL intercept

Example

When the browser receives a server certificate signed by an unknown CA or with a hostname that does not match the URL hostname, it shows a security alert popup. This popup can be leveraged as an SSL level splashing mechanism. Various combinations of the ssl.forward_proxy.* properties can be used to force the Security Alert popup and provide additional information.

The security alert popup can be forced by a hostname mismatch or by using an unknown CA as follows:

```
<SSL-Intercept>
  ssl.forward_proxy.hostname("WE ARE WATCHING YOU") ssl.forward_proxy.issuer_keyring(new-private-ca) \
    ssl.forward_proxy.splash_text("This session is being monitored.") ssl.forward_proxy.splash_url
(http://example.com/ssl-intercept-policy.html)
```

streaming.fast_cache()

Enables/disables fast-caching feature on Windows Media Client.

Effectively, setting yes enables the Windows Media Client to stream content faster than the bitrate of the content if this capability is supported in the streaming server. Setting no will cause the Windows Media Client to stream the content at the bitrate speed of the content.

Syntax

```
streaming.fast_cache(yes|no)
```

where:

- yes: (Default behavior) Proxy will advertise the com.microsoft.wm.fastcache token in the Supported header for non-local responses returned to a Windows Media Client if that token is included in the response from the streaming server. For local responses, the proxy will advertise the com.microsoft.wm.fastcache token in the Supported header.
- no: Proxy will not advertise the com.microsoft.wm.fastcache token in the Supported header for both local and non-local responses returned to a Windows Media Client.

Layer and Transaction Notes

- Layers: <Proxy>
- Transactions: all proxies

Example

Disable fast-caching for WM-RTSP traffic.

```
<Proxy>
  client.protocol=rtsp streaming.fast_cache(no)
```

streaming.rtmp.tunnel_encrypted()

Determines whether encrypted Flash traffic is tunneled or accelerated. By default, RTMPE and RTMPTE traffic is not tunneled; incoming data is decrypted, the connections are accelerated, and the outgoing data is encrypted. Because encryption is CPU intensive, you might want to write policy to turn it off. Or, if there are issues with accessing a specific site, you can write policy to tunnel the encrypted RTMP traffic to that site. If a connection is tunneled due to this type of policy, the Active Sessions Detail column will show Encrypted, tunneled by policy.

Syntax

```
streaming.rtmp.tunnel_encrypted(yes|no)
```

Layer and Transaction Notes

- Layers: `<Proxy>`
- Transactions: Flash

Example

RTMPE or RTMPTE connections to 10.12.13.14 will be tunneled (not accelerated).

```
<Proxy>
  url.address=10.12.13.14 streaming.rtmp.tunnel_encrypted(yes)
```

See Also

- *SGOS Administration Guide*, "Managing Streaming Media" chapter

streaming.transport()

Determines the upstream transport mechanism to be used for this streaming transaction. This setting is not definitive. The ability to use the specified transport mechanism depends on the capabilities of the selected forwarding host.

If a connection is encrypted (RTMPE or RTMPTE), the outgoing connection will also be encrypted, using the transport specified in the policy. RTMPE uses the streaming.transport(tcp) property and RTMPTE uses the streaming.transport(http) property. By changing the transport mechanism you can convert traffic from RTMPE to RTMPTE or vice versa.

> **Note:** This property is not applicable to the Smooth Streaming proxy.

Syntax

```
streaming.transport(auto|tcp|http)
```

where:

- `auto`: Use the default transport for the upstream connection, as determined by the originating transport and the capabilities of any selected forwarding host.
- `tcp`: Use TCP as the upstream transport mechanism.
- `http`: Use HTTP as the upstream transport mechanism.

Layer and Transaction Notes

- Layers: `<Forward>`
- Transactions: streaming

See Also

- Conditions:"bitrate=" on page 83, "live=" on page 192, "streaming.client=" on page 283, "streaming.content=" on page 284

supplier.allowed_countries()

Allows IP addresses based on geographical location; depending on the property values you specify, connections are allowed to all countries, no countries, or specific countries. If you allow connections to all countries or no countries in policy, you can optionally override that rule by specifying countries in conjunction with an allow or deny value.

During a connection attempt to a domain, the DNS server returns one or more IP addresses associated with the domain. The appliance filters the returned IP addresses against policy and skips any that are not allowed.

Thus, this property applies before the appliance attempts to connect to the Origin Content Server (OCS); the appliance only attempts to connect to an OCS if and when an IP address is allowed per the policy.

> **Note:** Because the connection must be made before this property applies, it is not useful for making policy decisions based on the response payload. To make decisions based on response payload, use the "supplier.country" on page 290 condition. See the **Examples** for "supplier.country" on page 290 for details.

Syntax

```
supplier.allowed_countries(all|deny-all|country_name,..)
supplier.allowed_countries.country_name(allow|deny)
supplier.allowed_countries[country_name_list](allow|deny)
```

where:

- `all`: (Default behavior) All countries are allowed unless overridden later in policy. If you do not specify supplier.allowed_countries(all), all countries are allowed unless overridden later in policy
- *`deny-all`*: All countries are denied unless overridden later in policy.
- *`country_name`*: Country name. If a country name includes punctuation or a space (such as United States), enclose the name in single quotation marks ('United States'). Alternatively, use the ISO two-letter code name, such as US for the United States.

  To see the list of countries in the geolocation database and their ISO code names, go to https://*IP_address:port*/Geolocation/Countries.
- `allow`: Specified countries are allowed and overrides any previous instances of supplier.allowed_countries(deny-all).
- `deny`: Sspecified countries are denied and overrides any previous instances of supplier.allowed_countries(all).

Layer and Transaction Notes

- Layers: `<Cache>`, `<Proxy>` , `<SSL>`
- Transactions: all

Example 1

Allow connections to all countries, with a subsequent override that denies access to China and Hong Kong for a given user group.

```
<Proxy>
  supplier.allowed_countries(all)

define condition Level_1
  realm=SAML1 group="Level1"
end condition Level_1

<Proxy>
  condition=Level_1 supplier.allowed_countries[CN,HK](deny)
```

Example 2

Deny connections to all countries, with a subsequent override that allows access to the United States during a specific time of day.

```
<Proxy>
  supplier.allowed_countries(deny-all)

define condition Lunch_Break
  time=(1200..1259)
end condition Lunch_Break

<Proxy>
  condition=Lunch_Break supplier.allowed_countries.US(allow)
```

See Also

- Conditions: "supplier.country" on page 290

tenant()

Used in a <Tenant> layer in the Landlord policy slot in multi-tenant policy deployments to set the tenant slot's policy to be used for the transaction.

Syntax

`tenant(tenant_ID)`

Layer and Transaction Notes

- Layers: <Tenant>
- Commits after authentication, allowing the widest range of conditions to guard the property.

  If any policy gestures in a tenant policy slot evaluate before tenant() sets the correct tenant for the transaction, then those policy gestures in the tenant policy slot will not be applied to the transaction.

Example

Use the specified tenant slot for the transaction.

```
<Tenant>
  url=example.com tenant(tenant1)
```

See Also

- *Multi-Tenant Policy Deployment Guide*

tenant.connection()

Used in a <Tenant> layer in the Landlord policy slot in multi-tenant policy deployments to set the tenant slot's policy to be used for the transaction.

Syntax

```
tenant.connection(tenant_ID)
```

Layer and Transaction Notes

- Layers: <Tenant>
- Commits early in the transaction, so the property can only be guarded by a small set of conditions:
  - "client.address=" on page 87
  - "client.host=" on page 106
  - "client.interface=" on page 109
  - "client.interface.routing_domain=" on page 110
  - "proxy.address=" on page 198
  - "proxy.port=" on page 200
  - "service.name=" on page 276

Example

Use the tenant1 policy for specified client address.

```
<Tenant>
  client.address=10.0.1.1 tenant.connection(tenant1)
```

See Also

- *Multi-Tenant Policy Deployment Guide*

tenant.request_url()

Used in a <Tenant> layer in the Landlord policy slot in multi-tenant policy deployments to set the tenant slot's policy to be used for the transaction.

Syntax

`tenant.request_url(tenant_ID)`

Layer and Transaction Notes

- Layers: <Tenant>
- Commits before authentication, so this property can be used when per-tenant authentication policy is used in the tenant policy. This property commits later than "terminate_connection()" on the facing page and guards against more conditions.

Example

Use the specified tenant slot for the transaction.

```
<Tenant>
  url=example.com tenant.request_url(tenant1)
```

See Also

- *Multi-Tenant Policy Deployment Guide*

terminate_connection()

Drop the connection rather than return the exception response. The yes option terminates the connection instead of returning the response.

Syntax

```
terminate_connection(yes|no)
```

where:

- yes: Terminate the connection instead of returning the response.
- no: (Default behavior) Do not terminate the connection; return exception instead.

Layer and Transaction Notes

- Layers: <Exception>
- Transactions: HTTP, including HTTP/2 streams

trace.destination()

Used to change the default path to the trace output file. By default, policy evaluation trace output is written to an object in the cache accessible using a console URL of the following form:

https://*appliance_IP_address*:8081/Policy/Trace/path

Syntax

`trace.destination(path)`

where:

- *path*: By default, the path is default_trace.html. You can change *path* to a filename or directory path, or both. If only a directory is provided, the default trace filename is used.

Layer and Transaction Notes

- Layers: <Admin>, <Cache>, <Diagnostic>, <DNS-Proxy>, <Exception>, <Forward>, <Proxy> , <SSL>, <SSL-Intercept>
- Transactions: all

Example 1

Change directory location of trace output file to https://*appliance_IP_address*:8081/Policy/Trace/test/default_trace.html

```
<Proxy>
  trace.destination(test/)
```

Example 2

Change trace output file location to https://appliance_IP_address:8081/Policy/Trace/test/phase_2.html.

```
<Proxy>
  trace.destination(test/phase_2.html)
```

See Also

- Properties: "trace.request()" on page 579
- *ProxySG Log Fields and CPL Substitutions Reference*: https://techdocs.broadcom.com/content/broadcom/techdocs/us/en/symantec-security-software/web-and-network-security/proxysg/7-2/proxysg-log-fields-and-substitutions.html

trace.diagnostic()

This property is generated when `define probe` is compiled and is included in policy traces. You cannot write CPL with this property. See "define probe" on page 648 for details.

trace.header()

Specifies whether unlimited, full header trace output is generated for the current request.

By default, trace output is written to an object accessible using the following console URL:

https://*appliance_IP_address*:8081/Policy/Trace/default_trace.html

The trace output location can be controlled using "trace.destination()" on page 576.

Syntax

```
trace.header(yes|no)
```

where:

- yes: Generate full trace for the current request.
- no: (Default behavior) Limit header trace output to 2k. Header data beyond 2k is truncated.

Layer and Transaction Notes

- Layers: <Admin>, <Cache>, <Diagnostic>, <DNS-Proxy>, <Exception>, <Forward>, <Proxy> , <SSL>, <SSL-Intercept>
- Transactions: all

Example

;Generate full header trace details when a specific URL is requested.

```
<Proxy>
  url=//www.example.com/help trace.header(yes)
```

See Also

- Properties: "trace.destination()" on page 576, "trace.request()" on the facing page

trace.request()

Determines whether detailed trace output is generated for the current request. Trace output is generated at the end of a request, and includes request parameters, property settings, and the effects of all actions taken. Output tracing can be set conditionally by creating a rule that combines this property with conditions such as "url=" on page 298 or "client.address=" on page 87.

By default, trace output is written to an object accessible using the following console URL:

https://*appliance_IP_address*:8081/Policy/Trace/default_trace.html

The trace output location can be controlled using the "trace.destination()" on page 576 property.

> **Note:** Tracing is best used temporarily, such as for troubleshooting; the "log_message()" on page 608 action is best for on-going monitoring.

Syntax

```
trace.request(yes|no)
```

where:

- yes: Generate full trace details for the current request.
- no: (Default behavior) Do not generate trace output.

Layer and Transaction Notes

- Layers: <Admin>, <Cache>, <Diagnostic>, <DNS-Proxy>, <Exception>, <Forward>, <Proxy> , <SSL>, <SSL-Intercept>
- Transactions: all

Example

Generate trace details when a specific URL is requested.

```
<Proxy>
 url=//www.example.com/confidential trace.request(yes)
```

See Also

- Properties: "trace.destination()" on page 576

trace.session()

If set to `yes`, all transactions that terminate in the current session are traced. The default behavior is `no`.

Using `trace.session(yes)` in an `<SSL-Intercept>` layer enables tracing for the SSL intercept transaction, the SSL tunnel transaction, and (if intercepted) any and all HTTPS forward proxy transactions spawned on the session.

Syntax

```
trace.session(yes|no)
```

Layer and Transaction Notes

- Layers: `<Admin>`, `<Cache>`, `<Diagnostic>`, `<DNS-Proxy>`, `<Exception>`, `<Forward>`, `<Proxy>` , `<SSL>`, `<SSL-Intercept`, `<Tenant>`
- Transactions: all

Example

Generate trace details for sessions that contain the specified domain.

```
<SSL-Intercept>
  url.domain=example.com trace.session(yes)
```

transform.data_type()

Specifies an override for the parser used in URL_rewrite, active_content, and javascript transform actions for HTTP response data.

Details

The appliance uses the value declared in the Content-Type header to automatically select the appropriate parser, (text or HTML) to use when invoking specific transform policy actions.

The transform policy action invokes three different transformers: active_content, javascript and url_rewrite. Each transformer is used in policy with a definition: "define active_content" on page 630, "define javascript" on page 644, "define url_rewrite" on page 662.

Both active_content and javascript transformers allow you to modify an HTML file. active_content allows you to add or replace active content in an ASX or HTML file, while javascript allows you to add JavaScript to an HTML file.

"define url_rewrite" on page 662 is different, as it is not restricted to a specific file type. Rather, this transformation type can work in two ways:

- When a "define url_rewrite" on page 662 policy uses the HTML parser, HTML and XHTML content is searched for valid HTML tags containing relative or absolute URLs. If the matched URL exists outside of a valid HTML tag, it will not be transformed.
- "define url_rewrite" on page 662 can also transform any other text based file using the text parser. However, each instance of the matched absolute URL will be transformed, regardless if it's contained within an HTML tag. The text parser is unable to modify relative URLs.

Reverse proxy portal deployments commonly use a url_rewrite transform action to rewrite the links embedded in Web pages from internal to external addresses. If the Content-Type header value is declared as a type other than HTML, the text parser will be used instead of the HTML parser. As a result, the transform action will only look for the absolute form of the URLs defined in policy. Since HTML pages typically use the relative form of the URL, the page will not be transformed correctly and the site will appear broken to external users.

To adjust for this disparity, you can either correct the Web server hosting the content to properly identify the type, or you can use the "transform.data_type()" above policy gesture to specify the preferred transform parser to be used.

Syntax

```
transform.data_type(html|text|none|default)
```

where:

- `html`: Specifies to use the HTML transform parser for transform actions for content with HTML tags, such as HTML or XHTML.
- `text`: Specifies to use the text transform parser to alter textual content such as Javascript, JSON, XML, CSS.

- `default`: Specifies to use the default transformer identification, based on the content-type HTTP response header.
- `none`: No transformer is used.

Layer and Transaction Notes

- Layers: `<Proxy>`
- Transactions: HTTP response data affected by a "transform" on page 622 action in policy
- JavaScript within an HTML file and active content each require that the parser has knowledge of HTML tags. As such, if the text parser is used on these content types, "transform" on page 622 policy will not modify the content.

Example

Specify the override for the parser used in URL_rewrite for responses from the specified URL.

```
define url_rewrite my_rewrite
  rewrite_url_prefix "http://portal.example.com/host42/" "http://host42.example.com/resource"
end

define action my_action
  transform my_rewrite
end

<Proxy>
  url=http://portal.example.com/ action.my_action(yes) transform.data_type(html)
```

See Also

- Conditions: "request.header.header_name=" on page 219, "request.header.header_name.address=" on page 221, "request.x_header.header_name=" on page 241, "request.x_header.header_name.address=" on page 242, "response.header.header_name=" on page 248, "response.x_header.header_name=" on page 256, "server_url=" on page 269
- Definitions: "define active_content" on page 630, "define javascript" on page 644, "define url_rewrite" on page 662
- Actions: "transform" on page 622

trust_destination_ip()

Allow the appliance to honor client's destination IP when it intercepts client requests transparently.

The appliance will trust the client provided destination IP and not do the DNS lookup for the HOST value in appropriate cases. This feature will not apply (that is, existing behavior will be preserved) if the appliance:

- Receives the client requests in explicit proxy deployment cases.
- Has forwarding rules configured for the given HOST value.
- Will connect upstream on SOCKS.
- Will connect upstream using ICP.

Syntax

```
trust_destination_ip(yes|no)
```

Layer and Transaction Notes

- Layers: <Proxy>
- Transactions: all proxies

Example

Disable trusting destination IP.

```
<Proxy>
  proxy.address=10.10.167.0/24 trust_destination_ip(no)
```

ttl()

Sets the time-to-live (TTL) value of an object in the cache, in seconds. Upon expiration, the cached copy is considered stale and will be re-obtained from the origin server when next accessed. However, this property has an effect only if the following HTTP command line option is enabled: `Force explicit expirations: Never serve after.`

If the above option is not set, the appliance's freshness algorithm determines the time-to-live value.

> **Note:** `advertisement(yes)` overrides any `ttl()` value.

Syntax

`ttl(`*`seconds`*`)`

where:

- *seconds*: An integer, specifying the number of seconds an object remains in the cache before it is deleted. The maximum value is 4294967295, or about 136 years. The default value is specified by configuration.

Layer and Transaction Notes

- Layers: `<Cache>`

Example

Delete the specified cached objects after 30 seconds.

```
<Cache>
  url=//www.example.com/dyn_images ttl(30)
```

See Also

- Properties:"advertisement()" on page 343, "cache()" on page 374

ua_sensitive()

When set to yes, modifies caching behavior by declaring that the response for a given object is expected to vary based on the user agent used to retrieve the object.

Using ua_sensitive(yes) has the same effect as cache(no).

> **Note:** Remember that any conflict among CPL property settings is resolved by CPL evaluation logic, which uses the property value that was last set when evaluation ends.

Syntax

```
ua_sensitive(yes|no)
```

The default behavior is no.

Layer and Transaction Notes

- Layers: <Cache>
- Transactions: proxy transactions, which execute both <Cache> and <Proxy> layers. Does not apply to FTP over HTTP transactions

See Also

- Properties: "advertisement()" on page 343, "always_verify()" on page 345, "bypass_cache()" on page 373, "cache()" on page 374, "cookie_sensitive()" on page 390, "delete_on_abandonment()" on page 391, "direct()" on page 396, "dynamic_bypass()" on page 401, "force_cache()" on page 407, "pipeline()" on page 500, "refresh()" on page 503, "ttl()" on the previous page

user.login.log_out()

Log out the current user from the current IP address. This property is used to log out a user from the current IP address.

When this property is executed, the current login for the user is logged out. The user will need to re-authenticate at this IP address before future transactions can proceed.

Syntax

```
user.login.log_out(yes|no)
```

The default behavior is `yes`.

Layer and Transaction Notes

- Layers: `<Admin>`, `<Proxy>`
- Transactions: proxy, administrator

Example

Log out the user whenever they visit the log out page.

```
<Proxy>
  url="http://company.com/log_out.html" user.login.log_out(yes)
```

user.login.log_out_other()

Log out the current user from logins other than the current IP address. This property is used to log out any other logins of the user on IP addresses other than the current IP address. When this property is executed, the all logins of the user on IP address other than the current IP address are logged out. The user will need to re-authenticate at the other IP address before future transactions at those IP addresses can proceed.

Syntax

```
user.login.log_out_other(yes|no)
```

The default behavior is yes.

Layer and Transaction Notes

- Layers: <Admin>, <Cache>, <DNS-Proxy>, <Exception>, <Forward>, <Proxy> , <SSL>, <SSL-Intercept>
- Transactions: HTTP proxy

Example

Log out the user from other workstations if they are logged in more than once.

```
<Proxy>
  user.login.count=2.. user.login.log_out_other(yes)
```

user.realm.surrogate()

Specifies a surrogate realm for user authentication.

You can use this property in conjunction with the "realm=" on page 208 condition. A realm specified in this property is used for surrogate authentication in addition to any other realms specified in "realm=" on page 208 tests in policy.

For example, if Symantec Web Isolation is deployed upstream, the proxy can authenticate users based on identity and group membership defined in Web Isolation. In this case, you would configure a policy substitution realm on the proxy and include the "user.realm.surrogate()" above property in policy. Refer to the *SGOS Administration Guide* for details on creating substitution realms. For details on Web Isolation integration, refer to *Symantec Threat Isolation Platform Guide for Administrators*:

https://techdocs.broadcom.com/content/broadcom/techdocs/us/en/symantec-security-software/web-and-network-security/web-isolation/1-0/web-isolation-administration-guides.html

> **Note:** The VPM Group and User objects require a realm selection; as a result, the generated CPL for these objects include the "realm=" on page 208 condition as follows:
> - `realm=realm_name group="group_name"`
> - `realm=realm_name group="user_name"`
>
> If policy includes both the user.realm.surrogate() property and a VPM layer containing the **Group** object or **User** object, the surrogate realm is used in the group or user authentication.

Syntax

`user.realm.surrogate(`*`isolation_realm_name`*`|no)`

where:

- *`surrogate_realm_name`*: Surrogate authentication realm.
- no: Do not use a surrogate authentication realm.

Layer and Transaction Notes

- Layers: `<Proxy>`
- Transactions: all proxies

Example 1

The following example shows how to apply policy to requests from a surrogate realm.

```
; layer 1
<Proxy>
  user.realm.surrogate(surrogate_realm)
```

```
  ...
; layer 2
<Proxy> realm=corporate
  category=gambling exception(content_filter_denied)
```

Because layer 1 specifies the surrogate_realm, the proxy evaluates layer 2 as if the layer guard were realm=(corporate,surrogate_realm) and applies the content filtering policy to users in those realms.

Example 2

The following example shows how to require surrogate realm authentication for specific request destinations.

```
define subnet isolation_servers
  <list_of_IP_addresses>
end

define subnet internal_servers
  <list_of_IP_addresses>
end

; layer 1
<Proxy>
  client.address=isolation_servers authenticate.realm.surrogate(isolation_realm)

; layer 2
<Proxy>
  url.address=internal_servers authenticate.realm.surrogate(no)

; layer 3
<Proxy>
  user=joan realm=LDAP_realm ALLOW
```

In this example, `realm=LDAP_realm` is evaluated as `realm=(LDAP_realm, isolation_realm)`. If the user requests an address in the internal_server list, she must authenticate in the LDAP_realm per layer 2. If she requests any other address, layer 3 specifies that the transaction is allowed as long as she authenticates with isolation_realm.

See Also

- Conditions: "realm=" on page 208

webpulse.categorize.mode( )

Determines how dynamic categorization will be performed.

Syntax

```
webpulse.categorize.mode(none|realtime|background|default)
```

where:

- none: Suppress dynamic categorization for this request.
- realtime: Perform dynamic categorization in real-time; the request waits until the dynamic category is available from the service.
- background: Perform dynamic categorization in the background; the request is assigned the category 'pending', and continues to be processed without delay. Later, when the categorization service responds, the dynamically-determined category for the requested object is saved so that future requests for the object can make use of it.
- default: Restore the setting to the configuration-specified default (to undo the effect of a previous policy layer).

Layer and Transaction Notes

- Layers:<Cache>, <Exception>
- Transactions: all

Example

This example illustrates how the property is used to control dynamic categorization.

```
; Do not dynamically categorize this domain
<Cache>
  url.domain=symantec.com webpulse.categorize.mode(none)

; Serve this domain and categorize in the background
<Cache>
  url.domain=yahoo.com webpulse.categorize.mode(background)

; Categorize all other requests in real time
<Cache>
  webpulse.categorize.mode(realtime)
```

webpulse.categorize.send_headers()

Determines which HTTP headers in the client request should be sent to WebPulse.

Syntax

```
webpulse.categorize.send_headers(yes|no|auto)
webpulse.categorize.send_headers(header_name,..)
webpulse.categorize.send_headers.header_name(yes|no)
webpulse.categorize.send_headers[header_name,..](yes|no)
```

where:

- *header_name*: Header name, such as Referer, User-agent.
- auto: This value is set in the appliance configuration. If dynamic rating service information handling is enabled (via the command `#(config bluecoat) service send-request-info enable`), send all headers. If the setting is disabled, send no headers

Layer and Transaction Notes

- Layers: <Cache>, <Exception>
- Transactions: all

Example

Send all HTTP headers for any request.

```
<Cache>
  webpulse.categorize.send_headers(yes)
```

webpulse.categorize.send_url()

Determines which information contained in the URL should be sent to WebPulse.

Syntax

```
webpulse.categorize.send_url(full|path|host)
```

where:

- `full`: The entire URL string
- `path`: The URL minus any query string
- `host`: Only the host information contained in the URL

The default value is set in the appliance configuration. If dynamic rating service information handling is enabled (via the command `#(config bluecoat)` **`service send-request-info enable`**), the default is `full`. If the setting is disabled, the default is `path`.

Layer and Transaction Notes

- Layers: `<Cache>`, `<Exception>`
- Transactions: all

Example

Control information submitted to WebPulse.

```
; Send only hostname to WebPulse for this domain
<Cache>
  url.domain=symantec.com webpulse.categorize.send_url(host)

; Send path information in the URL to WebPulse
<Cache>
   url.domain=yahoo.com webpulse.categorize.send_url(path)

; Send full URL for all other requests
<Cache>
  webpulse.categorize.send_url(full)
```

webpulse.notify.malware()

Provides the ability to disable malware notification to WebPulse.

When the appliance sends a URL to Webpulse for categorization and WebPulse identifies the URL as malware, by default the appliance reports this malware rating to Webpulse so that the master WebPulse database can be updated. Updating the master database helps in two ways – it eliminates the need for dynamic categorization requests for that URL, and subsequent database updates for all WebFilter users (the WebPulse community) will include the malware rating for the URL. If you do not want to share the result of the URL categorization with the WebPulse community, you can disable malware notification and the appliance will not resport the information back to the WebPulse master database.

Syntax

```
webpulse.notify.malware(yes|no)
```

where:

- yes: (Default behavior) Send a notification to WebPulse.
- no: Disable malware notification to WebPulse.

Layer and Transaction Notes

- Layers: `<Cache>`, `<Exception>`
- Transactions: all transactions subject to URL categorization

Example

Disable malware notification to WebPulse:

```
<Cache>
  webpulse.notify.malware(no)
```

Actions

An action takes arguments and is wrapped in a user-named action definition block. When the action definition is called from a policy rule, any actions it contains operate on their respective arguments. Within a rule, named action definitions are enabled and disabled using the action( )property.

Actions take the following general form:

```
action(argument1, ...)
```

An action block is limited to the common subset among the allowed layers of each of the actions it contains. Actions appear only within action definitions. They cannot appear in `<Admin>` layers.

Argument Syntax

The allowed syntax for action arguments depends on the action.

- String: A string argument must be quoted if it contains whitespace or other special characters, such as log_message ("Access alert").
- Enumeration: Actions such as "delete()" on page 598 use as an argument a token specifying the transaction component on which to act. For example: a header name such as `.Referer`.
- Regular expression: Several actions take regular expressions. For more information about writing regular expressions, see "Regex Reference" on page 703.
- Variable substitution: The quoted strings in some action arguments can include variable substitution substrings. These include the various versions of the replacement argument of the "redirect()" on page 611 and "rewrite()" on page 615 actions, and the string argument in the "append()" on the facing page, "log_message()" on page 608, and set(header, string) actions. A variable substitution is a substring of the form:

  ```
  $(name)
  ```

  where `name` is one of the allowed substitution variables.

For a complete list of substitutions, refer to the *ProxySG Log Fields and CPL Substitutions Reference*: https://techdocs.broadcom.com/content/broadcom/techdocs/us/en/symantec-security-software/web-and-network-security/proxysg/7-2/proxysg-log-fields-and-substitutions.html

append()

Appends a new component to the specified header.

Syntax

`append(header,string)`

where:

- *header*: Header specified using the following form. For a list of recognized headers, including headers that support field repetition, see "Recognized HTTP Headers" on page 700
  - *request.header.header_name*—Identifies a recognized HTTP request header.
  - *response.header.header_name*—Identifies a recognized HTTP response header.
  - *request.x_header.header_name*—Identifies any request header, including custom headers.
  - *response.x_header.header_name*—Identifies any response header, including custom headers.
- *string*: Quoted string that can optionally include one or more variable substitutions.

Layer and Transaction Notes

- Layers: <Cache>,<Proxy>
- Transactions: HTTP proxy

Example

Append the specified string to the X-Forwarded-For header when users access certain sites.

```
<Proxy>
  condition=some_sites action.append_X-Forwarded_For(yes)

define condition some_sites
  list_of_url.domain_conditions
end

define action append_X-Forwarded_For
  append(request.header.X-Forwarded-For, "originating IP")
end action Append_X-Forwarded_For
```

See Also

- Conditions: "request.header.header_name=" on page 219, "request.header.header_name.address=" on page 221, "request.x_header.header_name=" on page 241, "request.x_header.header_name.address=" on page 242, "request.x_header.header_name=" on page 241, "response.x_header.header_name=" on page 256
- Actions: "delete()" on page 598, "delete_matching()" on page 600, "rewrite()" on page 615, "set()" on page 619
- *ProxySG Log Fields and CPL Substitutions Reference*: https://techdocs.broadcom.com/content/broadcom/techdocs/us/en/symantec-security-software/web-and-network-security/proxysg/7-2/proxysg-log-fields-and-substitutions.html

authenticate.persist_cookies()

Controls cookie persistence during user authentication.

Syntax

```
authenticate.persist_cookies(auto|no|yes)
```

where:

- auto: Cookie persistency value configured in the realm will be used.
- no: Session cookie will be used in authentication in this transaction.
- yes: Persistent cookie will be used in authentication in this transaction.

Layer and Transaction Notes

- Layers: <Proxy>

Example

Persistent cookies are used in requests to 'mycompany' web pages. Users do not have to log in again on other 'mycompany' web pages after they are already authenticated.

```
<Proxy>
  url.host.regex=mycompany authenticate.persist_cookies(yes)
```

delete()

Deletes all components of the specified header.

Syntax

```
delete(header)
```

where:

- header: Header specified using the following form. For a list of recognized headers, see "Recognized HTTP Headers" on page 700.
  - *request.header.header_name*—Identifies a recognized HTTP request header.
  - *response.header.header_name*—Identifies a recognized HTTP response header.
  - *request.x_header.header_name*—Identifies any request header, including custom headers.
  - *response.x_header.header_name*—Identifies any response header, including custom headers.
  - *exception.response.header.header_name*—Identifies a recognized HTTP response header from the exception response.

Layer and Transaction Notes

- Layers: Use with *exception.response.header.header_name* in <Proxy> or <Exception> layers; use with request or response headers in <Proxy> or <Cache> layers.
- Transactions: HTTP proxy

Example

For the test.com domain, delete the Referer request header and log the action taken.

```
<Proxy>
  url.domain=test.com action.DeleteReferer(yes)

define action DeleteReferer
  log_message("Referer header deleted: $(.Referer)")
  delete(request.header.Referer)
end
```

See Also

- Conditions: "request.header.header_name=" on page 219, "request.header.header_name.address=" on page 221, "request.x_header.header_name=" on page 241, "request.x_header.header_name.address=" on page 242, "request.x_

header.header_name=" on page 241, "response.x_header.header_name=" on page 256

- Actions: "append()" on page 595, "delete_matching()" on the next page, rewrite(*header,regex_pattern,replacement_component*), "set()" on page 619

delete_matching()

Deletes all components of the specified header that contain a substring matching a regular-expression pattern.

Syntax

```
delete_matching(header,regex_pattern)
```

where:

- *header*: Header specified using the following form. For a list of recognized headers, including headers that support field repetition, see "Recognized HTTP Headers" on page 700
  - *request.header.header_name*–Identifies a recognized HTTP request header.
  - *response.header.header_name*–Identifies a recognized HTTP response header.
  - *request.x_header.header_name*–Identifies any request header, including custom headers.
  - *response.x_header.header_name*–Identifies any response header, including custom headers.
  - *regex_pattern*: Quoted regular-expression pattern. For more information, see "Regex Reference" on page 703.
- *string*: Quoted string that can optionally include one or more variable substitutions.

Layer and Transaction Notes

- Layers: <Cache>, <Proxy>

Example

Delete the specified component of the specified header

```
define action deletecookie
 delete_matching(request.header.cookie, "^SMSESSION=.*")
end

<Proxy>
 url.domain=sample_site action.deletecookie(yes)
```

See Also

- Conditions: "request.header.header_name=" on page 219, "request.header.header_name.address=" on page 221, "request.x_header.header_name=" on page 241, "request.x_header.header_name.address=" on page 242, "request.x_header.header_name=" on page 241, "response.x_header.header_name=" on page 256

- Actions: "append()" on page 595, "delete()" on page 598, rewrite(*header,regex_pattern,replacement_component*), "set()" on page 619
- *ProxySG Log Fields and CPL Substitutions Reference*: https://techdocs.broadcom.com/content/broadcom/techdocs/us/en/symantec-security-software/web-and-network-security/proxysg/7-2/proxysg-log-fields-and-substitutions.html

diagnostic.stop(pcap)

> **Tip:** This action is not to be confused with the <Diagnostic> layer, which supports this policy action as well as other gestures that are used to include diagnostic information about the transaction.

Stops a running packet capture (PCAP) when a policy condition matches user transaction data. While troubleshooting issues, you may find that there is too much traffic to distinguish one request from another. PCAP filters help, but it can be difficult to stop the capture before the PCAP file size limitation is reached and still gather useful information. This policy action be used with unique policy conditions to stop the PCAP when required.

Start a packet capture with your desired filters, direction, and interface:

- In the Management Console (**Statistics > Advanced > Packet Capture > Start Packet Capture**)
- With CLI command **#pcap start**

Define unique conditions for testing to ensure that the desired information is included in the packet capture.

To save the completed capture, browse to https://*appliance_IP_address*:8082/PCAP/bluecoat.cap

Syntax

```
diagnostic.stop(pcap)
```

Layer and Transaction Notes

- Layers: <Admin>, <Cache>, <Diagnostic>, <DNS-Proxy>, <Exception>, <Forward>, <Proxy> , <SSL>, <SSL-Intercept>
- Transactions: all

Example

A user is reporting trouble authenticating to the domain. The following policy will stop the running PCAP when the test user triggers an authentication exception.

```
<Exception>
  user.authentication_error=(any) diagnostic.stop(pcap)
```

iterate()

Binds a user-defined label to policy rules for each iterator value.

Syntax

```
iterate(header)
 policy_rules
 ...
end
```

where:

- *header*: Header that the appliance recognizes and which uses the specified form.

Layer and Transaction Notes

- Layers: <Admin>, <Cache>, <DNS-Proxy>, <Exception>, <Forward>, <Proxy> , <SSL>, <SSL-Intercept>

Example

Delete client cookies with “Sample” prefix.

```
define action DeleteSampleCookies
  iterate(.Cookie)
    iterator.prefix="Sample" iterator.delete()
  end
end

<Proxy>
  action.DeleteSampleCookies(yes)
```

See Also

- Conditions: "iterator=" on page 186
- Properties: "log.rewrite.field_id()" on page 491
- Actions: "iterator.append()" on the next page, "iterator.delete()" on page 605

iterator.append()

Appends a new iterator value to the HTML header.

Syntax

`iterator.append(`*`string`*`)`

where:

- *`string`*: Quoted string that can optionally include one or more variable substitutions.

This method is not supported when iterating over a sub-value of an individual header, such as when a delimiter is specified in the iterate block, such as:

```
iterate(.Cookie)
  iterator.append("CookieName=CookieValue")
end
```

Layer and Transaction Notes

- Layers:<Cache>, <Proxy>

Example

Sign all cookies and add the specified value to the header.

```
define action sign_all
  iterate(response.header.Set-Cookie)
    iterator.append("$(iterator:rewrite(([^=]*)=([^;]*)(.*),BCSIG_$(1)=$(2:hmac)$(3)))")
  end
end

<Proxy>
  server_url.domain=x.com/ action.sign_all(yes)
```

iterator.delete()

Removes the `iterator` value from the HTML header.

Syntax

```
iterator.delete()
```

> **Note:** This method is not supported when iterating over a sub-value of an individual header, such as when a delimiter is specified in the "iterate()" on page 603 block, as in the following example:
>
> ```
> iterate(.Cookie)
>  iterator.delete();
> end
> ```

Layer and Transaction Notes

- Layers: `<Cache>`, `<Proxy>`

Example

See the example in "iterate()" on page 603.

iterator.rewrite()

Modifies cookie attributes inside the header being iterated over.

Syntax

`iterator.rewrite(`*`regex_pattern`*`,`*`replacement_string`*`)`

where:

- *`regex_pattern`*: Quoted regular expression pattern that is compared with cookie values in a header. If no value matches the *`regex_pattern`*, the pattern being iterated over is not modified.
- *`replacement_string`*: Quoted string that includes one or more variable substitutions. The string replaces the entire portion of the header that matches the *`regex_pattern`*. For more information, refer to the *ProxySG Log Fields and CPL Substitutions Reference*: https://techdocs.broadcom.com/content/broadcom/techdocs/us/en/symantec-security-software/web-and-network-security/proxysg/7-2/proxysg-log-fields-and-substitutions.html.

Layer and Transaction Notes

- Layers: `<Cache>`, `<Proxy>`

Example

Adds the Secure and HttpOnly attributes to a cookie header, and the expiration date of the cookie, if one exists, being set to midnight. Symantec recommends that you use the Secure and HttpOnly cookie attributes whenever possible.

```
define action add_secure_and_http_attributes_to_cookies
  iterate(response.header.Set-Cookie)
    ; Add both attributes if both are missing
    iterator.regex=!"; *Secure" iterator.regex=!"; *HttpOnly" iterator.rewrite
(".*","$(0);Secure;HttpOnly")
    ; If only Secure is missing,       add that
    iterator.regex=!"; *Secure" iterator.rewrite(".*","$(0);Secure")
    ; If only HttpOnly is      missing, add that
    iterator.regex=!"; *HttpOnly" iterator.rewrite(".*","$(0);HttpOnly")
  end
end

; If the cookie contains an expiration date, we will change it to expire tonight at midnight
; Cookies with no expiration date set will be unaffected
define action if_expiry_set_to_midnight
  iterate(response.header.Set-Cookie)
    iterator.regex="; *expires=" iterator.rewrite("(.*); *expires=([^;]*)(.*)",
"$(1)$(3);expires=$(cookie_date:next_date(00:00))")
   end
end
```

```
<Proxy>
  action.add_secure_and_http_attributes_to_cookies(yes) action.if_expiry_set_to_midnight(yes)
```

See Also

- Properties: "iterate()" on page 603

log_message()

Writes the specified string to the event log.

Events generated by `log_message( )` are viewed by selecting the Policy messages event logging level in the Management Console.

> **Note:** This is independent of access logging.

Syntax

`log_message(`*`string`*`)`

where:

- *`string`*: Quoted string that can optionally include one or more substitution variables.

Layer and Transaction Notes

- Layers: `<Admin>`, `<Cache>`, `<Diagnostic>`, `<DNS-Proxy>`, `<Exception>`, `<Forward>`, `<Proxy>` , `<SSL>`, `<SSL-Intercept>`

Example

Log the action taken, and include the original value of the Referer header.

```
define action DeleteReferer
  log_message("Referer header deleted: $(.Referer)")
  delete(.Referer)
end
```

See Also

- Properties: "access_log()" on page 333, "log.rewrite.field_id()" on page 491, l"log.suppress.field_id( )" on page 493
- Actions: "notify_email()" on the facing page, "notify_snmp()" on page 610
- *ProxySG Log Fields and CPL Substitutions Reference*: https://techdocs.broadcom.com/content/broadcom/techdocs/us/en/symantec-security-software/web-and-network-security/proxysg/7-2/proxysg-log-fields-and-substitutions.html

notify_email()

Sends an e-mail notification to the list of recipients specified in the Event Log mail configuration. The sender of the e-mail appears as *Primary_appliance_IP_address - configured_appliance_hostname*. You can specify multiple notify_email() actions, which may result in multiple mail messages for a single transaction.

When the transaction terminates, The e-mail is sent to the list of recipients specified in the Event Log mail configuration.

Syntax

```
notify_email(subject, body)
```

where:

- *subject*: Quoted string that can optionally include one or more substitution variables.
- *body*: Quoted string that can optionally include one or more substitution variables.

Layer and Transaction Notes

- Layers: <Admin>, <Cache>, <Diagnostic>, <Exception>, <Forward>, <Proxy> , <SSL>, <SSL-Intercept>
- Transactions: HTTP proxy

Example

E-mail administrators when users access the specified restricted sites.

```
define condition restricted_sites
  url.domain=a_very_bad_site
  ...
end

<Proxy>
  condition=restricted_sites action.email_notify_restricted(yes)

define action email_notify_restricted
  notify_email("restricted: ", "$(client.address) accessed URL: $(url)")
end
```

See Also

- Actions: "log_message()" on the previous page, "notify_snmp()" on the next page
- *ProxySG Log Fields and CPL Substitutions Reference*: https://techdocs.broadcom.com/content/broadcom/techdocs/us/en/symantec-security-software/web-and-network-security/proxysg/7-2/proxysg-log-fields-and-substitutions.html

notify_snmp()

Multiple notify_snmp actions may be specified, resulting in multiple SNMP traps for a single transaction.

The SNMP trap is sent when the transaction terminates.

Syntax

`notify_snmp(`*`message`*`)`

where:

- *`message`*: Quoted string that can optionally include one or more substitution variables.

Layer and Transaction Notes

- Layers: `<Admin>`, `<Cache>`, `<Diagnostic>`, `<DNS-Proxy>`, `<Exception>`, `<Forward>`, `<Proxy>` , `<SSL>`, `<SSL-Intercept>`
- Transactions: HTTP proxy

Example

Send SNMP traps when users access the specified restricted sites.

```
define condition restricted_sites
  url.domain=a_very_bad_site
  ...
end

<Proxy>
  condition=restricted_sites action.snmp_notify_restricted(yes)

define action snmp_notify_restricted
  notify_snmp("$(client.address) accessed restricted URL: $(url)")
end
```

See Also

- Actions: "log_message()" on page 608, "notify_email()" on the previous page
- *ProxySG Log Fields and CPL Substitutions Reference*: https://techdocs.broadcom.com/content/broadcom/techdocs/us/en/symantec-security-software/web-and-network-security/proxysg/7-2/proxysg-log-fields-and-substitutions.html

redirect()

Ends the current HTTP transaction and returns an HTTP redirect response to the client by setting the `policy_redirect` exception. Use this action to specify an HTTP 3xx response code, optionally set substitution variables based on the request URL, and generate the new Location response-header URL after performing variable substitution.

> **Note:** You cannot use a redirect to override an exception. Exceptions always override redirects.

FTP over HTTP requests are not redirected for Microsoft Internet Explorer clients. To avoid this issue, do not use the `redirect( )` action when the url.scheme=ftp condition is true. For example, if the http_redirect action definition contains a `redirect( )` action, you can use the following rule:

```
url.scheme=ftp action.http_redirect(no)
```

> **Note:** An error results if two `redirect( )` actions conflict. The error is noted at compile time if the conflicting actions are within the same action definition block. A runtime error is recorded in the event log if the conflicting actions are defined in different blocks.

> **Warning:** It is possible to put the browser into an infinite redirection loop if the URL that the browser is being redirected to also triggers a policy-based redirect response.

Syntax

```
redirect(response_code, regex_pattern, redirect_location)
```

where:

- *response_code*: HTTP redirect code used as the HTTP response code; supported codes are 301, 302, 305, and 307.
- *regex_pattern*: Quoted regular-expression pattern that is compared with the request URL based on an anchored match. If the regex_pattern does not match the request URL, the redirect action is ignored. A regex_pattern match sets the values for substitution variables. If no variable substitution is performed by the redirect_location string, specify ".* " for regex_pattern to match all request URLs. For more information about regular expressions, see "Regex Reference" on page 703.
- *redirect_location*: Quoted string that can optionally include one or more variable substitutions.

  This string is an absolute or relative url that is included in the redirect response, as the value of the Location: header. In normal usage, the redirect_location is an absolute url like http://www.example.com/, which instructs the client to

redirect to the specified URL. If the redirect_location does not begin with <scheme>://, where scheme is usually http, then it will be interpreted by the client as a relative URL, which is interpreted relative to the original request URL.

Layer and Transaction Notes

- Layers: , `<Cache>`, `<Proxy>`
- Transactions: HTTP proxy

Example

See the example in "request_redirect()" on the facing page.

See Also

- Conditions: "exception.id=" on page 137
- Actions: rewrite(url.host, *host_regex_pattern, replacement_host*), rewrite(url, *regex_pattern, redirect_location*), set (url.port, *port_number*)
- *ProxySG Log Fields and CPL Substitutions Reference*: https://techdocs.broadcom.com/content/broadcom/techdocs/us/en/symantec-security-software/web-and-network-security/proxysg/7-2/proxysg-log-fields-and-substitutions.html

request_redirect()

Only use this gesture for objects returned from the appliance itself, such as the accelerated_pac_base.pac. Do not apply to redirects for objects from an OCS (Origin Content Server). Use "redirect()" on page 611 for redirects to an OCS.

Syntax

`request_redirect (response_code, regex_pattern, redirect_location)`

where:

- *response_code*: HTTP redirect code used as the HTTP response code; supported codes are 301, 302, 305, and 307.
- *regex_pattern*: Quoted regular-expression pattern that is compared with the request URL based on an anchored match. If the regex_pattern does not match the request URL, the redirect action is ignored. A regex_pattern match sets the values for substitution variables. If no variable substitution is performed by the redirect_location string, specify ".* " for regex_pattern to match all request URLs. For more information about regular expressions, see Appendix E: "Using Regular Expressions".
- *redirect_location*: Quoted string that can optionally include one or more variable substitutions.

  This string is an absolute or relative url that is included in the redirect response, as the value of the Location: header. In normal usage, the redirect_location is an absolute url like http://www.example.com/, which instructs the client to redirect to the specified URL. If the redirect_location does not begin with <scheme>://, where scheme is usually http, then it will be interpreted by the client as a relative URL, which is interpreted relative to the original request URL. For more information, see Appendix D: "CPL Substitutions".

Layer and Transaction Notes

- Layers: `<Cache>`, `<Proxy>`

Example

Prevent a `Request could not be handled` exception when a redirect is necessary in combination with policy that requires a response from an upstream server.

```
<Proxy>
 ALLOW url.path.exact=/wpad.dat action.ReturnRedirect1(yes)

define action ReturnRedirect1
 request_redirect( 302, ".*", "http://proxy.company.com/accelerated_pac_base.pac")
end
```

See Also

- Conditions: "exception.id=" on page 137
- Actions: "rewrite()" on the facing page, "set()" on page 619
- *ProxySG Log Fields and CPL Substitutions Reference*: https://techdocs.broadcom.com/content/broadcom/techdocs/us/en/symantec-security-software/web-and-network-security/proxysg/7-2/proxysg-log-fields-and-substitutions.html

rewrite()

Rewrites the request URL, URL host, or components of the specified header if it matches the regular-expression pattern. This action is often used in conjunction with the URL rewrite form of the transform action in a server portal application.

> **Note:** The URL form of this action does not rewrite some URL components for Windows Media (MMS) transactions. The URL scheme, host, and port are restored to their original values and an error logged if the URL specified by redirect_location attempts to change these components.

An error results if the URL or URL host form of this action conflicts with another URL rewriting action. The error is noted at compile time if the conflicting actions are within the same action definition block. A runtime error is recorded in the event log if the conflicting actions are defined in different blocks.

HTTPS Limitations

Consider the following when planning to use the `rewrite( )` action for HTTPS traffic:

- To perform host rewrites for HTTPS requests, rewrite(url.host) rules need to be created in an <SSL-Intercept> layer.
- HTTPS interception requires that the appliance has an active SSL license and is configured with either protocol detection enabled, (explicit proxy deployment) or that the HTTPS proxy service is set to use the SSL Proxy engine (transparent proxy deployment).
- Header and URL path rewrites for SSL traffic can only occur if the appliance intercepts and decrypts that traffic. Otherwise, the SG cannot access to the unencrypted headers containing the URL path and destination port to perform the rewrite.

Syntax

```
rewrite(url, regex_pattern, redirect_location[, URL_form1,..URL_form3])
rewrite(url.host, regex_pattern, replacement_host[, URL_form1,..URL_form3])
rewrite(header, regex_pattern, replacement_component)
```

where:

- `url`: Specifies a rewrite of the entire URL.
- `url.host`: Specifies a rewrite of the host portion of the URL.
- *header*: Specifies the header to rewrite, using the following form. For a list of recognized headers, see "Recognized HTTP Headers" on page 700.

  - `.header_name`—Identifies a recognized HTTP request header.
  - `response.header.header_name`—Identifies a recognized HTTP response header.
  - `request.x_header.header_name`—Identifies any request header, including custom headers.
  - `response.x_header.header_name`—Identifies any response header, including custom headers.

- *regex_pattern*: Quoted regular-expression pattern that is compared with the URL, host or header as specified, based on an anchored match. If the regex_pattern does not match, the rewrite action is ignored. A regex_pattern match sets the values for substitution variables. If the rewrite should always be applied, but no variable substitution is required for the replacement string, specify ".* " for regex_pattern. For more information about regular expressions, see "Regex Reference" on page 703
- *redirect_location*: Quoted string that can optionally include one or more variable substitutions, which replaces the entire URL once the substitutions are performed. The resulting URL is considered complete, and replaces any URL that contains a substring matching the regex_pattern substring. Sub-patterns of the regex_pattern matched can be substituted in redirect_location using the $(n) syntax, where n is an integer from 1 to 32, specifying the matched sub-pattern. For more information, refer to *ProxySG Log Fields and CPL Substitutions Reference*: https://techdocs.broadcom.com/content/broadcom/techdocs/us/en/symantec-security-software/web-and-network-security/proxysg/7-2/proxysg-log-fields-and-substitutions.html
- *replacement_host*: Quoted string that can optionally include one or more variable substitutions, which replaces the host portion of the URL once the substitutions are performed. Note that the resulting host is considered complete, and it replaces the host in the URL forms specified. Sub-patterns of the regex_pattern matched can be substituted in replacement_host using the $(n) syntax, where n is an integer from 1 to 32, specifying the matched sub-pattern. For more information, refer to *ProxySG Log Fields and CPL Substitutions Reference*: https://techdocs.broadcom.com/content/broadcom/techdocs/us/en/symantec-security-software/web-and-network-security/proxysg/7-2/proxysg-log-fields-and-substitutions.html
- *URL_form*: List of up to three forms of the request URLs that will have the URL or host replaced. If this parameter is left blank, all three forms are rewritten. The following are the possible values:
  - `log`—Request URL used when generating log messages.
  - `cache`—Request URL used to address the object in the local cache.
  - `server`—Request URL sent to the origin server.
- *replacement_component*: Quoted string that can optionally include one or more variable substitutions, which replaces the entire component of the header matched by the regex_pattern substring. Sub-patterns of the regex_pattern matched can be substituted in replacement_component using the `$(n)` syntax, where n is an integer from 1 to 32, indicating the matched sub-pattern. For more information, refer to *ProxySG Log Fields and CPL Substitutions Reference*: https://techdocs.broadcom.com/content/broadcom/techdocs/us/en/symantec-security-software/web-and-network-security/proxysg/7-2/proxysg-log-fields-and-substitutions.html

Discussion

Any rewrite of the server form of the request URL must be respected by policy controlling upstream connections. The server form of the URL is tested by the"server_url=" on page 269 conditions, which are the only URL tests allowed in <Forward> layers.

All forms of the URL are available for access logging. The version of the URL that appears in a specific access log is selected by including the appropriate substitution variable in the access log format:

- c-uri—The original URL
- cs-uri—The log URL, used when generating log messages
- s-uri—The cache URL, used to address the object in the local cache
- sr-uri—The server URL, used in the upstream request

In the absence of actions that modify the URL, all of these substitution variables represent the same value.

Layer and Transaction Notes

- Layers: <Cache>, <Proxy> , <SSL-Intercept>
- Transactions: URL and host rewrites apply to all transactions. Header rewrites apply to HTTP transactions.

Example 1

Rewrite the request URL for HTTP transactions.

```
<Proxy>
  url.domain=//www.example.com/ action.HTTP_rewrite(yes)

define action HTTP_rewrite
  rewrite(url, "^http://www\.example\.com/(.*)", "http://www.server1.example.com/$(1)")
end
```

Example 2

Rewrite the request URL for HTTPS transactions.

```
<SSL-Intercept>
  url.domain=//www.example.com/ action.HTTPS_rewrite(yes)

define action HTTPS_rewrite
  rewrite( url.host, "(.*)example.com(.*)", "$(1)server1.example.com$(2)" )
end
```

See Also

- Conditions: "request.header.header_name=" on page 219, "request.header.header_name.address=" on page 221, "request.x_header.header_name=" on page 241, "request.x_header.header_name.address=" on page 242, "response.header.header_name=" on page 248, "response.x_header.header_name=" on page 256, "server_url=" on page 269
- Actions: "append()" on page 595, "delete()" on page 598, "delete_matching()" on page 600, "redirect()" on page 611, "set()" on the facing page, "transform" on page 622
- Definitions: "define url_rewrite" on page 662
- *ProxySG Log Fields and CPL Substitutions Reference*: https://techdocs.broadcom.com/content/broadcom/techdocs/us/en/symantec-security-software/web-and-network-security/proxysg/7-2/proxysg-log-fields-and-substitutions.html

set()

Sets the specified header to the specified string after deleting all components of the header.

HTTPS Limitations

If the HTTP CONNECT method is used to tunnel a HTTPS connection, the URL path is encrypted and unavailable when the client browser is using a proxy for the connection. As a result, only the headers inside the CONNECT request can be set: these headers include host, port, and other headers using the forward.http_connect parameter. These headers are not encrypted.

Syntax

```
set(header, string)
set(url.port, port_number [, URL_form1,..URL_form3])
```

where:

- header: Specifies the header to set, using the following form. For a list of recognized headers, see "Recognized HTTP Headers" on page 700.
  - `.header_name`–Sets a recognized HTTP request header.
  - `exception.response.header.header_name`–Sets a recognized HTTP response header from the exception response.
  - `exception.response.x_header.header_name`–Sets any response header from the exception response, including custom headers.
  - `forward.http_connect.header.header_name`–Sets a recognized header_name in an HTTP CONNECT request.
  - `forward.http_connect.x_header.header_name`–Sets any HTTP CONNECT request header, including custom headers.
  - `icap_reqmod.request.x_header.header_name`–Sets an ICAP request header for REQMOD.
  - `icap_respmod.request.x_header.header_name`–Sets an ICAP request header for RESPMOD.
  - `request.header.header_name`–Sets a recognized HTTP request header.
  - `request.x_header.header_name`–Sets any request header, including custom headers.
  - `response.header.header_name`–Sets a recognized HTTP response header.
  - `response.x_header.header_name`–Sets any response header, including custom headers.

- *string*: Quoted string that can optionally include one or more variable substitutions, which replaces the specified header components once the substitutions are performed.
- `url.port`: Specifies the URL port to set.
- *port_number*: Port that the request URL is set to. The value can be an integer between 1 and 65535.
- *URL_form*: List of up to three forms of the request URLs that have the port number set. If this parameter is left blank, all three forms of the request URL are rewritten. The possible values are the following:
  - `log`–Request URL used when generating log messages.
  - `cache`–Request URL used to address the object in the local cache.
  - `server`–Request URL sent to the origin server.

Discussion

Any change to the server form of the request URL must be respected by policy controlling upstream connections. The server form of the URL is tested by the"server_url=" on page 269 conditions, which are the only URL tests allowed in `<Forward>` layers.

All forms of the URL are available for access logging. The version of the URL that appears in a specific access log is selected by including the appropriate substitution variable in the access log format:

- c-uri–The original URL.
- cs-uri–The log URL, used when generating log messages.
- s-uri–The cache URL, used to address the object in the local cache.
- sr-uri–The server URL, used in the upstream request.

In the absence of actions that modify the URL, all of these substitution variables represent the same value.

Layer and Transaction Notes

- Use with `exception.response.header.header_name` in `<Proxy>` or `<Exception>` layers; otherwise use only from `<Proxy>`, `<SSL-Intercept>`, or `<Cache>` layers.
- When used with headers, applies to HTTP transactions.
- When used in an <SSL-Intercept> layer, only `set (url.port)` may be used.
- When used with `url.port`, applies to all transactions.

Example

Modifies the URL port component to 8081 for requests sent to the server and cache.

```
set(url.port, 8081, server, cache)
```

See Also

- Conditions: "request.header.header_name=" on page 219, "request.header.header_name.address=" on page 221, "request.x_header.header_name=" on page 241, "request.x_header.header_name.address=" on page 242, "response.header.header_name=" on page 248, "response.x_header.header_name=" on page 256, "server_url=" on page 269
- Actions: "append()" on page 595, "delete()" on page 598, "delete_matching()" on page 600, "redirect()" on page 611, rewrite(url.host, *regex_pattern, replacement_host*), rewrite(url, *regex_pattern, redirect_location*)
- *ProxySG Log Fields and CPL Substitutions Reference*: https://www.symantec.com/docs/DOC11251

transform

Invokes "define active_content" on page 630 , "define javascript" on page 644, or "define url_rewrite" on page 662. The invoked transformer takes effect only if the transform action is used in a "define action" on page 628 block, and that block is in turn enabled by an "action()" on page 337 property.

> **Note:** Any transformed content is not cached, in contrast with content that has been sent to a virus scanning server. This means the transform action can be safely triggered based on any condition, including client identity and time of day.

Syntax

`transform` *`transformer_id`*

where:

- *`transformer_id`*: User-defined identifier for a transformer definition block. This identifier is not case-sensitive.

Layer and Transaction Notes

- Layers: `<Cache>`, `<Proxy>`

Example

Transform the specified `active_content` definition.

```
; The transform action is part of an action block enabled by a rule.
<Proxy>
  url.domain=!my_site.com action.strip_active_content(yes)

; transformer definition
define active_content strip_with_indication
  tag_replace applet <<EOT
  <B>APPLET content has been removed</B>
  EOT
  tag_replace embed <<EOT
  <B>APPLET content has been removed</B>
  EOT
  tag_replace object <<EOT
  <B>OBJECT content has been removed</B>
  EOT
  tag_replace script <<EOT
  <B>SCRIPT content has been removed</B>
  EOT
end
```

```
define action strip_active_content
  ; the transform action invokes the transformer
  transform strip_with_indication
end
```

See Also

- Properties: "action()" on page 337, "transform.data_type()" on page 581
- Definitions: "define action" on page 628, "define active_content" on page 630, "define url_rewrite" on page 662

validate()

Invoke the specified CAPTCHA validator.

Syntax

`validate(`*`validator_name`*`)`

where:

Layer and Transaction Notes

- Layers: <Admin>, <Cache>, <DNS-Proxy>, <Exception>, <Forward>, <Proxy> , <SSL>, <SSL-Intercept>
- Transactions: HTTP proxy

Example

Implement two-factor authentication using an existing IWA realm and CAPTCHA.

```
; authenticate using the specified realm
<Proxy>
  authenticate(iwa_realm)

; for URLs where the content filter cannot determine the category, perform second authentication step
using specified CAPTCHA validator
<Proxy>
  category=unavailable validate(CAPTCHA_1)
```

See Also

- Actions: "validate.form()" on the facing page, "validate.mode()" on page 626

validate.form()

Specify a custom CAPTCHA form to use for a validator.

Syntax

`validate.form(`*`form_name`*`)`

Layer and Transaction Notes

- Layers: <Admin>, <Cache>, <DNS-Proxy>, <Exception>, <Forward>, <Proxy> , <SSL>, <SSL-Intercept>
- Transactions: HTTP proxy

Example

Add the form to policy after invoking the validator.

```
; use the specified CAPTCHA validator
<Proxy>
  category=unavailable validate(CAPTCHA_1)

; use the specified CAPTCHA validation form
<Proxy>
  validate.form(CAPTCHA_form1)
```

See Also

- Actions: "validate()" on the previous page, "validate.mode()" on the next page

validate.mode()

Use a Common Domain Cookie to prevent recurring CAPTCHA challenges when changing hosts within a browsing session.

Syntax

`validate.mode(`*mode*`)`

where:

- `mode`: One of the following authentication modes:
  - `form-cookie`: A form is presented to collect the user's credentials. The cookies are set on the OCS domain only, and the user is presented with the form for each new domain. This mode is most useful in reverse proxy scenarios where there is a limited number of domains.
  - `form-cookie-redirect`: A form is presented to collect the user's credentials. The user is redirected to the authentication virtual URL before the form is presented. The authentication cookie is set on both the virtual URL and the OCS domain. The user is only challenged when the credential cache entry expires.

Layer and Transaction Notes

- Layers: `<Admin>`, `<Cache>`, `<DNS-Proxy>`, `<Exception>`, `<Forward>`, `<Proxy>` , `<SSL>`, `<SSL-Intercept>`
- Transactions: HTTP proxy

Example

```
; allow connection when user authenticates against specified realm
<Proxy>
  allow authenticate(iwa_realm)

; allow connection with the specified validator
; use form-cookie-redirect authentication redirect mode
<Proxy>
  allow validate(CAPTCHA_1) validate.mode(form-cookie-redirect)
```

See Also

- Actions: "validate()" on page 624, "validate.form()" on the previous page

Destinations

In policy files, definitions serve to bind a set of conditions, actions, or transformations to a user-defined label.

Two types of definitions exist:

- Named definitions—Explicitly referenced by policy.
- Anonymous definitions—Apply to all policy evaluation and are not referenced directly in rules.

There are two types of anonymous definitions: DNS and RDNS restrictions.

Definition Names

There are various types of named definitions. Each of these definitions is given a user-defined name that is then used in rules to refer to the definitions. The user-defined labels used with definitions are not case-sensitive. Characters in labels may include the following:

- letters
- numbers
- space
- period
- underscore
- hyphen
- forward slash
- ampersand

The first character of the name must be a letter or underscore. If spaces are included, the name must be a quoted string.

Only alphanumeric, underscore, and dash characters can be used in the name given to a defined action.

The remainder of this section lists the definitions and their accepted values. It also provides tips as to where each definition can be used and examples of how to use them.

define action

Binds a user-defined label to a sequence of action statements. The "action()" on page 337 property has syntax that allows for individual action definition blocks to be enabled and disabled independently, based on the policy evaluation for the transaction. When an action definition block is enabled, any action statements it contains operate on the transaction as indicated by their respective arguments. See "Actions" on page 594 for more information about the various action statements available.

> **Note:** Action statements that must be performed in a set sequence and cannot overlap should be listed within a single action definition block.

Syntax

```
define action label
  list_of_action_statements
end
```

where:

- *label*: User-defined identifier for an action definition. Only alphanumeric, underscore, and dash characters can be used in the label given to a defined action.
- *list_of_action_statements*: List of actions to be carried out in sequence. See "Actions" on page 594 for the available actions.

Layer and Transaction Notes

Each action statement has its own timing requirements and layer applicability. The timing requirements for the overall action are the strictest required by any of the action statements contained in the definition block.

Similarly, the layers that can reference an action definition block are the layers common to all the action statements in the block.

Action statements that are not appropriate to the transaction will be ignored.

Example

Clear the `From` and `Referer` headers (which normally could be used to identify the user and where they clicked from) in any request to specified URLs.

```
define url condition scrub_headers_list
  example.com
  test.com
  ..
end
```

```
define action scrub_private_info
  set(request.header.From, "")
  set(request.header.Referer, "")
end

<proxy> condition=scrub_headers_list
  action.scrub_private_info(yes)
```

Notice that the object on which the "set()" on page 619 action operates is given in the first argument, and then appropriate values follow, in this case, the new value for the specified header. This is common to many of the actions.

See Also

- Properties: "action()" on page 337
- Definitions: "define active_content" on the next page, "define url_rewrite" on page 662

define active_content

Defines rules for removing or replacing active content in HTML or ASX documents. This definition takes effect only if it is invoked by a "transform" on page 622 action in a "define action" on page 628 block, and that block is in turn enabled an "action ()" on page 337 property as a result of policy evaluation.

Active content transformation acts on the following four HTML elements in documents: <applet>, <embed>, <object>, and <script>. In addition, a script transformation removes any JavaScript content on the page. For each tag, the replacement can either be empty (thus deleting the tag and its content) or new text that replaces the tag. Multiple tags can be transformed in a single active content transformer. Pages served over an HTTPS tunneled connection are encrypted so the content cannot be modified.

> **Note:** Transformed content is not cached, in contrast with content that has been sent to a virus scanning server. Therefore, a transformer can be safely triggered based on any condition, including client identity and time of day.

Syntax

```
define active_content transformer_id
  tag_replace HTML_tag_name << text_end_delimiter
    [replacement_text]
    text_end_delimiter
      [tag_replace ...]
  ...
end
```

where:

- *transformer_id*: User-defined identifier for a transformer definition block. Used to invoke the transformer using the transform action in a "define action" on page 628 definition block.
- *HTML_tag_name*: Name of an HTML tag to be removed or replaced, as follows:
  - applet–Operates on the <applet> element, which places a Java applet on a Web page.
  - embed–Operates on the <embed> element, which embeds an object, such as a multimedia file, on a Web page.
  - object–Operates on the <object> element, which places an object, such as an applet or media file, on a Web page.
  - script–Operates on the <script> element, which adds a script to a Web page. Also removes any JavaScript entities, strings, or events that may appear on the page.

  If the tag_replace keyword is repeated within the body of the transformer, multiple HTML tags can be removed or replaced.

- *text_end_delimiter*: User-defined token that does not appear in the replacement text and does not use quotes or whitespace. The delimiter is defined on the first line, after the required double angle brackets (<<). All text that follows, up to the second use of the delimiter, is used as the replacement text.
- replacement_text: Either blank, to remove the specified tag, or new text (including HTML tags) to replace the tag.

Layer and Transaction Notes

- Layers: <Proxy>

Example

Remove active content from requests other than to my_site, and replace with the specified message.

```
<Proxy>
  url.domain=!my_site.com action.strip_active_content(yes)

define active_content strip_with_indication
  tag_replace applet <<EOT
     <B>APPLET content has been removed</B>
  EOT
  tag_replace embed <<EOT
     <B>APPLET content has been removed</B>
  EOT
  tag_replace object <<EOT
     <B>OBJECT content has been removed</B>
  EOT
  tag_replace script <<EOT
     <B>SCRIPT content has been removed</B>
  EOT
end

define action strip_active_content
  transform strip_with_indication
end
```

See Also

- Properties: "action()" on page 337, "transform.data_type()" on page 581
- Actions: "transform" on page 622
- Definitions: "define action" on page 628, "define url_rewrite" on page 662

define application_protection_set

Bind a specified label to a set content nature detection engines, and then specify the action to take on the defined set.

> **Note:** The engines parse SOAP and Multipart content.

Syntax

```
define application_protection_set label
engine=engine_name keyword=(property1,..)
engine=engine_name keyword=property
..
end

http.request.detection.label(block|monitor|ignore)
```

where:

- *label* - Your custom name for the set.
- *engine_name* - The content nature detection engine.
- *keyword* - Optional specifiers:
  - language - One or more languages that the specified engine supports.
  - host - One or more operating systems.
  - part - The part of the HTTP request that the engine scans:

Name	Value
name - All argument names found in the URL query string, post body (URL-encoded and multipart-form encoded formats), or cookie.	value - All named and unnamed argument values found in URL query string, post body (URL-encoded and multipart-form encoded formats), or cookie.
query_arg_name - All argument names found in the URL query string.	query_arg - All named and unnamed argument values found in the URL query string.
arg_name - All argument names found in both the URL query string and the post body (URL-encoded and multipart-form encoded formats).	arg - All named and unnamed argument values found in both the URL query string and the post body (URL-encoded and multipart-form encoded formats).
cookie_name - All argument names found in all Cookie and Cookie2 headers.	cookie - All named and unnamed argument values found in all Cookie and Cookie2 headers.

Name	Value
`post_arg_name` - All argument names found in the post body (URL-encoded and Multipart-form encoded formats)	`post_arg` - All named and unnamed argument values found in the post body (URL-encoded and Multipart-form encoded formats).
`header_name` - All header names.	`header` - All header values.
`path` - Path of the URL. This attribute does not have `name=value` format.	

- rule - Use the rule keyword to enable or disable certain blacklist or analytics_filter engine rules. The rule keyword is not applicable to other engines. Rules have the following format:

  AF-####-# or BL-####-#

  In the preceding example, AF refers to analytics_filter and BL refers to blacklist. The ####-## represents the rule and sub-pattern identifiers. When the blacklist and analytics_filter engines flag a detection within a request, the matching rules are included in the block or monitor details access log field(s).

- version - Specify the command injection engine version:

  - 2 - The legacy version used in versions prior to 6.6.5.1. This version targets chained command sequences, and requires command-separation characters to be present in the payload to be effective.
  - 3 - The current default version. The command injection engine detects a wider set of attacks, including non-chained command injection payloads. Symantec recommends that you use this version.

- property - A supported property:

Engine	CPL
Blacklist	engine=blacklist rule=(BL-2000-0, BL-2000-1)
Analytics Filter	engine=analytics_filter rule=(AF-1000-0, AF-1000-1)
SQL injection	engine=injection.sql
Cross-site scripting	engine=xss
Code injection	engine=injection.code language=(java, php, ssi, javascript) Note: You do not have to use parentheses when specifying a single value.
HTML injection	engine=injection.html
Directory traversal	engine=reference.directory_traversal

Engine	CPL
Command injection	engine=injection.command host=(linux, windows, osx) version=2 | 3 Note: You do not have to use parentheses when specifying a single value.

> **Note:** All engines except `blacklist` and `analytics_filter` accept the optional `part=` attribute.

- block - Denies the request and logs the action.
- monitor - Allows the request and logs the action.
- ignore - Allows the request and does not log the action.

Layer and Transaction Notes

- Layers: `<Admin>`, `<Cache>`, `<DNS-Proxy>`, `<Exception>`, `<Forward>`, `<Proxy>` , `<SSL>`, `<SSL-Intercept>`
- Transactions: HTTP proxy

Example

Define a set of engines for application protection and set them to block.

```
define application_protection_set engines
  engine=blacklist
  engine=analytics_filter
  engine=reference.directory_traversal
  engine=xss
  engine=injection.command host=(windows, osx)
  engine=injection.html
  engine=injection.sql
  engine=injection.code language=java
end

...

http.request.detection.engines(block)
```

See Also

- Properties: "http.request.detection.other()" on page 454

define category

Binds a user-defined label to a set of conditions for use in a "category=" on page 85 expression.

For condition definitions, the manner in which the condition expressions are listed is significant. Multiple condition expressions on one line, separated by whitespace, are considered to have a Boolean AND relationship. However, the lines of condition expressions are considered to have a Boolean OR relationship.

Performance optimized condition definitions are available for testing large numbers of URLs. See "define server_url.domain condition" on page 652"define url condition" on page 658, and "define url.domain condition" on page 660.

Syntax

```
define condition label
  condition_expression
  ...
end
```

where:

- *label*: User-defined identifier for a condition definition. Used to call the definition from "action()" on page 337.
- *condition_expression*: Any of the conditions available in a rule. The layer and timing restrictions for the defined condition depend on the layer and timing restrictions of the contained expressions.

  "condition=" on page 113 is one of the expressions that can be included in the body of a define condition definition block. In this way, one condition definition block can call another condition-related definition block, so that they are in effect nested. Circular references generate a compile error.

Layer and Transaction Notes

- The layers that can reference a condition definition are the layers common to all the condition statements in the block.
- A condition can be evaluated for any transaction. The condition evaluates to true if all the condition expressions on any line of the condition definition apply to that transaction and evaluate to true. Condition expressions that do not apply to the transaction evaluate to false.

Example

This example illustrates a simple virus scanning policy designed to prevent some traffic from going to the scanner. Some file types are assumed to be at low risk of infection (some virus scanners will not scan certain file types), and some are assumed to have already been scanned when they were loaded on the company's servers.

> **Note:** The following policy is not a security recommendation, but an illustration of a technique. If you choose to selectively direct traffic to your virus scanner, you should make your own security risk assessments based on current information and knowledge of your virus scanning vendor's capabilities.

```
define condition extension_low_risk ; file types assumed to be low risk
  url.extension=(asf,asx,gif,jpeg,mov,mp3,ram,rm,smi,smil,swf,txt,wax,wma,wmv,wvx)
end

define condition internal_prescanned ; will be prescanned so we can assume safe
  server_url.domain=internal.myco.com server_url.extension=(doc,dot,hlp,html)
  server_url.domain=internal.myco.com response.header.Content-Type=(text,application/pdf)
end

define condition white_list
  condition=extension_low_risk
  condition=internal_prescanned
end

<Cache>
  condition=!internal_white_list action.virus_scan(true)

define action virus_scan
  response.icap_service("ICAP_server") ; configured service name
end
```

See Also

- Conditions: "category=" on page 85, "condition=" on page 113
- Properties: "action()" on page 337
- Definitions: "define server_url.domain condition" on page 652, "define url condition" on page 658, "define url.domain condition" on page 660

define condition

Binds a user-defined label to a set of conditions for use in a "condition=" on page 113 expression.

For condition definitions, the manner in which the condition expressions are listed is significant. Multiple condition expressions on one line, separated by whitespace, are considered to have a Boolean AND relationship. However, the lines of condition expressions are considered to have a Boolean OR relationship.

Performance optimized condition definitions are available for testing large numbers of URLs. See "define url condition" on page 658, "define url.domain condition" on page 660, and "define server_url.domain condition" on page 652.

Syntax

```
define condition label
  condition_expression,...
  ...
end
```

where:

- *label*: User-defined identifier for a condition definition. Used to call the definition from an "action()" on page 337 property.
- *condition_expression*: Any of the conditions available in a rule. The layer and timing restrictions for the defined condition depend on the layer and timing restrictions of the contained expressions.

  "condition=" on page 113 is one of the expressions that can be included in the body of a define condition definition block. In this way, one condition definition block can call another condition-related definition block, so that they are in effect nested. Circular references generate a compile error.

Layer and Transaction Notes

- The layers that can reference a condition definition are the layers common to all the condition statements in the block.
- A condition can be evaluated for any transaction. The condition evaluates to true if all the condition expressions on any line of the condition definition apply to that transaction and evaluate to true. Condition expressions that do not apply to the transaction evaluate to false.

Example

This example illustrates a simple virus scanning policy designed to prevent some traffic from going to the scanner. Some file types are assumed to be at low risk of infection (some virus scanners will not scan certain file types), and some are assumed to have already been scanned when they were loaded on the company's servers.

> **Note:** The following policy is not a security recommendation, but an illustration of a technique. If you choose to selectively direct traffic to your virus scanner, you should make your own security risk assessments based on current information and knowledge of your virus scanning vendor's capabilities.

```
define condition extension_low_risk ; file types assumed to be low risk.
  url.extension=(asf,asx,gif,jpeg,mov,mp3,ram,rm,smi,smil,swf,txt,wax,wma,wmv,wvx)
end

define condition internal_prescanned ; will be prescanned so we can assume safe
  server_url.domain=internal.myco.com server_url.extension=(doc,dot,hlp,html)
  server_url.domain=internal.myco.com response.header.Content-Type=(text, application/pdf)
end

define condition white_list
  condition=extension_low_risk
  condition=internal_prescanned
end

<Cache>
  condition=!internal_white_list action.virus_scan(true)

define action virus_scan
  response.icap_service( "ICAP_server" ) ; configured service name
end
```

See Also

- Conditions: "category=" on page 85, "condition=" on page 113
- Properties: "action()" on page 337

define constraint_set

Specify field constraints that, when violated, will trigger a block or monitor action. The constraints are applied to the HTTP request field attributes after they are normalized.

Use this gesture in conjunction with "http.request.detection.constraint_set()" on page 451.

When a transaction violates constraints, it is blocked or monitored. The bcreporterwarp_v1 access log format includes the following fields that log information about the constraint violation:

- the `x-bluecoat-waf-attack-family` field shows `Constraint Violation`
- the `x-bluecoat-waf-block-details` or `x-bluecoat-waf-monitor-details` field shows details with the following syntax:

  `"{""detection"":""constraint"",""part"":""{name|query_arg_name|query_arg|arg_name|arg|cookie_name|cookie|post_arg_name|post_arg|header_name|header|path}"",""line"":""`*`constraint_set_defn_cpl_line`*`"",""data"":""`*`matched_data`*`""}"`

Syntax

```
define constraint_set constraint_id
  part="attribute" [pattern.string_modifier="string"] {key|value|path}.modifier=constraint
end
```

where:

- *`constraint_id`*: Name for the constraint set.
- *`attribute`*: Supported HTTP attribute.

Name	Value
`name` - All argument names found in the URL query string, post body (URL-encoded and multipart-form encoded formats), or cookie.	`value` - All named and unnamed argument values found in URL query string, post body (URL-encoded and multipart-form encoded formats), or cookie.
`query_arg_name` - All argument names found in the URL query string.	`query_arg` - All named and unnamed argument values found in the URL query string.
`arg_name` - All argument names found in both the URL query string and the post body (URL-encoded and multipart-form encoded formats).	`arg` - All named and unnamed argument values found in both the URL query string and the post body (URL-encoded and multipart-form encoded formats).
`cookie_name` - All argument names found in all Cookie and Cookie2 headers.	`cookie` - All named and unnamed argument values found in all Cookie and Cookie2 headers.

Name	Value
`post_arg_name` - All argument names found in the post body (URL-encoded and Multipart-form encoded formats)	`post_arg` - All named and unnamed argument values found in the post body (URL-encoded and Multipart-form encoded formats).
`header_name` - All header names.	`header` - All header values.
`path` - Path of the URL. This attribute does not have `name=value` format.	

- *`string_modifier`*: One of the supported string modifiers listed below.
  - `exact` - match the string exactly
  - `prefix` - match the start of a string
  - `regex` - regular expression match; see "Regex Reference" on page 703
  - `substring` - match part of a string
  - `suffix` - match the end of a string
- *`string`*: String to search for in the specified `part`.
- *`key`*: Request `part`'s specified key to constrain.
- *`value`*: Request `part`'s specified value to constrain.
- *`path`*: Request `part`'s specified path to constrain.
- *`modifier`*: Valid constraint type. See the following table for examples.
- *`constraint`*: Valid constraint for the constraint type. See the following table for examples.

> **Note:** The following modifiers apply to `key` and `value`, such as `key.length=`. `path` is used only if `part=path`.

Modifier	Constraints	Examples
`length` - Matches if the key or value field length is the specified exact number of characters, or the field length is within the specified inclusive range(s).	*`exact_length(s)`* *`range(s)`* *`mix_of_exact_and_range`*	`key.length="10..100"` `key.length="15,20"` `value.length="10,..4"` `value.length="50..60"`

Modifier	Constraints	Examples
range - Matches if the key or value is the specified exact number, or the numbers is within the specified inclusive range(s). **Note:** If the match is not an integer, a constraint violation occurs. See the **Example** for details.	*exact_number(s)* *range(s)* *mix_of_exact_and_range*	key.range="10" key.range="15,25" value.range="20..30" value.range="20,30.."
count - Matches if the key or value occurs the specified number of times or within the specified inclusive range(s). **Note:** The "data" section of x-bluecoat-waf-block-details or x-bluecoat-waf-monitor-details access log field displays "-" for this modifier. See the **Example** for details.	*exact_number(s)* *range(s)* *mix_of_exact_and_range*	key.count="20" key.count="1..9" value.count="2,10.." value.count="..6"
type - Matches if the key or value is the specified type.	utf8 ascii number integer	key.type="utf8" key.type="ascii" value.type="number" value.type="integer"
regex - Matches if the key or value is equal to the specified regular expression.	*regular_expression*	key.regex=\"Content-Length\" value.regex=\"[0-9]+e\^[0-9]+\"

Layer and Transaction Notes

- Layers: <Proxy>
- Transactions: HTTP proxy

Example

Define constraints for specified parts of requests, and then block or monitor transactions that violate constraints.

```
; define constraints for the specified parts of requests to block
define constraint_set c_block
  part="post_arg_name" pattern.regex="pid_.*" value.range="1..10"
  part="post_arg_name" pattern.regex="nid_.*" key.count="1" value.range="1..100"
```

```
  part="query_arg" value.length="..15"
end

; use value.count instead of key.count so there is at most 1 Content-Type header in this example
: because "Content-Type" becomes the key for a list of values
; multiple "Content-Type" headers still keep the key count at most 1
; but the count of values increases for each one
; key.count is higher for scenarios where the pattern can match multiple strings
; i.e., regex pattern "Content-?Type" will match both Content-Type and ContentType
; if a request has both Content-Type and ContentType in the request, key.count will go to 2 in that
example

define constraint_set c_monitor
  part="header_name" pattern.regex="Content-Type" value.count="..1"
  part="query_arg_name" pattern.exact="large_number" value.regex="[0-9]+e\^[0-9]+"
  part="header" pattern.regex="text/html" key.regex="Content-Type"
end

; block and log transaction details if constraints defined by c_block are violated
; monitor and log transaction details if constraints defined by c_monitor are violated
<Proxy>
  http.request.detection.constraint_set.c_block(block) http.request.detection.constraint_set.c_monitor
(monitor)
```

Refer to the following for examples of policy operation's effect on different requests and log output. In all cases, the `x-bluecoat-waf-attack-family` field shows `"Constraint Violation"`.

Request properties	Policy verdict	`x-bluecoat-waf-block|monitor-details` access log field output
POST argument `pid_123=44`	Blocked because 44 violates `pattern.regex="pid_.*" value.range="1..10` constraint.	`x-bluecoat-waf-block-details "[{""detect"":""constraint"",""part"":""post_arg"",""line"":""part=\"post_arg_name\" pattern.regex=\"pid_.*\" value.range=\"1..10\""",""data"":""44""}]"`
POST argument `pid_2=5.897`	Blocked because the range must be an integer.	`x-bluecoat-waf-block-details "[{""detect"":""constraint"",""part"":""post_arg"",""line"":""part=\"post_arg_name\" pattern.regex=\"pid_.*\" value.type=\"integer\""",""data"":""5.897""}]"`
query argument `fff=1234567890123456`	Blocked because the number of characters in the `fff=` field violates the `value.length="..15"` constraint.	`x-bluecoat-waf-block-details "[{""detect"":""constraint"",""part"":""query_arg"",""line"":""part=\"query_arg\" value.length=\"..15\""",""data"":""1234567890123456""}]"`

Request properties	Policy verdict	x-bluecoat-waf-block|monitor-details access log field output
Two Content-Type headers	Monitored because more than one Content-Type header violates pattern.regex="[Cc]ontent-[Tt]ype" key.count="..1" constraint.	x-bluecoat-waf-monitor-details "[{""detect"":""constraint"",""part"":""header"",""line"":""part=\"header_name\" pattern.regex=\"[Cc]ontent-[Tt]ype\" key.count=\"..1\""",""data"":""-""}]"
Query argument large_number=123	Monitored because 123 does not match the regex in the constraint definition: "[0-9]+e\^[0-9]+"	x-bluecoat-waf-monitor-details "[{""detect"":""constraint"",""part"":""query_arg"",""line"":""part=\"query_arg_name\" pattern.exact=\"large_number\" value.regex=\"[0-9]+e\^[0-9]+\""",""data"":""123""}]"

See Also

- Properties: "http.request.detection.constraint_set()" on page 451
- *ProxySG Log Fields and CPL Substitutions Reference*: https://techdocs.broadcom.com/content/broadcom/techdocs/us/en/symantec-security-software/web-and-network-security/proxysg/7-2/proxysg-log-fields-and-substitutions.html

define javascript

Define a JavaScript transformer, which adds JavaScript that you supply to HTML responses.

Syntax

```
define javascript transformer_id
  javascript_statement[,javascript-_statement,..]
  ...
end
```

where:

- *transformer_id*: User-defined identifier for a transformer definition block. Used to invoke the transformer using the "transform" on page 622 action in a "define action" on page 628 definition block.
- *javascript_statement*: Statement with the following syntax:

  javascript-statement ::= section-type [tag_attributes='attributes'] replacement

  where:

  - section-type ::= prolog | onload | epilog

    Specifies whether to insert the JavaScript block at the beginning (prolog) or the end of the HTML page (epilog), and if the script should be executed when parsing is complete and the page is loaded (onload).

  - tag_attributes='attributes'

    (applicable to prolog and epilog sections only) Adds the specified attributes within the <script> start tag. Double quotes (") are supported, but you need only use them if the attribute value includes single quotes.

    If supplied in CPL, the specified attributes will replace the content that the appliance inserts by default.

  - replacement ::= << endmarker newline lines-of-text newline endmarker

Layer and Transaction Notes

- Layers: `<Proxy>`

Example 1

The following is an example of a JavaScript transformer that adds a message to the top of each Web page, used as part of a simple content filtering application:

```
define javascript
  js_transformer
    onload <<EOS
```

```
      var msg = "This site is restricted. Your access has been logged.";
      var p = document.createElement("p");
      p.appendChild(document.createTextNode(msg));
      document.body.insertBefore(p, document.body.firstChild);
    EOS
end

define action js_action
  transform js_transformer
end

<Proxy>
  category=restricted action.js_action(yes)
```

The VPM uses JavaScript transformers to implement popup ad blocking.

Example 2

The following is an example of a JavaScript transformer that adds the specified attributes inside the <script> tag. The JavaScript block is added to the start of the page.

```
define javascript
  js_transformer2
    prolog tag_attributes='type="text/javascript" src="http://mysite/myscript.js"' <<EOS
    EOS
end

define action js_action2
  transform js_transformer2
end

<Proxy>
  action.js_action2(yes)
```

See Also

- Properties: "action()" on page 337, "transform.data_type()" on page 581
- Actions: "transform" on page 622
- Definitions: "define action" on page 628

define policy

A policy definition defines a named policy macro, which is a sequence of policy layers that can be called by name from other layers. All layers in a policy macro must be of the same type, which is declared on the first line of the definition.

A policy macro call (`policy.`*`macro_name`*) is similar to a CPL property setting: it is only evaluated if all the conditions on the rule line are true. When a macro call is evaluated, all of the layers in the corresponding policy definition are evaluated, setting some properties. (A policy macro that sets no properties has no effect when evaluated.)

When a rule is matched during policy evaluation, all of the property settings and macro calls in that rule are evaluated from left to right, with later property settings overriding earlier property settings. This means that all property settings before a macro call act as defaults, and all property settings after the macro call act as overrides.

Note the following about `define policy`:

- A policy definition can contain calls to other policy macros; however, recursive calls and circular call chains are not allowed.
- A policy definition cannot contain other definitions.
- The calling layer must have the same type as the policy macro.

Syntax

```
define layer_type policy macro_name
  layer1
  layer2
  ...
end
```

Layer and Transaction Notes

- Layers: `<Admin>`, `<Cache>`, `<DNS-Proxy>`, `<Exception>`, `<Forward>`, `<Proxy>` , `<SSL>`, `<SSL-Intercept>`
- Transactions: HTTP proxy

Example

Call the policy macro from another layer.

```
define proxy policy WebAccessPolicy
  <Proxy>
    DENY hour=9..17 category=NotBusinessRelated
    DENY category=IllegalOrOffensive
end

<Proxy> url.address=TheInternet
  group=Operator ALLOW
```

```
  group=Employee policy.WebAccessPolicy
DENY
```

define probe

Collect diagnostics (policy trace, debug logs) when a specified policy condition (created via "define condition" on page 637) is met. Additionally, you can specify settings within the `define probe` definition to display the trace and log details in the Advanced URL or Syslog, and be notified when the diagnostics become available.

You can also troubleshoot specific issues by specifying settings within the `define probe` definition. The following settings are required:

- condition
- target
- expiry

For other settings, you do not need to specify a value. If a value is not specified, the settings take the default.

Syntax

```
define probe case_label
  condition=condition_label
  target=system_log:log_level[,system_log:log_level]
  policy_trace={yes|no}
  limit=transaction_limit
  limit.session=session_limit
  alert=alert_channel:{first|last|both}
  delivery={hold|syslog}
  scope={session|transaction}
  expiry[.utc]=expiry_time
end
```

where:

- *case_label*: User-defined label assigned to the probe definition to identify it from any other probe definitions. The *case_label* is reflected in the `diagnostic_probe` field in Splunk alerting and is the name of the trace in the Advanced URL.
- *condition_label*: Name of a "define condition" on page 637 block that exists elsewhere in policy. This condition identifies the traffic to which the diagnostic probe is applied. Condition labels are case-insensitive.
- `target`: Enable logging for a specified system and logging level:
    - *system_log*: System or sub-system. Possible values are:
        - `http`: System for HTTP traffic.
        - `ssl`: System for SSL traffic

- *log_level*: Logging level for the specified logs to collect:
  - off: Disable logging.
  - error: Log only errors.
  - debug: Logs debug-level information and errors.
  - all: Logs all information and errors.

To understand which logs are captured by each *log_level* value, view the following table which maps the *log_level* value to the selection of logs that are captured for both the http and ssl systems.

Log Level	Captured Logs for HTTP System Log	Captured Logs for SSL System Log
Error	◦ LOG_ASSERT ◦ LOG_CRIT ◦ LOG_EMRG ◦ LOG_ALERT ◦ LOG_WARNING	◦ SSLPROXYERROR ◦ SSLPROXYWARN
Debug	◦ LOG_DEBUG ◦ LOG_INFO ◦ LOG_NOTICE	◦ SSLPROXYNOTICE ◦ SSLPROXYINFO
All	◦ LOG_EVENT ◦ LOG_ALWAYS	◦ SSLPROXYALWAYS

- limit: (Optional) Maximum number of transactions for which the current diagnostics are collected.
  - *transaction_limit*: Default value is 30 and maximum value is 50.
- limit.session: (Optional) Maximum number of transactions whose diagnostic information can be collected for the current session. If scope is not set to session, this setting is ignored. It is useful to limit captures to the first few transactions in a session, such as transaction handoffs. This limit applies to the first transaction that matches the session scope probe conditions and subsequent transactions within the session.
  - *session_limit* - Default is 10 and the maximum value is the *transaction_limit* value.

    As an example, a *transaction_limit* of 40 and a *session_limit* of 10 means that up to 400 transactions could be traced.
- alert: (Optional) Notify the administrator when the diagnostics bundle becomes available, or when the *transaction_limit* is reached. The default is none.

- *alert_channel* - Notification method of event-log or none to not receive any notifications. When event-log is selected, the alert message is sent to the event log. If you have configured the event log to write alert messages that have a log level of policy, informational, or verbose to the event log, then the event log will forward the alert messages to the syslog. For information on configuring the event-log, see the #(config) **event-log** command in the *SGOS Command Line Interface Reference*.
  - first - Send notifications when the first diagnostic trace associated with the case is available.
  - last - Send notifications when all traces (up to *transaction_limit*) have been collected.
  - both - Send both first and last notifications.

  For more information on these alerts, see the "About Diagnostics Probe Alerts" topic in the *SGOS Administration Guide*.

- delivery: (Optional) Delivery method for the diagnostic bundle:
  - hold - (Default) Display diagnostics information on the advanced URL page at https://*IP_address*:*port*/Diagnostic/Trace.
  - syslog - Send diagnostics to the configured Syslog upload method specified by the # (config diagnostics) **syslog** CLI subcommands. The command must be enabled before installing policy. Refer to the *Command Line Interface Reference* for details.
- scope: (Optional) When diagnostics are collected:
  - session: When the specified condition matches for a transaction, diagnostics are collected for all subsequent transactions in the session or up to the *session_limit* (if specified).

    > **Note:** The subsystem diagnostics that are generated for a session takes the maximum level required by the session scope probes that applied to that session. For example, if probe A requires an HTTP diagnostic level of ERROR and probe B requires an HTTP diagnostic level of DEBUG, a session with both probes applied will collect DEBUG diagnostics from the HTTP subsystem.

  - transaction: (Default) Each transaction is evaluated independently to determine if diagnostics are to be collected.
- *expiry_time*: Time after which collection of the diagnostic bundle stops, expressed in format *YYYYMMDD*:*HHMM*. By default, the time is local unless .utc is specified.

Layer and Transaction Notes

- Layers: The probe definitions are not called or referenced by other layers. The probe generates a <Diagnostic> layer that is visible in a policy trace.
- Transactions: All transactions that occur in the specified target systems.

Example

When traffic matches the `my_traffic_selection` condition:

- A diagnostics bundle (consisting of HTTP debug log, all SSL logs, and policy trace) is collected for up to ten transactions, and up to five transactions in a session.
- Diagnostics collection stops at 11:50 PM on January 1, 2020.
- The administrator is notified via Syslog when the diagnostics information is available on the Advanced URL page.

```
define condition my_traffic_selection
  condition_expressions
end

define probe case123
  condition=my_traffic_selection
  scope=session
  target=http:debug,ssl:all
  policy_trace=yes
  limit=10
  limit.session=5
  alert=syslog:both
  delivery=hold
  expiry=20200101:2350
end
```

After the policy has been applied, view the compiled policy in the CLI by using the `#(config)` **`show policy executable`** command.

```
#(config) show policy executable
<Diagnostic probe:case123> diagnostic.required.case123=true
    condition=my_traffic_selection trace.session(yes) \
         trace.diagnostic.HTTP(debug)  trace.diagnostic.SSL(all)
    diagnostic.disable(case123)
```

define server_url.domain condition

Binds a user-defined label to a set of domain-suffix patterns for use in a "condition=" on page 113 expression. Using this definition block allows you to quickly test a large set of server_url.domain= conditions. Although the "define condition" on page 637 definition block could be used in a similar way to encapsulate a set of domain suffix patterns, this specialized definition block provides a substantial performance boost.

The manner in which the URL patterns and any condition expressions are listed is significant. Each line begins with a URL pattern and, optionally, one or more condition expressions, all of which have a Boolean AND relationship. Each line inside the definition block is considered to have a Boolean OR relationship with other lines in the block.

> **Note:** This condition is for use in the <Forward> layers and takes into account the effect of any "rewrite()" on page 615 actions on the URL. Because any rewrites of the URL intended for servers or other upstream devices must be respected by <Forward> layer policy, conditions that test the unrewritten URL are not allowed in <Forward> layers. Instead, this condition is provided.

Syntax

```
define server_url.domain condition label
  domain_suffix_pattern [condition_expression,...]
  ...
end
```

where:

- *label*: User-defined identifier for a domain condition definition. Used in "condition=" on page 113.
- *domain_suffix_pattern*: URL pattern that includes a domain name (domain), as a minimum. See the "url=" on page 298 condition reference for a complete description.
- *condition_expression*: Condition expression, using any of the conditions available in a rule, that are allowed in a <Forward> layer. For more information, see "Conditions" on page 65.

"condition=" on page 113 is one of the expressions that can be included in the body of a `define server_url.domain condition` definition block, following a URL pattern. In this way, one `define server_url.domain condition` definition block can call another condition-related definition block, so that they are in effect nested. See the example in the "define condition" on page 637 definition block topic. Any referenced condition must be valid in a <Forward> layer.

Layer and Transaction Notes

- Layers: <Forward>
- Transactions: all

Example

Define a set of domains that are allowed, and allow access to the OCS only for those domains.

```
define server_url.domain condition allowed
  inventory.example.com
  affinityclub.example.com
end

<Forward>
   condition=!allowed access_server(no)
```

See Also

- Condition: "condition=" on page 113, server_url.domain=
- Definitions: "define url.domain condition" on page 660

define string

Define a named, multi-line character string. Note the following about this gesture:

- Between define string and end, blank lines and comment lines are ignored.
- Lines beginning with > characters contain text that is added to the string; the leading > character is
- ignored.
- Leading white space before the > character is ignored.
- You cannot use a backslash (\) to continue a line. The \ character is treated literally.

A string name can be used as the optional third argument to the exception( ) property. This overrides the format field of the exception. In this usage, the string can contain substitutions, which are expanded when the exception is generated.

Syntax

```
define string string_name
>first_line_of_text
>second_line_of_text

;comments and blank lines ignored
>third_line_of_text
end
```

Layer and Transaction Notes

- Layers: <Admin>, <Cache>, <DNS-Proxy>, <Exception>, <Forward>, <Proxy> , <SSL>, <SSL-Intercept>
- Transactions: HTTP proxy

Example

Return a 200 HTTP response of type text/html where the HTML is defined by the string-definition-name Message. Substitutions of the form $(...) within the string definition are expanded.

```
define string Message
><html>
><head>
><title>Notice</title>
><meta http-equiv=refresh content="10;$(url)">
></head>
><body>
>There are cookies in the lunch room. Help yourself.
></body>
></html>
```

```
end

<Proxy>
  condition=ShouldBeNotified exception(notify,"",Message)
```

See Also

- Properties:"exception()" on page 402

define subnet

Binds a user-defined label to a set of IP addresses or IP subnet patterns. Use a subnet definition label with any of the conditions that test part of the transaction as an IP address, including: "client.address=" on page 87, "proxy.address=" on page 198, "request.header.header_name.address=" on page 221, "request.x_header.header_name.address=" on page 242, and server_url.address=.

The listed IP addresses or subnets are considered to have a Boolean OR relationship, no matter whether they are all on one line or separate lines.

Syntax

```
define subnet label
  {ip_address|ip_address_range|ip_address_wildcards|subnet}
  ...
end
```

where:

- *ip_address*: Client IP address; for example, 10.25.198.0
- *ip_address_range*: IP address range; for example, 192.0.2.0-192.0.2.255
- *ip_address_wildcards*: IP address specified using wildcards in any octet(s); for example, 10.25.*.0 or 10.*.*.0
- *subnet*: Subnet specification; for example, 10.25.198.0/24

Layer and Transaction Notes

- Layers: <Admin>, <Cache>, <Diagnostic>, <DNS-Proxy>, <Exception>, <Forward>, <Proxy> , <SSL>, <SSL-Intercept>
- Transactions: HTTP proxy

Example

Deny access when the client address is not in the defined subnet.

```
define subnet local_net
  1.2.3.4  1.2.3.5 ; can list individual IP addresses
  2.3.4.0/24  2.3.5.0/24  ; or subnets
  2.3.4.0-2.3.4.255  ; or an IP address range
  2.3.*.*  ; or IP address wildcards
end

<Proxy>
  client.address=!local_subnet deny
```

See Also

- Conditions: "client.address=" on page 87, "proxy.address=" on page 198, "request.header.header_name.address=" on page 221, "request.x_header.header_name.address=" on page 242 , server_url.address=
- Information on wildcards: http://www.symantec.com/docs/TECH241521
- Information on IP address ranges: http://www.symantec.com/docs/TECH241929

define url condition

> **Tip:** This definition block is not to be confused with the "url=" on page 298 condition.

Binds a user-defined label to a set of URL prefix patterns for use in a "condition=" on page 113 expression. Using this definition block allows you to quickly test a large set of "url=" on page 298 conditions. Although the define condition definition block could be used in a similar way to encapsulate a set of URL prefix patterns, this specialized definition block provides a substantial performance boost.

The manner in which the URL patterns and any condition expressions are listed is significant. Each line begins with a URL pattern suitable to a "url=" on page 298 condition and, optionally, one or more condition expressions, all of which have a Boolean AND relationship. Each line inside the definition block is considered to have a Boolean OR relationship with other lines in the block.

Syntax

```
define url condition label1
  url_prefix_pattern [condition_expression,...]
  ...
end
```

where:

- *label*: User-defined identifier for a prefix condition definition.
- *url_prefix_pattern*: URL pattern that includes at least a portion of *scheme://host:port/path*

  where:

  - scheme–URL scheme (http, https, ftp, mms, or rtsp) followed by a colon (:).
  - host–Host name or IP address, optionally preceded by two forward slashes (//). Host names must be complete; for example, url=http://www will fail to match a URL such as http://www.example.com. This use of a complete host instead of simply a domain name (such as example.com) marks the difference between the prefix and domain condition definition blocks.
  - port–Port number between 1 and 65535.
  - path–Forward slash (/) followed by one or more full directory names.

  Accepted prefix patterns include the following:

  - scheme://host
  - scheme://host:port

- scheme://host:port/path
- scheme://host/path
- //host
- //host:port
- //host:port/path
- //host/path
- host
- host:port
- host:port/path
- host/path

- *condition_expression*: Condition expression using any of the conditions available in a rule. For more information, see "Conditions" on page 65. The layer and timing restrictions for the defined condition depend on the layer and timing restrictions of the contained expressions.

  "condition=" on page 113 is one of the expressions that can be included in the body of a `define url condition=` definition block, following a URL pattern. In this way, one prefix definition block can call another condition-related definition block, so that they are in effect nested. See the example in "define condition" on page 637.

Layer and Transaction Notes

- Layers: `<Cache>`, `<Exception>`, `<Proxy>` , `<SSL>`, `<SSL-Intercept>`
- Transactions: HTTP proxy

Example

See the example in "define action" on page 628.

See Also

- Conditions: "condition=" on page 113, "url=" on page 298
- Definitions:"define url.domain condition" on the next page

define url.domain condition

Binds a user-defined label to a set of domain-suffix patterns for use in a "condition=" on page 113 expression. Using this definition block allows you to test a large set of server_url.domain= conditions very quickly. Although the define condition definition block could be used in a similar way to encapsulate a set of domain suffix patterns, this specialized definition block provides a substantial performance boost.

For domain and URL definitions, the manner in which the URL patterns and any condition expressions are listed is significant. Each line begins with a URL pattern and, optionally, one or more condition expressions, all of which have a Boolean AND relationship. Each line inside the definition block is considered to have a Boolean OR relationship with other lines in the block.

Syntax

```
define url.domain condition label1
  domain_suffix_pattern [condition_expression,...]
  ...
end
```

where:

- *label*: User-defined identifier for a domain condition definition. Used in "condition=" on page 113.
- *domain_suffix_pattern*: URL pattern suitable to "user.domain=" on page 316 that includes a domain name (domain), as a minimum. See "url=" on page 298 for a complete description.
- *condition_expression*: Condition expression using any of the conditions available in a rule. For more information, see "Conditions" on page 65. The layer and timing restrictions for the defined condition depend on the layer and timing restrictions of the contained expressions.

  "condition=" on page 113 is one of the expressions that can be included in the body of a `define url.domain condition=` definition block, following a URL pattern. In this way, one domain definition block can call another condition-related definition block, so that they are in effect nested. See the example in "define condition" on page 637.

Layer and Transaction Notes

- Layers: `<Cache>`, `<Exception>`, `<Proxy>` , `<SSL>`, `<SSL-Intercept>`
- Transactions: HTTP proxy

Example

Allow requests to the specified sites.

```
define url.domain condition allowed_sites
 inventory.example.com method=GET
 affinityclub.example.com
end
```

```
<proxy>
 condition=allowed_sites allow
```

See Also

- Conditions: "condition=" on page 113, "server_url=" on page 269
- Definitions: "define url condition" on page 658, "define server_url.domain condition" on page 652

define url_rewrite

Defines rules for rewriting URLs in HTTP responses. The URLs are either included in HTTP response headers—Location, Content-Location, and Refresh—or embedded in tags within HTML, CSS, JavaScript, XML, JSON, XHTML, and ASX documents. In addition to rewriting URLs, you can also rewrite arbitrary JavaScript.

This transformer takes effect only if it is invoked by a transform action in "define action" on page 628, and that block is called from "action()" on page 337. If policy includes more than one `action(yes)` property containing url_rewrite rules, only the last rewrite action takes effect.

For each URL found within an HTTP response, the url_rewrite transformer converts the URL into absolute form, and applies all rewrite statements to the URL being considered. If it finds a match, it replaces the substring in the rule. Each statement is applied to pre-transformed content (that is, they do not transform URLs that match as a result of a previous rewrite statement), except for `rewrite_url_prefix`, which is cumulative (transformed URLs are subject to subsequent matching rewrite statements). See "How Multiple Rewrite Statements Affect Transformation" on the facing page for examples.

> **Note:** Pages served over an HTTPS tunneled connection are encrypted; thus, URLs embedded within them cannot be rewritten.

Transformed content is not cached (although the original object can be cached), in contrast with content that has been sent to a content scanning server. This means that any transformer can be safely triggered based on any condition, including client identity and time of day.

Syntax

```
define url_rewrite transformer_id
  rewrite_statement "replacement" "match"
  ...
end
```

where:

- *`transformer_id`*: User-defined identifier for a transformer definition block. Used to invoke the transformer using the transform action in a define action block.
- *`rewrite_statement "replacement" "match"`*: Rewrite rule comprising a statement followed by the replacement string and the string to match in the URL. Matching is case-insensitive. Supported rules follow:

> **Note:** You can specify a port for *server_url_substring*; however, if the port is 80 or 443, do not specify the port. For traffic arriving on ports 80 and 443, the ProxySG appliance removes the port numbers from the URL before the `url_rewrite` policy is applied and policy will not match if these ports are specified. For example, `rewrite_url_prefix "https://internal.example.org/" "https://www.example.com:443"` will not match and the URL will not be rewritten. Instead, write `rewrite_url_prefix "https://internal.example.org/" "https://www.example.com"` to match and rewrite the URL.

- `rewrite_url_substring "client_url_substring" "server_url_substring"`

  This rule matches the specified *server_url_substring* in the URL and replaces it with the specified *client_url_substring*. The comparison is done against original normalized URLs embedded in the document.

- `rewrite_url_prefix "client_url_substring" "server_url_substring"`

  This rule looks for the specified *server_url_substring* in the URL prefix string and replaces it with the specified *client_url_substring*. The comparison is done against original normalized URLs embedded in the document.

- `rewrite_script_substring "client_substring" "server_substring"`

  This rule matches the specified *server_substring* in JavaScript files and content inside the `<script> </script>` tags in HTML files. The substrings can be of any pattern inside any unrecognized tag or attribute, including those that cannot validly contain URLs. Matches are replaced with the specified *client_substring*.

- `rewrite_script_regex "client_substring_with_back_ref" "server_regex_substring"`

  This rule performs a regular expression match for the specified *server_regex_substring* in JavaScript files and content inside the `<script> </script>` tags in HTML files. Matches are replaced with the specified string including back references, *client_substring_with_back_ref*.

  Note the following about regular expression usage in the replacement string:

  - Perl's regular expression text patterns are supported
  - `\1` and `$1` back references are not supported
  - Escape special characters with a backslash; to escape double quotation marks, use `\x22`

  See "Regex Reference" on page 703 for more information on writing regular expressions.

How Multiple Rewrite Statements Affect Transformation

For `rewrite_url_substring`, `rewrite_url_prefix`, and `rewrite_script_substring`, each statement applies to the page contents separately. Consider the following example:

```
define url_rewrite rewrite1
  rewrite_url_prefix "http://example" "http://10.1.1.1"
  rewrite_url_prefix "http://server" "http://example"
end
```

This policy makes the following transformations in a page containing `http://10.1.1.1` and `example`:

- Per the first rewrite statement, `http://10.1.1.1` is transformed to `http://example`
- Per the second rewrite statement, `http://example` is transformed to `http://server`

The second `rewrite` statement does not transform the `http://example` resulting from the first `rewrite` statement.

For `rewrite_script_regex`, each statement applies cumulatively. Consider the following example:

```
define url_rewrite rewrite2
  rewrite_script_regex "example" "10\.1\.1\.1"
  rewrite_script_regex "server" "example"
end
```

This policy makes the following transformations in a page containing `10.1.1.1` and `example`:

- Per the first `rewrite` statement, `10.1.1.1` is transformed to `example`
- Per the second `rewrite` statement:
  - `example` (in pre-transformed content) is transformed to `server`
  - `example` (transformed by the first rewrite statement) is transformed again to `server`

Layer and Transaction Notes

- Layers: `<Proxy>`

Example

For requests to the specified URL, transform the response URL in Referer headers.

```
define url_rewrite example_portal
  rewrite_url_prefix "http://www.example.com/" "http://www.server1.example.com/"
end

define action example_server_portal
  ; request rewriting
  rewrite( url, "^http://www\.example\.com/(.*)", "http://www.server1.example.com/$(1)" )
  rewrite( request.header.Referer, "^http://www\.example\.com/(.*)",
"http://www.server1.example.com/$(1)" )

  ; response rewriting
  transform example_portal
```

```
end

<Proxy> ; apply rewrites for example.com
  url=example.com/ action.example_server_portal(yes)
```

See Also

- Actions: "transform" on page 622
- Definitions: "define action" on page 628, "define active_content" on page 630
- Properties: "action()" on page 337, "transform.data_type()" on page 581

define xml_*schema-type*_schema

Defines an object referable by "http.request.detection.xml.schema.schema_name()" on page 460 to validate the request XML body and take the specified action.

Syntax

```
define xml_schema-type_schema
 xml_elements
end
```

where:

- *schema-type*: One of the following:
  - dtd–Define the document structure with a list of legal elements and attributes.
  - xsd–Specify how to formally describe the elements in XML.

Layer and Transaction Notes

- Layers: <Admin>, <Cache>, <DNS-Proxy>, <Exception>, <Forward>, <Proxy> , <SSL>, <SSL-Intercept>
- Transactions: HTTP proxy

Example

Allow XML with the specified schema definition and log the request.

```
define xml_xsd_schema example-1
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="order">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="description" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
end

<proxy>
  http.request.detection.xml.schema.example-1(monitor)
```

restrict dns

This definition restricts DNS lookups and is useful in installations where access to DNS resolution is limited or problematic. The definition has no name because it is not directly referenced by any rules. It is global to policy evaluation and intended to prevent any DNS lookups caused by policy. It does not suppress DNS lookups that might be required to make upstream connections.

If the domain specified in a URL matches any of the domain patterns specified in *domain_list*, no DNS lookup is done for any "category=" on page 85, url=, url.address=, url.domain=, or url.host= test.

The special domain "." matches all domains, and therefore can be used to restrict all policy-based DNS lookups.

If a lookup is required to evaluate the trigger, the trigger evaluates to false.

A `restrict dns` definition may appear multiple times in policy. The compiler attempts to coalesce these definitions, and may emit various errors or warnings while coalescing if the definition is contradictory or redundant.

Syntax

```
restrict dns
  list_of_restricted_domains
 except
  list_of_exempted_domains
end
```

where:

- *list_of_restricted_domains*: Domains for which DNS lookup is restricted.
- *list_of_exempted_domains*: Domains exempt from the DNS restriction. Policy is able to use DNS lookups when evaluating policy related to these domains.

Layer and Transaction Notes

- Layers: `<Admin>`, `<Cache>`, `<Diagnostic>`, `<DNS-Proxy>`, `<Exception>`, `<Forward>`, `<Proxy>` , `<SSL>`, `<SSL-Intercept>`
- Transactions: all

Example

Restrict DNS resolution to all but the specified domain.

```
restrict dns
  domain1
  domain2
   ...
except
```

```
  other_domain
end
```

See Also

- Conditions: "category=" on page 85, "server_url=" on page 269, "url=" on page 298
- Definitions: "restrict rdns" on the facing page

restrict rdns

This definition restricts reverse DNS lookups and is useful in installations where access to reverse DNS resolution is limited or problematic. The definition has no name. It is global to policy evaluation and is not directly referenced by any rules.

If policy includes this definition, RDNS lookups are allowed regardless of what is specified in the `#(config) policy restrict-rdns` CLI command.

If the requested URL specifies the host in IP form, no reverse DNS lookup is performed to match any "category=" on page 85, url=, url.domain=, or url.host= condition.

The special token all matches all subnets, and therefore can be used to restrict all policy-based reverse DNS lookups.

If a lookup is required to evaluate the trigger, the trigger evaluates to false.

A restrict rdns definition may appear multiple times in policy. The compiler attempts to coalesce these definitions, and may emit various errors or warnings while coalescing if the definition is contradictory or redundant.

Syntax

```
restrict rdns
  restricted_subnet_list
  restricted_ip_address_wildcards
  restricted_ip_address_range
except
  exempted_subnet_list
  exempted_ip_address_wildcards
  exempted_ip_address_range
end
```

where:

- *restricted_subnet_list*: Subnets for which reverse DNS lookup is restricted.
- *restricted_ip_address_wildcards*: IP address (specified using wildcards in any octet(s); for example, 10.25.*.0 or 10.*.*.0) for which reverse DNS lookup is restricted.
- *restricted_ip_address_range*: Range of IP addresses (for example, 192.0.2.0-192.0.2.255) for which reverse DNS lookup is restricted.
- *exempted_subnet_list*: Subnets exempt from the reverse DNS restriction. Policy is able to use reverse DNS lookups when evaluating policy related to these subnets.
- *restricted_ip_address_wildcards*: IP address exempt from the reverse DNS restriction, specified using wildcards in any octet(s).
- *restricted_ip_address_range*: Range of IP addresses exempt from the reverse DNS restriction.

Layer and Transaction Notes

- Layers: `<Admin>`, `<Cache>`, `<Diagnostic>`, `<DNS-Proxy>`, `<Exception>`, `<Forward>`, `<Proxy>` , `<SSL>`, `<SSL-Intercept>`
- Transactions: all

Example

Restrict reverse DNS resolution for all but the 10.10.100.0/24 subnet:

```
restrict rdns
  all
except
  10.10.100.0/24
end
```

See Also

- Conditions: "category=" on page 85, "server_url=" on page 269, "url=" on page 298
- Definitions: "restrict dns" on page 667
- Information on wildcards: http://www.symantec.com/docs/TECH241521
- Information on IP address ranges: http://www.symantec.com/docs/TECH241929

Variables

Variables represent values that you set and test. You can override a variable's default value and test the new overridden value in conditions. You can also override and test the variable multiple times in policy, which has a cumulative effect when subsequent values override earlier ones.

A variable test can be in the same layer where the variable is set, or it can be in a different layer. The only restriction on setting variables more than once in policy is that all condition tests must occur later than the overrides in policy evaluation order. See "Identify Warnings and Errors in Variables Policy" on the next page for some of the compilation errors that could occur if policy violates this restriction.

Variables Syntax and Usage

You must set a variable before you test it in conditions because actions that modify the same transaction field are executed at the same checkpoint in policy source code. To test a variable that is set in a different layer, the layer including the variable test must precede the layer where the variable is set.

To set a variable—that is, override the default value—use the form:

`variable.`*`variable_name`*`(`*`pattern_expression`*`)`

The following example sets a Threat Risk Level of 5:

```
variable.url.threat_risk.effective_level(5)
```

To test a variable, use the form:

`variable.`*`variable_name`*`=`*`pattern_expression`*

The following example tests for Threat Risk Levels 7 through 9:

```
variable.url.threat_risk.effective_level=7..9
```

A variable accepts one or more of the following pattern expressions:

- String; enclose a string argument in quotation marks if it contains whitespace or special characters, such as `variable.time_quota_name("Monthly Quota")`.
- Substitution string, such as `variable.volume_quota_limit("$(sr-bytes)")`.
- Integer, such as `variable.url.threat_risk.effective_level(10)`.

  > **Note:** Variable tests accept integer ranges, but you cannot set a variable to a range.

- Single arguments, such as `variable.guest_volume_quota_exceeded(true)`.

Use Variables in Access Logs and in Substitutions

You can include variables in access log fields. Use the form:

`$(variable.variable_name)`

You can also include variables in substitutions in CPL and exception pages.

For details on modifying access log formats and using substitutions, refer to *ProxySG Log Fields and CPL Substitutions Reference*: https://techdocs.broadcom.com/content/broadcom/techdocs/us/en/symantec-security-software/web-and-network-security/proxysg/7-2/proxysg-log-fields-and-substitutions.html

Rules for Writing Variables Policy

To prevent compilation errors related to variables timing, adhere to the following rules:

- Precede all tests for a variable with all overrides for the variable in policy evaluation order.

  > **Note:** Policy evaluation order depends on the order of policy compilation and the order in policy source files.

- Use variables only in the same layer type for which they are already defined. For example, all instances of"server.certificate.hostname.threat_risk.effective_level" on page 678 must be in the `<SSL-Intercept>` layer.

Identify Warnings and Errors in Variables Policy

Refer to the following table to identify and troubleshoot policy variables warnings and errors that could occur at compilation time.

Warning/Error Text	Warning/Error Meaning	Troubleshoot the Warning/Error
`Warning: Late condition guards early action`	The condition on a rule that sets a variable is late with respect to the property or action guarded by the variable condition.	Revise policy to use a different condition to perform the required action.
`Error: Unknown variable At 'A $(...) substitution, somewhere in the CPL source code'`	Policy includes a quota substitution variable but the quota library is disabled.	Enable the quota library. Issue the following CLI command: # `(config)`**`policy quota`**
`Error: variable not defined`	Policy includes quota variables but the quota library is disabled.	

Warning/Error Text	Warning/Error Meaning	Troubleshoot the Warning/Error
`Error: variable tested and set in the same layer`	The variable is set and tested in the same policy layer.	Set the variable in a policy layer that precedes, in policy evaluation order, the layer containing the variable condition. You can use Visual Policy Manager (VPM) layers if they are available. Re-order the layers as needed.
`Error: variable tested without being set in a previous layer`	A variable test exists in policy but the variable was not set.	
`Error: variable modified after being observed`	The variable is set after the condition test according to policy evaluation order.	
`Error: variable set in multiple layer types` or `Error: variable set and referenced in different layer types`	The variable is set and overridden in different layer types, or it is set and tested in different layer types.	Include all instances of the variable in the same layer type.

Use Policy Tools to Analyze Variables Policy

Examine policy to determine which variables are set and which variables are executed for specific transactions. For instructions on tracing policy, see "Testing and Troubleshooting" on page 732.

> **Note:** Policy trace displays policy in the builtin-prolog and builtin-epilog source files, which are internal to the appliance and not user-configurable.

Determine Which Variables are Set in Policy

For each transaction recorded in the trace, between the miss/match/late rule evaluations and connection details, look for instances of the line Assigned values of transaction variables. This line precedes a list of all variables, which indicates whether each variable is initialized or executed for the transaction.

If a variable is initialized for a transaction, the trace displays the default or overridden value. If a variable does not have to be initialized for a transaction, the trace says `(value undetermined)`.

Consider the following example:

```
Assigned values of transaction variables:
 dns.request.threat_risk.effective_level=(value undetermined)
 url.threat_risk.effective_level=7
 ...
 time_quota_enforced=FALSE
```

In the previous example:

- `dns.request.threat_risk.effective_level` was not initialized for the transaction.
- `url.threat_risk.effective_level` was initialized and the transaction URL's Threat Risk Level (according to the WebPulse service or as a result of one or more overrides) is 7.
- `time_quota_enforced` was initialized but time quota policy is not enabled for the transaction.

The policy trace includes quota variables only if the quota library is enabled. To enable the quota library, issue the `#(config)`**`policy quota`** command in the CLI.

Determine Which Variables are Executed

For each transaction recorded in the trace, look for policy rules preceded by `MATCH`.

Consider the following example:

```
start transaction -------------------
transaction ID=2314 type=http.proxy
...

MATCH:          url.domain=<domain> variable.url.threat_risk.effective_level(9)
...
     <Proxy>
MATCH:          DENY variable.url.threat_risk.effective_level=8..9

Assigned values of transaction variables:
  dns.request.threat_risk.effective_level=(value undetermined)
  url.threat_risk.effective_level=9
  request.header.Referer.url.threat_risk.effective_level=(value undetermined)
  server_url.threat_risk.effective_level=(value undetermined)
  server.certificate.hostname.threat_risk.effective_level=(value undetermined)
```

In the previous example:

- The `DENY variable.url.threat_risk.effective_level=8..9` rule was executed.
- `url.threat_risk.effective_level` was initialized and the transaction URL's Threat Risk Level (according to the WebPulse service or as a result of one or more overrides) is 9, which matches the `8..9` integer range.

Determine How Often Variable Policy is Executed

To determine the frequency with which rules that include a variable match proxied requests, or to verify if such a rule is ever executed, browse to the appliance's policy coverage page:

https://*appliance_IP_address*:8082/policy/coverage

For details on policy coverage, refer to TECH241425:

http://www.symantec.com/docs/TECH241425

Important Notes about Threat Risk Level Variables

- When writing Threat Risk Levels policy, you can use level 0 to override the WebPulse-reported level for a URL; however, keep in mind that the Management Console (**Statistics > Threat Risk Details**) reports level 0 as Low.

  For details on Threat Risk Levels, refer to the "Analyzing the Threat Risk of a URL" chapter in the *SGOS Administration Guide*, and the *Visual Policy Manager Reference*.

Important Notes about Quota Variables

- Time and volume quota policy variables are available when the quota library is enabled. To enable the quota library, issue the `#(config)policy quota` command in the CLI.

  For details on time and volume quotas, refer to the "Filtering Web Content" chapter in the *SGOS Administration Guide*.

- Although you can write policy to test quota variables, they are in fact input variables underlying the quota settings you defined in the VPM. Thus, depending on your requirements, they might not be useful or necessary to test. The existence of these variables in policy ensures that quota policy is in effect for matching transactions.

- The quotas implementation uses the following variables internally; do not include them in your policy:

  - `guest_time_recorded`
  - `guest_volume_quota_exceeded`
  - `guest_volume_quota_warning`
  - `guest_volume_quota_warning_exists`
  - `time_recorded`
  - `user_authentication_is_on`
  - `volume_quota_exceeded`
  - `volume_quota_warning`
  - `volume_quota_warning_exists`

dns.request.threat_risk.effective_level

Override the Threat Risk Level that WebPulse returns for the host associated with the DNS request.

The default value is the WebPulse value. The value range is 0..10.

Syntax

```
variable.dns.request.threat_risk.effective_level(integer)
variable.dns.request.threat_risk.effective_level=integer_or_range
```

where:

- *integer*: Value from 0 to 10.
- *integer_or_range*: Numeric range as described in " Variables Syntax and Usage" on page 671.

Layer and Transaction Notes

- Layers: <DNS-Proxy>
- Transactions: DNS proxy

Example

Define countries based on geographical locations of IP addresses. For DNS requests matching country_rule, set a Threat Risk Level of 10. Trace DNS requests when the URL's Threat Risk Level is set to 10.

```
define condition country_rule
  supplier.country=list_of_country_codes
end

<dns-proxy>
  condition=country_rule variable.dns.request.threat_risk.effective_level(10)

<dns-proxy>
  variable.dns.request.threat_risk.effective_level=10 trace.request(yes)
```

See Also

- Conditions: "url.threat_risk.level=" on page 310

request.header.Referer.url.threat_risk.effective_level

Override the Threat Risk Level that WebPulse returns for the URL identified in the Referer HTTP header of the current request.

The default value is the WebPulse value. The value range is 0..10.

Syntax

```
variable.request.header.Referer.url.threat_risk.effective_level(integer)
variable.request.header.Referer.url.threat_risk.effective_level=integer_or_range
```

where:

- *integer*: Value from 0 to 10.
- *integer_or_range*: Numeric range as described in " Variables Syntax and Usage" on page 671.

Layer and Transaction Notes

- Layers: <Proxy>
- Transactions: all

Example

for all requests with the specified Referer header value,; set the Threat Risk Level to 0. when the Threat Risk Level of the Referer URL is set to 8 through 10,

; log the transaction to the specified access log

```
<proxy>
  request.header.Referer.url=URL variable.request.header.Referer.url.threat_risk.effective_level(0)

<proxy>
  variable.request.header.Referer.url.threat_risk.effective_level=8..10 access_log.<log_name>(yes)
```

See Also

- Conditions: "url.threat_risk.level=" on page 310

server.certificate.hostname.threat_risk.effective_level

Override the Threat Risk Level that WebPulse returns for the hostname associated with the server certificate.

The default value is the WebPulse value. The value range is 0..10.

Syntax

```
variable.server.certificate.hostname.threat_risk.effective_level(integer)
variable.server.certificate.hostname.threat_risk.effective_level=integer_or_range
```

where:

- *integer*: Value from 0 to 10.
- *integer_or_range*: Numeric range as described in " Variables Syntax and Usage" on page 671.

Layer and Transaction Notes

- Layers: <SSL-Intercept>
- Transactions: all

Example

Set the specified Threat Risk Level for the requests.

```
; condition defining when content filtering category is unavailable
; for the hostname extracted from x.509 certificate

define condition unavailable
  server.certificate.hostname.category=unavailable
end

; for requests matching previous condition, set Threat Risk Level of 0
<ssl-intercept>
  condition=unavailable variable.server.certificate.hostname.threat_risk.effective_level(0)

; execute specified action when server cert hostname is Threat Risk Level 0
<ssl-intercept>
  variable.server.certificate.hostname.threat_risk.effective_level=0 action.log_unavailable_category
(yes)

; write the specified message to the event log
define action log_unavailable_category
  log_message( "Content filtering unavailable to test $(server.certificate.hostname)" )
end
```

See Also

- Conditions: "url.threat_risk.level=" on page 310

server_url.threat_risk.effective_level

Override the Threat Risk Level that WebPulse returns for a request URL that was possibly rewritten.

The default value is the WebPulse value. The value range is 0..10.

Syntax

```
variable.server_url.threat_risk.effective_level(integer)
variable.server_url.threat_risk.effective_level=integer_or_range
```

where:

- *integer*: Value from 0 to 10.
- *integer_or_range*: Numeric range as described in " Variables Syntax and Usage" on page 671.

Layer and Transaction Notes

- Layers: <Forward>
- Transactions: all

Example

Set the specified Threat Risk Level for the requests.

```
; when a P2P client is in use, set the server URL to Threat Risk Level to 5
; but when the P2P client is BitTorrent, set the server URL Threat Risk Level to 7
<forward>
  p2p.client=yes variable.server_url.threat_risk.effective_level(5)
  p2p.client=bittorrent variable.server_url.threat_risk.effective_level(7)

; log requests with server URL Threat Risk Levels 5 through 7 to the specified log
<forward>
  variable.server_url.threat_risk.effective_level=5..7 access_log.<log_name>(yes)
```

See Also

- Conditions: "url.threat_risk.level=" on page 310

time_quota_enforced

When time quota policy exists in the VPM, this variable is set and determines whether the time quota policy is evaluated and enforced. The default value is false.

If other time quota variables (discussed in this chapter) are assigned appropriate values, then time quota policy is in effect.

Syntax

```
variable.time_quota_enforced(true|false)
variable.time_quota_enforced=true|false
```

Layer and Transaction Notes

- Layers: <Proxy>
- Transactions: all

Example

Enforce time quota for the specified users.

```
; define condition for specified user
define condition user
  realm=auth_realm user=user_name
end

; when user condition matches, enforce time quota for user access to YouTube
; time quota is measured and refreshed daily
; user is allowed 60 minutes a day
; user is warned when they reach 75% or 45 minutes of usage per day
<proxy>
  condition=user url.domain="youtube.com" variable.time_quota_enforced(true) variable.time_quota_name
("YouTube") variable.time_quota_frequency("daily") variable.time_quota_limit(60) variable.time_quota_
warning_limit(45)
```

time_quota_frequency

When time quota policy exists in the VPM, this variable is set and specifies the quota's validity period. The quota is refreshed at the end of the specified interval. The default value is `hourly`.

Syntax

```
variable.time_quota_frequency(weekly|daily|hourly)
variable.time_quota_frequency=weekly|daily|hourly
```

Layer and Transaction Notes

- Layers: `<Proxy>`
- Transactions: all

Example

See the example for "time_quota_enforced" on the previous page.

time_quota_limit

When time quota policy exists in the VPM, this variable is set and specifies the maximum number of minutes that the quota allows. The default value is 0.

Syntax

```
variable.time_quota_limit(integer)
variable.time_quota_limit=integer
```

where:

- *integer*: Value from 0 to 10.

> **Note:** Although this variable accepts any integer value, it is only useful when it is:
> - Set to a value other than the default.
> - Greater than the value of"time_quota_warning_limit" on page 685, because the quotas implementation displays a warning message to users when they reach a specified ratio of the quota. For example, if you set this variable to 60, set "time_quota_warning_limit" on page 685 to a value less than 60.
> - Less than the equivalent value of "time_quota_frequency" on the previous page, because time quota tracking refreshes at the end of the interval that variable specifies. For example, if you set"time_quota_frequency" on the previous page to `hourly`, set this variable to a value no greater than 60.

Layer and Transaction Notes

- Layers: `<Proxy>`
- Transactions: all

Example

See the example for "time_quota_enforced" on page 681.

See Also

- Variables: "time_quota_warning_limit" on page 685

time_quota_name

In the VPM, you can define multiple quotas with unique names, but only one time quota can be in effect per transaction. This variable records the name of the quota in effect for the associated transaction. The default value is ignored.

Syntax

```
variable.time_quota_name(string)
variable.time_quota_name=string
```

where:

- *string*: Name of the quota.

> **Note:** For this variable to be useful, set it to a value other than the default.

Layer and Transaction Notes

- Layers: <Proxy>
- Transactions: all

Example

See the example for "time_quota_enforced" on page 681.

time_quota_warning_limit

When time quota policy exists in the VPM, this variable is set and specifies the warning threshold, expressed in minutes. When users reach this threshold, they receive a warning message. The default value is 0.

Syntax

```
variable.time_quota_warning_limit(integer)
variable.time_quota_warning_limit=integer
```

where:

- *integer_or_range*: Numeric range as described in " Variables Syntax and Usage" on page 671.

> **Note:** Although this variable accepts any integer value, it is only useful when it is:
> - Set to a value other than the default.
> - Less than the value of "time_quota_limit" on page 683, because the quotas implementation displays the warning message to users when they reach a specified ratio of the quota. For example, if you set"time_quota_limit" on page 683 to 60, set this variable to a value less than 60.

Layer and Transaction Notes

- Layers: <Proxy>
- Transactions: all

Example

See the example for "time_quota_enforced" on page 681.

See Also

- Variables: "time_quota_limit" on page 683

url.threat_risk.effective_level

Override the Threat Risk Level that WebPulse returns for the request URL, excluding URLs that might have been rewritten.

The default value is the WebPulse value. The value range is 0..10.

Syntax

```
variable.url.threat_risk.effective_level(integer)
variable.url.threat_risk.effective_level=integer_or_range
```

where:

- *integer*: Value from 0 to 10.
- *integer_or_range*: Numeric range as described in " Variables Syntax and Usage" on page 671.

Layer and Transaction Notes

- Layers: <Proxy>
- Transactions: all

Example

Set the Threat Risk Levels for URLs from the specified domains and deny requests where the URL's Threat Risk Level is set to 8 or 9 .

```
<proxy>
  url.domain=domain1 variable.url.threat_risk.effective_level(9)
  url.domain=domain2 variable.url.threat_risk.effective_level(8)

<proxy>
  variable.url.threat_risk.effective_level=8..9 deny
```

See Also

- Conditions: "url.threat_risk.level=" on page 310

volume_quota_enforced

When volume quota policy exists in the VPM, this variable is set and determines whether the volume quota policy is evaluated and enforced. The default value is false.

If other volume quota variables (discussed in this chapter) are assigned appropriate values, then the volume quota policy is in effect.

Syntax

```
variable.volume_quota_enforced(true|false)
variable.volume_quota_enforced=true|false
```

Layer and Transaction Notes

- Layers: <Proxy>
- Transactions: all

Example

Guest users are subject to a volume quota that is measured and refreshed hourly. They can download 50 MB or 5242880 bytes an hour and are warned when they reach 75% of the quota or 3932160 bytes in the hour

```
<proxy>
  user.is_guest=yes variable.volume_quota_enforced(true) variable.volume_quota_name("guest")
variable.volume_quota_frequency("hourly") \
    variable.volume_quota_limit(5242880) variable.volume_quota_warning_limit(3932160)
```

volume_quota_frequency

When volume quota policy exists in the VPM, this variable is set and specifies the quota's validity period. The quota is refreshed at the end of the specified interval. The default value is `hourly`.

Syntax

```
variable.volume_quota_frequency(daily|hourly|weekly)
variable.volume_quota_frequency=daily|hourly|weekly
```

Layer and Transaction Notes

- Layers: `<Proxy>`
- Transactions: all

Example

See the example for "volume_quota_enforced" on the previous page.

volume_quota_limit

When volume quota policy exists in the VPM, this variable is set and specifies the maximum number of bytes that the quota allows. The default value is 0.

Syntax

```
variable.volume_quota_limit(integer)
variable.volume_quota_limit=integer
```

where:

- *integer*: Value from 0 to 10.

> **Note:** Although this variable accepts any integer value, it is only useful when it is:
> - Set to a value other than the default.
> - Greater than the value of `volume_quota_warning_limit`, because the quotas implementation displays a warning message to users when they reach a specified ratio of the quota. For example, if you set this variable to 5242880, set `volume_quota_warning_limit` to a value less than 5242880.

Layer and Transaction Notes

- Layers: `<Proxy>`
- Transactions: all

Example

See the example for "volume_quota_enforced" on page 687.

See Also

- Variables: "volume_quota_warning_limit" on page 691

volume_quota_name

In the VPM, you can define multiple quotas with unique names, but only one volume quota can be in effect per transaction. This variable records the quota name in effect for the associated transaction. The default value is ignored.

Syntax

```
variable.volume_quota_name(string)
variable.volume_quota_name=string
```

where:

- *string*: Name of the quota.

> **Note:** For this variable to be useful, set it to a value other than the default.

Layer and Transaction Notes

- Layers: <Proxy>
- Transactions: all

Example

See the example for "volume_quota_enforced" on page 687.

volume_quota_warning_limit

When volume quota policy exists in the VPM, this variable is set and specifies the warning threshold, expressed in bytes. When users reach this threshold, they receive a warning message. The default value is 0.

Syntax

```
variable.volume_quota_warning_limit(integer)
variable.volume_quota_warning_limit=integer
```

where:

- *integer*: Value from 0 to 10.

> **Note:** Although this variable accepts any integer value, it is only useful when it is:
> - Set to a value other than the default.
> - Less than the value of "volume_quota_limit" on page 689, because the quotas implementation displays the warning message to users when they reach a specified ratio of the quota. For example, if you set "volume_quota_limit" on page 689 to 5242880), set this variable to a value less than 5242880.

Layer and Transaction Notes

- Layers: <Proxy>
- Transactions: all

Example

See the example for "volume_quota_enforced" on page 687.

See Also

- Variables: "volume_quota_enforced" on page 687

Testing and Troubleshooting

If you are experiencing problems with your policy files or would like to monitor policy evaluation, you can do the following to troubleshoot policy:

- Policy trace–Allows you to examine how the appliance policy is applied to a particular request. This is appropriate if you want to monitor policy evaluation for single or multiple transactions over shorter periods of time.
- Policy coverage–Reports on the rules and objects that match user requests processed through the appliance's current policy. This is appropriate if you want to monitor evaluation for multiple transactions over longer periods of time.

Overview of Policy Tracing

Tracing allows you to examine how the appliance policy is applied to a particular request. To configure tracing in a policy file, you use several policy language properties to enable tracing, set the verbosity level, and specify the path for output. Using appropriate conditions to guard the tracing rules, you can be specific about the requests for which you gather tracing information.

> **Note:** Use policy tracing for troubleshooting only. Tracing is best used temporarily for troubleshooting, while the "log_message()" on page 608 action is best for on-going monitoring. If tracing is enabled in a production setting, appliance performance degrades. After you complete troubleshooting, be sure to remove policy tracing.

CPL provides the following trace-related properties:

- "trace.request()" on page 579–Enables tracing and includes a description of the transaction being processed in the trace. No trace output is generated if this is set to no.
- "trace.destination()" on page 576–Directs the trace output to a user-named trace log.

In addition to policy tracing, you can report on the policy rules that are used in transactions. Code coverage shows the frequency with which certain pieces of policy are matched. (A 'piece' of policy specifically means any place in CPL where a condition, layer, or filter is evaluated.) With this information, policy can be reordered for better performance or deleted if policy is not useful.

Enabling Request Tracing

Use the "trace.request()" on page 579 property to enable request tracing. Request tracing logs a summary of information about the transaction: request parameters, property settings, and the effects of all actions taken. This property uses the following syntax:

```
trace.request(yes|no)
```

where:

- yes: Generate full trace details for the current request.
- no: (Default behavior) Do not generate trace output.

Example

Enable full tracing information for all transactions:

```
<cache>
  trace.request(yes)
```

Configuring the Path

Use the "trace.destination()" on page 576 property to configure where the appliance saves trace information. The trace destination can be set and reset repeatedly. It takes effect (and the trace is actually written) only when the appliance has finished processing the request and any associated response. Trace output is saved to an object that is accessible using a console URL in the following form:

https://*appliance_IP_address*:8081/Policy/Trace/*path*

where path is, by default, default_trace.html. This property allows you to change the destination. The property uses the following syntax:

`trace.destination(path)`

where:

- *path*: By default, the path is default_trace.html. You can change *path* to a filename or directory path, or both. If only a directory is provided, the default trace filename is used.

You can view policy statistics through the Management Console (**Statistics > Advanced > Policy > List of policy URLs**).

Example

Two destinations are configured for policy tracing information:

```
<Proxy>
  client.address=10.25.0.0/16 trace.destination(internal_trace.html)
  client.address=10.0.0.0/8 trace.destination(external_trace.html)
```

The console URLs for retrieving the information are:

https://*appliance_IP_address*:8081/Policy/Trace/internal_trace.html

https://*appliance_IP_address*:8081/Policy/Trace/external_trace.html

Using Trace Information to Improve Policies

To help you understand tracing, this section shows annotated trace output. These traces show the evaluation of specific requests against a particular policy.

For information on the trace result late, refer to TECH245782:

http://www.symantec.com/docs/TECH245782

Example

Refer to the example below. It is intended to be an illustration of most aspects of policy trace output. The example addresses the following policy requirements:

- DNS lookups are restricted except for a site being hosted.
- There is no access to reverse DNS so that is completely restricted.
- Any requests not addressed to the hosted site either by name or subnet should be rejected.
- FTP POST requests should be rejected.
- Request URLs for the hosted site are to be rewritten and a request header on the way into the site.

```
; DNS lookups are restricted except for one site that is being hosted
restrict dns
except
my_site.com
end

; No access to RDNS
restrict rdns
all
end

define subnet my_subnet
  10.11.12.0/24
end

<Proxy>
  trace.request(yes)

<Proxy>
  deny url.host.is_numeric=no url.domain=!my_site.com
  deny url.address=!my_subnet

<Proxy>
  deny ftp.method=STOR

<Proxy>
  url.domain=my_site.com action.test(yes)
```

```
define action test
  set(request.x_header.test, "test")
  rewrite(url, "(.*)\.my_site.com", "$(1).his_site.com")
  end
```

Since "trace.request()" on page 579 is set to yes, a policy trace is performed when client requests are evaluated.

The following is the trace output produced for an HTTP GET request for http://www.my_site.com/home.html.

The line numbers shown at the left do not appear in actual trace output. They are added here for annotation purposes.

```
1   start transaction ------------------------------
2     CPL Evaluation Trace:
3               <Proxy>
4       MATCH:     trace.request(yes)
5               <Proxy>
6       miss:     url.domain=!//my_site.com/
7       miss:     url.address=!my_subnet
8               <Proxy>
9       n/a  :     ftp.method=STOR
10              <Proxy>
11      MATCH:     url.domain=//my_site.com/ action.foo(yes)
12   connection: client.address=10.10.0.10 proxy.port=36895
13   time: 2003-09-11 19:36:22 UTC
14   GET http://www.my_site.com/home.html
15   DNS lookup was unrestricted
16   rewritten URL(s):
17   cache_url/server_url/log_url=http://www.his_site.com/
18   User-Agent: Mozilla 8.6 (Non-compatible)
19   user: unauthenticated
20   set header= (request)
21    value='test'
22  end transaction -------------------------------
```

Notes:

- Lines 1 and 22 are delimiters indicating where the trace for this transaction starts and ends.
- Line 2 introduces the rule evaluation part of the trace. A rule evaluation part is generated when "trace.request()" on page 579 is set to yes.
- Lines 3 to 4 and 10 to 11 show rule matches, and are included when "trace.request()" on page 579 is set to yes.
- Lines 5 to 9 show rule misses, and are included when "trace.request()" on page 579 is set to yes.
- Line 9 shows how a rule (containing an FTP specific condition) that is not applicable to this transaction (HTTP) is marked as n/a.
- Lines 12 to 21 are generated as a result of "trace.request()" on page 579.
- Line 12 shows client related information.

- Line 13 shows the time the transaction was processed.
- Line 14 is a summary of the request line.
- Line 15 indicates that DNS lookup was attempted during evaluation, and was unrestricted. This line only appears if there is a DNS restriction and a DNS lookup was required for evaluation.
- Lines 16 and 17 indicate that the request URL was rewritten, and show the effects.
- Line 19 indicates that the user was not required to authenticate. If authentication had been required, the user identity would be displayed.
- Lines 20 and 21 show the results of the header modification action.

The following is a trace of the same policy, but for a transaction in which the request URL has an IP address instead of a hostname.

```
1   start transaction ------------------------------
2     CPL Evaluation Trace:
3               <Proxy>
4       MATCH:     trace.request(yes)
5               <Proxy>
6       miss:      url.host.is_numeric=no
7       miss:      url.address=!my_subnet
8               <Proxy>
9       n/a  :     ftp.method=STOR
10              <Proxy>
11      miss:      url.domain=//my_site.com/
12   connection: client.address=10.10.0.10 proxy.port=36895
13   time: 2003-09-11 19:33:34 UTC
14   GET http://10.11.12.13/home.html
15   DNS lookup was restricted
16   RDNS lookup was restricted
17   User-Agent: Mozilla 8.6 (Non-compatible)
18   user: unauthenticated
19  end transaction --------------------------------
```

This shows many of the same features as the earlier trace, but has the following differences:

- Line 12–The URL requested had a numeric host name.
- Lines 15 and 16–Both DNA and RDNS lookups were restricted for this transaction.
- Line 11–Because RDNS lookups are restricted, the rule missed; no rewrite action was used for the transaction and no rewrite action is reported in the transaction summary (lines 12-18).

Trace output can be used to determine the cause of action conflicts that may be reported in the event log. For example, consider the following policy fragment:

```
<Proxy>
  trace.request(yes)
```

```
<Proxy> action.set_header_1(yes)
  [Rule] action.set_header_2(yes)
    action.set_header_3(yes)

define action set_header_1
  set(request.x_header.Test, "one")
end

define action set_header_2
  set(request.x_header.Test, "two")
end

define action set_header_3
  set(request.x_header.Test, "three")
end
```

Because they all set the same header, these actions will conflict. In this example, the conflict is obvious because all the actions are enabled in the same layer. However, conflicts can also arise when actions are enabled by completely independent portions of policy. If an action conflict occurs, one of the actions is dropped and an event log entry is made similar to the following:

```
Policy: Action discarded, 'set_header_1' conflicts with an action already committed
```

The conflict is reflected in the following trace of a request for //www.my_site.com/home.html:

```
1  start transaction ------------------------------
2    CPL Evaluation Trace:
3              <Proxy>
4      MATCH: trace.request(yes)
5              <Proxy> action.set_header_1(yes)
6                [Rule] action.set_header_2(yes)
7      MATCH:         action.set_header_1(yes)
8      MATCH:         action.set_header_2(yes)
9      MATCH:         action.set_header_3(yes)
10 connection: client.address=10.10.0.10 proxy.port=36895
11 time: 2003-09-12 15:56:39 UTC
12 GET http://www.my_site.com/home.html
13   User-Agent: Mozilla 8.6 (Non-compatible)
14 user: unauthenticated
15 Discarded Actions:
16   set_header_1
17   set_header_2
18 set header=set_header_3 (request)
19   value='three'
20 end transaction --------------------------------
```

Notes:

- Layer and section guard expressions are indicated in the trace (lines 7 and 8) before any rules subject to the guard (line 9).
- Line 15 indicates that actions were discarded due to conflicts.
- Lines 16 and 17 show the discarded actions.
- Line 18 shows the remaining action, while line 19 shows the effect of the action on the header value.

Determining Which Policy Rules are Matched in Transactions

You can use policy coverage to report on the rules that match user requests processed through the appliance's current policy. Unlike a policy trace, which you enable and disable through CPL, policy coverage is always enabled and running. The appliance resets the policy coverage counter whenever new policy is installed.

To determine which rules match proxied requests and the frequency with which the rules are 'hit', display the current policy coverage in the Management Console (select **Statistics > Advanced** and scroll down to **Policy**. Then, click **Show Policy Coverage**).

The Policy Coverage page displays all policy (Visual, Local, Central and Forward) on the appliance in CPL. The number of times that each rule is hit is listed to the left of each policy item that can be tracked. The following is an example of the output on the Policy Coverage page:

```
: ; Installed Policy -- compiled at: Mon, 03 Mar 2014 14:21:10 UTC
: ;     Default proxy policy is DENY
:
: ; Policy Rules
: <Proxy>
34:     authenticate(local) authenticate.force(no) authenticate.mode(origin)


: <Proxy>
34:     authenticate.guest("guest", 0, "local")

0: <Proxy>  user.is_guest=yes (0)
0:     DENY url.domain=//www.google.com/ (0)
0:     DENY streaming.client=yes (0)

: <Proxy>
1:     DENY url.domain=//www.tinydeal.com/ (1)
```

> **Note:** Domains in a define section do not have conditions, so they are not included in coverage. For example, consider a transaction involving a domain in the following section:
>
> ```
> define url.domain condition my_domains
>    example.com
>    company.com
>    test.com
> end
> ```

Policy coverage tracks when the `my_domains` condition is hit; however, it does not track transactions involving specific URLs within the `my_domains` condition.

For more information on policy coverage, refer to TECH241425:

http://www.symantec.com/docs/TECH241425

Recognized HTTP Headers

This section lists all recognized HTTP 1.1 headers and indicate how the appliance is able to interact with them. For each header, columns show whether the header appears in request or response forms, and whether the "append()" on page 595, "delete()" on page 598, "rewrite()" on page 615, or "set()" on page 619 actions can be used to change the header.

Recognized headers can be used with the "request.header.header_name.exists=" on page 223 and "response.header.header_name=" on page 248 conditions. Headers not shown in these tables must be tested with the "request.x_header.header_name.address=" on page 242 and "response.x_header.header_name=" on page 256 conditions.

In addition, the `Client-IP`, `Host`, and `X-Forwarded-For` header fields take address values, so they can be used with `.header_name.address=` conditions.

Symantec uses the ELFF `#Remark` directive to record the serial number and name of the appliance in an ELFF-formatted access log.

Header Field	Request/Response Form	Allowed Actions		
		`rewrite( )` `set( )`	`append( )`	`delete( )`
Accept	Request	X	X	X
Accept-Charset	Request	X	X	X
Accept-Encoding	Request	X	X	X
Accept-Language	Request	X	X	X
Accept-Ranges	Response	X	X	X
Age	Response			
Allow	Request/Response	X	X	X
Authorization	Request			
Cache-Control	Request/Response	X	X	X
Client-IP	Request	X		X
Connection	Request/Response			
Content-Encoding	Request/Response		X	
Content-Language	Request/Response			
Content-Length	Request/Response			
Content-Location	Request/Response	X		X
Content-MD5	Request/Response			
Content-Range	Request/Response			
Content-Type	Request/Response	X		

Header Field	Request/Response Form	Allowed Actions		
Cookie	Request	X	X	X
Cookie2	Request	X		X
Date	Request/Response			
ETag	Response	X		X
Expect	Request		X	
Expires	Request/Response	X		X
From	Request	X		X
Host	Request			
If-Match	Request		X	
If-Modified-Since	Request			
If-None-Match	Request		X	
If-Range	Request			
If-Unmodified-Since	Request			
Last-Modified	Request/Response			
Location	Response	X		X
Max-Forwards	Request			
Meter	Request/Response	X		X
Pragma	Request/Response	X		X
Proxy-Authenticate	Response		X	
Proxy-Authorization	Request			X
Proxy-Connection	Request			
Range	Request			X
Referer	Request	X		X
Retry-After	Response	X		X
Server	Response	X		X
Set-Cookie	Response	X	X	X
Set-Cookie2	Response	X	X	X
TE	Request		X	
Trailer	Request/Response		X	
Transfer-Encoding	Request/Response			
Upgrade	Request/Response			
User-Agent	Request	X		X
Vary	Response	X	X	X
Via	Request/Response	X	X	X

Header Field	Request/Response Form	Allowed Actions		
Warning	Request/Response	X	X	X
WWW-Authenticate	Response			

The following table lists custom headers that are recognized by the appliance.

Header Field	Request/Response Form	Allowed Actions
Authentication-Info	Response	`append( )`
Front-End-Https	Request/Response	`rewrite( )` `set( )` `delete( )`
ICAP-Reqmod	Request	`set( )`
ICAP-Respmod	Response	`set( )`
Proxy-Support	Response	Cannot be modified.
P3P	Response	`rewrite( )` `set( )` `delete( )`
Refresh	Response	`rewrite( )` `set( )` `delete( )`
X-BlueCoat-Error	Request/Response	Cannot be modified.
X-BlueCoat-Via	Request/Response	`delete( )`
X-Forwarded-For	Request	Cannot be modified.

Regex Reference

Regular expressions can be used for complex pattern matching.

> **Note:** Avoid using a regular expression when a non-regular expression alternative is available. Regular expressions are almost always less effective and more error prone than non-regular expressions. For instance, instead of using the regular expression "^[^:]*://.*\.bluecoat\.com/.*$", write "url.domain=bluecoat.com".

The following conditions use regular-expression arguments:

- All triggers with the .regex qualifier (for example, url.regex=)
- Request and response header triggers (for example, "request.header.header_name.address=" on page 221, "request.x_header.header_name.address=" on page 242, "response.header.header_name=" on page 248, "response.x_header.header_name=" on page 256)

The following CPL actions include regular-expression arguments:

- "delete_matching()" on page 600
- "redirect()" on page 611
- "rewrite()" on page 615

The regular expression support in the appliance described in this appendix is based on the Perl-compatible regular expression libraries (PCRE) by Philip Hazel. The text of this appendix is based on the PCRE documentation.

A regular expression (or RE) is a pattern that is matched against a subject string from left to right. Most characters stand for themselves in a pattern, and match the corresponding characters in the subject. The power of regular expressions comes from the ability to include alternatives and repetitions in the pattern. These are encoded in the pattern by the use of metacharacters, which do not stand for themselves, but instead are interpreted in some special way. For details of the theory and implementation of regular expressions, consult Jeffrey Friedl's *Mastering Regular Expressions*, Third Edition, published by O'Reilly (ISBN 0-596-00289-0).

The appliance uses a Regular Expression Engine (RE ENGINE) to evaluate regular expressions.

This section covers the following subjects:

- Syntax and semantics, including a table of metacharacters
- Differences between the RE ENGINE and Perl

Regular Expression Syntax

Regular expressions can contain both special and ordinary characters. Most ordinary characters, like 'A', 'a', or '3', are the simplest regular expressions; they simply match themselves. You can concatenate ordinary characters, so 'last' matches the characters 'last'. (In the rest of this section, regular expressions are written in a fixed-width font, usually without quotes, and strings to be matched are 'in single quotes'.)

Some characters, like | or (, are special. Special characters, called metacharacters, either stand for classes of ordinary characters, or affect how the regular expressions around them are interpreted. The metacharacters are described in the following table.

Metacharacter	Description
`(?i)`	Evaluate the expression following this metacharacter in a case-insensitive manner.
`.`	(Dot) In the default mode, this matches any character except a newline. (Note that newlines should not be detected when using regular expressions in CPL.)
`^`	(Circumflex or caret) Matches the start of the string. $ Matches the end of the string.
`$`	Matches the end of the string.
`*`	Causes the resulting RE to match zero (0) or more repetitions of the preceding RE, as many repetitions as are possible. ab* will match 'a', 'ab', or 'a' followed by any number of 'b's.
`+`	Causes the resulting RE to match one (1) or more repetitions of the preceding RE. ab+ will match 'a' followed by any non-zero number of 'b's; it will not match just 'a'.
`?`	Causes the resulting RE to match 0 or 1 repetitions of the preceding RE. ab? will match either 'a' or 'ab'.
`*?, +?, ??`	The *, +, and ? qualifiers are all greedy; they match as much text as possible. Sometimes this behavior isn't desired. If the RE /page1/.*/ is matched against /page1/heading/images/, it will match the entire string, and not just /page1/heading/. Adding ? after the qualifier makes it perform the match in non-greedy or minimal fashion; matching as few characters as possible. Using .*? in the previous expression will match only /page1/heading/.
`{m,n}`	Causes the resulting RE to match from m to n repetitions of the preceding RE, attempting to match as many repetitions as possible. For example, a{3,5} will match from 3 to 5 'a' characters.

Metacharacter	Description
{m,n}?	Causes the resulting RE to match from m to n repetitions of the preceding RE, attempting to match as few repetitions as possible. This is the non-greedy version of the previous qualifier. For example, on the 6-character string 'aaaaaa', a {3,5} will match 5 'a' characters, while a{3,5}? will only match 3 characters.
\	Either escapes special characters (permitting you to match characters like '*?+&$'), or signals a special sequence; special sequences are discussed below.
[]	Used to indicate a set of characters. Characters can be listed individually, or a range of characters can be indicated by giving two characters and separating them by a '-'. Special characters are not active inside sets. For example, [akm$] will match any of the characters 'a', 'k', 'm', or '$'; [a-z] will match any lowercase letter and [a-zA-Z0-9] matches any letter or digit. Character classes such as \w or \S (defined below) are also acceptable inside a range. If you want to include a] or a - inside a set, precede it with a backslash. Characters not within a range can be matched by including a ^ as the first character of the set; ^ elsewhere will simply match the '^' character.
|	A|B, where A and B can be arbitrary REs, creates a regular expression that will match either A or B. This can be used inside groups (see below) as well. To match a literal '|', use \\|, or enclose it inside a character class, like [|].
(...)	Matches whatever regular expression is inside the parentheses, and indicates the start and end of a group; the contents of a group can be retrieved after a match has been performed, and can be matched later in the string with the \number special sequence, described below. To match the literals '(' or ')', use \(or \), or enclose them inside a character class: [(] [)].

Regular Expression Details

This section describes the syntax and semantics of the regular expressions supported. Regular expressions are also described in most Perl documentation and in a number of other books, some of which have copious examples. Jeffrey Friedl's *Mastering Regular Expressions*, published by O'Reilly (ISBN 0-596-00289-0), covers them in great detail. The description here is intended as reference documentation.

There are two different sets of metacharacters: those that are recognized anywhere in the pattern except within square brackets, and those that are recognized in square brackets. Outside square brackets, the metacharacters are:

Metacharacter	Description
\	general escape character with several uses
^	assert start of subject (or line, in multiline mode)
$	assert end of subject (or line, in multiline mode)
.	match any character except newline (by default)
[	start character class definition
|	start of alternative branch
(	start subpattern
)	end subpattern
?	extends the meaning of "(" also 0 or 1 quantifier also quantifier minimizer
*	0 or more quantifier
+	1 or more quantifier
{	start min/max quantifier

The part of a pattern that is in square brackets is called a "character class." In a character class the only metacharacters are:

Metacharacter	Description
\	general escape character
^	negate the class, but only if the first character
-	indicates character range
]	terminates the character class

The following sections describe the use of each of the metacharacters.

Backslash

The backslash character has several uses. If it is followed by a non-alphanumeric character, it takes away any special meaning that character might have. This use of backslash as an escape character applies both inside and outside character classes.

For example, if you want to match a "*" character, you write "*" in the pattern. This applies whether or not the following character would otherwise be interpreted as a metacharacter, so it is always safe to precede a non-alphanumeric with "\" to specify that it stands for itself. In particular, if you want to match a backslash, you write "\\".

An escaping backslash can be used to include a white space or "#" character as part of the pattern.

A second use of backslash provides a way of encoding non-printing characters in patterns in a visible manner. There is no restriction on the appearance of non-printing characters, apart from the binary zero that terminates a pattern; but when a pattern is being prepared by text editing, it is usually easier to use one of the following escape sequences than the binary character it represents. For example, \a represents "alarm", the BEL character (hex 07).

The handling of a backslash followed by a digit other than 0 is complicated. Outside a character class, RE ENGINE reads it and any following digits as a decimal number. If the number is less than 10, or if there have been at least that many previous capturing left parentheses in the expression, the entire sequence is taken as a back reference. A description of how this works is given later, following the discussion of parenthesized subpatterns.

Inside a character class, or if the decimal number is greater than 9 and there have not been that many capturing subpatterns, RE ENGINE re-reads up to three octal digits following the backslash, and generates a single byte from the least significant 8 bits of the value. Any subsequent digits stand for themselves. For example, \040 is another way of writing a space

Note that octal values of 100 or greater must not be introduced by a leading zero, because no more than three octal digits are ever read. All the sequences that define a single byte value can be used both inside and outside character classes. In addition, inside a character class, the sequence "\b" is interpreted as the backspace character (hex 08). Outside a character class it has a different meaning (see below).

The third use of backslash is for specifying generic character types:

- \d Any decimal digit
- \D Any character that is not a decimal digit
- \s Any white space character
- \S Any character that is not a white space character
- \w Any word character
- \W Any non-word character

Each pair of escape sequences partitions the complete set of characters into two disjoint sets. Any given character matches one, and only one, of each pair.

A "word" character is any letter or digit or the underscore character; that is, any character that can be part of a Perl "word."

These character-type sequences can appear both inside and outside character classes. They each match one character of the appropriate type. If the current matching point is at the end of the subject string, all of them fail, since there is no character to match.

The fourth use of backslash is for certain simple assertions. An assertion specifies a condition that has to be met at a particular point in a match, without consuming any characters from the subject string. The use of subpatterns for more complicated assertions is described below. The back slashed assertions are

- \b Word boundary
- \B Not a word boundary
- \A Start of subject (independent of multiline mode)
- \Z End of subject or newline at end (independent of multiline mode)
- \z End of subject (independent of multiline mode)

These assertions might not appear in character classes (but note that "\b" has a different meaning, namely the backspace character, inside a character class).

A word boundary is a position in the subject string where the current character and the previous character do not both match \w or \W (i.e. one matches \w and the other matches \W), or the start or end of the string if the first or last character matches \w, respectively.

The \A, \Z, and \z assertions differ from the traditional circumflex and dollar (described below) in that they only ever match at the very start and end of the subject string, whatever options are set. The difference between \Z and \z is that \Z matches before a newline that is the last character of the string as well as at the end of the string, whereas \z matches only at the end. (Newlines should not be detected when using regular expressions in CPL.)

Circumflex and Dollar

Regular expressions are anchored in the CPL actions redirect() and rewrite(), and unanchored in all other CPL and command uses of regular-expression patterns. In a regular expression that is by default unanchored, use the circumflex and dollar (^ and $) to anchor the match at the beginning and end.

Circumflex need not be the first character of the pattern if a number of alternatives are involved, but it should be the first thing in each alternative in which it appears if the pattern is ever to match that branch. If all possible alternatives start with a circumflex, that is, if the pattern is constrained to match only at the start of the subject, it is said to be an "anchored" pattern. (There are also other constructs that can cause a pattern to be anchored.)

A dollar character is an assertion that is true only if the current matching point is at the end of the subject string, or immediately before a newline character that is the last character in the string (by default). Dollar need not be the last character of the pattern if a number of alternatives are involved, but it should be the last item in any branch in which it appears. Dollar has no special meaning in a character class.

Period (Dot)

Outside a character class, a dot in the pattern matches any one character in the subject, including a non-printing character, but not (by default) newline. (Note that newlines should not be detected when using regular expressions in CPL.) The handling of dot is entirely independent of the handling of circumflex and dollar, the only relationship being that they both involve newline characters. Dot has no special meaning in a character class.

Square Brackets

An opening square bracket introduces a character class, terminated by a closing square bracket. A closing square bracket on its own is not special. If a closing square bracket is required as a member of the class, it should be the first data character in the class (after an initial circumflex, if present) or escaped with a backslash.

A character class matches a single character in the subject; the character must be in the set of characters defined by the class, unless the first character in the class is a circumflex, in which case the subject character must not be in the set defined by the class. If a circumflex is actually required as a member of the class, ensure it is not the first character, or escape it with a backslash.

For example, the character class [aeiou] matches any lowercase vowel, while [^aeiou] matches any character that is not a lowercase vowel. Note that a circumflex is just a convenient notation for specifying the characters, which are in the class by enumerating those that are not. It is not an assertion: it still consumes a character from the subject string, and fails if the current pointer is at the end of the string.

A class such as [^a] will always match a newline. (Newlines should not be detected when using regular expressions in CPL.)

The minus (hyphen) character can be used to specify a range of characters in a character class. For example, [d-m] matches any letter between d and m, inclusive. If a minus character is required in a class, it must be escaped with a backslash or appear in a position where it cannot be interpreted as indicating a range, typically as the first or last character in the class. It is not possible to have the character "]" as the end character of a range, since a sequence such as [w-] is interpreted as a class of two characters. The octal or hexadecimal representation of "]" can, however, be used to end a range.

Ranges operate in ASCII collating sequence. They can also be used for characters specified numerically, for example [\000-\037].

The character types \d, \D, \s, \S, \w, and \W might also appear in a character class, and add the characters that they match to the class. For example, [\dABCDEF] matches any hexadecimal digit. A circumflex can conveniently be used with the upper case character types to specify a more restricted set of characters than the matching lower case type. For example, the class [^\W_] matches any letter or digit, but not underscore.

All non-alphanumeric characters other than \, -, ^ (at the start) and the terminating] are non-special in character classes, but it does no harm if they are escaped.

Vertical Bar

Vertical bar characters are used to separate alternative patterns. For example, the pattern

gilbert|sullivan

matches either "gilbert" or "sullivan." Any number of alternatives might appear, and an empty alternative is permitted (matching the empty string). The matching process tries each alternative in turn, from left to right, and the first one that succeeds is used. If the alternatives are within a subpattern (defined below), "succeeds" means matching the rest of the main pattern as well as the alternative in the subpattern.

Lowercase-Sensitivity

By default, CPL conditions that take regular-expression arguments perform a case-insensitive match. In all other places where the the appliance performs a regular-expression match, the match is case sensitive.

In CPL, use the ".case_sensitive" condition modifier for case sensitivity, rather than relying on Perl syntax.

Override the default for case sensitivity by using the following syntax:

(?i) Sets case-insensitive matching mode.

(?-i) Sets case-sensitive matching mode.

The scope of a mode setting depends on where in the pattern the setting occurs. For settings that are outside any subpattern (see the next section), the effect is the same as if the options were set or unset at the start of matching. The following patterns all behave in exactly the same way:

(?i)abc

a(?i)bc

ab(?i)c

abc(?i)

In other words, such “top level” settings apply to the whole pattern (unless there are other changes inside subpatterns). If there is more than one setting of the same option at the top level, the rightmost setting is used.

If an option change occurs inside a subpattern, the effect is different. This is a change of behavior in Perl 5.005. An option change inside a subpattern affects only that part of the subpattern that follows it, so (a(?i)b)c matches abc and aBc and no other strings (assuming the default is case sensitive). By this means, options can be made to have different settings in different parts of the pattern. Any changes made in one alternative do carry on into subsequent branches within the same subpattern. For example (a(?i)b|c) matches "ab", "aB", "c", and "C", even though when matching "C" the first branch is abandoned before the option setting. This is because the effects of option settings happen at compile time. This avoids some strange side-effects.

Subpatterns

Subpatterns are delimited by parentheses (round brackets), which can be nested. Marking part of a pattern as a subpattern does two things:

It localizes a set of alternatives.

For example, the pattern cat(aract|erpillar|) matches one of the words “cat”, “cataract”, or “caterpillar”. Without the parentheses, it would match “cataract”, “erpillar” or the empty string.

It sets up the subpattern as a capturing subpattern (as defined above). When the whole pattern matches, that portion of the subject string that matched the subpattern is passed back to the caller via the ovector argument of RE Engine_exec(). Opening parentheses are counted from left to right (starting from 1) to obtain the numbers of the capturing subpatterns.

For example, if the string “the red king” is matched against the pattern the ((red|white) (king|queen)) the captured substrings are “red king”, “red”, and “king”, and are numbered 1, 2, and 3.

The fact that plain parentheses fulfill two functions is not always helpful. There are times when a grouping subpattern is required without a capturing requirement. If an opening parenthesis is followed by “?:”, the subpattern does not do any capturing, and is not counted when computing the number of any subsequent capturing subpatterns. For example, if the string “the white queen” is matched against the pattern the ((?:red|white)(king|queen)) the captured substrings are “white queen” and “queen,” and are numbered 1 and 2. The maximum number of captured substrings is 99, and the maximum number of all subpatterns, both capturing and non-capturing, is 200.

As a convenient shorthand, if any option settings are required at the start of a non-capturing subpattern, the option letters might appear between the “?” and the “:”. Thus the two patterns (?i:saturday|sunday) and (?:(?i)saturday|sunday) match exactly the same set of strings. Because alternative branches are tried from left to right, and options are not reset until the end of the subpattern is reached, an option setting in one branch does affect subsequent branches, so the above patterns match “SUNDAY” as well as “Saturday”.

Repetition

Repetition is specified by quantifiers, which can follow any of the following items:

- A single character, possibly escaped by the . metacharacter
- A character class
- A back reference (see next section)
- A parenthesized subpattern (unless it is an assertion - see below)

The general repetition quantifier specifies a minimum and maximum number of permitted matches, by giving the two numbers in curly brackets (braces), separated by a comma. The numbers must be less than 65536, and the first must be less than or equal to the second. For example, z{2,4} matches “zz”, “zzz”, or “zzzz.” A closing brace on its own is not a special character. If the second number is omitted, but the comma is present, there is no upper limit; if the second number and the comma are both omitted, the quantifier specifies an exact number of required matches. Thus [aeiou]{3,} matches at least 3 successive vowels, but might match many more, while \d{8} matches exactly 8 digits. An opening curly bracket that appears in a position where a quantifier is not allowed, or one that does not match the syntax of a quantifier, is taken as a literal character. For example, {,6} is not a quantifier, but a literal string of four characters.

The quantifier {0} is permitted, causing the expression to behave as if the previous item and the quantifier were not present. For convenience (and historical compatibility) the three most common quantifiers have single-character abbreviations:

- * Equivalent to {0,}
- + Equivalent to {1,}
- ? Equivalent to {0,1}

It is possible to construct infinite loops by following a subpattern that can match no characters with a quantifier that has no upper limit, for example (a?)*

Earlier versions of Perl gave an error at compile time for such patterns. However, because there are cases where this can be useful, such patterns are now accepted, but if any repetition of the subpattern does in fact match no characters, the loop is forcibly broken.

By default, the quantifiers are “greedy,” that is, they match as much as possible (up to the maximum number of permitted times) without causing the rest of the pattern to fail. The classic example of where this gives problems is in trying to match comments in C programs. These appear between the sequences /* and */ and within the sequence, individual * and / characters might appear. An attempt to match C comments by applying the following pattern fails because it matches the entire string due to the greediness of the .* item.

/*.**/

to the string

/* first command */ not comment /* second comment */

However, if a quantifier is followed by a question mark, then it ceases to be greedy, and instead matches the minimum number of times possible, so the following pattern does the right thing with the C comments.

/*.*?*/

The meaning of the various quantifiers is not otherwise changed, just the preferred number of matches. Do not confuse this use of question mark with its use as a quantifier in its own right. Because it has two uses, it can sometimes appear doubled, as below, which matches one digit by preference, but can match two if that is the only way the rest of the pattern matches.

\d??\d

When a parenthesized subpattern is quantified with a minimum repeat count that is greater than 1 or with a limited maximum, more store is required for the compiled pattern, in proportion to the size of the minimum or maximum.

If a pattern starts with .* then it is implicitly anchored, since whatever follows will be tried against every character position in the subject string. RE ENGINE treats this as though it were preceded by \A.

When a capturing subpattern is repeated, the value captured is the substring that matched the final iteration. For example, after the following expression has matched “tweedledum tweedledee” the value of the captured substring is “tweedledee”.

(tweedle[dume]{3}\s*)+

However, if there are nested capturing subpatterns, the corresponding captured values might have been set in previous iterations. For example, after

/(a|(b))+/

matches “aba” the value of the second captured substring is “b”.

Back References

Outside a character class, a backslash followed by a digit greater than 0 (and possibly further digits) is a back reference to a capturing subpattern earlier (i.e., to its left) in the pattern, provided there have been that many previous capturing left parentheses.

However, if the decimal number following the backslash is less than 10, it is always taken as a back reference, and causes an error only if there are not that many capturing left parentheses in the entire pattern. In other words, the parentheses that are referenced need not be to the left of the reference for numbers less than 10. See the section entitled “Backslash” above for further details of the handling of digits following a backslash.

A back reference matches whatever actually matched the capturing subpattern in the current subject string, rather than anything matching the subpattern itself. So the following pattern matches “sense and sensibility” and “response and responsibility,” but not “sense and responsibility.”

(sens|respons)e and \1ibility

There might be more than one back reference to the same subpattern. If a subpattern has not actually been used in a particular match, then any back references to it always fail. For example, the following pattern always fails if it starts to match “a” rather than “bc.” Because there might be up to 99 back references, all digits following the backslash are taken as part of a potential back reference number. If the pattern continues with a digit character, then some delimiter must be used to terminate the back reference.

(a|(bc))\2

A back reference that occurs inside the parentheses to which it refers fails when the subpattern is first used, so, for example, (a\1) never matches. However, such references can be useful inside repeated subpatterns. For example, the following pattern matches any number of “a”s and also “aba”, “ababaa” etc. At each iteration of the subpattern, the back reference matches the character string corresponding to the previous iteration. In order for this to work, the pattern must be such that the first iteration does not need to match the back reference. This can be done using alternation, as in the example above, or by a quantifier with a minimum of zero.

(a|b\1)+

Assertions

An assertion is a test on the characters following or preceding the current matching point that does not actually consume any characters. The simple assertions coded as \b, \B, \A, \Z, \z, ^ and $ are described above. More complicated assertions are coded as subpatterns. There are two kinds: those that look ahead of the current position in the subject string, and those that look behind it.

An assertion subpattern is matched in the normal way, except that it does not cause the current matching position to be changed. Lookahead assertions start with (?= for positive assertions and (?! for negative assertions. For example, the following expression matches a word followed by a semicolon, but does not include the semicolon in the match.

\w+(?=;)

The following expression matches any occurrence of “example” that is not followed by “bar”.

example(?!bar)

Note that the apparently similar pattern that follows does not find an occurrence of “bar” that is preceded by something other than “example”; it finds any occurrence of “bar” whatsoever, because the assertion (?!example) is always true when the next three characters are “bar”. A lookbehind assertion is needed to achieve this effect.

(?!example)bar

Lookbehind assertions start with (?<= for positive assertions and (?<! for negative assertions. For example, the following expression does find an occurrence of “bar” that is not preceded by “example”. The contents of a lookbehind assertion are restricted such that all the strings it matches must have a fixed length.

(?<!example)bar

However, if there are several alternatives, they do not all have to have the same fixed length. Thus (?<=bullock|donkey) is permitted, but (?<!dogs?|cats?) causes an error at compile time. Branches that match different length strings are permitted

only at the top level of a lookbehind assertion. This is an extension compared with Perl 5.005, which requires all branches to match the same length of string. An assertion such as (?<=ab(c|de)) is not permitted, because its single branch can match two different lengths, but it is acceptable if rewritten to use two branches:

(?<=abc|abde)

The implementation of lookbehind assertions is, for each alternative, to temporarily move the current position back by the fixed width and then try to match. If there are insufficient characters before the current position, the match is deemed to fail.

Assertions can be nested in any combination. For example, the following expression matches an occurrence of “baz” that is preceded by “bar” which in turn is not preceded by “example”.

(?<=(?<!example)bar)baz

Assertion subpatterns are not capturing subpatterns, and might not be repeated, because it makes no sense to assert the same thing several times. If an assertion contains capturing subpatterns within it, these are always counted for the purposes of numbering the capturing subpatterns in the whole pattern. Substring capturing is carried out for positive assertions, but it does not make sense for negative assertions.

Assertions count towards the maximum of 200 parenthesized subpatterns.

Once-Only Subpatterns

With both maximizing and minimizing repetition, failure of what follows normally causes the repeated item to be re-evaluated to see if a different number of repeats allows the rest of the pattern to match. Sometimes it is useful to prevent this, either to change the nature of the match, or to cause it fail earlier than it otherwise might, when the author of the pattern knows there is no point in carrying on.

Consider, for example, the pattern \d+example when applied to the subject line

123456bar

After matching all 6 digits and then failing to match “example,” the normal action of the matcher is to try again with only 5 digits matching the \d+ item, and then with 4, and so on, before ultimately failing. Once-only subpatterns provide the means for specifying that once a portion of the pattern has matched, it is not to be re-evaluated in this way, so the matcher would give up immediately on failing to match “example” the first time. The notation is another kind of special parenthesis, starting with (?> as in this example:

(?>\d+)bar

This kind of parenthesis “locks up” the part of the pattern it contains once it has matched, and a failure further into the pattern is prevented from backtracking into it. Backtracking past it to previous items, however, works as normal.

An alternative description is that a subpattern of this type matches the string of characters that an identical standalone pattern would match, if anchored at the current point in the subject string.

Once-only subpatterns are not capturing subpatterns. Simple cases such as the above example can be though of as a maximizing repeat that must swallow everything it can. So, while both \d+ and \d+? are prepared to adjust the number of digits they match in order to make the rest of the pattern match, (?>\d+) can only match an entire sequence of digits.

This construction can of course contain arbitrarily complicated subpatterns, and it can be nested.

Conditional Subpatterns

It is possible to cause the matching process to obey a subpattern conditionally or to choose between two alternative subpatterns, depending on the result of an assertion, or whether a previous capturing subpattern matched or not. The two possible forms of conditional subpattern are

(?(condition)yes-pattern)

(?(condition)yes-pattern|no-pattern)

If the condition is satisfied, the yes-pattern is used; otherwise the no-pattern (if present) is used. If there are more than two alternatives in the subpattern, a compile-time error occurs.

There are two kinds of condition. If the text between the parentheses consists of a sequence of digits, then the condition is satisfied if the capturing subpattern of that number has previously matched. Consider the following pattern, which contains non-significant white space to make it more readable and to divide it into three parts for ease of discussion:

(\( )? [^()]+ (?(1) \))

The first part matches an optional opening parenthesis, and if that character is present, sets it as the first captured substring. The second part matches one or more characters that are not parentheses. The third part is a conditional subpattern that tests whether the first set of parentheses matched or not. If they did, that is, if subject started with an opening parenthesis, the condition is true, and so the yes-pattern is executed and a closing parenthesis is required. Otherwise, since no-pattern is not present, the subpattern matches nothing. In other words, this pattern matches a sequence of non-parentheses, optionally enclosed in parentheses.

If the condition is not a sequence of digits, it must be an assertion. This might be a positive or negative lookahead or lookbehind assertion. Consider this pattern, again containing non-significant white space, and with the two alternatives on the second line:

(?(?=[^a-z]*[a-z])

\d{2}[a-z]{3}-\d{2}|\d{2}-\d{2}-\d{2})

The condition is a positive lookahead assertion that matches an optional sequence of non-letters followed by a letter. In other words, it tests for the presence of at least one letter in the subject. If a letter is found, the subject is matched against the first alternative; otherwise it is matched against the second. This pattern matches strings in one of the two forms dd-aaa-dd or dd-dd-dd, where aaa are letters and dd are digits.

Comments

The sequence (?# marks the start of a comment which continues up to the next closing parenthesis. Nested parentheses are not permitted. The characters that make up a comment play no part in the pattern matching at all.

Performance

Certain items that might appear in patterns are more efficient than others. It is more efficient to use a character class like [aeiou] than a set of alternatives such as (a|e|i|o|u). In general, the simplest construction that provides the required behavior is usually the most efficient. Remember that non-regular expressions are simpler constructions than regular expressions, and are thus more efficient in general.

Regular Expression Engine Differences From Perl

This section describes differences between the RE ENGINE and Perl 5.005.

Normally “space” matches space, formfeed, newline, carriage return, horizontal tab, and vertical tab. Perl 5 no longer includes vertical tab in its set of white-space characters. The \v escape that was in the Perl documentation for a long time was never in fact recognized. However, the character itself was treated as white space at least up to 5.002. In 5.004 and 5.005 it does not match \s.

RE ENGINE does not allow repeat quantifiers on lookahead assertions. Perl permits them, but they do not mean what you might think. For example, (?!a){3} does not assert that the next three characters are not “a”. It just asserts that the next character is not “a” three times.

Capturing subpatterns that occur inside negative lookahead assertions are counted, but their entries in the offsets vector are never set. Perl sets its numerical variables from any such patterns that are matched before the assertion fails to match something (thereby succeeding), but only if the negative lookahead assertion contains just one branch.

Though binary zero characters are supported in the subject string, they are not allowed in a pattern string because it is passed as a normal C string, terminated by zero. The escape sequence “\0” can be used in the pattern to represent a binary zero.

The following Perl escape sequences are not supported: \l, \u, \L, \U, \E, \Q. In fact these are implemented by Perl’s general string handling and are not part of its pattern-matching engine.

The Perl \G assertion is not supported as it is not relevant to single pattern matches.

RE ENGINE does not support the (?{code}) construction.

There are at the time of writing some oddities in Perl 5.005_02 concerned with the settings of captured strings when part of a pattern is repeated. For example, matching “aba” against the pattern /^(a(b)?)+$/ sets $2 to the value “b”, but matching “aabbaa” against /^(aa(bb)?)+$/ leaves $2 unset. However, if the pattern is changed to /^(aa(b(b))?)+$/ then $2 (and $3) get set. In Perl 5.004 $2 is set in both cases, and that is also true of RE ENGINE.

Another as yet unresolved discrepancy is that in Perl 5.005_02 the pattern /^(a)?(?(1)a|b)+$/ matches the string “a”, whereas in RE ENGINE it does not. However, in both Perl and RE ENGINE /^(a)?a/ matched against “a” leaves $1 unset.

RE ENGINE provides some extensions to the Perl regular expression facilities: Although lookbehind assertions must match fixed length strings, each alternative branch of a lookbehind assertion can match a different length of string. Perl 5.005 requires them all to have the same length.

When regular expressions are used to match a URL, a space character matches a %20 in the request URL. However, a %20 in the regular-expression pattern will not match anything in any request URL, because "%20" is normalized to " " in the subject string before the regex match is performed.

Regex Syntax

Regular expressions can contain both special and ordinary characters. Most ordinary characters, like 'A', 'a', or '3', are the simplest regular expressions; they simply match themselves. You can concatenate ordinary characters, so 'last' matches the characters 'last'. (In the rest of this section, regular expressions are written in a fixed-width font, usually without quotes, and strings to be matched are 'in single quotes'.)

Some characters, like | or (, are special. Special characters, called metacharacters, either stand for classes of ordinary characters, or affect how the regular expressions around them are interpreted. The metacharacters are described in the following table.

Metacharacter	Description
`(?i)`	Evaluate the expression following this metacharacter in a case-insensitive manner.
`.`	(Dot) In the default mode, this matches any character except a newline. (Note that newlines should not be detected when using regular expressions in CPL.)
`^`	(Circumflex or caret) Matches the start of the string. $ Matches the end of the string.
`$`	Matches the end of the string.
`*`	Causes the resulting RE to match zero (0) or more repetitions of the preceding RE, as many repetitions as are possible. ab* will match 'a', 'ab', or 'a' followed by any number of 'b's.
`+`	Causes the resulting RE to match one (1) or more repetitions of the preceding RE. ab+ will match 'a' followed by any non-zero number of 'b's; it will not match just 'a'.
`?`	Causes the resulting RE to match 0 or 1 repetitions of the preceding RE. ab? will match either 'a' or 'ab'.
`*?, +?, ??`	The *, +, and ? qualifiers are all greedy; they match as much text as possible. Sometimes this behavior isn't desired. If the RE /page1/.*/ is matched against /page1/heading/images/, it will match the entire string, and not just /page1/heading/. Adding ? after the qualifier makes it perform the match in non-greedy or minimal fashion; matching as few characters as possible. Using .*? in the previous expression will match only /page1/heading/.
`{m,n}`	Causes the resulting RE to match from m to n repetitions of the preceding RE, attempting to match as many repetitions as possible. For example, a{3,5} will match from 3 to 5 'a' characters.

Metacharacter	Description
{m,n}?	Causes the resulting RE to match from m to n repetitions of the preceding RE, attempting to match as few repetitions as possible. This is the non-greedy version of the previous qualifier. For example, on the 6-character string 'aaaaaa', a {3,5} will match 5 'a' characters, while a{3,5}? will only match 3 characters.
\	Either escapes special characters (permitting you to match characters like '*?+&$'), or signals a special sequence; special sequences are discussed below.
[]	Used to indicate a set of characters. Characters can be listed individually, or a range of characters can be indicated by giving two characters and separating them by a '-'. Special characters are not active inside sets. For example, [akm$] will match any of the characters 'a', 'k', 'm', or '$'; [a-z] will match any lowercase letter and [a-zA-Z0-9] matches any letter or digit. Character classes such as \w or \S (defined below) are also acceptable inside a range. If you want to include a] or a - inside a set, precede it with a backslash. Characters not within a range can be matched by including a ^ as the first character of the set; ^ elsewhere will simply match the '^' character.
|	A|B, where A and B can be arbitrary REs, creates a regular expression that will match either A or B. This can be used inside groups (see below) as well. To match a literal '|', use |, or enclose it inside a character class, like [|].
(...)	Matches whatever regular expression is inside the parentheses, and indicates the start and end of a group; the contents of a group can be retrieved after a match has been performed, and can be matched later in the string with the \number special sequence, described below. To match the literals '(' or ')', use \(or \), or enclose them inside a character class: [(] [)].

Regex Details

This section describes the syntax and semantics of the regular expressions supported. Regular expressions are also described in most Perl documentation and in a number of other books, some of which have copious examples. Jeffrey Friedl's *Mastering Regular Expressions*, published by O'Reilly (ISBN 0-596-00289-0), covers them in great detail. The description here is intended as reference documentation.

There are two different sets of metacharacters: those that are recognized anywhere in the pattern except within square brackets, and those that are recognized in square brackets. Outside square brackets, the metacharacters are:

Metacharacter	Description
\	general escape character with several uses
^	assert start of subject (or line, in multiline mode)
$	assert end of subject (or line, in multiline mode)
.	match any character except newline (by default)
[	start character class definition
|	start of alternative branch (start subpattern) end subpattern
?	extends the meaning of "(" also 0 or 1 quantifier also quantifier minimizer
*	0 or more quantifier
+	1 or more quantifier
{	start min/max quantifier

The part of a pattern that is in square brackets is called a character class. In a character class the only metacharacters are:

Metacharacter	Description
\	general escape character
^	negate the class, but only if the first character
-	indicates character range
]	terminates the character class

The following sections describe the use of each of the metacharacters.

Backslash

The backslash character has several uses. If it is followed by a non-alphanumeric character, it takes away any special meaning that character might have. This use of backslash as an escape character applies both inside and outside character classes.

For example, if you want to match a "*" character, you write "*" in the pattern. This applies whether or not the following character would otherwise be interpreted as a metacharacter, so it is always safe to precede a non-alphanumeric with "\" to specify that it stands for itself. In particular, if you want to match a backslash, you write "\\".

An escaping backslash can be used to include a white space or "#" character as part of the pattern.

A second use of backslash provides a way of encoding non-printing characters in patterns in a visible manner. There is no restriction on the appearance of non-printing characters, apart from the binary zero that terminates a pattern; but when a pattern is being prepared by text editing, it is usually easier to use one of the following escape sequences than the binary character it represents. For example, \a represents "alarm", the BEL character (hex 07).

The handling of a backslash followed by a digit other than 0 is complicated. Outside a character class, RE ENGINE reads it and any following digits as a decimal number. If the number is less than 10, or if there have been at least that many previous capturing left parentheses in the expression, the entire sequence is taken as a back reference. A description of how this works is given later, following the discussion of parenthesized subpatterns.

Inside a character class, or if the decimal number is greater than 9 and there have not been that many capturing subpatterns, RE ENGINE re-reads up to three octal digits following the backslash, and generates a single byte from the least significant 8 bits of the value. Any subsequent digits stand for themselves. For example, \040 is another way of writing a space

Note that octal values of 100 or greater must not be introduced by a leading zero, because no more than three octal digits are ever read. All the sequences that define a single byte value can be used both inside and outside character classes. In addition, inside a character class, the sequence "\b" is interpreted as the backspace character (hex 08). Outside a character class it has a different meaning (see below).

The third use of backslash is for specifying generic character types:

- \d : Any decimal digit
- \D : Any character that is not a decimal digit
- \s : Any white space character
- \S : Any character that is not a white space character
- \w : Any word character
- \W : Any non-word character

Each pair of escape sequences partitions the complete set of characters into two disjoint sets. Any given character matches one, and only one, of each pair.

A "word" character is any letter or digit or the underscore character; that is, any character that can be part of a Perl "word."

These character-type sequences can appear both inside and outside character classes. They each match one character of the appropriate type. If the current matching point is at the end of the subject string, all of them fail, since there is no character to match.

The fourth use of backslash is for certain simple assertions. An assertion specifies a condition that has to be met at a particular point in a match, without consuming any characters from the subject string. The use of subpatterns for more complicated assertions is described below. The back slashed assertions are

- \b : Word boundary
- \B : Not a word boundary
- \A : Start of subject (independent of multiline mode)
- \Z : End of subject or newline at end (independent of multiline mode)
- \z : End of subject (independent of multiline mode)

These assertions might not appear in character classes (but note that "\b" has a different meaning, namely the backspace character, inside a character class).

A word boundary is a position in the subject string where the current character and the previous character do not both match \w or \W (i.e. one matches \w and the other matches \W), or the start or end of the string if the first or last character matches \w, respectively.

The \A, \Z, and \z assertions differ from the traditional circumflex and dollar (described below) in that they only ever match at the very start and end of the subject string, whatever options are set. The difference between \Z and \z is that \Z matches before a newline that is the last character of the string as well as at the end of the string, whereas \z matches only at the end. (Newlines should not be detected when using regular expressions in CPL.)

Circumflex and Dollar

Regular expressions are anchored in the CPL actions redirect() and rewrite(), and unanchored in all other CPL and command uses of regular-expression patterns. In a regular expression that is by default unanchored, use the circumflex and dollar (^ and $) to anchor the match at the beginning and end.

Circumflex need not be the first character of the pattern if a number of alternatives are involved, but it should be the first thing in each alternative in which it appears if the pattern is ever to match that branch. If all possible alternatives start with a circumflex, that is, if the pattern is constrained to match only at the start of the subject, it is said to be an "anchored" pattern. (There are also other constructs that can cause a pattern to be anchored.)

A dollar character is an assertion that is true only if the current matching point is at the end of the subject string, or immediately before a newline character that is the last character in the string (by default). Dollar need not be the last character of the pattern if a number of alternatives are involved, but it should be the last item in any branch in which it appears. Dollar has no special meaning in a character class.

Period (Dot)

Outside a character class, a dot in the pattern matches any one character in the subject, including a non-printing character, but not (by default) newline. (Note that newlines should not be detected when using regular expressions in CPL.) The handling of dot is entirely independent of the handling of circumflex and dollar, the only relationship being that they both involve newline characters. Dot has no special meaning in a character class.

Square Brackets

An opening square bracket introduces a character class, terminated by a closing square bracket. A closing square bracket on its own is not special. If a closing square bracket is required as a member of the class, it should be the first data character in the class (after an initial circumflex, if present) or escaped with a backslash.

A character class matches a single character in the subject; the character must be in the set of characters defined by the class, unless the first character in the class is a circumflex, in which case the subject character must not be in the set defined by the class. If a circumflex is actually required as a member of the class, ensure it is not the first character, or escape it with a backslash.

For example, the character class [aeiou] matches any lowercase vowel, while [^aeiou] matches any character that is not a lowercase vowel. Note that a circumflex is just a convenient notation for specifying the characters, which are in the class by enumerating those that are not. It is not an assertion: it still consumes a character from the subject string, and fails if the current pointer is at the end of the string.

A class such as [^a] will always match a newline. (Newlines should not be detected when using regular expressions in CPL.)

The minus (hyphen) character can be used to specify a range of characters in a character class. For example, [d-m] matches any letter between d and m, inclusive. If a minus character is required in a class, it must be escaped with a backslash or appear in a position where it cannot be interpreted as indicating a range, typically as the first or last character in the class. It is not possible to have the character "]" as the end character of a range, since a sequence such as [w-] is interpreted as a class of two characters. The octal or hexadecimal representation of "]" can, however, be used to end a range.

Ranges operate in ASCII collating sequence. They can also be used for characters specified numerically, for example [\000-\037].

The character types \d, \D, \s, \S, \w, and \W might also appear in a character class, and add the characters that they match to the class. For example, [\dABCDEF] matches any hexadecimal digit. A circumflex can conveniently be used with the upper case character types to specify a more restricted set of characters than the matching lower case type. For example, the class [^\W_] matches any letter or digit, but not underscore.

All non-alphanumeric characters other than \, -, ^ (at the start) and the terminating] are non-special in character classes, but it does no harm if they are escaped.

Vertical Bar

Vertical bar characters are used to separate alternative patterns. For example, the pattern

```
gilbert|sullivan
```

matches either "gilbert" or "sullivan." Any number of alternatives might appear, and an empty alternative is permitted (matching the empty string). The matching process tries each alternative in turn, from left to right, and the first one that succeeds is used. If the alternatives are within a subpattern (defined below), "succeeds" means matching the rest of the main pattern as well as the alternative in the subpattern.

Lowercase-Sensitivity

By default, CPL conditions that take regular-expression arguments perform a case-insensitive match. In all other places where the the appliance performs a regular-expression match, the match is case sensitive.

> **Note:** In CPL, use the `.case_sensitive` condition modifier for case sensitivity, rather than relying on Perl syntax.

Override the default for case sensitivity by using the following syntax:

- (?i) Sets case-insensitive matching mode.
- (?-i) Sets case-sensitive matching mode.

The scope of a mode setting depends on where in the pattern the setting occurs. For settings that are outside any subpattern (see the next section), the effect is the same as if the options were set or unset at the start of matching. The following patterns all behave in exactly the same way:

- (?i)abc
- a(?i)bc
- ab(?i)c
- abc(?i)

In other words, such “top level” settings apply to the whole pattern (unless there are other changes inside subpatterns). If there is more than one setting of the same option at the top level, the rightmost setting is used.

If an option change occurs inside a subpattern, the effect is different. This is a change of behavior in Perl 5.005. An option change inside a subpattern affects only that part of the subpattern that follows it, so (a(?i)b)c matches abc and aBc and no other strings (assuming the default is case sensitive). By this means, options can be made to have different settings in different parts of the pattern. Any changes made in one alternative do carry on into subsequent branches within the same subpattern. For example (a(?i)b|c) matches "ab", "aB", "c", and "C", even though when matching "C" the first branch is abandoned before the option setting. This is because the effects of option settings happen at compile time. This avoids some strange side-effects.

Subpatterns

Subpatterns are delimited by parentheses (round brackets), which can be nested. Marking part of a pattern as a subpattern does two things:

- It localizes a set of alternatives.

  For example, the pattern cat(aract|erpillar|) matches one of the words “cat”, “cataract”, or “caterpillar”. Without the parentheses, it would match “cataract”, “erpillar” or the empty string.

- It sets up the subpattern as a capturing subpattern (as defined above). When the whole pattern matches, that portion of the subject string that matched the subpattern is passed back to the caller via the ovector argument of RE Engine_exec (). Opening parentheses are counted from left to right (starting from 1) to obtain the numbers of the capturing subpatterns.

  For example, if the string “the red king” is matched against the pattern the ((red|white) (king|queen)) the captured substrings are “red king”, “red”, and “king”, and are numbered 1, 2, and 3.

The fact that plain parentheses fulfill two functions is not always helpful. There are times when a grouping subpattern is required without a capturing requirement. If an opening parenthesis is followed by “?:”, the subpattern does not do any capturing, and is not counted when computing the number of any subsequent capturing subpatterns. For example, if the string “the white queen” is matched against the pattern the ((?:red|white)(king|queen)) the captured substrings are “white queen” and “queen,” and are numbered 1 and 2. The maximum number of captured substrings is 99, and the maximum number of all subpatterns, both capturing and non-capturing, is 200.

As a convenient shorthand, if any option settings are required at the start of a non-capturing subpattern, the option letters might appear between the “?” and the “:”. Thus the two patterns (?i:saturday|sunday) and (?:(?i)saturday|sunday) match exactly the same set of strings. Because alternative branches are tried from left to right, and options are not reset until the end of the subpattern is reached, an option setting in one branch does affect subsequent branches, so the above patterns match “SUNDAY” as well as “Saturday”.

Repetition

Repetition is specified by quantifiers, which can follow any of the following items:

- A single character, possibly escaped by the . metacharacter
- A character class
- A back reference (see next section)
- A parenthesized subpattern (unless it is an assertion - see below)

The general repetition quantifier specifies a minimum and maximum number of permitted matches, by giving the two numbers in curly brackets (braces), separated by a comma. The numbers must be less than 65536, and the first must be less than or equal to the second. For example, z{2,4} matches “zz”, “zzz”, or “zzzz.” A closing brace on its own is not a special character. If the second number is omitted, but the comma is present, there is no upper limit; if the second number and the comma are both omitted, the quantifier specifies an exact number of required matches. Thus [aeiou]{3,} matches at least 3 successive vowels, but might match many more, while \d{8} matches exactly 8 digits. An opening curly bracket that appears in a position where a quantifier is not allowed, or one that does not match the syntax of a quantifier, is taken as a literal character. For example, {,6} is not a quantifier, but a literal string of four characters.

The quantifier {0} is permitted, causing the expression to behave as if the previous item and the quantifier were not present. For convenience (and historical compatibility) the three most common quantifiers have single-character abbreviations:

- * Equivalent to {0,}
- + Equivalent to {1,}
- ? Equivalent to {0,1}

It is possible to construct infinite loops by following a subpattern that can match no characters with a quantifier that has no upper limit, for example (a?)*

Earlier versions of Perl gave an error at compile time for such patterns. However, because there are cases where this can be useful, such patterns are now accepted, but if any repetition of the subpattern does in fact match no characters, the loop is forcibly broken.

By default, the quantifiers are “greedy,” that is, they match as much as possible (up to the maximum number of permitted times) without causing the rest of the pattern to fail. The classic example of where this gives problems is in trying to match comments in C programs. These appear between the sequences /* and */ and within the sequence, individual * and / characters might appear. An attempt to match C comments by applying the following pattern fails because it matches the entire string due to the greediness of the .* item

```
/\*.*\*/
```

to the string

```
/* first command */ not comment /* second comment */
```

However, if a quantifier is followed by a question mark, then it ceases to be greedy, and instead matches the minimum number of times possible, so the following pattern does the right thing with the C comments.

```
/\*.*?\*/
```

The meaning of the various quantifiers is not otherwise changed, just the preferred number of matches. Do not confuse this use of question mark with its use as a quantifier in its own right. Because it has two uses, it can sometimes appear doubled, as below, which matches one digit by preference, but can match two if that is the only way the rest of the pattern matches.

```
\d??\d
```

When a parenthesized subpattern is quantified with a minimum repeat count that is greater than 1 or with a limited maximum, more store is required for the compiled pattern, in proportion to the size of the minimum or maximum.

If a pattern starts with .* then it is implicitly anchored, since whatever follows will be tried against every character position in the subject string. RE ENGINE treats this as though it were preceded by \A.

When a capturing subpattern is repeated, the value captured is the substring that matched the final iteration. For example, after the following expression has matched “tweedledum tweedledee” the value of the captured substring is “tweedledee”.

```
(tweedle[dume]{3}\s*)+
```

However, if there are nested capturing subpatterns, the corresponding captured values might have been set in previous iterations. For example, after

```
/(a|(b))+/
```

matches “aba” the value of the second captured substring is “b”.

Back References

Outside a character class, a backslash followed by a digit greater than 0 (and possibly further digits) is a back reference to a capturing subpattern earlier (i.e., to its left) in the pattern, provided there have been that many previous capturing left parentheses.

However, if the decimal number following the backslash is less than 10, it is always taken as a back reference, and causes an error only if there are not that many capturing left parentheses in the entire pattern. In other words, the parentheses that are referenced need not be to the left of the reference for numbers less than 10. See the section entitled “Backslash” above for further details of the handling of digits following a backslash.

A back reference matches whatever actually matched the capturing subpattern in the current subject string, rather than anything matching the subpattern itself. So the following pattern matches “sense and sensibility” and “response and responsibility,” but not “sense and responsibility.”

```
(sens|respons)e and \1ibility
```

There might be more than one back reference to the same subpattern. If a subpattern has not actually been used in a particular match, then any back references to it always fail. For example, the following pattern always fails if it starts to match “a” rather than “bc.” Because there might be up to 99 back references, all digits following the backslash are taken as part of a potential back reference number. If the pattern continues with a digit character, then some delimiter must be used to terminate the back reference.

```
(a|(bc))\2
```

A back reference that occurs inside the parentheses to which it refers fails when the subpattern is first used, so, for example, (a\1) never matches. However, such references can be useful inside repeated subpatterns. For example, the following pattern matches any number of “a”s and also “aba”, “ababaa” etc. At each iteration of the subpattern, the back reference matches the character string corresponding to the previous iteration. In order for this to work, the pattern must be such that the first iteration does not need to match the back reference. This can be done using alternation, as in the example above, or by a quantifier with a minimum of zero.

```
(a|b\1)+
```

Assertions

An assertion is a test on the characters following or preceding the current matching point that does not actually consume any characters. The simple assertions coded as \b, \B, \A, \Z, \z, ^ and $ are described above. More complicated assertions are coded as subpatterns. There are two kinds: those that look ahead of the current position in the subject string, and those that look behind it.

An assertion subpattern is matched in the normal way, except that it does not cause the current matching position to be changed. Lookahead assertions start with (?= for positive assertions and (?! for negative assertions. For example, the following expression matches a word followed by a semicolon, but does not include the semicolon in the match.

```
\w+(?=;)
```

The following expression matches any occurrence of “example” that is not followed by “bar”.

```
example(?!bar)
```

Note that the apparently similar pattern that follows does not find an occurrence of “bar” that is preceded by something other than “example”; it finds any occurrence of “bar” whatsoever, because the assertion (?!example) is always true when the next three characters are “bar”. A lookbehind assertion is needed to achieve this effect.

```
(?!example)bar
```

Lookbehind assertions start with (?<= for positive assertions and (?<! for negative assertions. For example, the following expression does find an occurrence of “bar” that is not preceded by “example”. The contents of a lookbehind assertion are restricted such that all the strings it matches must have a fixed length.

```
(?<!example)bar
```

However, if there are several alternatives, they do not all have to have the same fixed length. Thus (?<=bullock|donkey) is permitted, but (?<!dogs?|cats?) causes an error at compile time. Branches that match different length strings are permitted only at the top level of a lookbehind assertion. This is an extension compared with Perl 5.005, which requires all branches to match the same length of string. An assertion such as (?<=ab(c|de)) is not permitted, because its single branch can match two different lengths, but it is acceptable if rewritten to use two branches:

```
(?<=abc|abde)
```

The implementation of lookbehind assertions is, for each alternative, to temporarily move the current position back by the fixed width and then try to match. If there are insufficient characters before the current position, the match is deemed to fail.

Assertions can be nested in any combination. For example, the following expression matches an occurrence of “baz” that is preceded by “bar” which in turn is not preceded by “example”.

```
(?<=(?<!example)bar)baz
```

Assertion subpatterns are not capturing subpatterns, and might not be repeated, because it makes no sense to assert the same thing several times. If an assertion contains capturing subpatterns within it, these are always counted for the purposes of numbering the capturing subpatterns in the whole pattern. Substring capturing is carried out for positive assertions, but it does not make sense for negative assertions.

Assertions count towards the maximum of 200 parenthesized subpatterns.

Once-Only Subpatterns

With both maximizing and minimizing repetition, failure of what follows normally causes the repeated item to be re-evaluated to see if a different number of repeats allows the rest of the pattern to match. Sometimes it is useful to prevent this, either to

change the nature of the match, or to cause it fail earlier than it otherwise might, when the author of the pattern knows there is no point in carrying on.

Consider, for example, the pattern \d+example when applied to the subject line

```
123456bar
```

After matching all 6 digits and then failing to match “example,” the normal action of the matcher is to try again with only 5 digits matching the \d+ item, and then with 4, and so on, before ultimately failing. Once-only subpatterns provide the means for specifying that once a portion of the pattern has matched, it is not to be re-evaluated in this way, so the matcher would give up immediately on failing to match “example” the first time. The notation is another kind of special parenthesis, starting with (?> as in this example:

```
(?>\d+)bar
```

This kind of parenthesis “locks up” the part of the pattern it contains once it has matched, and a failure further into the pattern is prevented from backtracking into it. Backtracking past it to previous items, however, works as normal.

An alternative description is that a subpattern of this type matches the string of characters that an identical standalone pattern would match, if anchored at the current point in the subject string.

Once-only subpatterns are not capturing subpatterns. Simple cases such as the above example can be though of as a maximizing repeat that must swallow everything it can. So, while both \d+ and \d+? are prepared to adjust the number of digits they match in order to make the rest of the pattern match, (?>\d+) can only match an entire sequence of digits.

This construction can of course contain arbitrarily complicated subpatterns, and it can be nested.

Conditional Subpatterns

It is possible to cause the matching process to obey a subpattern conditionally or to choose between two alternative subpatterns, depending on the result of an assertion, or whether a previous capturing subpattern matched or not. The two possible forms of conditional subpattern are

```
(?(condition)yes-pattern)
(?(condition)yes-pattern|no-pattern)
```

If the condition is satisfied, the yes-pattern is used; otherwise the no-pattern (if present) is used. If there are more than two alternatives in the subpattern, a compile-time error occurs.

There are two kinds of condition. If the text between the parentheses consists of a sequence of digits, then the condition is satisfied if the capturing subpattern of that number has previously matched. Consider the following pattern, which contains non-significant white space to make it more readable and to divide it into three parts for ease of discussion:

```
( \( )? [^()]+ (?(1) \) )
```

The first part matches an optional opening parenthesis, and if that character is present, sets it as the first captured substring. The second part matches one or more characters that are not parentheses. The third part is a conditional subpattern that tests whether the first set of parentheses matched or not. If they did, that is, if subject started with an opening parenthesis, the condition is true, and so the yes-pattern is executed and a closing parenthesis is required. Otherwise, since no-pattern is not

present, the subpattern matches nothing. In other words, this pattern matches a sequence of non-parentheses, optionally enclosed in parentheses.

If the condition is not a sequence of digits, it must be an assertion. This might be a positive or negative lookahead or lookbehind assertion. Consider this pattern, again containing non-significant white space, and with the two alternatives on the second line:

```
(?(?=[^a-z]*[a-z])
\d{2}[a-z]{3}-\d{2}|\d{2}-\d{2}-\d{2} )
```

The condition is a positive lookahead assertion that matches an optional sequence of non-letters followed by a letter. In other words, it tests for the presence of at least one letter in the subject. If a letter is found, the subject is matched against the first alternative; otherwise it is matched against the second. This pattern matches strings in one of the two forms dd-aaa-dd or dd-dd-dd, where aaa are letters and dd are digits.

Comments

The sequence (?# marks the start of a comment which continues up to the next closing parenthesis. Nested parentheses are not permitted. The characters that make up a comment play no part in the pattern matching at all.

Performance

Certain items that might appear in patterns are more efficient than others. It is more efficient to use a character class like [aeiou] than a set of alternatives such as (a|e|i|o|u). In general, the simplest construction that provides the required behavior is usually the most efficient. Remember that non-regular expressions are simpler constructions than regular expressions, and are thus more efficient in general.

Regular Expression Engine Differences From Perl

This section describes differences between the RE ENGINE and Perl 5.005.

- Normally “space” matches space, formfeed, newline, carriage return, horizontal tab, and vertical tab. Perl 5 no longer includes vertical tab in its set of white-space characters. The \v escape that was in the Perl documentation for a long time was never in fact recognized. However, the character itself was treated as white space at least up to 5.002. In 5.004 and 5.005 it does not match \s.
- RE ENGINE does not allow repeat quantifiers on lookahead assertions. Perl permits them, but they do not mean what you might think. For example, (?!a){3} does not assert that the next three characters are not “a”. It just asserts that the next character is not “a” three times.
- Capturing subpatterns that occur inside negative lookahead assertions are counted, but their entries in the offsets vector are never set. Perl sets its numerical variables from any such patterns that are matched before the assertion fails to match something (thereby succeeding), but only if the negative lookahead assertion contains just one branch.

- Though binary zero characters are supported in the subject string, they are not allowed in a pattern string because it is passed as a normal C string, terminated by zero. The escape sequence “\0” can be used in the pattern to represent a binary zero.
- The following Perl escape sequences are not supported: \l, \u, \L, \U, \E, \Q. In fact these are implemented by Perl’s general string handling and are not part of its pattern-matching engine.
- The Perl \G assertion is not supported as it is not relevant to single pattern matches.
- RE ENGINE does not support the (?{code}) construction.
- There are at the time of writing some oddities in Perl 5.005_02 concerned with the settings of captured strings when part of a pattern is repeated. For example, matching “aba” against the pattern /^(a(b)?)+$/ sets $2 to the value “b”, but matching “aabbaa” against /^(aa(bb)?)+$/ leaves $2 unset. However, if the pattern is changed to /^(aa(b(b))?)+$/ then $2 (and $3) get set. In Perl 5.004 $2 is set in both cases, and that is also true of RE ENGINE.
- Another as yet unresolved discrepancy is that in Perl 5.005_02 the pattern /^(a)?(?(1)a|b)+$/ matches the string “a”, whereas in RE ENGINE it does not. However, in both Perl and RE ENGINE /^(a)?a/ matched against “a” leaves $1 unset.
- RE ENGINE provides some extensions to the Perl regular expression facilities: Although lookbehind assertions must match fixed length strings, each alternative branch of a lookbehind assertion can match a different length of string. Perl 5.005 requires them all to have the same length.

Note: When regular expressions are used to match a URL, a space character matches a %20 in the request URL. However, a %20 in the regular-expression pattern will not match anything in any request URL, because "%20" is normalized to " " in the subject string before the regex match is performed.

Testing and Troubleshooting

If you are experiencing problems with your policy files or would like to monitor policy evaluation, you can do the following to troubleshoot policy:

- Policy trace–Allows you to examine how the appliance policy is applied to a particular request. This is appropriate if you want to monitor policy evaluation for single or multiple transactions over shorter periods of time.
- Policy coverage–Reports on the rules and objects that match user requests processed through the appliance's current policy. This is appropriate if you want to monitor evaluation for multiple transactions over longer periods of time.

Overview of Policy Tracing

Tracing allows you to examine how the appliance policy is applied to a particular request. To configure tracing in a policy file, you use several policy language properties to enable tracing, set the verbosity level, and specify the path for output. Using appropriate conditions to guard the tracing rules, you can be specific about the requests for which you gather tracing information.

> **Note:** Use policy tracing for troubleshooting only. Tracing is best used temporarily for troubleshooting, while the "log_message()" on page 608 action is best for on-going monitoring. If tracing is enabled in a production setting, appliance performance degrades. After you complete troubleshooting, be sure to remove policy tracing.

CPL provides the following trace-related properties:

- "trace.request()" on page 579–Enables tracing and includes a description of the transaction being processed in the trace. No trace output is generated if this is set to no.
- "trace.destination()" on page 576–Directs the trace output to a user-named trace log.

In addition to policy tracing, you can report on the policy rules that are used in transactions. Code coverage shows the frequency with which certain pieces of policy are matched. (A 'piece' of policy specifically means any place in CPL where a condition, layer, or filter is evaluated.) With this information, policy can be reordered for better performance or deleted if policy is not useful.

Enabling Request Tracing

Use the "trace.request()" on page 579 property to enable request tracing. Request tracing logs a summary of information about the transaction: request parameters, property settings, and the effects of all actions taken. This property uses the following syntax:

```
trace.request(yes|no)
```

where:

- yes: Generate full trace details for the current request.
- no: (Default behavior) Do not generate trace output.

Example

Enable full tracing information for all transactions:

```
<cache>
  trace.request(yes)
```

Configuring the Path

Use the "trace.destination()" on page 576 property to configure where the appliance saves trace information. The trace destination can be set and reset repeatedly. It takes effect (and the trace is actually written) only when the appliance has finished processing the request and any associated response. Trace output is saved to an object that is accessible using a console URL in the following form:

https://*appliance_IP_address*:8081/Policy/Trace/*path*

where path is, by default, default_trace.html. This property allows you to change the destination. The property uses the following syntax:

`trace.destination(path)`

where:

- *path*: By default, the path is default_trace.html. You can change *path* to a filename or directory path, or both. If only a directory is provided, the default trace filename is used.

You can view policy statistics through the Management Console (**Statistics > Advanced > Policy > List of policy URLs**).

Example

Two destinations are configured for policy tracing information:

```
<Proxy>
  client.address=10.25.0.0/16 trace.destination(internal_trace.html)
  client.address=10.0.0.0/8 trace.destination(external_trace.html)
```

The console URLs for retrieving the information are:

https://*appliance_IP_address*:8081/Policy/Trace/internal_trace.html

https://*appliance_IP_address*:8081/Policy/Trace/external_trace.html

Using Trace Information to Improve Policies

To help you understand tracing, this section shows annotated trace output. These traces show the evaluation of specific requests against a particular policy.

For information on the trace result late, refer to TECH245782:

http://www.symantec.com/docs/TECH245782

Example

Refer to the example below. It is intended to be an illustration of most aspects of policy trace output. The example addresses the following policy requirements:

- DNS lookups are restricted except for a site being hosted.
- There is no access to reverse DNS so that is completely restricted.
- Any requests not addressed to the hosted site either by name or subnet should be rejected.
- FTP POST requests should be rejected.
- Request URLs for the hosted site are to be rewritten and a request header on the way into the site.

```
; DNS lookups are restricted except for one site that is being hosted
restrict dns
except
my_site.com
end

; No access to RDNS
restrict rdns
all
end

define subnet my_subnet
  10.11.12.0/24
end

<Proxy>
  trace.request(yes)

<Proxy>
  deny url.host.is_numeric=no url.domain=!my_site.com
  deny url.address=!my_subnet

<Proxy>
  deny ftp.method=STOR

<Proxy>
  url.domain=my_site.com action.test(yes)
```

```
define action test
  set(request.x_header.test, "test")
  rewrite(url, "(.*)\.my_site.com", "$(1).his_site.com")
  end
```

Since "trace.request()" on page 579 is set to yes, a policy trace is performed when client requests are evaluated.

The following is the trace output produced for an HTTP GET request for http://www.my_site.com/home.html.

The line numbers shown at the left do not appear in actual trace output. They are added here for annotation purposes.

```
1   start transaction ------------------------------
2     CPL Evaluation Trace:
3               <Proxy>
4       MATCH:     trace.request(yes)
5               <Proxy>
6       miss:     url.domain=!//my_site.com/
7       miss:     url.address=!my_subnet
8               <Proxy>
9       n/a  :     ftp.method=STOR
10              <Proxy>
11      MATCH:     url.domain=//my_site.com/ action.foo(yes)
12   connection: client.address=10.10.0.10 proxy.port=36895
13   time: 2003-09-11 19:36:22 UTC
14   GET http://www.my_site.com/home.html
15   DNS lookup was unrestricted
16   rewritten URL(s):
17   cache_url/server_url/log_url=http://www.his_site.com/
18   User-Agent: Mozilla 8.6 (Non-compatible)
19   user: unauthenticated
20   set header= (request)
21    value='test'
22  end transaction -------------------------------
```

Notes:

- Lines 1 and 22 are delimiters indicating where the trace for this transaction starts and ends.
- Line 2 introduces the rule evaluation part of the trace. A rule evaluation part is generated when "trace.request()" on page 579 is set to yes.
- Lines 3 to 4 and 10 to 11 show rule matches, and are included when "trace.request()" on page 579 is set to yes.
- Lines 5 to 9 show rule misses, and are included when "trace.request()" on page 579 is set to yes.
- Line 9 shows how a rule (containing an FTP specific condition) that is not applicable to this transaction (HTTP) is marked as n/a.
- Lines 12 to 21 are generated as a result of "trace.request()" on page 579.
- Line 12 shows client related information.

- Line 13 shows the time the transaction was processed.
- Line 14 is a summary of the request line.
- Line 15 indicates that DNS lookup was attempted during evaluation, and was unrestricted. This line only appears if there is a DNS restriction and a DNS lookup was required for evaluation.
- Lines 16 and 17 indicate that the request URL was rewritten, and show the effects.
- Line 19 indicates that the user was not required to authenticate. If authentication had been required, the user identity would be displayed.
- Lines 20 and 21 show the results of the header modification action.

The following is a trace of the same policy, but for a transaction in which the request URL has an IP address instead of a hostname.

```
1   start transaction ------------------------------
2     CPL Evaluation Trace:
3               <Proxy>
4       MATCH:     trace.request(yes)
5               <Proxy>
6       miss:      url.host.is_numeric=no
7       miss:      url.address=!my_subnet
8               <Proxy>
9       n/a  :     ftp.method=STOR
10              <Proxy>
11      miss:      url.domain=//my_site.com/
12   connection: client.address=10.10.0.10 proxy.port=36895
13   time: 2003-09-11 19:33:34 UTC
14   GET http://10.11.12.13/home.html
15   DNS lookup was restricted
16   RDNS lookup was restricted
17   User-Agent: Mozilla 8.6 (Non-compatible)
18   user: unauthenticated
19  end transaction --------------------------------
```

This shows many of the same features as the earlier trace, but has the following differences:

- Line 12–The URL requested had a numeric host name.
- Lines 15 and 16–Both DNA and RDNS lookups were restricted for this transaction.
- Line 11–Because RDNS lookups are restricted, the rule missed; no rewrite action was used for the transaction and no rewrite action is reported in the transaction summary (lines 12-18).

Trace output can be used to determine the cause of action conflicts that may be reported in the event log. For example, consider the following policy fragment:

```
<Proxy>
  trace.request(yes)
```

```
<Proxy> action.set_header_1(yes)
  [Rule] action.set_header_2(yes)
    action.set_header_3(yes)

define action set_header_1
  set(request.x_header.Test, "one")
end

define action set_header_2
  set(request.x_header.Test, "two")
end

define action set_header_3
  set(request.x_header.Test, "three")
end
```

Because they all set the same header, these actions will conflict. In this example, the conflict is obvious because all the actions are enabled in the same layer. However, conflicts can also arise when actions are enabled by completely independent portions of policy. If an action conflict occurs, one of the actions is dropped and an event log entry is made similar to the following:

```
Policy: Action discarded, 'set_header_1' conflicts with an action already committed
```

The conflict is reflected in the following trace of a request for //www.my_site.com/home.html:

```
1  start transaction -------------------------------
2    CPL Evaluation Trace:
3              <Proxy>
4      MATCH: trace.request(yes)
5              <Proxy> action.set_header_1(yes)
6                [Rule] action.set_header_2(yes)
7      MATCH:         action.set_header_1(yes)
8      MATCH:         action.set_header_2(yes)
9      MATCH:         action.set_header_3(yes)
10 connection: client.address=10.10.0.10 proxy.port=36895
11 time: 2003-09-12 15:56:39 UTC
12 GET http://www.my_site.com/home.html
13   User-Agent: Mozilla 8.6 (Non-compatible)
14 user: unauthenticated
15 Discarded Actions:
16   set_header_1
17   set_header_2
18 set header=set_header_3 (request)
19   value='three'
20 end transaction --------------------------------
```

Notes:

- Layer and section guard expressions are indicated in the trace (lines 7 and 8) before any rules subject to the guard (line 9).
- Line 15 indicates that actions were discarded due to conflicts.
- Lines 16 and 17 show the discarded actions.
- Line 18 shows the remaining action, while line 19 shows the effect of the action on the header value.

Determining Which Policy Rules are Matched in Transactions

You can use policy coverage to report on the rules that match user requests processed through the appliance's current policy. Unlike a policy trace, which you enable and disable through CPL, policy coverage is always enabled and running. The appliance resets the policy coverage counter whenever new policy is installed.

To determine which rules match proxied requests and the frequency with which the rules are 'hit', display the current policy coverage in the Management Console (select **Statistics > Advanced** and scroll down to **Policy**. Then, click **Show Policy Coverage**).

The Policy Coverage page displays all policy (Visual, Local, Central and Forward) on the appliance in CPL. The number of times that each rule is hit is listed to the left of each policy item that can be tracked. The following is an example of the output on the Policy Coverage page:

```
: ; Installed Policy -- compiled at: Mon, 03 Mar 2014 14:21:10 UTC
: ;     Default proxy policy is DENY
:
: ; Policy Rules
: <Proxy>
34:     authenticate(local) authenticate.force(no) authenticate.mode(origin)


: <Proxy>
34:     authenticate.guest("guest", 0, "local")

0: <Proxy>  user.is_guest=yes (0)
0:     DENY url.domain=//www.google.com/ (0)
0:     DENY streaming.client=yes (0)

: <Proxy>
1:     DENY url.domain=//www.tinydeal.com/ (1)
```

> **Note:** Domains in a define section do not have conditions, so they are not included in coverage. For example, consider a transaction involving a domain in the following section:
>
> ```
> define url.domain condition my_domains
>    example.com
>    company.com
>    test.com
> end
> ```

Policy coverage tracks when the `my_domains` condition is hit; however, it does not track transactions involving specific URLs within the `my_domains` condition.

For more information on policy coverage, refer to TECH241425:

http://www.symantec.com/docs/TECH241425