Content Policy Language Reference

Version

Contact Information

Americas:

Symantec Corporation
350 Ellis Street
Mountain View, CA 94043

Rest of the World:

Symantec Limited
Ballycoolin Business Park
Blanchardstown, Dublin 15, Ireland

Document Number: 231-03019
Document Revision:

Contents

Preface: Introducing the Content Policy Language

About the Document Organization xiii
Notes and Warnings xv

Chapter 1: Overview of Content Policy Language 17

Concepts 17
- Transactions 18
- Policy Model 18
- Role of CPL 19

CPL Basics 19
- Comments 20
- Rules 20
- Notes 21
- Quoting 22
- Layers 22
- Sections 23
- Definitions 24
- Referential Integrity 26
- Substitutions 26

Writing Policy Using CPL 27
- Authentication and Denial 27
- Installing Policy 28
- CPL General Use Characters and Formatting 29

Troubleshooting Policy 29
Upgrade/Downgrade Issues 30
- CPL Syntax Deprecations 30
- Conditional Compilation 31

Chapter 2: Managing Content Policy Language 33

Understanding Transactions and Timing 33
- <Admin> Transactions 33
- <Proxy> Transactions 34
- <DNS-Proxy> Transactions 35
- <Cache> Transactions 36
- <Exception> Transaction 36
- <Forwarding> Transactions 36
- <SSL> Transactions 37

Timing 37
Understanding Layers 38
<Admin> Layers 38
<Cache> Layers 39
<Exception> Layers 40
<Forward> Layers 41
<Proxy> Layers 41
<DNS-Proxy> Layers 42
<SSL-Intercept> Layers 42
<SSL> Layers 43
Layer Guards 43
Timing 44
Understanding Sections 44
[Rule] 45
[url] 46
[url.domain] 46
[url.regex] 46
[server_url.domain] 46
Section Guards 47
Defining Policies 47
Blacklists and Whitelists 48
General Rules and Exceptions to a General Rule 48
Best Practices 50

Chapter 3: Condition Reference 53

Condition Syntax 53
Pattern Types 54
Unavailable Conditions 55
Layer Type Restrictions 55
Global Restrictions 55
Condition Reference 56
admin.access= 57
ami.config.threat-protection.malware-scanning.config_setting= 58
appliance.id= 59
attribute.name= 60
authenticated= 66
bitrate= 67
category= 69
client.address= 70
client.address.country= 72
client.address.login.count= 73
client.certificate.common_name= 74
client.certificate.requested= 75
client.certificate.subject= 76
client.certificate.subject_directory_attribute 77

client.connection.dscp= 79
client.connection.negotiated_cipher= 80
client.connection.negotiated_cipher.strength= 81
client.connection.negotiated_ssl_version= 82
client.effective_address= 83
client.effective_address.country= 84
client.effective_address.is_overridden= 85
client.host= 86
client.host.has_name= 87
client.protocol= 88
condition= 89
console_access= 91
content_management= 92
data_leak_detected= 93
date[.utc]= 94
day= 95
dns.client_transport= 96
dns.request.address= 97
dns.request.category= 98
dns.request.class= 99
dns.request.name= 100
dns.request.opcode= 101
dns.request.type= 102
dns.response.a= 103
dns.response.aaaa= 104
dns.response.cname= 105
dns.response.code= 106
dns.response.nodata= 107
dns.response.ptr= 108
exception.id= 109
ftp.method= 111
group= 112
has_attribute.name= 115
has_client= 117
health_check= 118
hour= 119
http.connect= 121
http.connect.User-Agent= 122
http.method= 123
http.method.custom= 124
http.method.regex= 125
http.request.apparent_data_type= 126
http.request.body.size= 128
http.request.body.max_size_exceeded= 129
http.request.data= 130
http.request.detection.result.application_protection_set= 131
http.request.detection.result.validation= 132
http.request[ ].modifier= 133
http.request_line.regex= 135

http.request.version= 136
http.response.apparent_data_type= 137
http.response.code= 139
http.response.data= 140
response.icap.apparent_data_type= 141
http.response.version= 142
http.transparent_authentication= 143
http.websocket= 144
icap_error_code= 145
icap_method.header.header_name= 146
is_healthy.health_check_name= 148
iterator= 149
ldap.attribute.ldap_attribute_name= 150
ldap.attribute.ldap_attribute_name.as_number= 151
ldap.attribute.ldap_attribute_name.count= 152
ldap.attribute.ldap_attribute_name.exists= 153
live= 154
minute= 155
month= 156
proxy.address= 157
proxy.card= 159
proxy.port= 160
p2p.client= 161
raw_url.regex= 162
raw_url.host.regex= 163
raw_url.path.regex= 164
raw_url.pathquery.regex= 165
raw_url.port.regex= 166
raw_url.query.regex= 167
realm= 168
release.id= 170
release.version= 171
request.header.content-length.as_number= 172
request.header.header_name= 173
request.header.header_name.address= 175
request.header.*header_name*.exists= 176
request.header.*header_name*.count= 177
request.header.*header_name*.length= 178
request.header.Referer.url= 179
request.header.Referer.url.category= 182
request.header.Referer.url.host.is_private= 183
request.icap.apparent_data_type= 184
request.raw_headers.count= 185
request.raw_headers.length= 186
request.raw_headers.regex= 187
request.x_header.header_name= 188
request.x_header.header_name.address= 189
request.x_header.*header_name*.count= 190
request.x_header.*header_name*.exists= 191

request.x_header.*header_name*.length= ... 192
response.header.content-length.as_number= ... 193
response.header.header_name= ... 194
response.raw_headers.count= ... 195
response.raw_headers.length= ... 196
response.raw_headers.regex= ... 197
response.x_header.header_name= ... 198
risk_score= ... 199
server.certificate.hostname= ... 200
server.certificate.hostname.category= ... 202
server.certificate.subject= ... 203
server.connection.dscp= ... 204
server.connection.negotiated_cipher= ... 205
server.connection.negotiated_cipher.strength= ... 206
server.connection.negotiated_ssl_version= ... 207
server_url= ... 208
server_url.category= ... 211
server_url.host.is_private= ... 212
service.group= ... 213
service.name= ... 214
socks= ... 215
socks.accelerated= ... 216
socks.method= ... 217
socks.version= ... 218
source.port= ... 219
ssl.proxy_mode= ... 220
streaming.client= ... 221
streaming.content= ... 222
streaming.rtmp.app_name= ... 223
streaming.rtmp.method= ... 224
streaming.rtmp.page_url= ... 225
streaming.rtmp.stream_name= ... 226
streaming.rtmp.swf_url= ... 227
time= ... 228
tunneled= ... 230
url= ... 231
url.application.name= ... 239
url.application.operation= ... 240
url.category= ... 241
url.host.is_private= ... 242
user= ... 243
user.authentication_error= ... 245
user.authorization_error= ... 246
user.domain= ... 247
user.is_guest= ... 248
user.login.address= ... 249
user.login.count= ... 250
user.login.time= ... 251
user.regex= ... 252

user.x509.issuer= 253
user.x509.serialNumber= 254
user.x509.subject= 255
virus_detected= 256
weekday= 257
year= 258

Chapter 4: Property Reference 259

Property Reference 259
access_log( ) 260
access_server( ) 261
action( ) 262
adn.connection.dscp( ) 263
adn.server( ) 264
adn.server.optimize( ) 265
adn.server.optimize.inbound( ) 266
adn.server.optimize.outbound( ) 267
advertisement( ) 268
allow 269
always_verify( ) 270
attack_detection.failure_weight( ) 271
authenticate() 272
authenticate.authorization_refresh_time( ) 273
authenticate.charset( ) 274
authenticate.credential_refresh_time() 275
authenticate.credentials.address( ) 276
authenticate.guest( ) 277
authenticate.force( ) 278
authenticate.force_307_redirect() 279
authenticate.form( ) 280
authenticate.forward_credentials() 281
authenticate.forward_credentials.log() 282
authenticate.mode( ) 283
authenticate.new_pin_form() 285
authenticate.query_form() 286
authenticate.redirect_stored_requests() 287
authenticate.surrogate_refresh_time() 288
authenticate.tolerate_error() 289
authenticate.transaction 290
authenticate.use_url_cookie( ) 292
authorize.add_group() 293
authorize.tolerate_error() 294
bypass_cache( ) 295
cache( ) 296
check_authorization( ) 298
client.address.login.log_out_other() 299
client.certificate.require( ) 300
client.certificate.validate( ) 301
client.certificate.validate.check_revocation() 302

client.connection.dscp() 303
client.connection.encrypted_tap( ) 304
client.effective_address() 305
cookie_sensitive( ) 307
delete_on_abandonment( ) 308
deny( ) 309
deny.unauthorized( ) 310
detect_protocol( ) 311
direct( ) 312
dns.respond( ) 313
dns.respond.a( ) 314
dns.respond.aaaa( ) 315
dns.respond.ptr( ) 316
dynamic_bypass( ) 317
exception( ) 318
exception.autopad( ) 320
exception.format( ) 321
force_cache( ) 323
force_deny( ) 327
force_exception( ) 328
force_protocol( ) 329
forward( ) 330
forward.fail_open( ) 331
ftp.match_client_data_ip( ) 332
ftp.match_server_data_ip( ) 333
ftp.server_connection( ) 334
ftp.server_data( ) 335
ftp.transport( ) 336
ftp.welcome_banner( ) 337
http.allow_compression( ) 338
http.allow_decompression( ) 339
http.client.allow_encoding( ) 340
http.client.persistence( ) 341
http.client.recv.timeout( ) 342
http.compression_level( ) 343
http.force_ntlm_for_server_auth( ) 344
http.refresh.recv.timeout( ) 346
http.request.apparent_data_type.allow() 347
http.request.apparent_data_type.deny() 348
http.request.body.max_size() 349
http.request.detection.injection.sql() 350
http.request.detection.other() 351
"Supported HTTP Attributes" on page 133http.request.version( ) 352
http.response.parse_meta_tag.Cache-Control( ) 353
http.response.parse_meta_tag.Expires( ) 354
http.response.parse_meta_tag.pragma-no-cache( ) 355
http.response.version( ) 356
http.server.accept_encoding( ) 357
http.server.accept_encoding.allow_unknown() 358

http.server.connect_attempts( ) 359
http.server.connect_timeout( ) 360
http.server.persistence( ) 361
http.server.recv.timeout( ) 362
im.tunnel() 363
integrate_new_hosts( ) 364
log.rewrite.field-id( ) 365
log.suppress.field-id( ) 366
max_bitrate( ) 367
never_refresh_before_expiry( ) 368
never_serve_after_expiry( ) 369
notify_email.recipients() 370
pipeline( ) 371
reference_id( ) 372
reflect_ip( ) 373
refresh( ) 374
remove_IMS_from_GET( ) 375
remove_PNC_from_GET( ) 376
remove_reload_from_IE_GET( ) 377
request.icap_service( ) 378
request.icap_service.secure_connection() 380
response.icap_feedback( ) 382
response.icap_feedback.force_interactive( ) 384
response.icap_feedback.interactive( ) 386
response.icap_feedback.non_interactive( ) 388
response.icap_mirror 390
response.icap_service( ) 391
response.icap_service.force_rescan( ) 393
response.icap_service.secure_connection() 394
response.raw_headers.max_count() 396
response.raw_headers.max_length() 397
response.raw_headers.tolerate() 398
risk_score.maximum() 399
risk_score.other() 400
server.authenticate.basic() 401
server.authenticate.constrained_delegation() 403
server.authenticate.constrained_delegation.spn() 404
server.certificate.validate() 405
server.certificate.validate.check_revocation() 407
server.certificate.validate.ignore() 408
server.connection.client_keyring() 409
server.connection.dscp() 410
server_url.dns_lookup() 411
shell.prompt( ) 412
shell.realm_banner( ) 413
shell.welcome_banner( ) 414
socks.accelerate( ) 415
socks.authenticate( ) 416
socks.authenticate.force( ) 417

socks_gateway( ) 418
socks_gateway.fail_open( ) 419
ssl.forward_proxy( ) 420
ssl.forward_proxy.hostname( ) 421
ssl.forward_proxy.issuer_keyring( ) 422
ssl.forward_proxy.preserve_untrusted 423
ssl.forward_proxy.server_keyring( ) 424
ssl.forward_proxy.splash_text( ) 425
ssl.forward_proxy.splash_url( ) 426
streaming.fast_cache( ) 427
streaming.rtmp.tunnel_encrypted( ) 428
streaming.transport( ) 429
terminate_connection( ) 430
trace.destination( ) 431
trace.header( ) 432
trace.request( ) 433
transform.data_type() 434
trust_destination_ip() 436
ttl( ) 437
ua_sensitive( ) 438
user.login.log_out( ) 439
user.login.log_out_other( ) 440
webpulse.categorize.mode( ) 441
webpulse.categorize.send_headers() 442
webpulse.categorize.send_url() 443
webpulse.notify.malware() 444

Chapter 5: Action Reference 445

Argument Syntax 445
Action Reference 445
append( ) 446
delete( ) 447
delete_matching( ) 449
diagnostic.stop(pcap) 450
iterate() 451
iterator.append( ) 452
iterator.delete( ) 453
iterator.rewrite( ) 454
log_message( ) 455
notify_email( ) 456
notify_snmp( ) 457
redirect( ) 458
request_redirect( ) 460
rewrite( ) 462
set( ) 465
transform( ) 468

Chapter 6: Definition Reference 471

Definition Names 471

define action 472
define active_content 474
define category 476
define condition 478
define javascript 480
define policy 482
define server_url.domain condition 483
define string 485
define subnet 486
define url condition 487
define url.domain condition 489
define url_rewrite 491
restrict dns 494
restrict rdns 495

Appendix A: Glossary

Appendix B: Testing and Troubleshooting

Overview of Policy Tracing 503
Enabling Request Tracing 503
Using Trace Information to Improve Policies 504
Determining Which Policy Rules are Matched in Transactions 508

Appendix C: Recognized HTTP Headers

Appendix D: Using Regular Expressions

Regular Expression Syntax 515
Regular Expression Details 517
Backslash 518
Circumflex and Dollar 519
Period (Dot) 520
Square Brackets 520
Vertical Bar 521
Lowercase-Sensitivity 521
Subpatterns 522
Repetition 523
Back References 525
Assertions 525
Once-Only Subpatterns 527
Conditional Subpatterns 527
Comments 528
Performance 528
Regular Expression Engine Differences From Perl 528

Preface: Introducing the Content Policy Language

The Blue Coat *Content Policy Language* (CPL) is a powerful, flexible language that enables you to specify a variety of authentication, Web-access, and networking policies. ProxySGpolicy is written in CPL, and every Web request is evaluated based on the installed policy. The language is designed so that policies can be customized to an organization's specific set of users and unique enforcement needs.

CPL uses the settings created when you configured the ProxySG appliance to your specifications. You can also:

- Use predefined common actions and transformations.
- Define your own conditions and actions.
- Exert fine-grained control over various aspects of appliance behavior.
- Create policy layers, allowing for multiple policy decisions for each request.
- Have a specific condition trigger multiple actions.
- Create authentication-aware policy, including user and group configuration.
- Support multiple authentication realms.
- Configure policy event logging.
- Debug policy using built-in tools.

About the Document Organization

This document is organized for easy reference, and is divided into the following sections and chapters:

Chapter 1 – *Overview of Content Policy Language*	Provides an overview of CPL, including concepts, CPL basics, writing and troubleshooting policy and upgrade/downgrade issues.
Chapter 2 – *Managing Content Policy Language*	Discusses understanding transactions, timing, layers, and sections, defining policies, and best practices.
Chapter 3 – *Condition Reference*	Lists conditions that are supported by CPL and provides an explanation for the usage.
Chapter 4 – *Property Reference*	Lists properties that are supported by CPL and provides an explanation for the usage.
Chapter 5 – *Action Reference*	Lists actions that are supported by CPL and provides an explanation for the usage.
Chapter 6 – *Definition Reference*	Lists definitions that are supported by CPL and provides an explanation for the usage.
Appendix A – *Glossary*	Defines terms used in this manual.
Appendix B – *Testing and Troubleshooting*	Explains using policy trace properties.

Appendix C – *Recognized HTTP Headers*	Lists all recognized HTTP 1.1 headers and indicates how the appliance interacts with them.
Appendix D—*Using Regular Expressions*	Discusses regular expressions and how to use them.

Note: CPL substitutions documentation is available in the *ProxySG Log Fields and CPL Substitutions Reference*:

https://www.symantec.com/docs/DOC11251

Document Conventions

Conventions	Definition
Italics	The first use of a new or Blue Coat-proprietary term.
`Courier font`	Screen output. For example, command line text, file names, and Blue Coat CPL.
`Courier Italics`	A command line variable that is to be substituted with a literal name or value pertaining to the appropriate facet of your network system.
`Courier Boldface`	A Blue Coat literal to be entered as shown.
Arial Boldface	Screen elements in the Management Console.
`{ }`	One of the parameters enclosed within the braces must be supplied
`[ ]`	An optional parameter or parameters.
`|`	A separator between parameters. Use one of the parameters separated by the pipe character.

Notes and Warnings

The following is provided for your information and to caution you against actions that can result in data loss or personal injury:

Note:	Information to which you should pay attention.

Important:	Critical information that is not related to equipment damage or personal injury (for example, data loss).

WARNING:	**Used *only* to inform you of danger of personal injury or physical damage to equipment. An example is a warning against electrostatic discharge (ESD) when installing equipment.**

Chapter 1: *Overview of Content Policy Language*

The Blue Coat® Content Policy Language (CPL) is a programming language with its own concepts and rules that you must follow.

Topics in this Chapter

This chapter includes information about the following topics:

- "Concepts" on page 17
- "CPL Basics" on page 19
- "Writing Policy Using CPL" on page 27
- "Troubleshooting Policy" on page 29
- "Upgrade/Downgrade Issues" on page 30

Concepts

In Blue Coat 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
- Cache 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, Blue Coat 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 (except where the semicolon appears inside quotation marks (" ") as part of a trigger pattern expression or property setting).

For example:

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

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=`), response (`response.header.Content-Type=`), related user (`user=`, `group=`), or system state (`time=`).

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( )` 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_expression`*
- 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
  ```
- Long 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
```

Important: Within a `define action` or `define url_rewrite` 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 [action]> [layer_condition][layer_properties] ...
   layer_content
```

where:

- The *layer_type* defines the 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 38.
- The optional *action*, separated from the layer type by space, is a CPL User-defined Identifier (see "Understanding Sections" on page 44), basically an alphabetic character followed by alphanumeric or underscore characters.
- The optional *layer_condition* is a list of triggers, all of which must evaluate to true before the layer content is evaluated.
- The optional *layer_properties* is a 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.

- The *layer_content* is a list of rules, possibly organized in *sections*. (see following). 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.

You have the option of choosing one of two section types.

The first section has the form:

```
[section_type  [action]] [section_condition][section_properties]

   section_content
```

The second section, known as the Policy Substitution section, has the form:

```
[string[.case_sensitive]<substitution-expression>[comment_token][section_conditi
on][section_properties]

   section_content
```

Note: Performance benefits are most noticeable when the number of rules in the section are greater than 10.

---

where:

- The *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 action) are required.
- The optional *action*, separated from the section type by space, is a CPL User-defined Identifier similar to a layer action.

- The optional *comment_token*, seperated by a from the substitution_expression by a space, is an identifier or quoted string that is displayed in policy traces to identify the purpose of the layer.
- The optional *section_condition* is a list of triggers, all of which must evaluate to true before the section content is evaluated.
- The *section_properties* is a 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.
- The optional *section_content* is a list of rules, each of which begins with a string criterion followed by other CPL triggers and properties. The string criterion is compared aganist the value of the substitution-expression forth current transaction. Multiple rules may begin with the same string criterion. Rules with matching string criteria will be evaluated in order.
- The *substitution-expressions* can be found in Appendix D.
- The optional *.case_sensitive* is a modifier which requires a case sensitive match. By default, the match is case insensitive.

Note: The Policy Substution Section type 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 Chapter 6: "Definition Reference" on page 471 for more 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 the `action( )` property. 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: `url_rewrite` definitions, `active_content` definitions, and `javascript` definitions.

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, see the *ProxySG Log Fields and CPL Substitutions Reference*.

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 ProxySG 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 the VPM, Central and Local policy files is configurable. The recommended evaluation order is VPM, Local, Central. The Forward policy file is always last.

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 CPL 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 or 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* 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.
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= trigger that tests for that workstation's IP address.

For more information on generating and retrieving policy trace, see Appendix B: "Testing and Troubleshooting".

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 Issues

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

CPL Syntax Deprecations

As the power of CPL has increased, the CPL language has evolved. To allow continuous evolution, the CPL language constructs are now more regular and flexible. Older language constructs have been replaced with new constructs of equal or greater power.

However, this also implies 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 software versions, 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 6.5.x versions up to 6.5.8.9 with:

  ```
  #if release.version=6.5..6.5.8.9
  ; guarded rules
  ...
  #endif
  ```

  Protect policy introduced in 6.6.x with:

  ```
  #if release.version=6.6..
  ; guarded rules
  ...
  #endif
  ```

Chapter 2: *Managing Content Policy Language*

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

Topics of this Chapter

This chapter includes information about the following topics:

- "Understanding Transactions and Timing" on page 33
- "Understanding Layers" on page 38
- "Understanding Sections" on page 44
- "Defining Policies" on page 47
- "Best Practices" on page 50

Understanding Transactions and Timing

Transactions are classified as in several different types:

- `<Admin>`
- `<Tenant>`
- `<Proxy>`
- `<Cache>`
- `<DNS-Proxy>`
- `<Exception>`
- `<Forwarding>`
- `<SSL>`
- `<SSL-Intercept>`

Only a subset of layer types, conditions, properties, and actions is appropriate for each of these four transaction types.

<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 ProxySG 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.

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

<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 each of the protocol-specific proxy transactions.

Table 2.1: When Policy is Evaluated

Transaction Type	Policy is Evaluated....
Tunneled TCP transactions	before the connection is established to the origin server.
HTTP proxy transactions	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 transactions:	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 transactions	Policy is examined before the requested object is fetched.

Table 2.1: When Policy is Evaluated (Continued)

Transaction Type	Policy is Evaluated....
Real Media streaming transactions	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 transactions	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 transactions	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.

<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.

<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=` and `group=`, or the `authenticate()` property.

An HTTP cache transaction is examined in two stages:

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

<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.

<SSL> Transactions

Two kinds 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.

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 address (`client.address=`) or proxy port (`proxy.port=`), 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()` can be conditional on the URL (`url=`), but for MMS streaming, only the Host portion of the URL can be tested (`url.host=`). Recall the outline of the evaluation model for Windows Media transactions presented in "Understanding Transactions and Timing" on page 33.

Understanding Layers

Eight types of layers are allowed in any policy file. The layer type determines the kinds of transaction its rules will act upon. The token used in the header identifies the layer type.

- `<Admin>`—Used to define policy that controls access to the management console and the command line. Policy is not evaluated for serial console access or SNMP traffic, however.
- `<Tenant>`—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.
- `<Cache>`—Used to list policy rules that are evaluated during both cache and proxy transactions.
- `<Exception>`—Exception layers are evaluated when a proxy transaction is terminated by an exception.
- `<Forward>`—Forward layers are only evaluated when the current transaction requires an upstream connection. Forwarding policy is generally distinct and independent of other policies, and is often used as part of maintaining network topologies.
- `<Proxy>`—Used to list policy rules that are evaluated during a proxy transaction.
- `<DNS-Proxy>`—Used to define policy controlling `<DNS-Proxy>` transactions. Only `<DNS-Proxy>` transactions are evaluated against `<DNS-Proxy>` layers.
- `<SSL-Intercept>`—Used to list policy triggers and properties that define the conditions under which SSL connections are intercepted and decrypted or tunneled. This layer is evaluated by the SSL-Intercept transaction. Only the conditions, actions, and properties essential to make the tunnel or intercept decision are allowed in the SSL-Intercept layer.
- `<SSL>`—Used to store additional SSL triggers and properties that are unrelated to SSL interception. This layer, called the SSL Access layer in VPM, is evaluated by all cache and proxy transactions except the SSL-Intercept and `<DNS-Proxy>` transactions.

Important: Only a subset of the conditions, properties, and actions available in the policy language is permitted within each layer type; the remainder generate compile-time errors. Check the specific chapters in this manual to learn where the conditions, properties, and actions can be used.

<Admin> Layers

`<Admin>` layers hold policy that is executed by Administrator transactions. This policy is used to specify an authentication realm; to allow or deny administrative access based on the client's IP address, credentials, and type of administrator access requested (read or write); and to perform any additional logging for administrative access.

Important: When traffic is explicitly proxied, it arrives at the `<Admin>` layer with the client IP address set to the appliance's IP address; therefore, the `client.address=` condition is not useful for explicitly proxied traffic.

The syntax is:

```
<Admin ["comment"]> [admin_condition][admin_properties] ...
   admin_rules
```

where:

- The optional *"comment"*, separated from the layer type by space, is an identifier or quoted string that is displayed in policy traces to identify the purpose of the layer.
- The optional *admin_condition* is a list of triggers, all of which must evaluate to true before the layer content is evaluated. For more information on using conditions, see "Condition Reference" on page 56. See also the following Layer Guards section.
- The optional *admin_properties* is a 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 Chapter 4: "Property Reference" on page 259. See also the following Layer Guards section.
- *admin_rules* is a list of rules representing the decision to be made by this policy layer.

<Cache> Layers

<Cache> layers hold policy that 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=, user=, group=, 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).
- 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

The syntax is:

```
<Cache ["comment"]> [cache_condition][cache_properties] ...
   cache_rules
```

where:

- The optional *"comment"*, separated from the layer type by space, is an identifier or quoted string that is displayed in policy traces to identify the purpose of the layer.
- The optional *cache_condition* is a list of triggers, all of which must evaluate to true before the layer content is evaluated. For more information on using conditions, see "Condition Reference" on page 56. See also the following Layer Guards section.
- The optional *cache_properties* is a 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 Chapter 4: "Property Reference" on page 259. See also the following Layer Guards section.
- *cache_rules* is a list of rules representing the decision to be made by this policy layer.

<Exception> Layers

<Exception> layers are evaluated when a proxy transaction is terminated by an exception. This could be caused by a bad request (for example, the request URL names a non-existent server) or by setting the deny or exception() properties in policy. Policy in an exception layer can be used to control how access logging is performed for exceptions, such as authentication_failed. It can also be used to modify the HTTP response headers in the exception page that is sent to the client.

The syntax is:

```
<Exception ["comment"]> [exception_condition][exception_properties] ...
    exception_rules
```

where:

- The optional *"comment"*, separated from the layer type by space, is an identifier or quoted string that is displayed in policy traces to identify the purpose of the layer.
- The optional *exception_condition* is a list of triggers, all of which must evaluate to true before the layer content is evaluated. For more information on using conditions, see "Condition Reference" on page 56. See also the following Layer Guards section.
- The optional *exception_properties* is a 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 Chapter 4: "Property Reference" on page 259. See also the following Layer Guards section.
- *exception_rules* is a list of rules representing the decision to be made by this policy layer.

<Forward> Layers

<Forward> layers are evaluated when the current transaction requires an upstream connection (and only then: forward policy will not be evaluated for a cache hit). <Forward> layers use the server_url= tests rather than the url= tests so that they are guaranteed to honor any policy that rewrites the URL.

The syntax is:

```
<Forward ["comment"]> [forward_condition][forward_properties] ...
    forward_rules
```

where:

- The optional *"comment"*, separated from the layer type by space, is an identifier or quoted string that is displayed in policy traces to identify the purpose of the layer.
- The optional *forward_condition* is a list of triggers, all of which must evaluate to true before the layer content is evaluated. For more information on using conditions, see also the following Layer Guards section.
- The optional *forward_properties* is a 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 also the following Layer Guards section.
- *forward_rules* is a list of rules representing the decision to be made by this policy layer.

<Proxy> Layers

<Proxy> layers define policy for authenticating and authorizing users' requests for service over one of the configured proxy service ports. Proxy layer policy involves both client identity and content. Only proxy transactions are evaluated against <Proxy> layers.

Note: Policy for <DNS-Proxy> transactions is distinct from policy for other proxy services. See the following <DNS-Proxy> Layers section.

The syntax is:

```
<Proxy ["comment"]> [proxy_condition][proxy_properties] ...
    proxy_rules
```

where:

- The optional *"comment"*, separated from the layer type by space, is an identifier or quoted string that is displayed in policy traces to identify the purpose of the layer.
- The optional *proxy_condition* is a list of triggers, all of which must evaluate to true before the layer content is evaluated. For more information on using conditions, see also the following Layer Guards section.
- The optional *proxy_properties* is a 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. See also the following Layer Guards section.
- *proxy_rules* is a list of rules representing the decision to be made by this policy layer.

<DNS-Proxy> Layers

<DNS-Proxy> layers define policy controlling <DNS-Proxy> transactions. Only <DNS-Proxy> transactions are evaluated against <DNS-Proxy> layers.

The syntax is:

```
<DNS-Proxy ["comment"]> [dns_proxy_condition][dns_proxy_properties] ...
   dns_proxy_rules
```

where:

- The optional *"comment"*, separated from the layer type by space, is an identifier or quoted string that is displayed in policy traces to identify the purpose of the layer.
- The optional *dns_proxy_condition* is a list of triggers, all of which must evaluate to true before the layer content is evaluated. For more information on using conditions, see also the following Layer Guards section.
- The optional *dns_proxy_properties* is a 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 also the following Layer Guards section.
- *dns_proxy_rules* is a list of rules representing the decision to be made by this policy layer.

<SSL-Intercept> Layers

The <SSL-Intercept> layer lists the triggers and properties that define the interception conditions for SSL connections. This layer is evaluated by the SSL-Intercept transaction only.

The syntax is

```
<SSL-Intercept ["comment"]> [SSL-Intercept_condition][SSL-Intercept_properties]
...
   SSL-Intercept_rules
```

where:

- The optional *"comment"*, separated from the layer type by space, is an identifier or quoted string that is displayed in policy traces to identify the purpose of the layer.
- The optional *SSL-Intercept_condition* is a list of triggers, all of which must evaluate to true before the layer content is evaluated. For more information on using conditions, see also the following Layer Guards section.
- The optional *SSL-Intercept_properties* is a 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 also the following Layer Guards section.
- *SSL-Intercept_rules* is a list of rules representing the decision to be made by this policy layer.

<SSL> Layers

The <SSL> layer lists the triggers and properties that define the interception conditions for SSL connections. This layer is evaluated by all transactions except the SSL-Intercept transaction.

The syntax is

```
<SSL ["comment"]> [SSL_condition][SSL_properties] ...
    SSL_rules
```

where:

- The optional "comment", separated from the layer type by space, is an identifier or quoted string that is displayed in policy traces to identify the purpose of the layer.
- The optional *SSL_condition* is a list of triggers, all of which must evaluate to true before the layer content is evaluated. For more information on using conditions, see also the following Layer Guards section.
- The optional *SSL_properties* is a 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 also the following Layer Guards section.
- *SSL_rules* is a list of rules representing the decision to be made by this policy layer.

Layer Guards

Often, the same set of conditions or properties appears in every rule in a layer. For example, a specific user group for which a number of individual cases exist where some things are denied:

```
<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
```

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, as noted earlier, the following rule would result in a timing conflict error:

```
group=xyz authenticate(MyRealm)

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

The following layer would result in a similar error:

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

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. For example:

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

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=`, `url.domain=` 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.bluecoat.com/internal group=!bluecoat_employees deny
  url=www.bluecoat.com/proteus group=!bluecoat_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.bluecoat.com/internal group=!bluecoat_employees deny
  www.bluecoat.com/proteus group=!bluecoat_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.bluecoat.com/internal group=!bluecoat_employees deny
  url=www.bluecoat.com/proteus group=!bluecoat_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= 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. (The [url.domain] section replaces the [section used in previous versions of CPL.)

- 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

This section includes some guidelines for defining policies using CPL.

- Write an explicit layer header (`<Proxy>`, `<Cache>`, `<Admin>`, `<Forward>`, or `<Exception>`) before writing any rules or sections. The only constructs that should occur before the first layer header are the condition-related definitions and comments.
- Do not begin a policy file with a section, such as `[Rule]`. Ensure all sections occur within layers.
- Do not use `[Rule]` sections unnecessarily.
- 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 proper 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>
  ; Blue Coat 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 page 48) 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>
  ; Blue Coat 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 `response.icap_service( )` 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( )` actions conflict.
- Two `response.icap_service( )` 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 always the last policy file.) For example, suppose that service was limited to the corporate users identifiable by subnet. Placing a rule such as:

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

at the end of the Forward file ensures that no other policy overrides this restriction through accidental use of allow. 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` or an `exception()` property can be overridden by a subsequent allow in later rules, CPL provides `force_deny` and `force_exception()`. CPL does not offer `force_allow`, so while the error returned to the user may be reset by subsequent `force_deny` or `force_exception()` 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 KB article 000032877:

  http://bluecoat.force.com/knowledgebase/articles/Solution/000032877

Chapter 3: *Condition Reference*

This chapter will discuss what a condition is and the different types of a condition. The patterns of a condition will also be discussed and along with the restrictions.

Topics in this Chapter

This chapter includes information about the following topics:

- "Condition Syntax" on page 53
- "Pattern Types" on page 54
- "Unavailable Conditions" on page 55
- "Condition Reference" on page 56

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

- Policy rules.
- Section and layer headers, as guards; for example,

  ```
  [Rule] group=("bankabc\hr" || "cn=humanresources,ou=groups,o=westernnational")
  ```

- `define condition`, `define domain condition`, and `define url condition` 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`, 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 Appendix D: "Using Regular Expressions".

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 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=` and a condition definition) 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 494 and "restrict rdns" on page 495.

`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`.

Condition Reference

The remainder of this chapter lists the conditions and their accepted values. It also provides tips as to where each condition can be used and examples of how to use them.

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.

Syntax

```
admin.access=READ|WRITE
```

Layer and Transaction Notes

- Use in `<Admin>` layers.
- Applies to Administrative transactions.

Example

This example shows how administrative access can be controlled for two classes of users, Read_only_admins and Full_admins. In cases where a user is in both groups, that user inherits the larger set of permissions.

```
define condition Full_admins
  user=paul
  group=operations
end

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
```

Notes

- All administrative transactions require either READ or WRITE access, therefore a condition such as '`admin.access=(READ,WRITE)`' is always true, and can be deleted without changing the meaning of a CPL rule.
- This trigger replaces the use of `method=READ|WRITE` in the `<admin>` layer.

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

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

The configuration settings that are currently supported include:

- `enabled` (type name: boolean)
- `level` (type name: BC-Malware-Scanning-Scan-Level)
- `secure-connection` (type name: BC-Malware-Scanning-Secure-Connection)
- `failure-mode` (type name: BC-Malware-Scanning-Failure-Mode).

Syntax

```
ami.config.threat-protection.malware-scanning.config_setting ='(type_name
"setting_value")'

ami.config.threat-protection.malware-scanning.config_setting = yes|no
```

where:

- `config_setting` specifies the configuration setting name currently supported by this condition.
- `type_name` specifies the type of the setting value as defined in configuration. If the setting type is boolean, the second syntactical form is used. In that form, the `type_name` does not need to be specified.
- `setting_value` specifies value of the setting we want to match.

Layer and Transaction Notes

- Can be used in all layers.
- Applies to all transactions.

Example

The following example shows user policy that extends the threat-protection of low risk files. The second part of the example shows that the file is not virus-scanned when there is a private URL and the malware scanning level is set to **high-performance**.

```
<Proxy>
   ami.config.threat-protection.malware-scanning.level='(BC-Malware-Scanning-Scan-L
   evel "high-performance")'

   url.host.is_private=yes response.icap_service.secure_connection(no)
```

appliance.id=

Tests the serial number of the ProxySG 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[StringQualifiers]=String
```

where *StringQualifiers* equal [.exact | .prefix | .suffix | .substring | .regex][.case_sensitive]

Layer and Transaction Notes

- Use in `<Proxy>`, `<Cache>` layers
- Applies to all transactions.

Example

The following example shows policy that defines 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`* is a RADIUS or LDAP attribute. The *`name`* attribute's case-sensitivity depends on the type of authentication realm.
- RADIUS realm: The available attributes (see below) are always case-sensitive.
- LDAP realm: Case-sensitivity depends on the realm definition in configuration.
- *`value`*: An attribute value. For RADIUS, the values include:

Table 3.2: RADIUS values

RADIUS Attribute Name	CPL Gesture Name	Type (Possible Value)
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)

RADIUS Attribute Name	CPL Gesture Name	Type (Possible Value)
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)
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

RADIUS Attribute Name	CPL Gesture Name	Type (Possible Value)
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)
Acct-Terminate-Cause	attribute.Acct-Terminate-Cause	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)

RADIUS Attribute Name	CPL Gesture Name	Type (Possible Value)
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
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

RADIUS Attribute Name	CPL Gesture Name	Type (Possible Value)
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)
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)

RADIUS Attribute Name	CPL Gesture Name	Type (Possible Value)
Blue-Coat-Group	attribute.Blue-Coat-Group	String (Max length:20)

Layer and Transaction Notes

- Use in <Admin>, <Proxy>, <SSL-Intercept> and <Forward> layers.

  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.

- Applies to proxy and administrator transactions.
- This condition cannot be combined with the authenticate() or socks.authenticate() properties.

Example

```
; This example uses 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)
 ...

 ; This example uses the attribute property to determine permissions associated with
 ; RADIUS authentication.

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

 <Proxy>
 authenticate(RADIUSRealm)

; This rule would restrict non-authorized users.
 <Proxy>
 deny condition=!ProxyAllowed

; This rule would serve to override a previous denial and grant access to authorized
; users

 <Proxy>
 allow condition=ProxyAllowed
```

See Also

- Conditions: authenticated=, group=, has_attribute.*name*=, http.transparent_authentication=, realm=, user=, user.domain=, user.x509.issuer=, user.x509.serialNumber=, user.x509.subject=
- Properties: authenticate( ), authenticate.force( ), check_authorization( ), exception( ), socks.authenticate( ), socks.authenticate.force( )

authenticated=

True if authentication is requested for the current transaction and the requested realm's credentials have been verified; otherwise, false.

Syntax

```
authenticated=(yes|no)
```

Layer and Transaction Notes

- Use in `<Proxy>`, `<Forward>`, `<exception>`, `<admin>`, `<SSL-Intercept>` and `<SSL>` layers.
- Applies to proxy and administrator transactions.
- This condition cannot be combined with the `authenticate( )` property.

Example

```
; In this example, only users authenticated in any domain are granted access to a
; specific site.

<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)
 ; anyone else is unauthenticated


; This rule would restrict unauthorized users. Use this when overriding previously
; granted access.

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


; This rule would grant access and would be used to override a previous denial.
; It assumes a deny in a previous layer.

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

See Also

- Conditions: `attribute.`*`name`*`=`, `group=`, `has_attribute.`*`name`*`=`, `http.transparent_authentication=`, `realm=`, `user=`, `user.domain=`
- Properties: `authenticate( )`, `authenticate.force( )`, `check_authorization()`, `socks.authenticate( )`, `socks.authenticate.force( )`

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. This condition is only available if the current transaction is a Real Media or Windows Media transaction.

Syntax

```
bitrate={ [lower]..[upper]|exact_rate }
```

where:

- *lower*—Lower end of bandwidth range. Specify using an integer, in bits, kilobits (1000x), or megabits (1,000,000x), as follows: `integer | integerk | integerm`. If left blank, there is no lower limit on the test.
- *upper*—Upper end of bandwidth range. Specify using an integer, in bits, kilobits, or megabits, as follows: `integer | integerk | integerm`. If left blank, there is no upper limit on the test.
- *exact_rate*—Exact bandwidth to test. Specify using an integer, in bits, kilobits, or megabits, as follows: `integer | integerk | integerm`.

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

Layer and Transaction Notes

- Use in `<Cache>` and `<Proxy>` layers.
- Applies to streaming transactions.
- This condition can be used with the `max_bitrate( )` property.

Example

```
; Deny service for bit rates above 56k.

deny bitrate=!0..56k

; This example allows 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..
```

```
; In this form of the rule, we assume that the users are by default denied, and we
; are overriding 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=`, `streaming.client=`, `streaming.content=`
- Properties: `access_server( )`, `max_bitrate( )`, `streaming.transport( )`

category=

Tests the content categories of the requested URL as assigned by policy definitions or an installed content filter database.

Content categories can be assigned to URLs by policy (see "define category" on page 476), by a local database you maintain, or by a third-party database.

A URL that is not categorized is assigned the category `none`.

If a content filter provider is selected in configuration, but an error occurs in determining the category, the URL is assigned the category `unavailable` (in addition to any categories assigned directly by policy). This can be the result of either a missing database or license expiry. An additional category of `unlicensed` is assigned in the latter case.

A URL may have been assigned a list of categories. The `category=` condition is true if it matches any of the categories assigned to the URL.

You cannot use `category=` to test the category assigned by off-box content filtering services. These services have their own policy that must be managed separately.

Note: If `category=unlicensed` is true, `category=unavailable` is true.

Syntax

```
category={ none|unlicensed|unavailable|category_name1, category_name2, ...}
```

where *`category_name1, category_name2, ...`* represent category names defined by policy or the selected content filter provider. The list of currently valid category names is available both through the Management Console and CLI.

Layer and Transaction Notes

- Use in `<Proxy>`, `<Cache>`, `<Exception>`, `<SSL>`, and `<SSL-Intercept>` layers.
- This condition can be combined with the `authenticate( )` property, except when a Microsoft Media Streaming (MMS) over HTTP transaction is being evaluated.
- Applies to proxy transactions.

Example

```
; This example denies requests for games or sports related content.

<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=`
- Properties: `exception( )`

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 ProxySG 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_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 subnet definition block that binds a number of IP addresses or subnets

Layer and Transaction Notes

- Can be used in all layers.
- Unavailable if the transaction is not associated with a client.

Example

```
; Blacklisted workstation.
client.address=10.25.198.0 deny


; This example uses 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.protocol=`, `proxy.address=`, `proxy.card=`, `proxy.port=`
- Definitions: `define subnet`
- Information on wildcards: http://bluecoat.force.com/knowledgebase/articles/Solution/000010037
- Information on IP address ranges: http://bluecoat.force.com/knowledgebase/articles/Solution/000010950

client.address.country=

This condition 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.

Syntax

```
client.address.country=<"country_name">
```

where ***<"country_name">*** is the 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

- Can be used in all layers.
- Unavailable if the transaction is not associated with a client.

Examples

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

; 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

- client.address=
- client.effective_address=
- client.effective_address.country=

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([lower]..[upper]|exact)

where:

- *[lower]* is optionally the fewest number of logins that will match this condition.
- *[upper]* is optionally the most number of logins that will match this condition.
- *exact* is optionally the exact number of logins that will match this condition.

Layer and Transaction Notes

- Use in <Proxy>, <Forward>, <Exception>, <Admin>, <SSL-Intercept> and <SSL> layers.
- Applies to: All transactions

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.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. This condition is NULL for transactions that do not involve an SSL connection to the client.

Syntax

```
client.certificate.common_name[.exact][.case_sensitive]=string
client.certificate.common_name.length=value
client.certificate.common_name.prefix[.case_sensitive]=string
client.certificate.common_name.substring[.case_sensitive]=string
client.certificate.common_name.suffix[.case_sensitive]=string
client.certificate.common_name.regex[.case_sensitive]=regular_expression
```

Note: 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

- Valid layers: `<SSL>`
- Applies to: HTTPS forward and reverse proxy transactions

Example

`Client.certificate.common_name` is mainly used 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 ProxySG 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

- Use in `<SSL-Intercept>` layer.
- Applies to: SSL Intercept transactions

Example(s)

This condition is used to avoid intercepting SSL proxy traffic when a server requests a client certificate to authenticate the client. The reason is that client certificates are not supported in this configuration. When intercepting such traffic, the appliance generates an exception page. The policy below enables 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 NULL for transactions that do not involve an SSL connection to the client.

Syntax

```
client.certificate.subject[.exact][.case_sensitive]=string
client.certificate.subject.length=value
client.certificate.subject.prefix[.case_sensitive]=string
client.certificate.subject.substring[.case_sensitive]=string
client.certificate.subject.suffix[.case_sensitive]=string
client.certificate.subject.regex[.case_sensitive]=regular_expression
```

Note: 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

- Valid layers: `<SSL>`
- Applies to: HTTPS forward and reverse proxy transactions

Example

`Client.certificate.subject` is mainly used 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>]=criterion
```

Where:

- `attribute_name` is a name defined in a subject_directory_attribute definition.
- `modifier` is optional and consists of one of:
  - `exists`
  - `count` – evaluates to the number of times the attribute appears in the certificate
  - `string_condition_modifier` – substring, exact, prefix, suffix, regex, case_sensitive
- `criterion` is a type-appropriate test expression. For string attributes, this is a string or regex expression.

Note: The pattern expression supports substitutions in strings. You can specify a substitution expression with the `.exact`, `.substring`, `.prefix`, and `.suffix` string modifiers where they are available.

- `attribute_name` is the name given to specific client certificate attribute, in a policy definition. Attributes are defined in CPL using a Subject Directory Attribute definition with the following syntax:

  ```
  define subject_directory_attribute
      subject_directory_definition_list
  end
  ```

  Where `subject_directory_attribute_definition_list` is a list of subject directory attribute definitions, one per line.

  - Multiple definitions are allowed, provided that the attribute names defined are globally distinct.
  - Each certificate attribute definition has the form:

    ```
    type:<type><attribute_name> <OID>
    ```

  - The string: reference is optional in this release and analogous with `type:`.

Example

```
define subject_directory_attribute
    employeeType 2.16.840.1.113730.3.1.4
    country 2.16.840.1.101.2.1.5.61
end
```

Note: The attribute name cannot contain a period, and cannot be one of the condition modifier tokens (substring, exact, prefix, suffix, regex, case_sensitive, exists).

Layer and Transaction Notes

- Applies to `<SSL>` and `<Proxy>` layers

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
```

Note: The preceding policy must exist on a single line.

client.connection.dscp=

Test the client-side inbound DSCP value.

Syntax

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

where *dscp_value* is `0..63 | af11 | af12 | af13 | af21 | af22 | af23 | af31 | af32 | af33 | af41 | af42 | af43 | best-effort | cs1 | cs2 | cs3 | cs4 | cs5 | cs6 | cs7 | ef`

Layer and Transaction Notes

- Valid in `<Proxy>`, `<DNS-Proxy>`, `<Forward>` layers.
- Applies to all transactions.

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`*

where *`cipher-suite`* is one of the following:

- `none`
- one or more cipher suites that the appliance supports; refer to the "Managing X.509 Certificates" chapter in the *SGOSAdministration Guide* for information on the supported cipher suites. Refer to the following **Example** to see how to specify multiple cipher suites.

Layer and Transaction Notes

- Use in `<SSL>` layer.
- Applies to proxy transactions.

Example

This example implements the following policies:

1. DENY clients that are not using one of the exportable cipher suites.
2. Access log clients that are not using secure connections in `unsecure_log1`.

```
; 1
<SSL>
 ALLOW client.connection.negotiated_cipher= \
   (EXP-RC4-MD5 || EXP-RC2-CBC-MD5 || EXP-DES-CBC-SHA)
DENY
 ; 2
<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=(none||low||medium||high)
```

Layer and Transaction Notes

- Use in `<SSL>` layer.
- Applies to proxy transactions.

Example

This example implements the following policies:

1. DENY clients that do not have at least a medium cipher strength.
2. ALLOW clients using FTP irrespective of their cipher strength since FTP clients do not have a means to encrypt the traffic.

```
<SSL>
 ; 1
 ALLOW client.connection.negotiated_cipher.strength=(medium||high)
 ; 2
 ALLOW url.scheme=ftp
 DENY
```

Notes

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.

client.connection.negotiated_ssl_version=

Test the SSL version negotiated with a securely connected client.

Syntax

```
client.connection.negotiated_ssl_version=SSLV2|SSLV3|TLSV1|TLSV1.1|TLSV1.2
```

Layer and Transaction Notes

- Use in `<SSL>` and `<Proxy>` layers.
- Applies to proxy transactions.

Example

```
<SSL>
  client.connection.negotiated_ssl_version=SSLV3
```

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=`).

Syntax

```
client.effective_address=
                    ip_address|ip_address_range|ip_address_wildcards|subnet_label
```

where:

- *`ip_address`*—Effective 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 subnet definition block that binds a number of IP addresses or subnets

Layer and Transaction Notes

- Available in all layers.
- Unavailable if the transaction is not associated with a client.

Example

```
; Only allow traffic from 192.0.2.0
<proxy>
     client.effective_address=192.0.2.0 allow
     deny
```

See Also

- `client.address`
- `client.effective_address()`
- `client.effective_address.country=`
- Information on wildcards: http://bluecoat.force.com/knowledgebase/articles/Solution/000010037
- Information on IP address ranges: http://bluecoat.force.com/knowledgebase/articles/Solution/000010950

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.

Syntax

```
client.effective_address.country=<country name>
```

where *<country name>* is the 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

- Available in all layers.
- Unavailable if the transaction is not associated with a client.

Example

```
; 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
; Allow all if geolocation is disabled or database is not downloaded.
<proxy>
     client.address.country=Unavailable allow
```

See Also

- `client.effective_address=`
- `client.effective_address()`

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

Available in all layers.

Example

```
; Only allow traffic if effective client IP address was not overridden.
<proxy>
     client.effective_address.is_overridden=yes
     deny
```

See Also

- `client.address=`
- `client.effective_address()`
- `client.effective_address.country=`

client.host=

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

Syntax

```
client.host=hostname
client.host=domain-suffix
client.host.exact=string
client.host.length=value
client.host.prefix=string
client.host.substring=string
client.host.suffix=string
client.host.regex=regular_expression
```

Note: 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

- Use in `<Proxy>`, `<Forward>`, `<Exception>`, `<Admin>`, `<SSL>`, and `<SSL-Intercept>` layers.
- Applies to all proxy transactions, excluding DNS-proxy transactions.

Example

This example implements the following policies:

1. DENY all users that do not have an RDNS name that ends with `bluecoat.com`
2. DENY all users that have `test` in their RDNS name
3. DENY all users that have an RDNS name that ends with `example.bluecoat.com`. This is meant to include `bexample.bluecoat.com` and `b.example.bluecoat.com`.
4. DENY all users that have numbers in their RDNS name.
5. DENY all users that have an RDNS name that begins with `fnord`.

```
<Proxy>
 ALLOW
```

```
<Proxy>
 ; 1
 DENY client.host=!".bluecoat.com"
 ; 2
 DENY client.host.substring="test"
 ; 3
 DENY client.host.suffix="example.bluecoat.com"
 ; 4
 DENY client.host.regex="[0-9]*"
 ; 5
 DENY client.host.prefix="fnord."
```

client.host.has_name=

Test the status of the RDNS performed to determine client.host.

Syntax

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

Layer and Transaction Notes

- Use in <Proxy>, <Forward>, <Exception>, <Admin>, <SSL>, and <SSL-Intercept> layers.
- Applies to All proxy transactions, excluding DNS-proxy transactions

Example

This example implements the following policies:

1. DENY all users from subnetA if their RDNS name could not be resolved
2. 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>
 ; 1
 ALLOW client.address=subnetA client.host.has_name=yes

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

client.protocol=

Test the protocol used between the client and the ProxySG appliance.

Syntax

```
client.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
```

Notes

- `tcp` specifies a tunneled transaction
- `client.protocol=dns` is valid in the `<dns-proxy>` layer only.
- `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.
- `ms-turn`, `sip`, and `sips` will be returned either as a result of `force_protocol()` or as a result of successful detection of one of these protocols for which `detect_protocol()` was enabled. The protocol `sips` can also be returned as a result of the `ssl.forward_proxy(sips)` policy. The `tunneled=` trigger will report "no" for these types of traffic, even though the traffic is effectively still tunneled by the TCP tunnel or STunnel proxies. (Added in 6.5.9.10.)

Layer and Transaction Notes

- Use in `<Exception>`, `<Forward>`, `<proxy>`, `<SSL>`, and `<SSL-intercept>` layers.
- Applies to all transactions.
- Tests false if the transaction is not associated with a client.

See Also

- Conditions: `client.address=`, `proxy.address=`, `proxy.card=`, `proxy.port=`

condition=

Tests if the specified defined condition is true.

Syntax

`condition=`*`condition_label`*

where *`condition_label`* is the label of a custom condition as defined in a `define condition`, `define url.domain condition`, or `define url condition` definition block.

Layer and Transaction Notes

- Use in all layers.
- The defined conditions that are referenced may have usage restrictions, as they must be evaluated in the layer from which they are referenced.

Example

```
; 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

; 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 of a define url condition.
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
```

```
; Example of a define domain-suffix (or domain) condition
define url.domain condition test
 com ; Matches all domains ending in .com
end

 <Proxy>
 condition=test deny
```

See Also

- Definitions: `define condition`, `define url.domain condition`, `define url condition`
- Properties: `action.`*`action_label`*`( )`

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 ProxySG 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

- Use in `<Proxy>`, `<Cache>`, `<Exception>`, and `<SSL>` layers.
- Applies to HTTP transactions.

See Also

- Conditions: `admin.access=`

content_management=

Tests if the current request is a content management transaction.

Syntax

```
content_management=yes|no
```

Layer and Transaction Notes

- Use in `<Cache>`, `<Forward>`, and `<SSL>` layers.
- Applies to all transactions.

See Also

- Conditions: `category=`, `ftp.method=`, `http.method=`, `server_url=`
- Properties: `http.request.version( )`, `http.response.version( )`

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

- Supported in <Exception> and <Proxy> layers.

See Also

- Conditions: virus_detected

date[.utc]=

Tests true if the current time is within the `startdate..enddate` range, inclusive. The comparison is made against local time unless the `.utc` qualifier is specified.

syntax

```
date[.utc]=YYYYMMDD..YYYYMMDD
date[.utc]=MMDD..MMDD
```

Layer and Transaction Notes

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

See Also

- Conditions: `day=`, `hour=`, `minute=`, `month=`, `time=`, `weekday=`, `year=`

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`*—An 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`*—An 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`*—An 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, the following shorthand expression is available. While `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

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

Example

```
; Test for New Year's Day (January 1).
day=1 month=1

; This policy allows access to a special event site only during the days of
; the event.
; This form of the rule restricts 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=2003 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=2003 month=7 day=23..25 ; During the event
```

See Also

- Conditions: `date[.utc]=`, `hour=`, `minute=`, `month=`, `time=`, `weekday=`, `year=`

dns.client_transport=

Test the transport protocol of a proxied DNS query

Syntax

```
dns.client_transport=tcp|udp
```

Layer and Transaction Notes

- Use in `<DNS-Proxy>` layers.
- Applies to DNS proxy transactions

Example

This example implements the following policy:

1. Refuse all DNS queries that use the TCP protocol
2. Unless the query is coming from the subnet `10.9.8.0/24`

```
; 1,2
<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
```

Layer and Transaction Notes

- Use in `<DNS-Proxy>` layers.
- Applies to DNS proxy transactions.

Example

This example implements the following policies:

1. Refuse all DNS PTR queries for addresses in the `10.10.0.0/16` subnet
2. Respond with `host1.example.com` to DNS PTR queries for `10.9.8.1`
3. Respond with `host2.example.com` to DNS PTR queries for `10.9.8.2`

```
<DNS-Proxy>
 ; 1
 dns.request.address=10.10.0.0/16 dns.respond(refused)
 ; 2
 dns.request.address=10.9.8.1 dns.respond.ptr("host1.example.com")
 ; 3
 dns.request.address=10.9.8.2 dns.respond.ptr("host2.example.com")
```

See Also

- Information on wildcards:
  http://bluecoat.force.com/knowledgebase/articles/Solution/000010037
- Information on IP address ranges:
  http://bluecoat.force.com/knowledgebase/articles/Solution/000010950

dns.request.category=

Test the URL category of either the DNS queried hostname or IP address

Syntax

```
dns.request.category=none|unlicensed|unavailable|category_name1,
category_name2, ...
```

Layer and Transaction Notes

- Use in `<DNS-Proxy>` layers.
- Applies to DNS proxy transactions.

Example

This example implements the following policies:

1. Refuse all DNS type "A" queries from the Engineering subnet for category `HR_intranet_servers`.
2. Refuse all DNS type "A" 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
 ; 1
 client.address=Engineering \
 dns.request.category=HR_intranet_servers  dns.respond(refused)
 ; 2
 client.address=HR \
 dns.request.category=Engineering_intranet_servers  dns.respond(refused)
```

Notes

- Additional RDNS/DNS lookups are not performed to categorize the DNS query.

dns.request.class=

Test the QCLASS of the DNS query

Syntax

```
dns.request.class=any|ch|hs|in|none|numeric range from 0 to 65535
```

Layer and Transaction Notes

- Use in `<DNS-Proxy>` layers.
- Applies to DNS-proxy transactions.

Example

This example implements the following policy:

- 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=hostname
dns.request.name=domain-suffix
dns.request.name.exact=string
dns.request.name.length=value
dns.request.name.prefix=string
dns.request.name.substring=string
dns.request.name.suffix=string
dns.request.name.regex=regular_expression
```

Layer and Transaction Notes

- Use in `<DNS-Proxy>` layers.
- Applies to DNS proxy transactions.

Example

This example implements the following policies:

1. Refuse all queries for hostnames that end with `example.com`.
2. Permit queries for `host1.example.com`.

```
; 1
<DNS-Proxy>
 dns.request.name=.example.com dns.respond(refused)
; 2
<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|numeric range from 0 to 15
```

Layer and Transaction Notes

- Use in <DNS-Proxy> layers.
- Applies to DNS-proxy transactions

Example

This example implements the following policy:

- Refuse all DNS traffic that does not use the OPCODE: QUERY.

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

dns.request.type=

Test the `QTYPE` of the DNS query.

Syntax

```
dns.request.type=dns-qtype|numeric range from 0 to 65535
```

where *dns-qtype* is one of:

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

Layer and Transaction Notes

- Use in `<DNS-Proxy>` layers.
- Applies to DNS proxy transactions.

Example

```
<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
```

Layer and Transaction Notes

- Use in `<DNS-Proxy>` layers.
- Applies to DNS proxy transactions.

Example

This example implements the following policy:

- 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

- Information on wildcards:
  http://bluecoat.force.com/knowledgebase/articles/Solution/000010037
- Information on IP address ranges:
  http://bluecoat.force.com/knowledgebase/articles/Solution/000010950

dns.response.aaaa=

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

Syntax

```
dns.response.aaaa=ip-address[/prefix-len]
```

Layer and Transaction Notes

- Use in <DNS-Proxy> layers.
- Applies to DNS-proxy transactions.

Example

This example implements the following policy:

- This policy example specifies that 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)
```

Note: The DNS Proxy caches IPv6 AAAA records.

dns.response.cname=

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

Syntax

```
dns.response.cname=hostname
dns.response.cname=domain-suffix
dns.response.cname.exact=string
dns.response.cname.length=value
dns.response.cname.prefix=string
dns.response.cname.substring=string
dns.response.cname.suffix=string
dns.response.cname.regex=regular_expression
```

Layer and Transaction Notes

- Use in <DNS-Proxy> layers.
- Applies to DNS proxy transactions.

Example

This example implements the following policies:

1. Refuse all DNS queries that have .example.com in any of the CNAME RRs.
2. Permit host1.example.com.

```
; 1
<DNS-Proxy>
 dns.response.cname=.example.com dns.respond(refused)
; 2
<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=noerror|formerr|servfail|nxdomain|notimp|refused|yxdomain|yxr
rset|nxrrset|notauth|notzone|numeric range from 0 to 15
```

Layer and Transaction Notes

- Use in `<DNS-Proxy>` layers.
- Applies to DNS proxy transactions.

Example

This example implements the following policy:

- We have a DNS server that routinely responds with `yxdomain`, but some of our client machines do not handle that response code gracefully. Converting the `yxdomain` response to `nxdomain` seems to fix the problem.

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

dns.response.nodata=

Test whether the DNS response had no RRs

Syntax

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

Layer and Transaction Notes

- Use in `<DNS-Proxy>` layers.
- Applies to DNS-proxy transactions.

Example

This example implements the following policy:

- We have a DNS server that routinely sends back empty responses. Some of our clients fail to handle this gracefully. The server in question needs to be patched, but it is in another department, so, for the time being, convert empty response to `nxdomain`, which our clients can handle okay.

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

dns.response.ptr=

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

Syntax

```
dns.response.ptr=hostname
dns.response.ptr=domain-suffix
dns.response.ptr.exact=string
dns.response.ptr.length=value
dns.response.ptr.prefix=string
dns.response.ptr.substring=string
dns.response.ptr.suffix=string
dns.response.ptr.regex=regular_expression
```

Layer and Transaction Notes

- Use in `<DNS-Proxy>` layers.
- Applies to DNS proxy transactions.

Example

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

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* is either the name of a built-in exception of the form:

```
exception_id
```

or the 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( )` property, `exception.id=` can also test for exceptions returned by other CPL gestures, such as `policy_denied`, returned by the `deny( )` property and `policy_redirect` returned by the `redirect( )` action.

Layer and Transaction Notes

- Use in `<Exception>` layers.
- Applies to proxy transactions.

Example

This example illustrates how 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: `deny( ),deny.unauthorized( ),exception( )`
- Actions: a`uthenticate( ),authenticate.force( ),redirect( )`

ftp.method=

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

- `ftp.method=` evaluates to true if the request method matches any of the methods specified.
- `ftp.method=` evaluates to NULL if the request is not an FTP protocol request.

Syntax

```
ftp.method=ABOR|ACCT|ALLO|APPE|CDUP|CWD|DELE|EPRT|EPSV|HELP|LIST|MDTM|MKD|MODE|N
LST|NOOP|PASS|PASV|PORT|PWD|REST|RETR|RMD|RNFR|RNTO|SITE|SIZE|SMNT
|STOR|STOU|STRU|SYST|TYPE|USER|XCUP|XCWD|XMKD|XPWD|XRMD|OPEN
```

where:

- `ftp.method=` evaluates to true if the request method matches any of the methods specified.
- It evaluates to NULL if the request is not an FTP protocol request.

Layer and Transaction Notes

- Use in `<Proxy>`, `<Cache>`, and `<Exception>` layers.
- Applies to FTP transactions.

See Also

- Conditions: `category=`, `content_management=`, `http.method=`, `server_url=`, `socks.method=`

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=` 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( )` 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=` test. This reduces the number of extraneous queries to authentication services for group information that does not pertain to that realm.

Important: When using this condition:
- Use domain-qualified group names such as `domain\groupname`. Numerous non-domain-qualified group names in policy might cause delays in policy compilation.
- Make sure that group caching, which caches the group-name-to-SID mapping for each of the groups-of-interest, is enabled. You can enable the feature so that the delay occurs only during initial policy compilation when the appliance boots.
Use the CLI command `#(config security windows-domains)` **`group-cache enable`**.

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.
  - NTLM realm: Group names are of the form `Domain\groupname`, where `Domain` may be optional, depending on whether BCAAA or CAASNT is installed on the NT domain controller for the domain. Names are case-insensitive.
  - Local Password realm: Group names are up to 32 characters long, and underscores (_) and alphanumerics are allowed. Names are case-sensitive.
  - RADIUS realm: RADIUS does not support groups. Instead, *groups* in RADIUS environments are defined by assigning users a `ServiceType` attribute.
  - LDAP realm: Group definitions depend on the type of LDAP directory and LDAP schema. Generally, LDAP distinguished names are used in the following form: `cn=`*`proxyusers`*, `ou=`*`groups`*, `o=`*`companyname`*. Case-sensitivity depends on the realm definition configuration.
  - Certificate realm: Certificate realms provide authentication, but do not themselves provide authorization; instead they delegate group membership decisions to their configured authorization realm, which is either a Local Password realm or an LDAP realm. Group definitions should conform to the appropriate standards for the delegated authorization realm. Although the group used in policy is then a group from the delegated realm, to achieve performance benefits, the `group=` test should be preceded with a realm test for the certificate realm, not the delegated authorization realm.

- Sequence realm: A sequence realm is a configured list of subordinate realms to which the user credentials are offered, in the order listed. The user is considered authenticated when the offered credentials are valid in one of the realms in the sequence. Authorization of the user is done with respect to the subordinate realm in which authentication occurred. Group names might be valid names in any of the realms in the sequence, but for the `group=` test to evaluate to true, the group must be valid in the realm in which the user is actually authenticated. If the group is valid in all realms in the sequence, then the `group=` test must be preceded by a `realm=` test of the Sequence realm; otherwise, it should be preceded by a `realm=` test of the appropriate subordinate realm.

Layer and Transaction Notes

- Use in `<Proxy>`, `<Forward>`, `<exception>`, `<admin>`, `<SSL-Intercept>` and `<SSL>` layers.

  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.

- Applies to proxy and administrator transactions.

Example

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

realm=corp group=all_staff
```

```
; This example shows sample group tests for each type of realm. It does
; this by creating a condition in CPL that treats a group of administrators in
; each realm as equivalent, granting them permission to administer the Security
; Appliance. Recall that the <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=`, `has_attribute.`*`name`*`=`, `http.transparent_authentication=`, `realm=`, `user=`, `user.domain=`, `user.x509.issuer=`, `user.x509.serialNumber=`, `user.x509.subject=`

- Properties: `authenticate( ),authenticate.force( ),check_authorization( ), socks.authenticate( ),socks.authenticate.force( )`

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` property is set to `no`).

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

Important: 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`* is an LDAP or RADIUS attribute. Case-sensitivity for the attribute name depends on the realm definition in configuration.

The following RADIUS attribute names can be specified:

```
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

- Use in `<Admin>` and `<Proxy>` layers. RADIUS attributes can also be used in `<Forward>` and `<Exception>` layers.
- LDAP applies to proxy and administrate transactions. RADIUS attributes apply to all transactions.

Examples

This example allows users to access the proxy if they have the RADIUS attribute Callback-Number. The attribute could have any value, even null.

```
<Proxy>
 authenticate(RADIUSRealm)
<Proxy>
 allow has_attribute.Callback-Number=yes
```

The following policy allows users to access the proxy if they have the LDAP attribute ProxyUser. The attribute could have any value, even null. Generally this kind of policy would be established in the first proxy layer, and would set up either the blacklist or whitelist model, as desired.

```
<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`*`=`, `authenticated=`, `group=`, `http.transparent_authentication=`, `realm=`, `user=`, `user.domain=`
- Properties: `authenticate( )`, `authenticate.force( )`, `check_authorization( )`

has_client=

The `has_client=` condition is used to 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

- Use in `<forward>`, `<ssl>`, and `<cache>` layers.
- Applies to all transactions.

See Also

- Conditions: `client.address=`, `client.protocol=`, `proxy.address=`, `proxy.card=` `proxy.port=`, `streaming.client=`

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, and will be added automatically. In all other cases the complete health check name, including its prefix, must be entered.

Layer and Transaction Notes

- Valid layers: Forward, SSL
- Applies to: All transactions

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_name*`=`

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.

Note: To test against an inverted range, such as a range that crosses from one day into the next, the following shorthand expression is available. While `hour=(..06|19..)` specifies midnight to 6:59 a.m. and 7:00 p.m. to midnight, the policy language also recognizes `hour=19..06` as equivalent.

Layer and Transaction Notes

- Use in all layers.
- Using time-related conditions to control caching behavior in a `<Cache>` layer may cause thrashing of the cached objects.
- Applies to all transactions.

Example

```
; Tests for 3:00 a.m. to 1:59 p.m. UTC.
hour.utc=03..13
```

```
; The following example restricts 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.
```

```
<Proxy>
allow server_url.domain=xyz.com ; internal site always available
allow weekday=6..7       ; unrestricted weekends
allow hour=17..8; Inverted range for outside business hours
```

See Also

- Conditions: `date[.utc]=`, `day=`, `minute=`, `month=`, `time=`, `weekday=`, `year=`

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

- Use in `<Proxy>`, `<Forward>`, `<Exception>`, and `<SSL>` layers.
- Applies to proxy transactions

Example

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

http.connect.User-Agent=

(Introduced in SGOS 6.5.7.1) Tests which user agent is used to initiate an explicit proxy HTTP CONNECT request.

Syntax

```
http.connect.User-Agent[.case_sensitive]=regular_expression
http.connect.User-Agent.exact[.case_sensitive]=string
http.connect.User-Agent.prefix[.case_sensitive]=string
http.connect.User-Agent.substring[.case_sensitive]=string
http.connect.User-Agent.suffix[.case_sensitive]=string
http.connect.User-Agent.regex[.case_sensitive]=regular_expression
```

where:

- *regular_expression* is a regular expression. For more information, see Appendix D: "Using Regular Expressions" on page 515.
- *string* is any printable ASCII sequence, quote delimited

Layer and Transaction Notes

- Use in <Proxy> and <SSL-Intercept> layers.
- Applies to explicitly proxied HTTPS transactions.
- 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.

Example

```
; Inspect User-Agent header for specific browser and intercept SSL if matched
<ssl-intercept>
  http.connect.User-Agent.exact="<my_browser>" ssl.forward_proxy(yes)
```

http.method=

Tests HTTP request methods against any of a common set of HTTP methods. A CPL parse error is given if an unrecognized method is specified.

Syntax

```
http.method=GET|CONNECT|DELETE|HEAD|POST|PUT|TRACE|OPTIONS|TUNNEL|LINK|UNLINK
|PATCH|PROPFIND|PROPPATCH|MKCOL|COPY|MOVE|LOCK|UNLOCK|MKDIR|INDEX|RMDIR|COPY|
MOVE
```

where:

- `http.method=` evaluates to true if the request method matches any of the methods specified.
- `http.method=` evaluates to NULL if the request is not an HTTP protocol request.

Layer and Transaction Notes

- Use in `<Proxy>`, `<Cache>`, and `<Exception>` layers.
- Applies to HTTP and SSL-terminated HTTPS transactions.

See Also

- Conditions: `admin.access=`, `ftp.method=`, `socks.method=`
- Properties: `http.request.version( )`, `http.response.version( )`

http.method.custom=

Test the HTTP protocol method versus custom values.

Syntax

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

Layer and Transaction Notes

- Use in `<Proxy>`, `<Cache>`, and `<Exception>` layers.
- Applies to HTTP and SSL-terminated HTTPS transactions.

Example

This example implements the following policies:

1. Of the well-known HTTP methods only permit “GET” and “POST.”
2. Allow the custom HTTP method “MYMETHOD1” that one of our backend servers is using.
3. DENY all other HTTP methods.

```
<Proxy>
 ; 1
 ALLOW http.method=(GET||POST)
 ; 2
 ALLOW http.method.custom=MYMETHOD1
 ; 3
 DENY
```

http.method.regex=

Test the HTTP method using a regular expression.

Syntax

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

Layer and Transaction Notes

- Use in `<Proxy>`, `<Cache>`, and `<Exception>` layers.
- Applies to HTTP and SSL-terminated HTTPS transactions.

Example

This example implements the following policy:

- DENY any HTTP method that contains a decimal number.

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

http.request.apparent_data_type=

Used to test HTTP POSTs that include data against policy based on the apparent type of that content.

This condition allows you to control the types of files submitted during an HTTP POST, as identified by apparent data type. Unlike MIME or file extension-based policies, when policy examines the Apparent Data Type string in an HTTP, it looks at the first few bytes of a POST in order to identify the type of data being requested.

Syntax

```
http.request.apparent_data_type=(BMP|BZ2|CAB|EXE|FLASH|GIF|GZIP|HTML|ICC|JPG|MS
DOC|MRAR|MZIP|PDF|PNG|RAR|RTF|TAR|TIF|TTF|TXT|XML|ZIP)
```

- The following table details the data types used in this condition:

Label	Description	Common Extensions
BMP	BMP image	.bmp
BZ2	BZip2 archive	.bz2, .tbz2
CAB	MS Cab archive	.cab
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
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
TTF	True-Type font	.ttf
TXT	plain text	.txt
XML	XML file	.xml
ZIP	ZIP archive	.zip

Layer and Transaction Notes

- Use in `<Proxy>` layers.
- Recommended for Reverse Proxy deployments, where files are uploaded through the appliance to a back-end server.
- Applies to all HTTP and decrypted HTTPS POST requests.

Example

A reverse proxy administrator for a local community website would like to allow users to upload images to the website, but only in JPG format.

```
<Proxy>
  ALLOW http.request.apparent_data_type=JPG
  DENY
```

See Also:

```
http.response.apparent_data_type=<type>
request.icap.apparent_data_type=<type>
response.icap.apparent_data_type=<type>
http.request.apparent_data_type.allow(<type>,…)
http.request.apparent_data_type.deny(<type>, …)
```

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 the `http.request.body.max_size` condition, `http.request.body.size` does not enforce a maximum.

Syntax

```
http.request.body.size=N
```

- In the above example, *N* equals the number of bytes.
- Ranges are supported, separated by `..` (a value of `1000..2000` will trigger the rule for HTTP requests with content equal to a size 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

- Use in `<Exception>`, `<Cache>` and `<Proxy>` layers.
- Applies to all HTTP requests with body content.
- Evaluated after the Origin Content Server (OCS) responds. To deny similar requests before the OCS responds, use `http.request.body.max_size()` instead.

Example

When the body content size of an HTTP request transaction exceeds 10 MB, output a new entry to the custom log type, 'large_request':

```
<Proxy>
http.request.body.size=10485760.. access_log[large_request](yes)
```

http.request.body.max_size_exceeded=

Used in conjunction with the property, `http.request.body.max_size()`, 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

- Use in `<Exception>` layers.
- Applies to requests that match rules restricting the size of an HTTP request body using the CPL property `http.request.body.max_size()`.

Example

When the body of an HTTP request exceeds 5 MB, it will be denied and reported in the custom log type, 'large_request':

```
<Proxy>
http.request.body.max_size(5242880)

<Exception>
http.request.body.max_size_exceeded=yes access_log[large_request](yes)
```

http.request.data=

Allows you to inspect the contents of the body of an HTTP request. Content filters can use up to 8192 bytes from the HTTP request body to match.

Note: The content filter could also use the URL of an application to determine the application name by using the `url.application.name=` condition.

Syntax

`http.request.data.N[StringQualifiers] = String`

where:

- `N` equals the number of HTTP request body bytes to inspect, from 1..8192
- *`StringQualifiers`* equal [.exact | .prefix | .suffix | .substring | .regex][.case_sensitive]

Layer and Transaction Notes

- Use in `<Cache>` and `<Proxy>` layers.
- Applies to all HTTP transactions.

See Also

- `url.application.name=`

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.8192.substring="sql" DENY
```

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 `http.request.detection.result.application_protection_set=` to perform an action such as additional logging to manually identify the content of the request.

Syntax

```
http.request.detection.result.application_protection_set=[block|monitor]
```

Layer and Transaction Notes

- Use in `<Proxy>` layers.
- Applies to all transactions that have already been processed by a WAF application protection scan rule.

See Also

- `define application_protection_set`
- `http.request.detect.result.validation=`
- `http.request.log_details[header,body](yes|no)`
-

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)
```

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 `http.request.detection.result.validation=` to perform an action such as additional logging to manually identify the content of the request.

Syntax

```
http.request.detection.result.validation=[block|monitor]
```

Layer and Transaction Notes

- Use in `<Proxy>` layers.
- Applies to all transactions that have already been processed by a WAF validation rule:
  - `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)`

See Also

- `http.request.detect.result.application_protection_set=`
- `http.request.log_details[header,body](yes|no)`
-

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)
```

http.request[ ].*modifier*=

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()` 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[<attribute>,…].<modifier>[.case_sensitive]=pattern
```

where:

- *modifier*—Specifies a modifier pattern. The following options are available:
  - exact
  - substring
  - suffix
  - regex
  - count
  - prefix

  If you do not specify a modifier, the condition tests for an exact pattern match.
- `case_sensitive`—Specifies the test as case-sensitive. By default, the test is case-insensitive.
- `attribute`—A comma-separated list of the predefined content sources:

Table 3–1 Supported HTTP Attributes

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).

Table 3–1 Supported HTTP Attributes

Name	Value
`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).

Layer and Transaction Notes

- Use in `<Proxy>` layer.

Example

```
; 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
```

See Also

- Condition: `http.request.data=`
- Properties: `http.request.detection.injection.sql()`, `http.request.detection.other()`

http.request_line.regex=

Test the HTTP protocol request line.

Syntax

```
http.request_line.regex=regular_expression
```

Layer and Transaction Notes

- Use in `<Proxy>` and `<Exception>` layers.
- Applies to 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 the above syntax restrictions.

```
<Proxy>
 DENY("bad HTTP request line") \
  http.request_line.regex="\t|(^ )|( $)|(  )"
```

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
```

Layer and Transaction Notes

- Use in `<Proxy>`, `<Cache>`, and `<Exception>` layers.
- Applies to HTTP and SSL-terminated HTTPS transactions.

See Also

- Conditions: `http.response.code=`, `http.response.version=`
- Properties: `http.request.version( )`, `http.response.version( )`

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, matching the Apparent Data Type string in an HTTP request examines the first few bytes of a response from an OCS in order to identify the type of data being requested.

Syntax

```
http.response.apparent_data_type=(BMP|BZ2|CAB|EXE|FLASH|GIF|GZIP|HTML|ICC|JPG|MS
DOC|MRAR|MZIP|PDF|PNG|RAR|RTF|TAR|TIF|TTF|TXT|XML|ZIP)
```

Each Apparent Data Type defined in the syntax above can be interchanged with the file extension for that type, per the following chart:

Label	Description	Common Extensions
BMP	BMP image	.bmp
BZ2	BZip2 archive	.bz2, .tbz2
CAB	MS Cab archive	.cab
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
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
TTF	True-Type font	.ttf
TXT	plain text	.txt
XML	XML file	.xml
ZIP	ZIP archive	.zip

Layer and Transaction Notes

- Use in `<Cache>` and `<Proxy>` layers.
- Applies to All 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:

`response.icap.apparent_data_type=`
`http.request.apparent_data_type=`
`request.icap.apparent_data_type=`

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.

Replaces: `http.response_code`

syntax

`http.response.code=`*`nnn`*

where *`nnn`* is a standard numeric range test with values in the range 100 to 999 inclusive.

Layer and Transaction Notes

- Use in `<Proxy>`, `<Cache>`, and `<Exception>` layers.
- Applies to HTTP and SSL-terminated HTTPS transactions.

See Also

- Conditions: `http.request.version=`, `http.response.version=`
- Properties: `http.response.version( )`

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[StringQualifiers] = String
```

where:

- *N* equals the number of bytes, from 1..256
- *StringQualifiers* equal [.exact | .length | .prefix | .suffix | .substring | .regex | .hex][.case_sensitive]

Layer and Transaction Notes

- Use in <Cache>, <Proxy>, and <Exception> layers.
- Applies to all HTTP transactions (proxy, refresh, pipeline).
- As .regex values in policy can be costly in terms of system resources, Blue Coat 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, see http://www.garykessler.net/library/file_sigs.html.

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"
```

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.

Syntax

```
response.icap.apparent_data_type=(BMP|BZ2|CAB|EXE|FLASH|GIF|GZIP|HTML|ICC|JPG|MS
DOC|MRAR|MZIP|PDF|PNG|RAR|RTF|TAR|TIF|TTF|TXT|XML|ZIP)
```

Layer and Transaction Notes

- 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.
- Use in `<Cache>` and `<Proxy>` layers.
- Requires an ICAP response modification rule configured. Policy will fail to compile if this is not present.

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.

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:

```
http.request.apparent_data_type=
http.response.apparent_data_type=
response.icap.apparent_data_type=
```

http.response.version=

Tests the version of HTTP used by the origin server to deliver the response to the ProxySG appliance.

Syntax

```
http.response.version=0.9|1.0|1.1
```

Layer and Transaction Notes

- Use in <Proxy>, <Cache>, and <Exception> layers.
- Applies to HTTP and SSL-terminated HTTPS transactions.

See Also

- Conditions: http.request.version=, http.response.code=
- Properties: http.response.version( )

http.transparent_authentication=

This condition evaluates to true if HTTP uses transparent proxy authentication for this request.

The condition can be used with the `authenticate( )` or `authenticate.force( )` properties to select an authentication realm.

Syntax

```
http.transparent_authentication=yes|no
```

Layer and Transaction Notes

- Use in `<Proxy>` and `<Exception>` layers.
- Applies to HTTP and SSL-terminated HTTPS transactions.

See Also

- Conditions: `attribute.`*`name`*`=`, `authenticated=`, `group=`, `has_attribute.name=`, `realm=`, `user=`, `user.domain=`
- Properties: `authenticate( )`, `authenticate.force( )`, `authenticate.mode( )`, `check_authorization( )`

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. Previous versions of SGOS did not allow WebSocket handshakes to complete, but supported versions allow the handshake to complete successfully. Supported versions also detect WebSocket traffic and allow you to perform specific policy actions.

Syntax

```
http.websocket=yes|no
```

Layer and Transaction Notes

- Use in `<Proxy>`, `<Forward>`, and `<Exception>` layers.
- Applies to HTTP and HTTPS transactions.

See Also

- Conditions: `client.protocol=`

Example

The following example blocks all WebSocket traffic to testsite.com.

```
<Proxy>
   url.domain=testsite.com http.websocket=yes DENY
```

icap_error_code=

Test which ICAP error occurred. Rules containing this trigger do not match for a transaction that does not involve ICAP scanning.

Syntax

`any|none|<ICAP_error>`

where:

<ICAP_error> is one of `none, scan_timeout, decode_error, password_protected, insufficient_space, max_file_size_exceeded, max_total_size_exceeded, max_total_files_exceeded, file_extension_blocked, antivirus_load_failure, antivirus_license_expired, antivirus_engine_error, connection_failure, request_timeout, internal_error, server_error, server_unavailable`.

Layer and Transaction Notes

- Use in `<Proxy>` and `<Exception>` layers.
- Applies to all HTTP transactions (proxy, refresh, pipeline), FTP proxy transactions

Example

In this example, the administrator has chosen to allow access to files that fail scanning because of password protection.The scanning property must use the optional `fail_open` setting, and that the rules must also allow an error code of `none`. Also, the example assumes a user-defined exception page named `virus_scan_failure`.

```
<Proxy>
  response.icap_service(virus_scan, fail_open)
```

```
<Proxy>
  icap_error_code=!(none, password_protected) exception(virus_scan_failure)
```

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>.<matcher1>[.<matcher2>]

where:

- *<icap_method>* is either `icap_reqmod` or `icap_respmod`
- *<header_name>* is the name of the ICAP response header
- *<matcher1>* is one of the following optional matchers ([*<matcher2>*] denotes a second optional matcher):
    - exact[.case_sensitive] = <*string*>
    - prefix[.case_sensitive] = <*string*>
    - suffix[.case_sensitive] = <*string*>
    - substring[.case_sensitive] = <*string*>
    - regex[.case_sensitive] = <*regex*>
    - length = an integer between 0 - 8192 representing the length of the header value
    - exists = yes | no
    - as_number = a positive integer representing the header value converted to an unsigned 32-bit integer

The condition defaults to a regex match if no matcher is specified.

Notes

- Ranges are supported when defining `length` and `as_number` separated by `..` (a range of 1000..2000 will trigger the rule for a value between 1000 and 2000 bytes.). See *Condition Syntax* in this guide for information on using a double period with integer-based values.
- Because non-standard headers are common, policy written using numeric matchers could end up testing headers that contain strings. In these cases, strings are ignored and the policy takes no effect.

Layer and Transaction Notes

- Use in `<Proxy>`, `<Cache>`, and `<Exception>` layers.
- Applies to all HTTP transactions

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_name*=

Tests whether a health check is reporting as healthy.

Syntax

`is_healthy.`*`health_check_name`* `= yes|no`

where: *`health_check_name`*

Name of a specific health check. 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

- Use in `<Proxy>`, `<Cache>`, `<Forward>`, and `<SSL>` layers.
- Applies to: All transactions

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=`

iterator=

When referenced inside of an `iterate` definition block, this condition compares the text string passed against the current string value being iterated over.

Syntax

- `iterator=`*`string`*
- `iterator[.prefix|substring|suffix|regex|exact][.case_sensitive]=`*`string`*

Layer and Transaction Notes

- Use only inside of a `iterate` block.
- `iterator=` is the same as `iterator.regex=`

Example

```
define action DeleteSampleCookies
     iterate(request.header.Cookie)
        iterator.prefix="Sample" iterator.delete()
     end
end

<Proxy>
   action.DeleteSampleCookies(yes)
```

See Also

- Action: `iterate()`

ldap.attribute.*ldap_attribute_name*=

Compares strings with the value of the LDAP attribute obtained from the user's entry.

Syntax

```
ldap.attribute.ldap_attribute_name[.case_sensitive]=regular_expression
ldap.attribute.ldap_attribute_name.exact[.case_sensitive]=string
ldap.attribute.ldap_attribute_name.prefix[.case_sensitive]=string
ldap.attribute.ldap_attribute_name.substring[.case_sensitive]=string
ldap.attribute.ldap_attribute_name.suffix[.case_sensitive]=string
ldap.attribute.ldap_attribute_name.regex[.case_sensitive]=regular_expression
```

where:

- *ldap_attribute_name*—Name of a specific LDAP attribute

Layer and Transaction Notes

- Use in `<Proxy>`, `<Admin>`, `<Forward>`, `<Exception>`, `<SSL-Intercept> and <SSL>` layers.
- Applies to all transactions

Example(s)

```
; This example denies 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*`=`, `ldap.attribute.`*ldap_attribute_name*`.exists=`, `ldap.attribute.`*ldap_attribute_name*`.count=`, `ldap.attribute.`*ldap_attribute_name*`.as_number=`

ldap.attribute.*ldap_attribute_name*.as_number=

Converts the value of the attribute to an unsigned 32-bit integer, and then allows numerical tests to be done. There must be exactly one value in the list of values.

Syntax

`ldap.attribute.`*`ldap_attribute_name`*`.as_number = `*`integer-pattern`*

where:

- *`ldap_attribute_name`*—Name of a specific LDAP attribute

Layer and Transaction Notes

- Use in `<Proxy>`, `<Admin>`, `<Forward>`, `<Exception>`, `<SSL-Intercept>` and `<SSL>` layers.
- Applies to all transactions

Example(s)

```
; This example allows users with high priority only to access the proxy.

<Proxy>
   authenticate(LDAPRealm)

<Proxy>
   ALLOW ldap.attribute.UserPriority.as_number=0
   DENY
```

See Also

- Conditions: `ldap.attribute.`*`ldap_attribute_name`*`=has_attribute.`*`name`*`=`, `ldap.attribute.`*`ldap_attribute_name`*`.exists=`, `ldap.attribute.`*`ldap_attribute_name`*`.count=`

ldap.attribute.*ldap_attribute_name*.count=

Tests the number of values in a list for the named attribute.

Syntax

```
ldap.attribute.ldap_attribute_name.count = integer-pattern
```

where:

- *ldap_attribute_name*—Name of a specific LDAP attribute

Layer and Transaction Notes

- Use in `<Proxy>`, `<Admin>`, `<Forward>`, `<Exception>`, `<SSL-Intercept>` and `<SSL>` layers.
- Applies to all transactions

Example(s)

```
; This example denies 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.ldap_attribute_name=`, `ldap.attribute.ldap_attribute_name=`, `ldap.attribute.ldap_attribute_name.exists=`, `ldap.attribute.ldap_attribute_name.as_number=`

ldap.attribute.*ldap_attribute_name*.exists=

Checks if the named attribute exists in the user's entry.

Syntax

```
ldap.attribute.ldap_attribute_name.exists = {yes|no}
```

where:

- *ldap_attribute_name*—Name of a specific LDAP attribute

Layer and Transaction Notes

- Use in `<Proxy>`, `<Admin>`, `<Forward>`, `<Exception>`, `<SSL-Intercept>` and `<SSL>` layers.
- Applies to all transactions

Example(s)

```
; This example allows 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.ldap_attribute_name=`, `ldap.attribute.ldap_attribute_name=`, `ldap.attribute.ldap_attribute_name.count=`, `ldap.attribute.ldap_attribute_name.as_number=`

live=

Tests if the streaming content is a live stream.

Syntax

```
live=yes|no
```

Layer and Transaction Notes

- Use in `<Cache>` and `<Proxy>` layers.
- Applies to streaming transactions.

Example

```
; The following policy restricts access to live streams during morning hours.
; In this example, we use a policy layer to define policy just for the live streams.
; This example uses the restrict form and integrates with other <Proxy> layers.

 <Proxy>
   deny live=yes time=1200..0800 ; Policy for live streams
```

See Also

- Conditions: `bitrate=`, `streaming.client=`, `streaming.content=`
- Properties: `access_server( )`, `max_bitrate( )`, `streaming.transport( )`

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]={[`*`first_minute`*`]..[`*`last_minute`*`]|`*`exact_minute`*`}`

where:

- *`first_minute`*—An integer from 0 to 59, indicating the first minute of the hour that tests true. If left blank, minute 0 is assumed.
- *`last_minute`*—An integer from 0 to 59, indicating the last minute of the hour that tests true. If left blank, minute 59 is assumed.
- *`exact_minute`*—An 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

- Use in all layers.
- Using time-related conditions to control caching behavior in a `<Cache>` layer might cause thrashing of the cached objects.

Example

```
; Tests for the first 5 minutes of every hour.
minute=0..4
```

See Also

- Conditions: `date[.utc]=`, `day=`, `hour=`, `month=`, `time=`, `weekday=`, `year=`

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`*—An 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`*—An integer from 1 to 12, where 1 specifies January and 12 specifies December, specifying the last month that tests true. If left blank, December (month 12) is assumed.
- *`exact_month`*—An integer from 1 to 12, where 1 specifies January and 12 specifies December, 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

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

Example

```
; Tests for the year-end holiday season.

define condition year_end_holidays
month=12 day=25..
month=1 day=1
end_condition year_end_holidays
```

See Also

- Conditions: `date[.utc]=`, `day=`, `hour=`, `minute=`, `time=`, `weekday=`, `year=`

proxy.address=

Tests the destination address of the incoming IP packet.

If the transaction was explicitly proxied, `proxy.address=` 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, `proxy.address=` tests the destination address contained in the IP packet.

Note: This test might not be equivalent to testing the `server_url.address`. The `server_url.address` and `proxy.address` 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=` contains the address of the upstream proxy through which the request is to be handled.

Note: functions correctly for transparent transactions.

`proxy.card=`**Syntax**

`proxy.address=`*`ip_address`*`|`*`ip_address_range`*`|`*`ip_address_wildcards`*`|`*`subnet`*`|`*`subnet_label`*

where:

- *`ip_address`*—NIC address or subnet specification; for example, `10.1.198.54`
- *`ip_address_range`*—NIC address range; for example, `192.0.2.0-192.0.2.255`
- *`ip_address_wildcards`*—NIC address specified using wildcards in any octet(s); for example, `10.25.*.0` or `10.*.*.0`
- *`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

Layer and Transaction Notes

- Use in `<Proxy>`, `<Forward>`, `<exception>`, `<admin>`, `<DNS-Proxy>`, `<SSL>`, and `<SSL-Intercept>` layers.
- Applies to proxy transactions.

Example

```
; Service should be denied through proxy within the subnet 1.2.3.x.

 <Proxy>
 proxy.address=1.2.3.0/24 deny
```

See Also

- Conditions: `client.address=`, `client.protocol=`
- Definitions: `define subnet`
- Information on wildcards: http://bluecoat.force.com/knowledgebase/articles/Solution/000010037
- Information on IP address ranges: http://bluecoat.force.com/knowledgebase/articles/Solution/000010950

proxy.card=

Tests the ordinal number of the network interface card (NIC) used by a request.

Syntax

```
proxy.card=card_number
```

where *card_number* is an integer that reflects the installation order.

Layer and Transaction Notes

- Use in <Proxy>, <Forward>, <Exception>, <Admin>, <DNS-Proxy>, <SSL>, and <SSL-Intercept> layers.
- Applies to proxy transactions.

Example

```
; Deny all incoming traffic through proxy card 0.

 <Proxy>
   proxy.card=0 deny
```

See Also

- Conditions: client.address=, client.protocol=, proxy.address=, proxy.port=

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_of_port_numbers`*`]|`*`exact_port_number`*`}`

where:

- *`range_of_port_numbers`*—A 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`.
- *`exact_port_number`*—A single port number; for example, `80`. Can be a number between 1 and 65535.

Layer and Transaction Notes

- Use in `<Proxy>`, `<Forward>`, `<Exception>`, `<Admin>`, `<DNS-Proxy>`, `<SSL>`, and `<SSL-Intercept>` layers.
- Applies to proxy transactions.

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=`, `client.protocol=`, `proxy.address=`, `proxy.card=`, `proxy.port=`, `server_url.port=`

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

- Use in `<Proxy>`, `<Forward>`, and `<Exception>` layers.
- Applies to proxy transactions.

Example

```
<Proxy>
  p2p.client=gnutella
```

raw_url.regex=

Test the value of the raw request URL.

The `raw_url=` 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.

Syntax

```
raw_url.regex=regular_expression
```

Layer and Transaction Notes

- Use in `<Proxy>`, `<Cache>`, and `<Exception>` layers.
- Applies to proxy transactions.

Example

- Reject request as invalid if URL encodes letters and digits using hex escape sequences. Rationale: 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])"
```

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`".

Syntax

`raw_url.host.regex=`*`regular_expression`*

Layer and Transaction Notes

- Use in `<Proxy>`, `<Exception>`, and `<Cache>` layers.
- Applies to proxy transactions.

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`".

Syntax

```
raw_url.path.regex=regular_expression
```

Layer and Transaction Notes

- Use in `<Proxy>`, `<Exception>`, and `<Cache>` layers.
- Applies to proxy transactions.

Example

- Reject request as invalid if path encodes letters and digits using hex escape sequences. Rationale: 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`".

Syntax

```
raw_url.pathquery.regex=regular_expression
```

Layer and Transaction Notes

- Use in `<Proxy>`, `<Cache>`, and `<Exception>` layers.
- Applies to proxy transactions.

Example

- Reject request as invalid if `pathquery` encodes letters and digits using hex escape sequences. Rationale: 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.port.regex=

Test the value of the port component of the raw request URL.

The `raw_url.port=` condition is the original character string used to specify the port in the HTTP request. It is different from the `url.port=` condition because it is a string, not an integer, and because of the following:

- Leading zeroes are not removed. Thus, `raw_url.port.regex="^0"` is true if there are leading zeroes.
- If the port is specified as a naked colon, with no following port number, then the string is the empty string, and `raw_url.port.regex="^$"` will be true.

  If no port is specified, then no regex will match, and `raw_url.port.regex=!""` will be true.

Syntax

`raw_url.port.regex=`*`regular_expression`*

Layer and Transaction Notes

- Use in `<Proxy>`, `<Exception>`, and `<Cache>` layers.
- Applies to proxy transactions.

Example

- Reject request as invalid if port specifier is a naked colon or has leading zeroes.

```
<Proxy>
 exception(invalid_request) raw_url.port.regex=("^$" || "^0")
```

raw_url.query.regex=

`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. For example, "`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

- Use in `<Proxy>`, `<Exception>`, and `<Cache>` layers.
- Applies to proxy transactions.

Example

- Reject request as invalid if query encodes letters and digits using hex escape sequences. Rationale: 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])"
```

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=` 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, the `authenticate` property 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.

Note: When used in the `<Forward>` layer, authentication conditions 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.

Syntax

`realm=`*`realm_name`*

where *`realm_name`* is the name of an NTLM, Local Password, RADIUS, LDAP, Certificate, or Sequence realm. Realm names are case-insensitive for all realm types.

Layer and Transaction Notes

- Use in `<Proxy>`, `<Forward>`, `<Exception>`, `<Admin>`, `<SSL-Intercept>` and `<SSL>` layers.
- Applies to proxy and administrator transactions.

Example

```
; This example tests if the user has logged into realm corp and
; is authenticated in the specified group.
realm=corp group=all_staff

; This example uses the realm property to 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=`, `group=`, `has_attribute.`*`name`*`=`, `http.transparent_authentication=`, `user=`, `user.domain=`, `user.x509.issuer=`, `user.x509.serialNumber=`, `user.x509.subject=`
- Properties: `authenticate( )`, `authenticate.force( )`, `check_authorization( )`

release.id=

Tests the software release ID of the appliance.

Syntax

`release.id=`*`number`*

where *`number`* is a five-digit number that increases with each new release of SGOS.

Layer and Transaction Notes

- Can be used in any type of layer.

Example

```
; the condition below is only true if you are running a version of ProxySG appliance
; whose release id is 18000 or later

release.id=18000..
```

See Also

- Conditions: `release.version=`

release.version=

Tests the release version of the appliance.

Syntax

```
release.version={[minimum_version]..[maximum_version]|version}
```

where each of the versions is of the format:

```
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

- Can be used in any layer.

Example

```
; the condition below is only true if you are running SGOS
; release version 6.5 or later

release.version=6.5..

; the condition below is only true if you are running SGOS
; release version 6.6 or earlier

release.version=..6.6
```

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.

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=N
```

- In the above example, *N* equals the number of bytes.
- Ranges are supported when defining `as_number` separated by `..` (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

• Use in `<Proxy>` and `<Cache>` layers.

• Used 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*= instead. For streaming requests, only the User-Agent header is available.

Syntax

request.header.*header_name=regular_expression*

request.header.*header_name*.exact=*string*

request.header.*header_name*.prefix=*string*

request.header.*header_name*.substring=*string*

request.header.*header_name*.suffix=*string*

request.header.*header_name*.regex=*regular_expression*

where:

- *header_name*—A recognized HTTP header. For a complete list of recognized headers, see Appendix C: "Recognized HTTP Headers".
- *regular_expression*—A regular expression. For more information, see Appendix D: "Using Regular Expressions".
- *string*— Any printable ASCII sequence, quote delimited.

Layer and Transaction Notes

- Use in <Admin>, <Cache>, <Exception>, <Forwarding>, and <Proxy> layers.
- Applies to HTTP, Streaming transactions. For streaming requests. only the User-Agent header is available.

Example

```
;Example 1: deny access when request is sent with Pragma-no-cache header

 <Proxy>
 deny url=http://www.bluecoat.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

- Actions: `append( ),delete( ),delete_matching( ),rewrite( ),set( ), transform`
- Conditions: `request.header.`*`header_name`*`.address=,request.x_header.`*`header_name`*`=, request.x_header.`*`header_name`*`.address=, response.header.`*`header_name`*`=`
- Properties: `transform_data.type()`

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 subnet definition 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`*—`client-ip` and `X-Forwarded-For`.
- *`ip_address`*—IP address; for example, `10.1.198.46`.
- *`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.

Layer and Transaction Notes

- Use in `<Cache>` and `<Proxy>` layers.

Example

```
; In this example, we assume that there is a downstream appliance that
; identifies client traffic by putting the client's IP address in a request
; header.

; Here we'll deny access to some clients, based on the header value.

 <Proxy>
 ; Netscape's convention is to use the Client-IP header
 deny request.header.Client-IP.address=10.1.198.0/24 ; the subnet

 ; Blue Coat's convention is to use the extended header:
 deny request.header.X-Forwarded-For.address=10.1.198.12
```

See Also

- Actions: `append( ),delete( ),delete_matching( ),rewrite( ),set( )`
- Conditions: `request.header.`*`header_name`*`=, response.header.`*`header_name`*`=, response.x_header.`*`header_name`*`=`
- Definitions: `define subnet`
- Properties: `transform_data.type()`

request.header.*header_name*.exists=

Test whether a request header exists.

Syntax

```
request.header.header_name.exists=yes|no
```

Layer and Transaction Notes

- Valid layers: Proxy, Exception
- Applies to: HTTP proxy transactions

Example

Sample usage:

```
<Proxy>
  request.header.Accept.exists=yes
```

request.header.*header_name*.count=

Test the number of header values in the request for the given *header_name*.

Syntax

```
request.header.header_name.count=numeric range from 0 to 8192
```

Layer and Transaction Notes

- Use in <Proxy>, <Cache>, <Forwarding>, and <Exception> layers.
- Applies to HTTP proxy transactions.

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*.length=

Test the total length of the header values for the given *header_name*.

Syntax

```
request.header.header_name.length=numeric range from 0 to 8192
```

Layer and Transaction Notes

- Use in <Proxy>, <Forwarding>, and <Exception> layers.
- Applies to HTTP proxy transactions.

Example

- Deny HTTP requests with more than 2K of cookie data.

```
<Proxy>
 DENY("Too much Cookie data") request.header.Cookie.length = 2048..
```

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[.case_sensitive][.no_lookup]=prefix_pattern
request.header.Referer.url.domain[.case_sensitive][.no_lookup]=
                                                        domain_suffix_pattern
request.header.Referer.url.exact=string
request.header.Referer.url.prefix=string
request.header.Referer.url.substring=string
request.header.Referer.url.suffix=string
request.header.Referer.url.regex=regular_expression

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[.exact]=host
request.header.Referer.url.host.[prefix|substring|suffix]=string
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={[low_port_number]..[high_port_number]
                                 |exact_port_number}

request.header.Referer.url.query.regex[.case_sensitive]=regular_expression

request.header.Referer.url.scheme=url_scheme

request.header.Referer.url.host.has_name=yes|no|restricted|refused|nxdomain \
|error
request.header.Referer.url.is_absolute=yes|no
```

where all options are identical to `url=`, except for the URL being tested. For more information, see “url=” on page 231.

Discussion

The `request.header.Referer.url=` condition is identical to `url=`, except for the lack of a `define url` condition and `[url]` or `[url.domain]` sections.

Layer and Transaction Notes

- Use in `<Proxy>` and `<Exception>` layers.
- Applies to HTTP proxy transactions.

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)

; 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: `url=, server_url=`
- Definitions: `define subnet`
- Information on wildcards: http://bluecoat.force.com/knowledgebase/articles/Solution/000010037
- Information on IP address ranges: http://bluecoat.force.com/knowledgebase/articles/Solution/000010950

request.header.Referer.url.category=

Test the content filter categories of the Referer URL.

Syntax

```
request.header.Referer.url.category=none|unlicensed|unavailable|pending|
category_name1, category_name2, ...
```

Layer and Transaction Notes

- Use in `<Proxy>` and `<Exception>` layers.
- Applies to proxy transactions with a request URL.

Example

```
<Proxy>
   request.header.Referer.url.category=Sports
```

See Also

- Conditions: `category=`, `url.category=`, `server.certificate.hostname.category=`

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

- Use in `<Proxy>` and `<Exception>` layers.
- Applies to proxy transactions with a request URL.

Example

```
<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.

Syntax

```
request.icap.apparent_data_type=(BMP|BZ2|CAB|EXE|FLASH|GIF|GZIP|HTML|ICC|JPG|MS
DOC|MRAR|MZIP|PDF|PNG|RAR|RTF|TAR|TIF|TTF|TXT|XML|ZIP)
```

Layer and Transaction Notes

- Use in `<Cache>` and `<Proxy>` layers.
- Recommended for Reverse Proxy deployments, where files are uploaded through the appliance to a back-end server.
- Applies to all HTTP and decrypted HTTPS POST requests that have been processed by a ProxyAV Appliance running version 3.5 or higher.
- An ICAP Request Modification rule is required to handle the appliance-to-ICAP communication.

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.

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:

```
http.request.apparent_data_type=<type>
http.response.apparent_data_type=<type>
response.icap.apparent_data_type=<type>
http.request.apparent_data_type.allow(<type>,…)
http.request.apparent_data_type.deny(<type>, …)
```

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=numeric range from 0 to 8192
```

Layer and Transaction Notes

- Use in `<Proxy>` and `<Exception>` layers.
- Applies to HTTP proxy transactions.

Example

- Reject the request if it contains more than 40 request headers.

```
<Proxy>
 exception(invalid_request) request.raw_headers.count=40..
```

request.raw_headers.length=

Test the total length of all HTTP request headers.

This condition tests the total number of bytes of HTTP request header data, including the header names, values, delimiters, and newlines. The tally does not include the HTTP request line (which contains the request method) and it does not include the terminating blank line.

Syntax

```
request.raw_headers.length=numeric range from 0 to 8192
```

Layer and Transaction Notes

- Use in `<Proxy>` and `<Exception>` layers
- Applies to HTTP proxy transactions

Example

- Reject the request if it contains more than 4K of request header data.

```
<Proxy>
 exception(invalid_request) request.raw_headers.length=4096..
```

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

- Use in `<Proxy>` and `<Exception>` layers.
- Applies to HTTP proxy transactions.

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 request header (*header_name*) against a regular expression or a string. Any HTTP request header can be tested, including custom headers. To test recognized headers, use request.header.*header_name*= 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=regular_expression
request.x_header.header_name.exact=string
request.x_header.header_name.prefix=string
request.x_header.header_name.substring=string
request.x_header.header_name.suffix=string
request.x_header.header_name.regex=regular_expression
```

where:

- *header_name*—Any HTTP header, including custom headers.
- *regular_expression*—A regular expression. For more information, see Appendix D: "Using Regular Expressions".
- *string*— Any printable ASCII sequence, quote delimited.

Layer and Transaction Notes

- Use in <Cache>, <Exception>, <Forward>, and <Proxy> layers.

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.bluecoat.com request.x_header.Test.exact="test1"
```

See Also

- Actions: append( ), delete( ), delete_matching( ), rewrite( ), set( )
- Conditions: request.header.*header_name*=, request.header.*header_name*.address=, request.x_header.*header_name*.address=, response.x_header.*header_name*=

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=` 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:

- *`header_name`*—Any HTTP header, including custom headers.
- *`ip_address`*—IP address; for example, `10.1.198.0`.
- *`ip_address_range`*—IP address specified using wildcards in any octet(s); for example, `10.25.*.0` or `10.*.*.0`.
- *`ip_address_wildcards`*—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.

Layer and Transaction Notes

- Use in `<Cache>` and `<Proxy>` layers.

Example

```
; deny access if the request's custom header "Local" has the value 10.1.198.0
deny request.x_header.Local.address=10.1.198.0
```

See Also

- Actions: `append( ),delete( ),delete_matching( ),rewrite( ),set( )`
- Conditions: `request.header.`*`header_name`*`=`, `request.header.`*`header_name`*`.address=`, `response.x_header.`*`header_name`*`=`
- Definitions: `define subnet`
- Information on wildcards: http://bluecoat.force.com/knowledgebase/articles/Solution/000010037
- Information on IP address ranges: http://bluecoat.force.com/knowledgebase/articles/Solution/000010950

request.x_header.*header_name*.count=

Test the number of header values in the request for the given *header_name*.

Syntax

```
request.x_header.header_name.count=numeric range from 0 to 8192
```

Layer and Transaction Notes

- Use in <Proxy> and <Exception> layers.
- Applies to HTTP proxy transactions.

Example

- Deny abnormal HTTP requests with 2 or more host headers.

```
<Proxy>
 DENY("Too many Host headers") request.header.Host.count =
```

request.x_header.*header_name*.exists=

Test whether a request header exists.

Syntax

```
request.x_header.header_name.exists=yes|no
```

Layer and Transaction Notes

- Valid layers: Proxy, Exception
- Applies to: HTTP proxy transactions

Example

Sample usage:

```
<Proxy>
request.x_header.Accept.exists=yes
```

request.x_header.*header_name*.length=

Test the total length of the header values for the given *header_name*.

Syntax

```
request.x_header.header_name.length=numeric range from 0 to 8192
```

Layer and Transaction Notes

- Use in `<Proxy>` and `<Exception>` layers.
- Applies to HTTP proxy transactions.

Example

- Deny HTTP requests with more than 2K of cookie data.

```
<Proxy>
 DENY("Too much Cookie data") request.header.Cookie.length = 2048..
```

response.header.content-length.as_number=

This condition is used to 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=N
```

- In the above example, *N* equals the number of bytes.
- 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* in this guide for information on using a double period with integer-based values.

Layer and Transaction Notes

• Use in `<Proxy>` and `<Cache>` layers.

• Used where the Origin Content Server 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 response.x_header.*header_name*= instead.

Syntax

response.header.*header_name*=*regular_expression*

response.header.*header_name*.exact=*string*

response.header.*header_name*.prefix=*string*

response.header.*header_name*.substring=*string*

response.header.*header_name*.suffix=*string*

response.header.*header_name*.regex=*regular_expression*

where:

- *header_name*—A recognized HTTP header. For a list of recognized headers, see Appendix C: "Recognized HTTP Headers". For custom headers not listed, use condition response.x_header.*header_name* instead.
- *regular_expression*—A regular expression. For more information, see Appendix D: "Using Regular Expressions".
- *string*—Any printable ASCII sequence, quote delimited

Layer and Transaction Notes

- Use in <Cache>, <Proxy>, and <Exception> layers.

Example

```
; Test if the response's "Content-Type" header has the value "image/jpeg"
response.header.Content-Type.prefix="image/jpeg( |\t)*($|;)"
```

See Also

- Actions: append( ), delete( ), delete_matching( ), rewrite( ), set( )
- Conditions: request.header.*header_name*=, response.x_header.*header_name*=

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=N|..N|N..|N1..N2
```

where N is an unsigned integer.

Layer and Transaction Notes

- Use in `<Proxy>` and `<Exception>` layers.
- Applies to HTTP proxy transactions.

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=N|..N|N..|N1..N2
```

where:

`N` is an unsigned integer.

Layer and Transaction Notes

- Use in `<Proxy>` and `<Exception>` layers.
- Applies to HTTP proxy transactions.

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

- Use in `<Proxy>` and `<Exception>` layers.
- Applies to HTTP proxy transactions.

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=` instead so that typing errors can be caught at compile time.

Syntax

```
response.x_header.header_name=regular_expression
response.x_header.header_name.exact=string
response.x_header.header_name.prefix=string
response.x_header.header_name.substring=string
response.x_header.header_name.suffix=string
response.x_header.header_name.regex=regular_expression
```

where:

- *header_name*—Any HTTP header, including custom headers.
- *regular_expression*—A regular expression. For more information, see Appendix D: "Using Regular Expressions"
- *string*—Any printable ASCII sequence, quote delimited

Layer and Transaction Notes

- Use in `<Cache>`, `<Proxy>`, and `<Exception>` layers.

Example

```
; Tests if the custom header "Security" has the value of "confidential"
response.x_header.Security.substring="confidential"
```

See Also

- Actions: `append( )`, `delete( )`, `delete_matching( )`, `rewrite( )`, `set( )`
- Conditions: `request.x_header.header_name=`, `response.header.header_name=`

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` is the threshold at which a specified action is triggered.

Layer and Transaction Notes

- Use in `<Proxy>` layers.
- Applies to all HTTP transactions.

See Also

- `risk_score.maximum()`, `risk_score.other()`

Example

```
; Block the connection when the client hits one or more violations
; Assume a default risk-score value of 10 for each violation
<proxy>
  risk_score=10.. deny
```

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. This condition is NULL for transactions that do not involve an SSL connection to the server.

Syntax

```
server.certificate.hostname=domain-name
server.certificate.hostname=domain-suffix-pattern
server.certificate.hostname.exact=string
server.certificate.hostname.length=value
server.certificate.hostname.prefix=string
server.certificate.hostname.substring=string
server.certificate.hostname.suffix=string
server.certificate.hostname.regex=regular_expression
```

where:

domain-name

A domain name is one or more domain labels, separated by dots, such as 'example.com' or 'www.example.com'. It matches the hostname exactly.

domain-suffix-pattern

A domain suffix pattern is a 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'.

> *Note:* 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

- Valid layers: `<Proxy>`, `<SSL>`, `<SSL-Intercept>`
- Applies to: HTTPS forward and reverse proxy transactions, SSL Intercept transactions, SSL tunnel transactions

Example

This example uses 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=`, `server.certificate.subject=`
  Properties: `server.certificate.validate( )`, `server.certificate.validate.ignore( )`, `server.certificate.validate.check_revocation( )`

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. This condition is NULL for transactions that do not involve an SSL connection to the server.

Syntax

```
server.certificate.hostname.category=none|unlicensed|unavailable|pending|
category_name1, category_name2, ...
```

Layer and Transaction Notes

- Use in `<SSL>` layers.
- Applies to proxy transactions.

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: `server.certificate.hostname=, server.certificate.common_name=, server.certificate.subject=, category=, url.category=, request.header.Referer.url.category=`
- Properties: `server.certificate.validate( ), server.certificate.validate.ignore( ), server.certificate.validate.check_revocation( )`

server.certificate.subject=

Test the subject field of an SSL server certificate.

Syntax

```
server.certificate.subject[.exact][.case_sensitive]=string
server.certificate.subject.length=value
server.certificate.subject.prefix[.case_sensitive]=string
server.certificate.subject.substring[.case_sensitive]=string
server.certificate.subject.suffix[.case_sensitive]=string
server.certificate.subject.regex[.case_sensitive]=regular_expression
```

Note: 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

- Valid layers: `<SSL>`, `<SSL-Intercept>`
- Applies to: HTTPS forward and reverse proxy transactions, SSL Intercept transactions, SSL tunnel transactions

Example

Sample usage:

```
<SSL>
 server.certificate.subject=todo
```

server.connection.dscp=

Test the server-side inbound DSCP value.

Syntax

```
client.server.dscp = dscp_value
```

where *dscp_value* is `0..63 | af11 | af12 | af13 | af21 | af22 | af23 | af31 | af32 | af33 | af41 | af42 | af43 | best-effort | cs1 | cs2 | cs3 | cs4 | cs5 | cs6 | cs7 | ef`

Layer and Transaction Notes

- Valid in <Proxy>, <DNS-Proxy>, <Cache> layers.
- Applies to all transactions.

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
```

where *cipher-suite* is one of the following:

- `none`
- a cipher suite that the appliance supports; refer to the "Managing X.509 Certificates" chapter in the *SGOSAdministration Guide* for information on the supported cipher suites.

Layer and Transaction Notes

- Valid layers: `<SSL> and <Proxy>`
- Applies to: HTTPS forward and reverse proxy transactions, SSL tunnel transactions

Example

This example implements the following policies:

1. DENY requests to servers that are not using one of the EXP suites
2. Log access to servers that are not using secure connections in 'unsecure_log1'

```
; 1
<SSL>
  ALLOW server.connection.negotiated_cipher= \
   (EXP-RC4-MD5|| \
    EXP-RC2-CBC-MD5|| \
    EXP-DES-CBC-SHA)
DENY
; 2
<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=none|low|medium|high|export
```

Layer and Transaction Notes

- Use in `<SSL>` and `<Proxy>` layers.
- Applies to proxy transactions.

Example

This example implements the following policies:

1. Allow only server connections that have a medium or high cipher strength.

```
<Proxy>
DENY server.connection.negotiated_cipher.strength=(low|export)
```

Notes

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.

server.connection.negotiated_ssl_version=

Test the SSL version negotiated with a secure server.

Syntax

```
server.connection.negotiated_ssl_version=SSLV2|SSLV3|TLSV1|TLSV1.1|TLSV1.2
```

Layer and Transaction Notes

- Valid layers: `<SSL>`, `<Proxy>`.
- Applies to: HTTPS forward and reverse proxy transactions, SSL tunnel transactions

Example

Sample usage:

```
<SSL>
 server.connection.negotiated_ssl_version=SSLV3
```

server_url=

Tests 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, 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( )` actions. Because any rewrites of the URL intended for servers or other upstream devices must be respected by `<Forward>` layer policy, the `url=` conditions (except `url.host=`) 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( )` actions on the URL.

Syntax

```
server_url[.case_sensitive][.no_lookup]=prefix_pattern
server_url.domain[.case_sensitive][.no_lookup]=domain_suffix_pattern

server_url.exact=string
server_url.prefix=string
server_url.substring=string
server_url.suffix=string
server_url.regex=regular_expressionregular_expression

server_url.address=
           ip_address|ip_address_range|ip_address_wildcards|subnet|subnet_label

server_url.extension[.case_sensitive]=[.]filename_extension

server_url.host[.exact] [.no_lookup]=host
server_url.host.[prefix|substring|suffix]=string
server_url.host.regex=regular_expression
server_url.is_absolute=yes|no
server_url.host.is_numeric=yes|no
server_url.host.has_name=yes|no|restricted|refused|nxdomain|error
server_url.host.no_name=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={[low_port_number]..[high_port_number]|exact_port_number}

server_url.query.regex[.case_sensitive]=regular_expression

server_url.scheme=url_scheme
```

where all options are identical to `url=`, except for the URL being tested. For more information, see “url=” on page 231.

Discussion

The `server_url=` condition is identical to `url=`, except for the lack of a `define server_url` condition and `[server_url]` section. Most optimization in forwarding is done with `server_url.domain` conditions and sections.

Layer and Transaction Notes

- Use in `<Proxy>`, `<Cache>`, `<Forward>`, `<SSL>` and `<SSL-Intercept>` layers.
- Applies to all non-administrator transactions.

Example

```
; Test if the server URL includes this 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)

; 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

; 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

<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=`, `url=`
- Definitions: `define subnet`, `define server_url.domain condition`
- Properties: transform_data.type()
- Information on wildcards: http://bluecoat.force.com/knowledgebase/articles/Solution/000010037
- Information on IP address ranges: http://bluecoat.force.com/knowledgebase/articles/Solution/000010950

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 476), by a local database you maintain, or by a third-party database.

A URL that is not categorized is assigned the category `none`.

If a content filter provider is selected in configuration, but an error occurs in determining the category, the URL is assigned the category `unavailable` (in addition to any categories assigned directly by policy). Errors can occur because of a missing database or an expired license. An additional category of `unlicensed` is assigned in the latter case.

A URL may have been assigned a list of categories. The `server_url.category=` condition is true if it matches any of the categories assigned to the URL.

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.

Syntax

```
server_url.category={ none|unlicensed|unavailable|category_name1,
category_name2, ...}
```

where *`category_name1, category_name2, ...`* represent category names defined by policy or the selected content filter provider. The list of currently valid category names is available through the Management Console and the CLI.

Layer and Transaction Notes

- Use in `<Proxy>`, `<Cache>`, `<Exception>`, `<Forward>`, `<SSL>`, and `<SSL-Intercept>` layers.
- Applies to all transactions.

Example

```
<Forward>
server_url.category="Entertainment" forward(host)
```

See Also

- Conditions: `category=, server_url=`
- Properties: `access_server( )`

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

- Use in `<Proxy>`, `<Cache>`, `<Forward>`, `<SSL>` and `<SSL-Intercept>` layers.
- Applies to all proxy transactions with a request URL.

Example

```
<proxy>
   server_url.host.is_private=yes
```

service.group=

Test the service group associated with a transaction.

Syntax

```
service.group=group_name
```

Layer and Transaction Notes

- Valid layers: Proxy, Admin, Forward, Exception, SSL, SSL-Intercept, DNS-Proxy
- Applies to: All proxy transactions excluding DNS proxy transactions

Example

This example shows how to forward the request based on service group.

```
<forward>
service.group=standard forward(standard_proxy)
```

See Also

- Conditions: `service.name=`

service.name=

Test the service name associated with a transaction.

Syntax

```
service.name=service_name
```

Layer and Transaction Notes

- Valid layers: `<Proxy>`, `<Admin>`, `<Forward>`, `<Exception>`, `<SSL>`, `<DNS-Proxy>`, and `<SSL-Intercept>`
- Applies to: All proxy transactions excluding DNS proxy transactions

Example

This example shows how to forward the request based on service name.

```
<forward>
service.name=http_internal forward(internal_proxy)
```

See Also

- Conditions: `service.group=`

Discussion

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.

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 or IM. In particular, `socks=yes` remains true even in the resulting HTTP or IM transactions. Other conditions, such as `proxy.address` or `proxy.port`, do not maintain a consistent value across the SOCKS transaction and the later HTTP or IM transaction, so they cannot be reliably used to do this kind of cross-protocol testing.

Syntax

```
socks=yes|no
```

Layer and Transaction Notes

- Use in `<Proxy>`, `<Exception>`, `<Forward>`, `<SSL>` layers.
- Applies to all proxy transactions.

See Also

- Conditions: `socks.accelerate=`
- Properties: `socks_gateway( )`, `socks.accelerate( )`, `socks.authenticate( )`, `socks.authenticate.force( )`.

socks.accelerated=

Tests whether the SOCKS proxy will hand off this transaction to other protocol agents for acceleration.

Syntax

```
socks.accelerated={yes|http|aol-im|msn-im|yahoo-im|no}
```

where:

- `yes` is true only for SOCKS transactions that will hand off to another protocol-specific proxy agent.
- `no` implies the transaction is a SOCKS tunnel.
- `http` is true if the transaction will be accelerated by the http proxy.
- `aol-im` is true if the transaction will be accelerated by the aol-im proxy.
- `msn-im` is true if the transaction will be accelerated by the msn-im proxy.
- `yahoo-im` is true if the transaction will be accelerated by the yahoo-im proxy.

Layer and Transaction Notes

- Use in `<Proxy>` and `<Exception>` layers.
- Applies to SOCKS transactions.

See Also

- Conditions: `socks.method=`, `socks.version=`
- Properties: `socks_gateway( )`, `socks.accelerate( )`, `socks.authenticate( )`, `socks.authenticate.force( )`.

socks.method=

Tests the SOCKS protocol method name associated with the transaction.

Syntax

```
socks.method=CONNECT|BIND|UDP_ASSOCIATE
```

Layer and Transaction Notes

- Use in `<Proxy>` and `<Exception>` layers.
- Applies to SOCKS transactions.

See Also

- Conditions: `ftp.method=`, `http.method=`, `server_url=`, `socks.version=`
- Properties: `socks_gateway( )`, `socks.accelerate( )`, `socks.authenticate( )`, `socks.authenticate.force( )`.

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()` or `socks.authenticate.force()` is set during evaluation of a SOCKS 4/4a transaction, that transaction will be denied.

Syntax

```
socks.version=4..5
```

Layer and Transaction Notes

- Use in `<Proxy>`, `<Forward>`, and `<Exception>` layers.
- Applies to SOCKS transactions.
- Does not apply to administrator transactions.

Example

This example authenticates SOCKS v5 clients, and allows 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.method=`, `socks.version=`
- Properties: `socks_gateway( )`, `socks.accelerate( )`, `socks.authenticate( )`, `socks.authenticate.force( )`

source.port=

Test the port that the client connects from.

Syntax

```
source.port=port-number
```

Layer and Transaction Notes

- Valid layers: Admin, DNS-Proxy, Forward, Proxy, Exception, SSL, SSL-Intercept
- Applies to: Proxy transactions

Example

```
<Admin>
source.port=8080
```

ssl.proxy_mode=

Test if the ProxySG appliance is intercepting and decrypting an SSL connection.

Syntax

```
ssl.proxy_mode = yes|no|https-reverse-proxy|https-forward-proxy
```

where:

`yes` means the same as (`https-reverse-proxy||https-forward-proxy`)

Layer and Transaction Notes

- Use in `<Proxy>` and `<SSL>` layers.
- Applies to proxy transactions.

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|windows_media|real_media|quicktime|flash|ms_smooth|adobe_hds|apple_hls
```

where:

- `yes` is true if the user agent is recognized as a Windows Media, Real Media, Quicktime, Flash, Smooth Streaming, Apple HLS, or Adobe HDS player.
- `no` is true if the user agent is not recognized as a Windows Media, Real Media, Quicktime, Flash, Smooth Streaming, Apple HLS, or Adobe HDS player.
- other values are true if the user agent is recognized as a media player of the specified type.

Layer and Transaction Notes

- Use in `<Proxy>`, `<Cache>`, `<Forward>`, and `<Exception>` layers.
- Applies to HTTP and streaming transactions. Does not apply to administrator transactions.

Examples

This example forces 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)
```

This example disables ADN byte compression since most video traffic can't be compressed efficiently:

```
<Forward>
streaming.client=ms_smooth adn.server.optimize.compress(no)
```

See Also

- Conditions: `bitrate=`, `live=`, `streaming.content=`
- Properties: `access_server( )`, `max_bitrate( )`, `streaming.transport( )`

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|windows_media|real_media|quicktime|flash
```

where:

- `yes` is true if the content is recognized as Windows Media, Real Media, QuickTime content. Note: Smooth Streaming content is not detected with this condition.
- `no` is true if the content is not recognized as Windows Media, Real Media, or QuickTime content.
- other values are true if the streaming content is recognized as the specified type.

Layer and Transaction Notes

- Use in `<Proxy>`, `<Cache>`, and `<Exception>` layers.
- Applies to all transactions.

See Also

- Conditions: `bitrate=`, `live=`, `streaming.client=`
- Properties: `access_server()`, `max_bitrate( )`, `streaming.transport( )`

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

- Use in `<Proxy>`, `<Cache>`, and `<Exception>` layers.
- Applies to Flash streaming transactions.

Example

`http://www.my_site.com/videoplayer/videoplayer.html` which embeds a SWF; `http://www.my_site.com/videoplayer/swfs/videoplayer.swf`, which plays videos from application 'vod' and stream name sample.flv. (URL: `rtmp://www.my_site.com/vod/sample.flv`)

In the above example, 'vod' is the `streaming.rtmp.app_name`.

See Also

- Conditions: `streaming.rtmp.method=`, `streaming.rtmp.page_url=`, `streaming.rtmp.stream_name=`, `streaming.rtmp.swf_url=`

streaming.rtmp.method=

Identifies the logical protocol operation on which policy is being applied.

Syntax

```
streaming.rtmp.method=open|connect|play
```

Layer and Transaction Notes

- Use in `<Proxy>`, `<Cache>`, and `<Exception>` layers.
- Applies to Flash streaming transactions.

See Also

- Conditions: `streaming.rtmp.app_name=, streaming.rtmp.page_url=, streaming.rtmp.stream_name=, streaming.rtmp.swf_url=`

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`*–The requested URL
- *`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`

Layer and Transaction Notes

- Use in `<Proxy>`, `<Cache>`, and `<Exception>` layers.
- Applies to Flash transactions.

Example

```
streaming.rtmp.page_url=http://www.mysite.com/videoplayer/videoplayer.html
```

See Also

- Conditions: `streaming.rtmp.method=`, `streaming.rtmp.app_name=`, `streaming.rtmp.stream_name=`, `streaming.rtmp.swf_url=`
- Information on wildcards: http://bluecoat.force.com/knowledgebase/articles/Solution/000010037
- Information on IP address ranges: http://bluecoat.force.com/knowledgebase/articles/Solution/000010950

streaming.rtmp.stream_name=

Identifies the name of the Flash stream being requested.

Syntax

```
streaming.rtmp.stream_name=<string>
```

Layer and Transaction Notes

- Use in `<Proxy>`, `<Cache>`, and `<Exception>` layers.
- Applies to Flash transactions.

Example

```
streaming.rtmp.stream_name=sample.flv
```

The URL of the stream is rtmp://www.mysite.com/vod/sample.flv.

See Also

- Conditions: `streaming.rtmp.method=`, `streaming.rtmp.page_url=`, `streaming.rtmp.app_name=`, `streaming.rtmp.swf_url=`

streaming.rtmp.swf_url=

Identifies 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*–The requested URL
- *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`

Layer and Transaction Notes

- Use in `<Proxy>`, `<Cache>`, and `<Exception>` layers.
- Applies to Flash transactions.

Example

```
streaming.rtmp.swf_url=http://www.mysite.com/videoplayer/swfs/videoplayer.swf
```

See Also

- Conditions: `streaming.rtmp.method=`, `streaming.rtmp.page_url=`, `streaming.rtmp.stream_name=`, `streaming.rtmp.app_name=`
- Information on wildcards: http://bluecoat.force.com/knowledgebase/articles/Solution/000010037
- Information on IP address ranges: http://bluecoat.force.com/knowledgebase/articles/Solution/000010950

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

`time[.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 day into the next, the following shorthand expression is available. While `time=(..0600|1900..)` specifies 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

- Use in all layers.
- Using time-related conditions to control caching behavior in a `<Cache>` layer may cause thrashing of the cached objects.
- Applies to all transactions.

Example

```
; 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=`, `hour=`, `minute=`, `month=`, `weekday=`, `year=`

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

- Use in `<Proxy>`, `<Forward>`, `<Exception>`, and `<SSL>` layers.
- Applies to proxy transactions.

Example

This example denies 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=`, `socks.accelerated=`, `url.scheme=`

Properties: `sock.accelerate()`

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( )` 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=` tests are provided for use in the `<Forward>` layer. Those tests always take into account the effect of any `rewrite( )` actions on the URL.

Replaces: various `url_`*`xxx`* forms.

Syntax

```
url[.case_sensitive][.no_lookup]=prefix_pattern
url.domain[.case_sensitive][.no_lookup]=prefix_pattern|domain_suffix_pattern
url.exact[.case_sensitive]=string
url.prefix[.case_sensitive]=string
url.substring[.case_sensitive]=string
url.suffix[.case_sensitive]=string
url.regex[.case_sensitive]=regular_expression

url.address=
          ip_address|ip_address_range|ip_address_wildcards|subnet|subnet_label

url.extension[.case_sensitive]=[.]filename_extension

url.host[.exact][.no_lookup]=host
url.host.[prefix|substring|suffix]=string
url.host.regex=regular_expression
url.is_absolute=yes|no
url.host.is_numeric=yes|no
url.host.no_name=yes|no
url.host.has_name=yes|no|restricted|refused|nxdomain|error

url.path[.exact][.case_sensitive]=/string
url.path[.prefix][.case_sensitive]=/string
url.path[.substring|.suffix][.case_sensitive]=string
url.path.regex[case_sensitive]=regular_expression

url.port={[low_port_number]..[high_port_number]|exact_port_number}

url.query.regex[.case_sensitive]=regular_expression

url.query.exists=yes|no

url.query.[exact|prefix|suffix|substring]=string
```

`url.scheme=`*`url_scheme`*

where the URL test patterns are:

- *`prefix_pattern`*—A URL pattern that includes at least a portion of the following:

  `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`*—A URL pattern that includes a domain suffix, as a minimum, using the following 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 `http`, `https`, `ftp`, `mms`, `rtsp`, `icp`, `or 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`*—A 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*—A 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*—A port number, between 1 and 65535.
- *path_query*—The *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*—A 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*—A Perl regular expression. The expression must be quoted if it contains whitespace or any of the following: `& | ( ) < > { } ; ! . = " '`. For more information, see Appendix D: "Using Regular Expressions".

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.

In SGOS 6.5.6.1 and later, 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:

- `.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, either 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, bolding shows the components used in the comparison; `?q=test` is the included query component 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`*—A port number at the low end of the range to be tested. Can be a number between 1 and 65535.
- *`high_port_number`*—A port number at the high end of the range to be tested. Can be a number between 1 and 65535.
- *`exact_port_number`*—A 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 `url` definition block or a `[url]` section (see "define url condition" on page 487).

If you are testing a large number of URLs using the `url.domain=` condition, consider the performance benefits of a `url.domain` definition block or a `[url.domain]` section (see "define url.domain condition" on page 489).

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

- Use in `<Proxy>`, `<Cache>`, `<Exception>`, `<SSL>` and `<SSL-Intercept>` layers. `url.host=` can also be used in `<Forward>`.
- Applies to all non-administrator transactions.

Example

```
; 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.com

; 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)

; 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=`, `console_access=`, `content_management=`, `server_url=`

- Definitions: `define subnet`, `define url condition`, `define url.domain condition`
- Information on wildcards:
  http://bluecoat.force.com/knowledgebase/articles/Solution/000010037
- Information on IP address ranges:
  http://bluecoat.force.com/knowledgebase/articles/Solution/000010950

url.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

```
url.application.name=NAME
```

where `NAME` is defined by the Blue Coat Web Filter database. Use the **`show content-filter bluecoat applications`** CLI command to list the supported application names.

Layer and Transaction Notes

- Use in `<Proxy>`, `<Cache>`, `<Exception>` layers. In SGOS 6.5.6.1 and later, also use in `the <ssl-intercept>` layer.
- Applies to proxy transactions with a request URL.

Example

- The following policy blocks Hotmail.

```
<Proxy>
    url.application.name=Hotmail deny
```

- The following policy allows LinkedIn access but denies Hotmail.

```
; Allow LinkedIn application but deny Hotmail.
; Show an error page displaying the application name
define string my_format_string
> url.application.name= $(url.application.name)
  > and the details: $(exception.details)
end
<Proxy>
    url.application.name=Linkedin allow ; no explicit allows
   url.application.name=Hotmail exception( policy_denied, "To prevent data loss, personal email is prohibited by management.", my_format_string )
```

- The following policy blocks all social networking sites except for Facebook.

```
<Proxy>
url.application.name=Facebook allow
url.category=social-networking deny
```

See Also

- Conditions: `http.request.data=`, `url.application.operation=`, `url.category=`

url.application.operation=

Test the Blue Coat content filter application operation of the request URL.

Syntax

```
url.application.operation=OPERATION
```

where OPERATION is defined by the Blue Coat Web Filter database. Use the **show content-filter bluecoat operations** CLI command to list the supported application operations.

Layer and Transaction Notes

- Use in `<Proxy>`, `<Cache>`, `<Exception>` layers. In SGOS 6.5.6.1 and later, also use in `<ssl>` and `<ssl-intercept>` layers.
- Applies to proxy transactions with a request URL.

Example

The following policy allows attachment downloads, but blocks attachment uploads for all applications.

```
; Allow attachment downloads but not uploads.
; Show an error page displaying the application operation
define string my_format_string
> url.application.operation= $(url.application.operation)
  > and the details: $(exception.details)
end
<Proxy>
    url.application.operation=”Download Attachment” allow ; no explicit allows
    url.application.operation=”Upload Attachment” exception( policy_denied,
"Attachment uploads are not allowed.", my_format_string )
```

See Also

- Conditions: `url.application.name=`
- Conditions: `url.category=`

url.category=

Test the content filter categories of the request URL. Synonym for '`category=`'.

Syntax

```
url.category=none|unlicensed|unavailable|pending|category_name1, category_name2,
...
```

Layer and Transaction Notes

- Use in `<Cache>`, `<Proxy>`, `<Exception>`, `<SSL>`, and `<SSL-Intercept>` layers.
- Applies to proxy transactions with a request URL.

Example

```
<Cache>
  url.category=Sports
```

See Also

- Conditions: `category=`, `request.header.Referer.url.category=`, `server.certificate.hostname.category=`

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

- Use in `<Cache>`, `<Proxy>`, `<Exception>`, `<SSL>`, and `<SSL-Intercept>` layers.
- Applies to proxy transactions with a request URL.

Example

```
<Proxy>
   url.host.is_private=yes
```

user=

Tests the authenticated username associated with the transaction. This condition is only available if the transaction was authenticated (that is, the `authenticate( )` property was set to something other than `no`, and the `proxy_authentication( )` property was not set to `no`).

Syntax

`user=`*`user_name`*

where *`user_name`* is a username.

- IWA realm: Usernames are case-insensitive.

  In IWA this provides the flexibility of matching either a full username or relative username.

  For example:

  `user=bluecoat\mary.jones`

  matches a complete username, and

  `user=mary.jones`

  matches a relative name.
- 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=bluecoat,dc=com"`

  -or-

  `user="uid=mary.jones,ou=sales,o=bluecoat"`

  matches a complete username, and

  `user=mary.jones`

  matches a relative name.

Layer and Transaction Notes

- Use in `<Proxy>`, `<Forward>`, `<Exception>`, `<Admin>`, `<SSL-Intercept>` and `<SSL>`, layers.

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 user john.smith.
user=john.smith
```

See Also

- Conditions: `authenticated=`, `group=`, `has_attribute.`*`name`*`=`, `http.transparent_authentication=`, `realm=`, `user.domain=`, `user.x509.issuer=`, `user.x509.serialNumber=`, `user.x509.subject=`
- Properties: `authenticate( )`, `authenticate.force( )`, `check_authorization( )`, `deny.unauthorized( )`, `socks.authenticate( )`, `socks.authenticate.force( )`, `ssl.forward_proxy( )`

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>,...)
```

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>,...`* - specifies a single error or a group of errors. If the authentication error is one of the errors in the list then the condition will evaluate to true.

Layer and Transaction Notes

- Valid layers: Proxy, SSL, SSL-Intercept, Forward, Exception
- Applies to: Proxy transactions

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
```

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>,...)`

where:

- `any` - evaluates to true if there was 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>,...` - specifies a single error or a group of errors. If the authorization error is one of the errors in the list then the condition will evaluate to true.

Layer and Transaction Notes

- Valid layers: Proxy, SSL, SSL-Intercept, Forward, Exception
- Applies to: Proxy transactions

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( )` property is set to `no`).

Syntax

`user.domain=`*`windows_domain_name`*

where *`windows_domain_name`* is a Windows domain name. This name is case-insensitive.

Layer and Transaction Notes

- Use in `<Proxy>`, `<Forward>`, `<Exception>`, `<Admin>`, `<SSL-Intercept>` and `<SSL>`, layers.

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.
user.domain=all-staff
```

See Also

- Conditions: `authenticated=`, `group=`, `has_attribute.`*`name`*`=`, `http.transparent_authentication=`, `realm=`, `user=`, `user.x509.issuer=`, `user.x509.serialNumber=`, `user.x509.subject=`
- Properties: `authenticate( )`, `authenticate.force( )`, `check_authorization( )`, `deny.unauthorized( )`, `socks.authenticate( )`, `socks.authenticate.force( )`, `ssl.forward_proxy( )`

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

- Valid layers: Proxy, SSL-Intercept, SSL, Forward, Exception
- Applies to: Proxy transactions

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`*`|`*`ip_address_range`*`|`*`subnet`*`|`*`subnet_label`*

Layer and Transaction Notes

- Valid layers: `<proxy>`, `<admin>`, `<SSL-Intercept>`, `<forward>`, `<exception>`
- Applies to: All transactions

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://bluecoat.force.com/knowledgebase/articles/Solution/000010037
- Information on IP address ranges: http://bluecoat.force.com/knowledgebase/articles/Solution/000010950

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(`*`[lower]..[upper]|exact`*`)`

where:

- *`[lower]`* is optionally the fewest number of logins that will match this condition.
- *`[upper]`* is optionally the most number of logins that will match this condition.
- *`exact`* is optionally the exact number of logins that will match this condition.

Layer and Transaction Notes

- Valid layers: Proxy, Admin, SSL-Intercept, Forward, Exception
- Applies to: All transactions

Example

Log out the other logins of a user if the user is logged in more than once.

```
<proxy>
  user.login.count=2.. user.login.log_out_other(yes)
```

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(`*`[lower]..[upper]|exact`*`)`

where:

- *`[lower]`* is optionally the fewest number of seconds that will match this condition
- *`[upper]`* is optionally the most number of seconds that will match this condition
- *`exact`* is optionally the exact number of seconds that will match this condition

Layer and Transaction Notes

- Valid layers: Proxy, Admin, SSL-Intercept, Forward, Exception
- Applies to: All transactions

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)
```

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

Valid Layers: Proxy, SSL, Forward, Exception, Admin

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: Appendix D: "Using Regular Expressions".

user.x509.issuer=

Tests the issuer of the x509 certificate used in authentication to certificate realms. The `user.x509.issuer=` 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`* is an RFC2253 LDAP DN, appropriately escaped. Comparisons are case-sensitive.

Layer and Transaction Notes

- Use in `<Proxy>`, `<Forward>`, `<Exception>`, `<Admin>`, and `<SSL>` layers.

  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.

- Applies to proxy transactions.

See Also

- Conditions: `authenticated=`, `group=`, `has_attribute.`*`name`*`=`, `http.transparent_authentication=`, `realm=`, `user=`, `user.domain=`, `user.x509.serialNumber=`, `user.x509.subject=`
- Properties: `authenticate( )`, `authenticate.force( )`

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`* is a string representation of the certificate's serial number in HEX.

The string is always an even number of characters long, so if the number needs an odd number of characters to represent in hex, there is a leading zero. This can be up to 160 bits.

Layer and Transaction Notes

- Use in `<Proxy>`, `<Forward>`, `<Exception>`, `<Admin>`, and `<SSL>` layers.

  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.

- Applies to proxy transactions.

See Also

- Conditions: `authenticated=`, `group=`, `has_attribute.`*`name`*`=`, `http.transparent_authentication=`, `realm=`, `user=`, `user.domain=`, `user.x509.issuer=`, `user.x509.subject=`
- Properties: `authenticate( )`, `authenticate.force( )`

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`* is an RFC2253 LDAP DN, appropriately escaped.

Comparisons are case-sensitive.

Layer and Transaction Notes

- Use in `<Proxy>`, `<Forward>`, `<Exception>`, `<Admin>`, and `<SSL>` layers.

  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.

- Applies to proxy transactions.

See Also

- Conditions: `authenticated=`, `group=`, `has_attribute.`*`name`*`=`, `http.transparent_authentication=`, `realm=`, `user=`, `user.domain=`, `user.x509.issuer=`, `user.x509.serialNumber=`
- Properties: `authenticate( )`, `authenticate.force( )`

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

- Use in `<Proxy>` and `<Exception>` layers.
- Applies to All HTTP transactions (proxy, refresh, pipeline), FTP proxy transactions.

Example

```
<Proxy>
  virus_detected=yes
```

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]={[`*`first_weekday`*`]..[`*`last_weekday`*`]|`*`exact_weekday`*`}`

where:

- *`first_weekday`*—An 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.
- *`last_weekday`*—An 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.
- *`exact_weekday`*—An integer from 1 to 7, where 1 specifies Monday and 7 specifies Sunday, indicating the day of the week that tests true.

Note: When you want 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

- Use in all layers.
- Using time-related conditions to control caching behavior in a `<Cache>` layer may cause thrashing of the cached objects.
- Applies to all transactions.

Example

```
; Test for the weekend.
weekday=6..7

; Test for Saturday through Monday.
weekday=6..1
```

See Also

- Conditions: `date[.utc]=`, `day=`, `hour=`, `minute=`, `month=`, `time=`, `year=`

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 ProxySG 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]={[`*`first_year`*`]..[`*`last_year`*`]|`*`exact_year`*`}`

where:

- *`first_year`*—Four digits (*nnnn*) representing the start of a range of years; for example, `2002`.
- *`last_year`*—Four digits (*nnnn*) representing the end of a range of years. If left blank, all years from first_year on are assumed.
- *`exact_year`*—Four digits (*nnnn*) representing an exact year.

Note: To test against an inverted range of years, the following shorthand expression is available. While `year=(..1998|2003..)` specifies years up to and including 1998, and from 2003 on, the policy language also recognizes `year=2003..1998` as equivalent.

Layer and Transaction Notes

- Use in all layers.
- Using time-related conditions to control caching behavior in a `<Cache>` layer may cause thrashing of the cached objects.
- Applies to all transactions.

Example

```
; Tests for the years 2004 through 2006.
year=2004..2006
```

See Also

- Conditions: `date[.utc]=`, `day=`, `hour=`, `minute=`, `month=`, `time=`, `weekday=`, `year=`

Chapter 4: *Property Reference*

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.

Property Reference

The remainder of this chapter lists the properties and their accepted values. It also provides tips as to where each property can be used and examples of how to use them.

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 455.

Syntax

```
access_log(auto|no|log_name_list)
access_log.log_name(yes|no)
access_log[log_name_list](yes|no)
```

The default value is `auto`.

where:

- `auto`—use the default log for this protocol.
- `no`—turns off logging, either for this transaction or to the specified *`log_name`* or *`log_name_list`*.
- `yes`—turns on logging for this transaction to the specified *`log_name`* or *`log_name_list`*.
- *`log_name`*—an access log name as defined in configuration
- *`log_name_list`*—a list of access log names as defined in configuration, of the form:

  `log_name_1, log_name_2, ...`

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_name_list]( )` 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

- Use in all `<Forward>`, `<Proxy>`. `<Exception>`, `<Cache>` layers.
- Applies to proxy transactions.

See Also

- Properties: `log.suppress.`*`field-id`*, `log.rewrite.`*`field-id`*`( )`
- Actions: `log_message( )`

access_server( )

Determines whether the client can receive streaming content directly from the origin content server or other upstream device. Set to `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.

Note: 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.

Syntax

```
access_server(yes|no)
```

The default value is `yes`.

Layer and Transaction Notes

- Use in `<Proxy>`, `<Cache>`, `<DNS-proxy>`, and `<Forward>` layers.
- Applies to DNS, HTTP, SOCKS, and streaming transactions.

See Also

- Conditions: `bitrate=`, `live=`, `streaming.client=`, `streaming.content=`

action( )

Selectively enables or disables a specified define action block. The default value is `no`.

Note: Several define action 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)
```

The default value is `no` for all defined actions.

where *`action_label`* is the label of the `define action` 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()` enables the specified action block and disables all other actions 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

- Use in `<Cache>`, `<Proxy>`, and `<Exception>` layers. The actions specified in the action block must be appropriate to the referencing layer.

See Also

- Definitions: `define action`

adn.connection.dscp( )

This property controls the DSCP value for both upstream and downstream ADN tunneled packets.

Syntax

```
adn.connection.dscp(dscp_value)
```

where:

dscp_value

```
0..63 | af11 | af12 | af13 | af21 | af22 | af23 | af31 | af32 | af33 | af41 |
af42 | af43 | best-effort | cs1 | cs2 | cs3 | cs4 | cs5 | cs6 | cs7 | ef |
preserve
```

If the property specified is `preserve`, the ADN proxies (both branches and concentrators) preserves 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.

If the property specified is a specific 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.

The default value is `preserve`.

Layer and Transaction Notes

- Valid layers: Forward
- Applies to: All transactions

Example(s)

The 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( ), adn.server.dscp( ), server.connection.dscp( )`

adn.server( )

This property is used to enable or disable ADN service.

Syntax

```
adn.server(yes|no)
```

The default value is taken from the applicable service configuration.

Layer and Transaction Notes

- Valid layers: Forward
- Applies to: Proxy transactions

Example

This property allows control of whether an ADN service is enabled in the `<Forward>` layer. The following example shows the property used to disable ADN service.

```
<Forward>
  adn.server(no)
```

See Also

- Properties: `adn.server.optimize( )`

adn.server.optimize( )

If the value is set to yes then data both to and from the server are optimized.

This property is applied only if the application proxy server connection is forwarded over an ADN tunnel.

Syntax

```
adn.server.optimize(yes|no|byte_cache|compress)
adn.server.optimize.optimization-setting(yes|no)
adn.server.optimize[optimization-settings](yes|no)
```

Where optimization-setting is one of `byte_cache` or `compress`. The default value is taken from the applicable service configuration, which is dependent on the service.

Layer and Transaction Notes

- Valid layers: Forward
- Applies to: Proxy transactions

Example

This property allows control of the optimization of individual connections based on any policy conditions available in the `<Forward>` layer. The following example shows the property used to 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( ),adn.server.optimize.outbound( )`

adn.server.optimize.inbound( )

If the value is set to yes then data received from the server will be optimized.

This property is applied only if the application proxy server connection is forwarded over an ADN tunnel.

Syntax

```
adn.server.optimize.inbound(yes|no|byte_cache|compress)
adn.server.optimize.inbound.optimization-setting(yes|no)
adn.server.optimize.inbound[optimization-settings](yes|no)
```

where optimization-setting is one of `byte_cache` or `compress`. The default value is taken from the applicable service configuration, which is dependent on the service.

Layer and Transaction Notes

- Valid layers: Forward
- Applies to: Proxy transactions

Example

This property allows control of the optimization of data received over individual server connections based on any policy conditions available in the `<Forward>` layer. The following example shows the property used to 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( ),adn.server.optimize.outbound( )`

adn.server.optimize.outbound( )

If the value is set to yes then data sent to the server are optimized.

This property is applied only if the application proxy server connection is forwarded over an ADN tunnel.

Syntax

```
adn.server.optimize.outbound(yes|no|byte_cache|compress)
adn.server.optimize.outbound.optimization-setting(yes|no)
adn.server.optimize.outbound[optimization-settings](yes|no)
```

where optimization-setting is one of `byte_cache` or `compress`. The default value is taken from the applicable service configuration, which is dependent on the service.

Layer and Transaction Notes

- Valid layers: Forward
- Applies to: Proxy transactions

Example

This property allows control of the optimization of data sent over individual server connections based on any policy conditions available in the `<Forward>` layer. The following example shows the property used to 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: a`dn.server.optimize( ),adn.server.optimize.inbound( )`

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 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 value is `no`.

Layer and Transaction Notes

- Use in `<Cache>` layers.
- Applies to HTTP transactions, except FTP over HTTP transactions.

See Also

- Properties: `always_verify( ),cache( ),cookie_sensitive( ),pipeline( ),refresh( ), ttl( ),ua_sensitive( )`

allow

Allows the transaction to be served.

Allow can be overridden by the `access_server( )`, `deny( )`, `force_deny( )`, `authenticate( )`, `exception( )`, or `force_exception( )` properties or by the `redirect( )` action.

Allow overrides `deny( )` and `exception( )` properties.

Note: Caution should be exercised when using `allow` in layers evaluated after layers containing `deny`, to ensure that security is not compromised.

Syntax

```
allow
```

Layer and Transaction Notes

- Use in `<Proxy>`, `<Cache>`, `<SSL>`, and `<Admin>` layers. Use "access_server( )" on page 261.
- Applies to all transactions.

See Also

- Properties: `access_server( )`, `deny( )`, `force_deny( )`, `authenticate( )`, `exception( )`, `force_exception( )`
- Actions: `redirect( )`

always_verify( )

Determines whether each request for the objects at a particular URL must be verified with the origin 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 value is `no`.

Layer and Transaction Notes

- Use in `<Proxy>` layers.
- Use in `<Cache>` layers.
- Applies to HTTP proxy transactions, except FTP over HTTP transactions.

See Also

- Properties: `advertisement( ),bypass_cache( ),cache( ),cookie_sensitive( ), force_cache( ),pipeline( ),refresh( ),ttl( ),ua_sensitive( )`

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 attack-detection CLI

Syntax

```
attack_detection.failure_weight(<N>)
```

where:

- *N*—Sets 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

- Use in `<Proxy>` layers.
- Applies to HTTP proxy transactions, except FTP over HTTP transactions.

Example

Sets 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=`

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` option.

Syntax

```
authenticate(realm_name)
```

-or-

```
authenticate(no [, upstream_authentication|no_upstream_authentication])
```

where:

- `realm_name` is the name of a configured authentication realm.
- `upstream_authentication` indicates that offered proxy credentials should be passed upstream
- `no_upstream_authentication` indicates that offered proxy credentials should not be passed upstream

Layer and Transaction Notes

- Use in `<Proxy>` and `<Admin>` layers.
- Applies to Proxy and Administrative transactions.

Example

This example implements the following policy:

1. All traffic to `a.com` is authenticated.
2. All traffic to b.com is authenticated by an upstream proxy.
3. 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=`, `exception.id=`, `group=`, `has_attribute.name=`, `http.transparent_authentication=`, `realm=`, `user=`, `user.domain=`
- Properties: `authenticate.force( )`, `authenticate.mode( )`, `authenticate.use_url_cookie( )`, `check_authorization( )`, `socks.authenticate( )`, `socks.authenticate.force( )`

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 is the 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.

The default value is `1`.

Layer and Transaction Notes

- Valid layers: Proxy, Admin
- Applies to: Proxy transactions, Administrative transactions

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 ProxySG 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 the authenticate.charset property.

The default value is `ascii`.

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(charset)
```

where:

`charset` A MIME charset name. Any of the standard charset names for encodings commonly supported by Web browsers may be used.

One list of standard charset names is: http://www.iana.org/assignments/character-sets.

If you use Microsoft Windows, then you can use the `chcp` command in the Windows CLI to find out your active code page. Once you know the code page number n, you can use `windows-n` as the charset name.

The default value is `ascii`.

Layer and Transaction Notes

- Use in `<Proxy>` and `<Admin>` layers.
- Applies to HTTP proxy transactions.

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`* is the 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

- Valid layers: Proxy, Admin
- Applies to: Proxy transactions, Administrative transactions

Example

Set the credential refresh time to one hour.

```
<proxy>
  authenticate(myrealm) authenticate.credential_refresh_time(3600)
```

authenticate.credentials.address( )

Specify the number of seconds that surrogates can be trusted.

This property allows overriding the address used for authentication. It sets the IP address of the login session.

Syntax

`authenticate.credentials.address(`*`ip address`*

The default value is 0.0.0.0.

Layer and Transaction Notes

- Valid layers: <proxy>
- Applies to: Proxy transactions

Example

Set the address of the login to value from the `Client-IP` header.

```
<proxy>
  authenticate(myrealm)\ authenticate.credentials.address($(.Client-IP))
```

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`* specifies the substitution string to evaluate to determine the guest username, for example "`guest_$(client.address)`"
- `surrogate-refresh-time` (optional) specifies the 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` (optional) specifies the 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

- Valid layers: Proxy
- Applies to: Proxy transactions

Example

To log users in as guest automatically

```
<proxy>
  authenticate.guest(realm,guest_user) allow
```

To 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.force( )

This property controls the relation between authentication and denial.

Syntax

```
authenticate.force(yes|no)
```

The default value is `no`.

where:

- `yes` —Makes an `authenticate( )` higher priority than `deny( )` or `exception( )`. Use `yes` to ensure that user IDs are available for access logging (including denied requests).
- `no`—`deny( )` and `exception( )` have a higher priority than `authenticate( )`. This setting allows early denial.

Layer and Transaction Notes

- Use in `<Proxy>` and `<Admin>` layers and transactions.
- Does not apply to `<Cache>` layers or transactions.

See Also

- Conditions: `authenticated=`, `group=`, `has_attribute.name=`, `http.transparent_authentication=`, `realm=`, `user=`, `user.domain=`
- Properties: `authenticate( )`, `check_authorization( )`, `socks.authenticate( )`, `socks.authenticate.force( )`

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)
```

The default value is `no`.

Layer and Transaction Notes

- Valid layers: Proxy
- Applies to: HTTP proxy transactions

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

- Used in `<Proxy>` layers.
- Applies to HTTP proxy transactions.

Example

This example implements the following policy:

- 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()

(Introduced in version 6.6.4.1) 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 value of `#(config)security force-credential-forwarding` command sets the default for this property.

Layer and Transaction Notes

- Used in `<Proxy>` layers.
- Applies to HTTP proxy transactions.

See Also

- `authenticate.forward_credentials.log()`

Example

```
; forward credentials to myhost.com
<proxy>
  url.domain=//myhost.com/ authenticate.forward_credentials(yes)
```

authenticate.forward_credentials.log()

(Introduced in version 6.6.4.1) 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()` 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 value of `#(config)security force-credential-forwarding-logging` command sets the default for this property.

Layer and Transaction Notes

- Used in `<Proxy>` layers.
- Applies to HTTP proxy transactions.

See Also

- `authenticate.forward_credentials()`

Example

```
; forward credentials to myhost.com and log the action
<proxy>
  url.domain=//myhost.com/ authenticate.forward_credentials(yes) \
    authenticate.forward_credentials.log(yes)
```

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. Blue Coat 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 ProxySG 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: Blue Coat 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`* is 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.
- **`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.

- **origin**: The appliance acts like an OCS and issues OCS challenges. The authenticated connection serves as the surrogate credential.
- **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-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-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.
- **SG2**: The mode is selected automatically, based on the request.
- **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-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-redirect**: This is similar to `form-ip` except that the user is redirected to the authentication virtual URL before the form is presented.

Important:	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

- Use in `<Proxy>` layers
- Applies to proxy transactions.

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)
```

where *new-pin-form-name* is the name of a valid new-pin form

Layer and Transaction Notes

- Use in `<Proxy>` layers.
- Applies to HTTP proxy transactions.

Example

This example implements the following policy:

1. All traffic from subnet HR_subnet must use the new-pin form '`HR_new_pin_form`'
2. All traffic from subnet ENG_subnet must use the new-pin form E`NG_new_pin_form`
3. 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)
```

where *query-form-name* is the name of a valid query form.

Layer and Transaction Notes

- Use in `<Proxy>` layers.
- Applies to HTTP proxy transactions.

Example

This example implements the following policy:

1. All traffic from subnet HR_subnet must use the query form `HR_query_form`
2. All traffic from subnet ENG_subnet must use the query form `ENG_query_form`
3. 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

- Use in `<Proxy>` layers.
- Applies to HTTP proxy transactions.

Example

```
<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* is the 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

- Valid layers: Proxy, Admin
- Applies to: Proxy transactions, Administrative transactions

Example

Set the surrogate refresh time to one hour.

```
<proxy>
  authenticate(myrealm) authenticate.surrogate_refresh_time(3600)
```

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.

IMPORTANT NOTE: 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:

`authenticate.tolerate_error(`*`<error>`*`,?)` is equivalent to `authenticate.tolerate_error[`*`<error>`*`?](yes)`

- `yes` - specifies that the error is permitted and the transaction should proceed unauthenticated
- `no` - specifies that the error is not permitted and the transaction should terminate
- *`<error>`*`,...` - specifies a single error or a group of errors

Layer and Transaction Notes

- Valid layers: Proxy
- Applies to: Proxy transactions

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, search for Blue Coat Knowledge Base article 000013368 at https://bto.bluecoat.com/knowledgebase.
- Authentication schemes that use IP surrogates, (such as Proxy-IP and Origin-IP-Redirect) are not compatible with this configuration. Instead, we recommend 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)
```

Note: 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
```

Note: 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( )

This property is used to authenticate users who have third party cookies explicitly disabled.

Note: 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.

Syntax

```
authenticate.use_url_cookie(yes|no)
```

The default is `no`.

Layer and Transaction Notes

- Use in `<Proxy>` layers.
- Applies to HTTP proxy transactions.

See Also

Properties: `authenticate.mode( )`

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>,...)`

where:

<group>`,...` - specifies the group or list of groups to add to the user

Layer and Transaction Notes

- Valid layers: Proxy, Admin
- Applies to: All transactions

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)
```

authorize.tolerate_error()

Specify to allow certain errors during user authorization.

This property can be used to support attempting authorization but allowing the transaction to proceed if authorization fails for the specified error.

Note: If this property is not explicitly specified in policy, it defaults to allowing the same errors as specified in any `authenticate.tolerate_error` properties.

Syntax

`authorize.tolerate_error.<error>(yes|no)`

`authorize.tolerate_error[<error>,...](yes|no)`

`authorize.tolerate_error(<error>,...)`

where:

authorize.tolerate_error(<*error*>,?) is equivalent to `authorize.tolerate_error[<error>?](yes)`

- `yes` - specifies that the error is permitted and the transaction should proceed without authorization data
- `no` - specifies that the error is not permitted and the transaction should terminate
- `<error>,...` - specifies a single error or a group of errors

Layer and Transaction Notes

- Valid layers: Proxy
- Applies to: Proxy transactions

Example

Add a user to a default group if authentication succeeded but authorization failed because of 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, the `bypass_cache` 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 is `no`.

Layer and Transaction Notes

- Use only in `<Proxy>` layers.
- Applies to HTTP, HTTPS, FTP over HTTP, and transparent FTP transactions.

Example

```
; Bypass the cache for requests from this client IP address.
client.address=10.25.198.0 bypass_cache(yes)
```

See Also

- Properties: `advertisement( ),always_verify( ),cache( ),cookie_sensitive( ), direct( ),dynamic_bypass( ),force_cache( ),pipeline( ),refresh( ),ttl( ), ua_sensitive( )`

cache( )

Controls HTTP and FTP caching behavior. A number of CPL properties affect caching behavior.

- If `bypass_cache(yes)` is set, then the cache is not accessed and the value of `cache( )` is irrelevant.
- If `cache(yes)` is set, then the `force_cache` 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)
```

The default is `yes`.

where:

- `yes`—Specifies the 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

- Use only in `<Cache>` layers.
- Applies to proxy transactions.

Example

```
; Prevent objects at this URL from being added to the cache.
url=http://www.example.com/docs cache(no)
; This example shows use of cache(yes) 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( ),always_verify( ),bypass_cache( ),cookie_sensitive( ), direct( ),dynamic_bypass( ),force_cache( ),pipeline( ),refresh( ),ttl( ), ua_sensitive( )`

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 is `no`.

Layer and Transaction Notes

- Use in `<Proxy>` and `<Cache>` layers.
- Applies to HTTP and RTSP proxy transactions.

See Also

- Conditions: `authenticated=`, `group=`, `has_attribute.name=`, `http.transparent_authentication=`, `realm=`, `user=`, `user.domain=`
- Properties: `authenticate( )`, `authenticate.force( )`

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 value is `yes`.

Layer and Transaction Notes

- Valid layers: Proxy, Admin
- Applies to: Proxy transactions, Administrative transactions

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.require( )

Controls whether a certificate is requested from the client during SSL negotiations.

Syntax

```
client.certificate.require(yes|no)
```

For HTTPS Forward Proxy transactions, the default value is `no`. For HTTPS Reverse Proxy transactions the default value is the same as the value of the `verify-client` attribute for the corresponding reverse proxy service.

Layer and Transaction Notes

- Use in `<SSL>` and `<Proxy>` layers.
- Applies to HTTPS forward and reverse proxy transactions.

Example

This HTTPS forward proxy policy implements consent certificates. The client Web browser is required to send a client certificate. The user has a choice of two predefined certificates: one certificate gives the proxy permission to intercept the SSL session, and the other certificate denies the proxy this permission. Thus, the human end-user gives explicit consent to have the SSL session intercepted.

```
<SSL> ssl.proxy_mode=https-forward-proxy
  client.certificate.require(yes)
  <SSL> ssl.proxy_mode=https-forward-proxy
  OK client.certificate.common_name = "Yes decrypt my data"
  FORCE_DENY
```

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, Blue Coat recommends that you do not disable certificate validation. If youmust do so, disable it only for specific, trusted URLs, for example, using the `url=` condition. Including `client.certificate.validate(no)` in policy disables all certificate validation forthe affected transactions, including checks for the validity of the certificate (such as trust chainand 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

- Use in `<SSL>` layers.
- Applies to HTTPS forward and reverse proxy transactions.

Example

```
<ssl>
  url.domain=”example.com” client.certificate.validate(no)
```

See Also

- Properties: `server.certificate.validate( )`

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` the certificate will be checked through OCSP if available, otherwise it is checked against locally installed revocation list.
- `ocsp` checks the certificate through OCSP.
- `no` the certificate is not checked for revocation.
- `local` checks the certificate against the locally installed revocation list.

The default value is `auto`.

Layer and Transaction Notes

- Valid layers: `SSL`.
- Applies to HTTPS forward and reverse proxy transactions.

Example

```
Sample usage:

<SSL>
  client.certificate.validate.check_revocation(local)
```

client.connection.dscp()

Controls client side outbound QoS/DSCP value.

Syntax

```
client.connection.dscp(dscp_value)
```

where *dscp_value* is `0..63 | af11 | af12 | af13 | af21 | af22 | af23 | af31 | af32 | af33 | af41 | af42 | af43 | best-effort | cs1 | cs2 | cs3 | cs4 | cs5 | cs6 | cs7 | ef | echo | preserve`

The special value `preserve` means to track the incoming DSCP value on the primary server connection and use that as the value when sending packets on the client connections. The special value `echo` means the outbound packet's DSCP value will use the same value as the inbound packet's DSCP value.

The default value is `preserve`.

Layer and Transaction Notes

- Valid in <Proxy> and <DNS-Proxy> layers.
- Applies to all transactions.

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 disabled Encrypted Tap. 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*—Disables Encrypted Tap
- *<interface>*—Set the Ethernet interface to output the tapped traffic to. The form is:
  - <adapter>:<interface>

Layer and Transaction Notes

- Applies only to client connections.
- Available in the SSL access policy layer.

Example

```
<ssl-intercept>
ssl.forward_proxy(stunnel)
<ssl>
client.connection.encrypted_tap(1:0)
<ssl>
server.certificate.validate(no)
```

See Also

- *Conditions:* .*header_name*=, .*header_name*.address=, request.x_header.*header_name*=, request.x_header.*header_name*.address=, response.header.*header_name*=, response.x_header.*header_name*=, server_url=
- *ProxySG Log Fields and CPL Substitutions Reference*

client.effective_address()

Lets you change the IP address that the `client.effective_address=` condition evaluates against. To handle cases where HTTP request header fields are not present or do not contain valid IP addresses, you can set multiple parameters. Policy will then use these parameters in the specified order to look up the effective client IP address.

This property will not change the effective client IP address if you attempt to set it to a value that is not an IP address.

Syntax

`client.effective_address(default | <IP_address> | <IP_address>, <IP_address>, ...)`

where:

- `default`—Resets `client.effective_address` to the `client.address`.
- *`<IP_address>`*—A text string representation of the IP address. This string can contain substitutions. If the first string is not a valid IP address, the next one in the list (if one exists) is attempted.

Layer and Transaction Notes

- Available in all layers.
- Applies to all transactions.

Example

The following example uses substitutions 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("$(request.header.x-forwarded-for)", \
          "$(request.x_header.x-client-ip)")
<Proxy>

     client.effective_address=192.0.2.0 deny
```

Use Case: Use the client IP address from the HTTP CONNECT request in subsequent HTTPS transactions

In this example, an organization has deployed a load balancer that does not decrypt SSL in front of the appliance but performs network address translation (NAT). The organization wants tunneled SSL transactions to inherit the effective client IP address set in the HTTP CONNECT request header.

When client traffic goes through the load balancer, the load balancer does the following:

- NATs the client IP address of HTTP and HTTPS transactions.
- Inserts the original client IP address in the X-Forwarded-For (XFF) header of HTTP transactions.
- Initiates an HTTP CONNECT transaction to tunnel the HTTPS transactions to the appliance.

The appliance is configured to use the address in the XFF header field as the client IP address. When the appliance intercepts HTTPS traffic, one of the following occurs, depending on policy configuration:

- The appliance extracts the client IP address from the XFF headers in the HTTPS transactions.
- The HTTPS transactions inherit the XFF header value from the parent HTTP CONNECT transaction.

To use the client IP address from HTTP transactions sfor each subsequent HTTPS transaction, the administrator installs the following policy. If SSL interception occurs for HTTPS traffic within the HTTP CONNECT transaction, the HTTPS transaction inherits the effective client IP address from the parent HTTP CONNECT transaction

```
<Proxy>
   client.protocol=http client.effective_address("$(request.header.x-forwarded-for)")
```

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.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.header.x-forwarded-for)")
```

See Also

- `client.address=`
- `client.effective_address.country=`
- `client.effective_address.is_overridden=`
- `client.protocol=`
- *ProxySG Log Fields and CPL Substitutions Reference*

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 `cache(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

- Use in `<Cache>` layers.
- Applies to HTTP proxy transactions, except FTP over HTTP transactions.

See Also

- Properties: `advertisement( ),always_verify( ),bypass_cache( ),cache( ),direct( ), force_cache( ),pipeline( ),refresh( ),ttl( ),ua_sensitive( )`

delete_on_abandonment( )

Specifies whether an object should be cached if client connection is interrupted before fetching of the object is completed.

If set to `yes`, specifies that 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 abandoned, and any prior instance of the object is deleted from the cache.

If set to `no`, specifies that 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.

If set to `auto`, it behaves like `yes` if the connection is over an ADN or bandwidth-gain is enabled, and it behaves like `no` otherwise.

This property is ignored for Flash VOD caching.

Syntax

```
delete_on_abandonment(yes|no|auto)
```

The default value is `auto`.

Layer and Transaction Notes

- Use in `<Cache>` layers.
- Applies to proxy transactions.

Example(s)

Sample usage:

```
<Cache>
delete_on_abandonment(auto)
```

See Also

- Properties: `advertisement( ),always_verify( ),bypass_cache( ),cache( ), cookie_sensitive( ),direct( ),dynamic_bypass( ),force_cache( ),pipeline( ), refresh( ),ttl( ),ua_sensitive( )`

deny( )

Denies service.

Denial can be overridden by `allow` or `exception( )`. To deny service in a way that cannot be overridden by a subsequent `allow`, use `force_deny( )` or `force_exception( )`.

The relation between `authenticate( )` and `deny( )` is controlled by the `authenticate.force )` property. By default, `deny( )` overrides `authenticate( )`. 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( )` and `deny( )` is controlled by the `socks.authenticate.force( )` property. By default, `deny( )` overrides `socks.authenticate( )`.

Syntax

```
deny
deny(details)
```

where *details* is a string defining a message to be displayed to the user. The details string may contain CPL substitution variables.

Discussion

The `deny(details)` property is equivalent to `exception(policy_denied, details)`. The identity of an exception being returned can be tested in an `<Exception>` layer using `exception.id=`.

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( )` instead to give the user an opportunity to retry the request with different credentials.

Layer and Transaction Notes

- Use in `<Cache>`, `<Proxy>`, `<SSL>`, and `<Admin>` layers. In `<Forward>` layers, use "access_server( )" on page 261.
- Applies to all transactions.

Example

```
deny url.address=10.25.100.100
```

See Also

- Condition: `exception.id=`
- Properties: `allow, authenticate.force( ),deny.unauthorized( ),force_deny( ), never_refresh_before_expiry( ),never_serve_after_expiry( ), remove_IMS_from_GET( ),remove_PNC_from_GET( ),remove_reload_from_IE_GET( ), socks.authenticate( ),socks.authenticate.force( ), exception.format( )`
- *ProxySG Log Fields and CPL Substitutions Reference*.

deny.unauthorized( )

The `deny.unauthorized` property instructs the ProxySG 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( )` 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* is a string defining a message to be displayed to the user. The details string may contain CPL substitution variables.

Discussion

If current policy contains rules that use the `authenticate()` or `authenticate.force( )` 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

- Use in `<Proxy>` layers.
- Applies to HTTP transactions. For other protocols, the property is the equivalent to `deny( )`.

See Also

- Conditions: `exception.id=`
- Properties: `deny( )`, `exception( )`, `force_deny( )`, `force_exception( )`, `exception.format( )`
- *ProxySG Log Fields and CPL Substitutions Reference*

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(all|none)
detect_protocol(protocol_list)
detect_protocol.protocol(yes|no)
detect_protocol[protocol_list](yes|no)
```

where:

- *protocol_list* is a comma separated list of *protocol*
- *protocol* is one of sip, sips, ms-turn, http, bittorrent, edonkey, fasttrack, gnutella, epmapper, https, or SSL.

The default value is `all`.

Layer and Transaction Notes

- Use in `<Proxy>` and `<SSL-Intercept>` layers.
- Applies to SOCKS, HTTP, TCP Tunnel and SSL Intercept transactions.

Example

```
<Proxy>
  detect_protocol(gnutella)
```

See Also

- Properties: `force_protocol( )`

direct( )

Used to prevent requests from being forwarded to a parent proxy or SOCKS server, when the ProxySG 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

- Use in `<Proxy>` and `<Cache>` layers.
- Does not apply to FTP over HTTP or transparent FTP transactions.

See Also

- Properties: `bypass_cache( ), dynamic_bypass( ), force_cache(), forward( ), reflect_ip( )`

dns.respond( )

Terminates a proxied DNS query with the given DNS RCODE.

Syntax

```
dns.respond(noerror|formerr|servfail|nxdomain|notimp|refused|yxdomain|yxrrset|nx
rrset|notauth|notzone|numeric range from 0 to 15)
```

Layer and Transaction Notes

- Use in `<DNS-Proxy>` layers.
- Applies to DNS proxy transactions.

Example

This example implements the following policy:

1. DNS queries using QTYPEs other than “PTR” or “A” are considered “not implemented.”
2. Any DNS query for a host ending in example.com is refused.

```
<DNS-Proxy>
 ; 1
 dns.request.type=!(A||PTR) dns.respond(notimp)
<DNS-Proxy>
 ; 2
 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

- Use in `<DNS-Proxy>` layers.
- Applies to DNS-proxy transactions.

Example

This example implements the following policies:

1. DNS queries for `host1.example.com` are resolved to `10.10.10.1` with a TTL of 7200 seconds.
2. 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>
  ; 1
  dns.request.name=host1.example.com dns.respond.a(10.10.10.1, 7200)
  ; 2
  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 IP-address) in the answer section of the response. You can also replace `[<ip-address>]*` with the `vip` 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])
```

Layer and Transaction Notes

- Use in `<DNS-Proxy>` layers.
- Applies to DNS proxy transactions.

Example(s)

These examples implement the following policies:

- The following policy specifies 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)
```

- The following policy specifies that 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)`. 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)
```

Note: The DNS Proxy caches IPv6 AAAA records.

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

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

- Use in `<DNS-Proxy>` layers.
- Applies to DNS-proxy transactions.

Example

This example implements the following policies:

1. Reverse DNS queries for `10.10.10.1` are resolved to `host1.example.com` with a TTL of 7200 seconds.
2. Reverse DNS queries for `10.10.10.2` are resolved to `host2.example.com` with the default TTL.

```
<DNS-Proxy>
 ; 1
 dns.request.address=10.10.10.1 dns.respond.ptr(host1.example.com, 7200)
 ; 2
 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 dynamic bypass methodology.

The `dynamic_bypass(yes)` property takes precedence over `authenticate()`; however, a committed denial takes precedence over `dynamic_bypass(yes)`.

Syntax

```
dynamic_bypass(yes|no)
```

The default value is `no`.

Layer and Transaction Notes

- Use in `<Proxy>` layers.
- Applies to transparent HTTP transactions only.

See Also

- Properties: `advertisement( ),always_verify( ),bypass_cache( ),cache( ), cookie_sensitive( ),delete_on_abandonment( ),direct( ),force_cache(),pipeline( ), refresh( ),ttl( ),ua_sensitive( )`

exception( )

Selects a built-in or user-defined response to be returned to the user.

The `exception( )` property is overridden by `allow` or `deny( )`. To set an exception that cannot be overridden by `allow`, use `force_exception( )`.

The identity of the exception being returned can be tested in an `<Exception>` layer using `exception.id=`.

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`*—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`*—A text string that is substituted for `$(exception.details)` within the selected exception.
- *`string_name`*—A string name, as defined by `define string`, that is substituted for `$(exception.format)`. The named string overrides the format field of the exception. The string can contain substitutions.

Layer and Transaction Notes

- Use in `<Cache>`, `<Proxy>`, `<SSL>`, and `<Admin>` layers.
- Applies to all transactions.

Example

```
<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=`
- Properties: `allow`, `deny( )`, `deny.unauthorized( )`, `exception.autopad( )`, `force_deny( )`, `force_exception( )`, `exception.format( )`
- Definitions: `string`
- *ProxySG Log Fields and CPL Substitutions Reference*

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)
```

where:

- `yes`—Enables auto-padding.
- `no`—Disables auto-padding.

The default value is `yes`.

Layer and Transaction Notes

- Use in `<Exception>` layers only.
- Applies to HTTP transactions.

See Also

- Conditions: `exception.id=`
- Properties: `exception(),force_exception(). exception.format()`

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)`—is the name of the CPL string defined using the define string <string_name> syntax.
- `default`—specifies to use the standard exception format specified in the ProxySG appliance configuration.

The default value is `default`.

Layer and Transaction Notes

- Use in `<Cache>, <Proxy>, <SSL>, and <Admin>` layers only.
- Applies to all transactions.

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(), exception(), exception.autopad(), force_deny(), force_exception()`
- `Definitions: string`

force_cache( )

Specify one or more reasons for forcing the caching of HTTP responses that would otherwise be 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)`, and `ua_sensitive(no)`. For streaming proxies, the value of the `force_cache( )` property is ignored unless `bypass_cache(no)` is set.

The `force_cache( )` property is typically used in conjunction with conditions. See Example 3 below.

Syntax

```
force_cache.<reason>(yes|no) <-adds/removes one reason
force_cache[reason1,reason2,...](yes|no) <-adds/removes multiple reasons
force_cache(reason1,reason2,...) <-resets reasons
force_cache(all|no) <-forces caching for all supported reasons or clears all
```

By default, no responses are forcibly cached. The default caching behavior for HTTP or streaming protocols is restored using `force_cache(no)`.

The `<reason>` can be specified using any of the following strings. The HTTP proxy supports all nine reasons, while streaming proxies have limited support, as indicated below.

`response-no-cache`: 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`: Refers to 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.

`private`: Refers to the `Cache-Control: private` response header/meta tag. Supported on Windows Media over RTSP or HTTP, but not on RealMedia over RTSP or HTTP.

`expired`: The HTTP response has `Expires` header and its value is in the past. Not supported on streaming proxies.

`set-cookie`: Includes both `Set-Cookie` and `Set-Cookie2` response headers. Not supported on streaming proxies.

`vary`: Refers to the `vary` response header. Not supported on streaming proxies.

`unknown-transfer-encoding`: The `Transfer-Encoding` response header value is unknown. Not supported on streaming proxies.

`missing-http-version`: The first line of the HTTP response does not contain "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 the `force_cache(personal-pages)` property can override it. Not supported on streaming proxies.

Layer and Transaction Notes

- Use only in `<Cache>` layer.
- Applies to proxy transactions, which execute both `<Cache>` and `<Proxy>` layers.

Examples

Example 1

```
; force caching for set-cookie reason for all responses
<Cache>
    force_cache.set-cookie(yes)
```

Example 2

```
; force caching for both expired & private reasons for all responses
<Cache>
     force_cache[expired, private](yes)
```

Example 3

This example incorporates conditions: unique conditions produce different actions—the cache is forced for different reasons. Note that if both conditions are met, both force_cache reasons will still be on.

```
; force caching for expired & private reason 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

The `force_cache(reason1,reason2,...)` syntax (without a `yes|no`) resets the reasons.

```
<Cache>
   force_cache(set-cookie) <=== resets the reason to set-cookie only here
<Cache>
```

```
force_cache(private)    <=== resets the reason to private only here
```

At the end, only the private reason is on (the last one specified). Because this syntax overwrites any previous reason that was specified, it could prevent you from adding other policy that you want to run independently to trigger caching for different reasons. Use this syntax with care.

See Also

- Properties: `advertisement( ),always_verify( ),bypass_cache( ),cache( ), cookie_sensitive( ),dynamic_bypass( ),pipeline( ),refresh( ),ttl( ), ua_sensitive( )`

force_deny( )

The `force_deny( )` property is similar to `deny( )` except that it:

- Cannot be overridden by an allow.
- Overrides any pending termination (that is, if a `deny( )` has already been matched, and a `force_deny` or `force_exception` 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` is a 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

- Use in `<Cache>`, `<Proxy>`, `<SSL>`, and `<Admin>` layers.
- Applies to all transactions.

See Also

- Conditions: `exception.id=`
- Properties: `deny()`, `force_exception()`
- *ProxySG Log Fields and CPL Substitutions Reference*

force_exception( )

The `force_exception( )` property is similar to exception except that it:

- Cannot be overridden by an `allow`.
- Overrides any pending termination (that is, if a `deny( )` has already been matched, and a `force_deny( )` or `force_exception( )` 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`*—A text string that is substituted for `$(exception.details)` within the selected exception.
- *`string_name`*—A string name, as defined by `define string`, that is substituted for `$(exception.format)`. The named string overrides the format field of the exception. The string can contain substitutions.

Layer and Transaction Notes

- Use in `<Cache>`, `<Proxy>`, `<SSL>`, and `<Admin>` layers.
- Applies to all transactions.

See Also

- Conditions: `exception.id=`
- Properties: `deny( )`, `exception( )`, `exception.autopad( )`, `force_deny( )`, `exception.format( )`
- *ProxySG Log Fields and CPL Substitutions Reference*

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|sip|sips|
ms-turn)
```

The default value is **no**.

Layer and Transaction Notes

- Use in `<Proxy>` and `<SSL-Intercept>` layers.
- Applies to SOCKS, HTTP, TCP Tunnel and SSL Intercept transactions.

Example

```
<Proxy>
  force_protocol(gnutella)
```

See Also

- Properties: `detect_protocol( )`

forward( )

Determines forwarding behavior.

There is a global configuration setting (`config>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 (`config>forwarding>failure-mode`) for the default forward failure mode. The `forward.fail_open( )` 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 ProxySG 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`—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.

The default value is `default`, as the only token in the *`alias_list`*.

Layer and Transaction Notes

- Use only in `<Forward>` layers.
- Applies to all transactions except administrator, instant messaging, and SOCKS.

See Also

- Properties: `direct( )`, `dynamic_bypass( )`, `reflect_ip( )`, `refresh( )`, `socks_gateway( )`, `socks_gateway.fail_open( )`, `streaming.transport( )`

forward.fail_open( )

Controls whether the ProxySG 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 (`config>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`—Terminate the request if the specified forwarding host or any designated backup or default cannot be contacted.

The default value is `no`.

Layer and Transaction Notes

- Use only in `<Forward>` layers.
- Applies to all transactions except administrator, instant messaging, and SOCKS.

See Also

- Properties: `bypass_cache( ), dynamic_bypass( ), forward( ), reflect_ip( ), socks_gateway( ), socks_gateway.fail_open( )`

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

- Use in `<Proxy>` layers.
- Applies to FTP proxy transactions.

Example

```
<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

- Use in `<Proxy>` layers.
- Applies to FTP proxy transactions.

Example

```
<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`.

Layer and Transaction Notes

- Use in `<Proxy>` and `<Cache>`layers.
- Applies to FTP transactions.

See Also

- Properties: `ftp.server_data( )`, `ftp.transport( )`

ftp.server_data( )

Determines the type of data connection to be used with this FTP 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.

Note: The `port` and `pasv` arguments have been deprecated. If you install existing policy with these arguments, they will automatically get converted to `active` and `passive`.

Layer and Transaction Notes

- Use in `<Forward>` layers.
- Applies to FTP transactions.

See Also

- Properties: `ftp.server_connection( ), ftp.transport( )`

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)
```

The default value is `auto`.

where:

- `auto`—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

- Use in `<Forward>` layers.
- Applies only to WebFTP transactions where the client uses the HTTP protocol to request a URL with an `ftp:` schema.

See Also

- Properties: `ftp.server_connection( ),ftp.server_data( )`

ftp.welcome_banner( )

Sets the welcome banner for a proxied FTP transaction.

Syntax

```
ftp.welcome_banner(default | no | substitution-string)
```

where:

- `default` means use the setting on the ProxySG appliance
- `no` means do not return a welcome banner

Layer and Transaction Notes

- Use in `<Proxy>` layers.
- Applies to FTP proxy transactions.

Example

This example implements the following policies:

1. All requests from HR_subnet get the FTP welcome banner *"client's address*: Welcome to *this appliance"*
2. All requests from ENG_subnet get the default FTP welcome banner.
3. All other requests get no FTP welcome banner.

```
define subnet HR_subnet
 10.10.0.0/16
end

define subnet ENG_subnet
 10.9.0.0/16
end

<Proxy>
 ; 1
 client.address=HR_subnet \
  ftp.welcome_banner("$(client.address): Welcome to $(appliance.name)")
 ; 2
 client.address=ENG_subnet ftp.welcome_banner(default)
 ; 3
 ftp.welcome_banner(no)
```

See Also

- *ProxySG Log Fields and CPL Substitutions Reference*

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

- Use in `<Proxy>` and `<Cache>` layers.
- Applies to all HTTP transactions (proxy, refresh, pipeline)

Example

```
<Proxy>
  http.allow_compression(yes)
```

See Also

- Properties: `http.allow_decompression( )`

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

- Use in `<Proxy> and <Cache>` layers.
- Applies to all HTTP transactions (proxy, refresh, pipeline).

Example

```
<Proxy>
  http.allow_decompression(yes)
```

See Also

- Properties: `http.allow_compression( )`

http.client.allow_encoding( )

Determines which encodings are allowed in the response sent to the client.

Syntax

```
http.client.allow_encoding(encoding_or_client_list)
http.client.allow_encoding.encoding(yes|no)
http.client.allow_encoding[encoding_list](yes|no)
```

where:

- *encoding_or_client_list* is a comma separated list of *encoding* or *client_list*
- *encoding_list* is a comma separated list of *encoding*
- *encoding* is one of gzip, or deflate
- *client* is replaced by the list of encodings specified in the client's request

The default value is *client_list*.

Layer and Transaction Notes

- Use in <Proxy> and <Cache> layers.
- Applies to all HTTP transactions (proxy, refresh, pipeline).

Example

```
<Proxy>
  http.client.allow_encoding(gzip)
```

See Also

- Properties: http.server.accept_encoding( )

http.client.persistence( )

Controls persistence of the connection to the HTTP client.

If set to no, after the current transaction is complete, the client connection will be dropped.

Syntax

```
http.client.persistence(yes|no|preserve)
```

The `preserve` option reflects the server's persistence to the client connection. The default value is taken from HTTP configuration, which is "yes" by default.

Layer and Transaction Notes

- Use in `<Proxy>` and `<Exception>` layers.
- Applies to HTTP proxy transactions.

Example

This property allows control of the persistence of individual client connections based on any policy conditions available in the `<Proxy>` or `<Exception>` layers. The following example shows the property used to 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( )`

http.client.recv.timeout( )

Sets the socket timeout for receiving bytes from the client.

Syntax

```
http.client.recv.timeout(auto | recv-timeout)
```

Layer and Transaction Notes

- Use in `<Proxy>` layers.
- Applies to HTTP proxy transactions.

Example

This example implements the following policies:

1. Requests from HR_subnet get a receive timeout of 200 seconds.
2. Any request heading to a host that ends in `example.com` gets a receive timeout of 20 seconds.
3. 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>
 ; 1
 client.address=HR_subnet http.client.recv.timeout(200)
<Forward>
 ; 2
 server_url.domain=example.com http.server.recv.timeout(20) \
  http.refresh.recv.timeout(auto)
 ; 3
 http.refresh.recv.timeout(300)
```

http.compression_level( )

Determines the compression level used by HTTP Proxy when `http.allow_compression` is true.

Syntax

```
http.compression_level(low|medium|high)
```

The default value is low.

Layer and Transaction Notes

- Use in `<Proxy>` and `<Cache>` layers.
- Applies to all HTTP transactions (proxy, refresh, pipeline).

Example

```
<Proxy>
   http.compression_level(medium)
```

See Also

- Properties: `http.allow_compression( )`

http.force_ntlm_for_server_auth( )

`http.force_ntlm_for_server_auth( )` is a fine grained control of the global configuration that can be set or unset through CLI commands `http force-ntlm` or `http no force-ntlm`.

The `force_ntlm` commands are used to work around the Microsoft limitation that Internet Explorer does not allow origin content server (OCS) NTLM authentication through a ProxySG appliance when explicitly proxied.

To correct this problem, Blue Coat 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 `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 ProxySG 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 ProxySG.

Syntax

```
http.force_ntlm_for_server_auth(yes|no)
```

This property overrides the default specified in configuration.

where:

- `yes`—Allows Internet Explorer clients explicitly proxied through an appliance to participate in NTLM authentication.
- `no`—The `Proxy-Support: Session-based-authentication` header is used to respond to 401 authentication-type challenges.

Layer and Transaction Notes

- Use in `<Proxy>` layers.
- Applies to HTTP proxy transactions.

Example

This example implements the following policies:

- 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>
 ; 1
 client.address=HR_subnet http.force_ntlm_for_server_auth(no)
 ; 2
 url.domain=example.com http.force_ntlm_for_server_auth(yes)
 ; 3
 http.force_ntlm_for_server_auth(no)10.10.0.0/16
end
```

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)
```

Layer and Transaction Notes

- Use in `<Cache>` and `<Forward>` layers.
- Applies to HTTP refresh transactions.

Example

This example implements the following policies:

1. Requests from HR_subnet get a receive timeout of 200 seconds.
2. Any request heading to a host that ends in `example.com` gets a receive timeout of 20 seconds.
3. 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>
 ; 1
 client.address=HR_subnet http.client.recv.timeout(200)
<Forward>
 ; 2
 server_url.domain=example.com http.server.recv.timeout(20) \
  http.refresh.recv.timeout(auto)
; 3
 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.

Syntax

```
http.request.apparent_data_type.allow( <type>,...)
http.request.apparent_data_type.allow[ <type>,...](yes|no)
http.request.apparent_data_type.allow.<type>(yes|no)
```

Where <type> is an ADT in the supported list:
(BMP|BZ2|CAB|EXE|FLASH|GIF|GZIP|HTML|ICC|JPG|MSDOC|MRAR|MZIP|PDF|PNG|RAR|RTF|TAR|TIF|TTF|TXT|XML|ZIP)

Layer and Transaction Notes

- Use in `<Proxy>` layers.
- Does not involve ICAP services.
- Recommended for Reverse Proxy deployments, where files will be uploaded through the ProxySG to a back-end web server.
- Applies to HTTP POST requests.

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:

```
http.request.apparent_data_type=<type>
http.response.apparent_data_type=<type>
request.icap.apparent_data_type=<type>
response.icap.apparent_data_type=<type>
http.request.apparent_data_type.deny(<type>, …)
```

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.

Syntax

```
http.request.apparent_data_type.deny( <type>,...)
http.request.apparent_data_type.deny[ <type>,...](yes|no)
http.request.apparent_data_type.deny.<type>(yes|no)
```

Where <type> is a data type from the supported list:
(BMP|BZ2|CAB|EXE|FLASH|GIF|GZIP|HTML|ICC|JPG|MSDOC|MRAR|MZIP|PDF|PNG|RAR|RTF|TAR|TIF|TTF|TXT|XML|ZIP)

Layer and Transaction Notes

- Use in `<Proxy>` layers.
- Does not involve ICAP services.
- Recommended for Reverse Proxy deployments, where files are uploaded through the ProxySG to a back-end web server.
- Applies to HTTP POST requests.

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:

```
http.request.apparent_data_type=<type>
http.response.apparent_data_type=<type>
request.icap.apparent_data_type=<type>
response.icap.apparent_data_type=<type>
http.request.apparent_data_type.allow(<type>, …)
```

- ❒
-

http.request.body.max_size()

This CPL property 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)
```

In the above example, *N* specifies the maximum number of bytes in the HTTP request body.

Layer and Transaction Notes

- Use in <Cache> and <Proxy> layers.
- Applies to all HTTP requests that include body content.
- Deny occurs after the HTTP request body received by the .
- When the connection is terminated using this property, the silent_denied exception is called. The user receives no feedback from the ProxySG Appliance.

Examples

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.injection.sql()

(Introduced in SGOS 6.5.2) Enables and defines settings for SQL injection detection in HTTP requests. This inspects up to the first 8k of the query string, cookie, and body within URL-encoded and Multipart-Form encoded HTTP requests for the specified argument names and values. The search occurs after the names and values are normalized.

If a match is found, the `risk_score` increments by 10. Verbose matching details can be found in the policy transaction log when trace logging is enabled.

Syntax

`http.request.detection.injection.sql[<attribute>](yes|no)`

where `<attribute>` is one of the following:

- `query` - Name and value in query string; includes all unnamed values.
- `cookie` - Name and value in Cookie and Cookie2 headers.
- `body` - Inspects up to the first 8 kB of request body data after the body is decompressed, dechunked, and normalized. SQL injection detection is performed on the name and value pairs in the body, which is either URL-encoded and Multipart-form encoded.
  If no attributes are specified, all attributes are inspected.

Layer and Transaction Notes

- Use in `<Proxy>` layers.
- Applies to all HTTP transactions.

See Also

- `http.request.detection.other(), http.request=`

Example

```
; Enable SQL injection protection and inspect cookies and query string
<proxy>
http.request.detection.injection.sql[cookie,query](yes)

; The request is denied when one or more SQL injection match is found
<proxy>
risk_score=10.. deny
```

http.request.detection.other()

(Introduced in SGOS 6.5.2) 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.

Syntax

`http.request.detection.other.[attribute](yes|no)`

where:

- *`attribute`* is one of the following:
  - `null_byte` - Detects content that contains null bytes.
  - `invalid_form_data` - Detects invalid multipart/form-data evasion techniques.
  - `parameter_pollution` - Detects multiple instances of parameters with the same name.
- `multiple_encoding` - Detects multiple encoding requests.

Layer and Transaction Notes

- Use in `<proxy>` layer.
- Applies to all HTTP transactions.

Example

```
; Scan for null-byte injection attacks
<proxy>
http.request.detection.other.null_byte(yes)
; Scan for HTTP parameter pollution
<proxy>
http.request.detection.other.parameter_pollution(yes)
```

"Supported HTTP Attributes" on page 133

http.request.version( )

The `http.request.version( )` property 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)
```

With the `preserve` option, the 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 `http.request.version( )` and `http.response.version( )`.

Layer and Transaction Notes

- Use in `<Proxy>` and `<Cache>` layers.
- Applies to HTTP transactions.

See Also

- Conditions: `http.request.version=`
- Properties: `http.response.version( )`

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

- Use in `<Cache>` layers.
- Applies to HTTP proxy transactions, HTTP refresh transactions, and HTTP pipeline transactions.

Example

```
<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

- Use in `<Cache>` layers.
- Applies to HTTP proxy transactions, HTTP refresh transactions, and HTTP pipeline transactions.

Example

```
<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

- Use in `<Cache>` layers.
- Applies to HTTP proxy transactions, HTTP refresh transactions, and HTTP pipeline transactions

Example

```
<Proxy>
  http.response.parse_meta_tag.pragma-no-cache(yes)
```

http.response.version( )

The `http.response.version( )` property 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)
```

With the `preserve` option, the 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 `http.request.version( )` and `http.response.version( )`.

Layer and Transaction Notes

- Use in `<Proxy>` layers.
- Applies to HTTP transactions.

See Also

- Conditions: `http.response.version=`
- Properties: `http.request.version( )`

http.server.accept_encoding( )

Determines which encodings are allowed in an upstream request.

Syntax

```
http.server.accept_encoding(all)
http.server.accept_encoding(encoding_or_client_list)
http.server.accept_encoding.encoding(yes|no)
http.server.accept_encoding[encoding_list](yes|no)
```

where:

- *encoding_or_client_list* is a comma separated list of *encoding* or *client*
- *encoding_list* is a comma separated list of *encoding*
- *encoding* is one of gzip, deflate or identity
- all represents all encodings supported by the client or by the ProxySG appliance (currently gzip, deflate and identity)
- client will be replaced by the list of encodings specified in the client's request

The default value for requests from a client is *client*. For client-less transactions, the default with a valid compression license is all, otherwise identity.

Layer and Transaction Notes

- Use in <Proxy> and <Cache> layers.
- Applies to All HTTP transactions (proxy, refresh, pipeline).

Example

This example illustrates how the property is used to determine the accepted encodings. The conditions used are assumed to be defined elsewhere).

```
<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( ), http.server.accept_encoding.allow_unknown( )

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

- Use in `<Proxy>` and `<Cache>` layers.
- Applies to All HTTP transactions (proxy, refresh, pipeline).

Example

This example allows only encodings supported by the ProxySG appliance.

```
<Proxy>
 http.server.accept_encoding(all) http.server.accept_encoding.allow_unknown(no)
```

See Also

- Properties: `http.server.accept_encoding( )`

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 from 1 to 10)
```

Layer and Transaction Notes

- Use in `<Forward>` layers.
- Applies to all HTTP transactions (proxy, refresh, pipeline).

Example

```
<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` indicates the default connection timeout, and `N` is a number from 10 to 120 that specifes the number of seconds to wait before the connection times out.

Layer and Transaction Notes

- Use in `<Proxy>`, `<Forward>`, and `<SSL-Intercept>` layers.
- Applies to HTTP and HTTPS transactions.

Example

This 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(), http.server.recv.timeout()`

http.server.persistence( )

Controls persistence of the connection to the HTTP server.

Syntax

```
http.server.persistence(yes|no)
```

The `preserve` option reflects the client's persistence to the server connection. The default value is taken from HTTP configuration, which is "`yes`" by default.

Layer and Transaction Notes

- Use in `<Proxy>` and `<Cache>` layers.
- Applies to all 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( )`

http.server.recv.timeout( )

Sets the socket timeout for receiving bytes from the upstream host.

Syntax

```
http.server.recv.timeout(auto | recv-timeout)
```

Layer and Transaction Notes

- Use in `<Proxy>` and `<Forward>` layers.
- Applies to HTTP proxy transactions, HTTP pipeline transactions.

Example

This example implements the following policies:

1. Requests from HR_subnet get a receive timeout of 200 seconds
2. Any request heading to a host that ends in example.com gets a receive timeout of 20 seconds
3. 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>
 ; 1
 client.address=HR_subnet http.client.recv.timeout(200)
<Forward>
 ; 2
 server_url.domain=example.com http.server.recv.timeout(20) \
  http.refresh.recv.timeout(auto)
 ; 3
 http.refresh.recv.timeout(300)
```

im.tunnel()

Determines whether IM traffic will be tunneled.

Syntax

```
im.tunnel(yes|no)
```

where:

- *yes*—transaction associated with connection will be tunneled.
- *no* (default value)—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

- Valid layers: Proxy
- Applies to: Instant messaging transactions

Example(s)

Sample usage:

```
<Proxy>
im.tunnel(yes)
```

Notes

- The `unsupported_version` value was added in SG 5.x.

integrate_new_hosts( )

Determines whether to add new host addresses to health checks and load balancing.

Syntax

```
integrate_new_hosts(yes|no)
```

The default is `no`. If it is set to `yes`, any new host addresses encountered during DNS resolution of forwarding hosts are added to health checks and load balancing.

Layer and Transaction

- Use in `<Forward>` layers.
- Applies to everything but SOCKS and administrator transactions.

See Also

- Properties: `forward( )`

log.rewrite.*field-id*( )

The `log.rewrite.field-id` property 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 the each log.

Syntax

```
log.rewrite.field-id("substitution"|no)
log.rewrite.field-id[log_name_list]("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.
- *log_name_list*—A comma separated list of configured access logs, of the form:
- *log_name_1, log_name_2, ...*
- *substitution*—A 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*`[log_name_list]( )` 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

- <Cache>, <Exception>, <Forward>, <Proxy>
- Applies to all proxy transactions.

See Also

- Properties: `access_log( ),log.suppress.field-id()`
- *ProxySG Log Fields and CPL Substitutions Reference*

log.suppress.*field-id*( )

The `log.suppress.field-id( )` property 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[log_name_list](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.

- *`log_name_list`*—A comma separated list of configured access logs, of the form:
- *`log_name_1, log_name_2, ...`*
- `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

- <Cache>, <Exception>, <Forward>, <Proxy>
- Applies to all proxy transactions.

See Also

- Properties: `access_log( ),log.rewrite.field-id( )`

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)
```

The default value is `no`.

where:

- *`bitrate`*—Maximum bit rate allowed. Specify using an integer, in bits, kilobits (1000x), or megabits (1,000,000x), as follows: *`integer | integerk | integerm`*.
- `no`—Allows any bitrate.

Layer and Transaction Notes

- Use in `<Proxy>` and `<Cache>` layers.
- Applies to streaming transactions.

Example

```
; Client bit rate for streaming media cannot exceed 56 kilobits.
max_bitrate(56k)
```

See Also

- Conditions: `bitrate=`, `live=`, `streaming.content=`

never_refresh_before_expiry( )

The `never_refresh_before_expiry( )` property is similar to the CLI command:

```
SGOS#(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

- Use in `<Cache>` layers.
- Applies to proxy transactions.

See Also

- Properties: `never_serve_after_expiry( )`, `remove_IMS_from_GET( )`, `remove_PNC_from_GET( )`, `remove_reload_from_IE_GET( )`
- *Command Line Interface Reference* for information on the `http strict-expiration` command.

never_serve_after_expiry( )

The `never_serve_after_expiry( )` property is similar to the CLI command:

```
SGOS#(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

- Use in `<Cache>` layers.
- Applies to proxy transactions.

See Also

- Properties: `always_verify(),never_refresh_before_expiry()`
- *Command Line Interface Reference* provides information on the `http strict-expiration` command.

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(address1@domain.com, address2@domain.com, ..)
```

Layer and Transaction Notes

- Specifies the addresses to which the appliance will send email when a `notify_email` object is matched in policy.
- Can be referenced by rules in any layer.
- 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.

Example

```
<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(), notify_snmp()`
- *ProxySG Log Fields and CPL Substitutions Reference*

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

- Use in `<Cache>` layers.
- Applies to HTTP proxy transactions.

reference_id( )

(Introduced in SGOS 6.5.5.7) 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`*`)`

where *`policy_ID`* is the ID you specify for the rule.

Layer and Transaction Notes

- Use in `<Admin>` , `<Cache>`, `<Proxy>`, `<SSL>`, or `<SSL-Intercept>` layers.

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 ProxySG appliance must be in the routing path for the `reflect_ip` property to work properly.

Syntax

```
reflect_ip(auto|no|client|vip|ip_address)
```

The default value is `auto`.

where:

- `auto`—Might reflect the client IP address, based on a config setting.
- `no`—The appliance's IP address is used to originate upstream connections.
- `client`—The client's IP address is used in initiating upstream connections.
- `vip`—The appliance's VIP on which the client request arrived is used to originate upstream traffic.
- *`ip_address`*—A 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

- Use in `<Proxy>`, `<DNS-proxy>`, and `<Forward>` layers.
- Applies to 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( )`

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

- Use in `<Cache>` layers.

See Also

- Properties: `advertisement( ),always_verify( ),bypass_cache( ),cache( ), cookie_sensitive( ),direct( ),force_cache( ),never_refresh_before_expiry( ), Never_serve_after_expiry( ),ttl( ),ua_sensitive( )`

remove_IMS_from_GET( )

The `remove_IMS_from_GET( )` property is similar to the CLI command:

```
SGOS#(config) http substitute if-modified-since
```

except that it provides per transaction control to allow overriding the box-wide default set by the command.

Syntax

```
remove_IMS_from_GET(yes|no)
```

The default value is taken from configuration.

Layer and Transaction Notes

- Use in `<Cache>` layers.
- Applies to HTTP proxy transactions.

See Also

- Properties: `never_refresh_before_expiry( ),never_serve_after_expiry( ), remove_PNC_from_GET( ),remove_reload_from_IE_GET( )`
- The *Command Line Interface Reference* provides information on the `http substitute` command.

remove_PNC_from_GET( )

The `remove_PNC_from_GET` property is similar to the CLI command:

```
SGOS#(config) http substitute pragma-no-cache
```

except that it provides per transaction control to allow overriding the box-wide default set by the command.

Syntax

```
remove_PNC_from_GET(yes|no)
```

The default value is taken from configuration.

Layer and Transaction Notes

- Use in `<Cache>` layers.
- Applies to HTTP proxy transactions.

See Also

- Properties: `never_refresh_before_expiry()`, `never_serve_after_expiry( )`, `remove_IMS_from_GET( )`, `remove_reload_from_IE_GET( )`
- The *Command Line Interface Reference* provides information on the `http substitute` command.

remove_reload_from_IE_GET( )

The `remove_reload_from_IE_GET( )` property is similar to the CLI command:

```
SGOS#(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

- Use in `<Cache>` layers.
- Applies to HTTP proxy transactions.

See Also

- Properties: `never_refresh_before_expiry( ),never_serve_after_expiry( ), remove_IMS_from_GET( ),remove_PNC_from_GET( )`
- The *Command Line Interface Reference* provides information on the `http substitute` command.

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( servicename_1, [servicename_2, ...,] [fail_open|fail_closed]
)
```

```
request.icap_service( no )
```

Where:

- *servicename*–A configured ICAP service or service group that supports request modification.
- `fail_open`—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.

The default value is **fail_closed**.

Layer and Transaction Notes

- Valid layers: Cache, Proxy
- Applies to: HTTP proxy transactions, FTP proxy transactions

Example

This example implements the following Intellectual Property protection policy using IP scanners:

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( )`

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.servicename_1( yes | no | auto )
request.icap_service.secure_connection.[servicename_1, servicename_2, ...]( yes | no | auto )
```

Where:

- *servicename*—A configured ICAP service that supports request modification.
- *yes*—ICAP service(s) use secure connection
- *no*—ICAP service(s) use plain (nonsecure) connection
- *auto*—-—ICAP service(s) use global default connection setting

The default value is **auto**.

Layer and Transaction Notes

- Valid layers: Cache, Proxy
- Applies to: HTTP proxy transactions, FTP proxy transactions
-

Example(s)

This example implements the following policy:

- 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:** `icap_error_code=`
- **Properties:** `response.icap_service.secure_connection( )`

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.

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:

- `no` prevents any bytes from being delivered to the client until scanning completes. This option has good security characteristics, but the worst user experience.
- `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.
- `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.
- *`patience_delay`* is the 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_delay* is the 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.

The default value is `no`.

Layer and Transaction Notes

- Valid layers: Proxy
- Applies to: HTTP proxy transactions, FTP proxy transactions

Example

The following example assumes 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:** `response.icap_service( )`, `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 ProxySG 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 value is `no`.

Layer and Transaction Notes

- Valid layers: Proxy
- Applies to: HTTP proxy transactions

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_service( ), response.icap_feedback( ), response.icap_feedback.interactive( ), response.icap_feedback.non_interactive( )`

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:

- `no` prevents any bytes from being delivered to the client until scanning completes. This option has good security characteristics, but the worst user experience.
- `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.
- `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.
- *`patience_delay`* is the number of seconds before a patience page is delivered to the client. The allowed range is 5 to 65535. The default is 5.
- *`trickle_delay`* is the 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.

The default value is `no`.

Layer and Transaction Notes

- Valid layers: Proxy

- Applies to: 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_service( ), response.icap_feedback( ), 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:

- `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.
- `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`* is the 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.

Layer and Transaction Notes

- Valid layers: Proxy
- Applies to: HTTP proxy transactions

Example

The following example uses authentication and group membership to 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_service( )`, `response.icap_feedback( )`, `response.icap_feedback.interactive( )`, `response.icap_feedback.force_interactive( )`

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

- Use in `<proxy>` layers.
- Applies to HTTP transactions only.
- Requires that a `response.icap_service()` rule exists in a `<cache>` layer.
- Data handled by this action is not cached.
- In cases where communication between the ProxySG 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:** `response.icap_service( )`

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`* - A 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` - 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.

The default value is `fail_closed`.

Layer and Transaction Notes

- Valid layers: Cache
- Applies to: All HTTP transactions (proxy, refresh, pipeline), FTP proxy transactions

Example

This example implements the following Virus Scanning policy using ICAP response scanners:

- 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( )`

response.icap_service.force_rescan( )

(Introduced in version 6.5.9.2) 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` - The ICAP service scans cached objects every time they are requested.
- `no` - The ICAP service rescans cached objects only when the ICAP server's ISTAG has changed since the last scan. This is the default behavior.

Layer and Transaction Notes

- Valid layers: Proxy
- Applies to: All HTTP transactions (proxy, refresh, pipeline), FTP proxy transactions The internal Content Analysis service can be invoked with this policy action with the object name, `bluecoat-local-response` as the ICAP service name.

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

- **Actions:** `set()`

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.

Syntax

```
response.icap_service.secure_connection( yes | no | auto )
response.icap_service.secure_connection.servicename_1( yes | no | auto )
response.icap_service.secure_connection.[servicename_1, servicename_2, ...]( yes | no | auto )
```

Where:

- *servicename*—A configured ICAP service that supports response modification.
- *yes*—ICAP service(s) use secure connection
- *no*—ICAP service(s) use plain (nonsecure) connection
- *auto*—ICAP service(s) use global default connection setting

The default value is `auto`.

Layer and Transaction Notes

- Valid layers: Cache
- Applies to: HTTP proxy transactions, FTP proxy transactions

Example

This example implements the following policy:

- 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.

```
<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:** `icap_error_code= Properties: request.icap_service.secure_connection( )`

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.

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 ProxySG 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

- Use in `<Proxy>` and `<Cache>` layers.
- Applies to HTTP proxy transactions.

Example

```
<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 ProxySG 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

- Use in `<Proxy>` and `<Cache>` layers.
- Applies to HTTP proxy transactions.

Example

```
<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|invalid_status)
```

where:

- `none`—indicates that no deviations are tolerated
- `continue`—indicates that the response header parsing should tolerate a continuation line (white space) prior to the start of the first header
- `invalid_header`—(Added in) indicates that 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`—(Added in)indicates that the response header parsing should tolerate invalid status.

The default value is none.

Layer and Transaction Notes

- Use in `<Proxy>` and `<Cache>` layers.
- Applies to all 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)
  ; For all other servers, use strict response header parsing rules
  ; 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( ), response.raw_headers.max_length( )`

risk_score.maximum()

(Introduced in SGOS 6.5.2) 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` is the 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

- Use in `<Proxy>` layers.
- Applies to all HTTP transactions.

See Also

- `risk_score.other()`

Example

```
; Discontinue scan requests when two anomalies are detected
; Assume a default risk-score of 10
<proxy>
  risk_score_maximum(20)
```

risk_score.other()

(Introduced in SGOS 6.5.2) 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>* is one of the following attack types:
  - `null_byte`
  - `invalid_form_data`
  - `parameter_pollution`
  - `multiple_encoding`
- *risk_score_value* is the custom risk score you want to specify

Layer and Transaction Notes

- Use in `<Proxy>` layers.
- Applies to all HTTP transactions.

See Also

- `risk_score.maximum()`

Example

```
; Set the risk score for HPP attacks to 30
<proxy>
  risk_score.other.parameter_pollution(30)
```

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 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)
```

OR

```
server.authenticate.basic(origin[,username_substitution [,password_substitution ]])
```

OR

```
server.authenticate.basic(proxy[,username_substitution [,password_substitution ] ])
```

where:

- `no`—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`*—an optional substitution string which will be sent as the username instead of using the authenticated username.
- *`password_substitution`*—an optional substitution string which will be sent as the password instead of using the authenticated password.

The default value is `no`.

Layer and Transaction Notes

- Use in the `<Proxy>` layer.
- Applies to all HTTP proxy transactions.

Examples

- Authenticate to an upstream server using the user's BASIC credentials:

  ```
  <proxy>
  url.host.exact="webmail.company.com" server.authenticate.basic(origin)
  ```

- 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)")
```

- Authenticate to an upstream proxy using a fixed username and password:

```
<proxy>
url.host.exact="proxy.company.com" server.authenticate.basic(proxy,
"internaluser", "internalpassword")
```

- 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.

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, then Kerberos constrained delegation 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)
```

OR

```
server.authenticate.constrained_delegation(origin, iwa_realm)
```

OR

```
server.authenticate.constrained_delegation(proxy, iwa_realm)
```

where:

- `no`—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`*—is the name of a configured IWA realm that is used to acquire the Kerberos tickets.

The default value is `no`.

Layer and Transaction Notes

- Use in the `<Proxy>` layer.
- Applies to all HTTP proxy transactions.

Examples

- Authenticate to an upstream server with Kerberos Constrained Delegation:

```
<proxy>
url.host.exact="images.company.com"
server.authenticate.constrained_delegation(origin, iwa_realm_1)
```

- Authenticate to an upstream server with Kerberos Constrained Delegation:

```
<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`* (if non-standard)].

If the service principal name is different, then it can be explicitly set using the following property.

delegation 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`—the service principal name to use for Kerberos constrained delegation.

The default value is generated from the hostname of the upstream server.

Layer and Transaction Notes

- Use in the `<Proxy>` layer.
- Applies to all HTTP proxy transactions.

Examples

- 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.compa
ny.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, `server.certificate.validate()` 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, Blue Coat recommends that you do not disable certificate validation. If youmust do so, disable it only for specific, trusted URLs, for example, using the `url=` condition. Including `server.certificate.validate(no)` in policy disables all certificate validation forthe affected transactions, including checks for the validity of the certificate (such as trust chainand 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

- Use in `<SSL>` layers.
- Applies to HTTPS forward and reverse proxy transactions, SSL tunnel transactions.

Example

```
<ssl>
  url.domain=”example.com” server.certificate.validate(no)
```

See Also

- Properties: `server.certificate.validate.ignore( )`

server.certificate.validate.check_revocation()

Check SSL server certificates for revocation.

Syntax

```
server.certificate.validate.check_revocation(auto|ocsp|local|no)
```

where:

- `auto`—the certificate will be checked through OCSP if available, otherwise it will be checked against locally installed revocation list
- `ocsp`—checks the certificate through OCSP
- `local`—checks the certificate against the locally installed revocation list
- `no`—the certificate will not be checked for revocation

The default value is `auto`.

Layer and Transaction Notes

- Valid layers: SSL.
- Applies to HTTPS forward and reverse proxy transactions, SSL tunnel transactions.

Example

```
<SSL>
  server.certificate.validate.check_revocation(local)
```

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)        ; form 1
server.certificate.validate.ignore[flag, ...](yes|no)  ; form 2
server.certificate.validate.ignore(all|none)           ; form 3
server.certificate.validate.ignore(flag, ...)          ; form 4
```

where `flag` is one of expiration, untrusted_issuer, or hostname_mismatch

The default value is `none`.

Layer and Transaction Notes

- Use in `<SSL>` layers.
- Applies to: 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( )`

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

- Use in `<SSL>` and `<Proxy>` layers.

Example(s)

- 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>)
  ```

- 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>)
  ```

- 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>)
  ```

- 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)")
  ```

- 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* is `0..63 | af11 | af12 | af13 | af21 | af22 | af23 | af31 | af32 | af33 | af41 | af42 | af43 | best-effort | cs1 | cs2 | cs3 | cs4 | cs5 | cs6 | cs7 | ef | echo | preserve`

The special value `preserve` means to track the incoming DSCP value on the primary server connection and use that as the value when sending packets on the client connections. The special value `echo` means the outbound packet's DSCP value will use the same value as the inbound packet's DSCP value.

The default value is `preserve`.

Layer and Transaction Notes

- Valid in `<Proxy>`, `<DNS-Proxy>`, `<Cache>`, and `<Forward>` layers.
- Applies to: All transactions

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_url.dns_lookup()

Sets the global policy for IP connection type preference.

Syntax

`server_url.dns_lookup(`*`dns_lookup_value`*`)`

where dns_lookup_value is one of the following:

- IPv4-Only (this is the default)
- IPv6-Only
- Prefer-IPv4
- Prefer-IPv6

Layer and Transaction Notes

- Valid layers: Proxy, Forward
- Applies to: Transactions connecting upstream to a proxy or origin server (but not DNS Proxy transactions)

Example

This policy rule specifies that 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

- Use in `<Proxy>` layers.
- Applies to shell (Telnet) proxy transactions.

Example

This example implements the following policies:

1. All requests from HR_subnet get the Shell prompt "*client's address*: Welcome to *this appliance*."
2. All requests from ENG_subnet get the default Shell prompt.
3. 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>
 ; 1
 client.address=HR_subnet \
  shell.prompt("$(client.address): Welcome to $(appliance.name)")
 ; 2
 client.address=ENG_subnet shell.prompt(default)
 ; 3
 shell.prompt(no)
```

See Also

- *ProxySG Log Fields and CPL Substitutions Reference*

shell.realm_banner( )

Sets the realm banner for a proxied shell transaction.

Syntax

```
shell.realm_banner(substitution-string)
```

Layer and Transaction Notes

- Use in <Proxy> layers.
- Applies to shell (Telnet) proxy transactions.

Example

This example implements the following policies:

1. All requests from HR_subnet get the Shell realm banner *"client's address*: Welcome to *this appliance*."
2. All requests from ENG_subnet get the default Shell realm banner.
3. 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>
 ; 1
 client.address=HR_subnet \
  shell.realm_banner("$(client.address): Welcome to $(appliance.name)")
 ; 2
 client.address=ENG_subnet shell.realm_banner(default)
 ; 3
 shell.realm_banner(no)
```

See Also

- *ProxySG Log Fields and CPL Substitutions Reference*

shell.welcome_banner( )

Sets the welcome banner for a proxied shell transaction.

Syntax

```
shell.welcome_banner(substitution-string)
```

Layer and Transaction Notes

- Use in `<Proxy>` layers.
- Applies to shell (Telnet) proxy transactions.

Example

This example implements the following policies:

1. All requests from HR_subnet get the Shell welcome banner "*client's address*: Welcome to *this appliance*."
2. All requests from ENG_subnet get the default Shell welcome banner.
3. 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>
 ; 1
 client.address=HR_subnet \
  shell.welcome_banner("$(client.address): Welcome to $(appliance.name)")
 ; 2
 client.address=ENG_subnet shell.welcome_banner(default)
; 3
 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)
```

The default value is `auto`.

where:

- `no`—The SOCKS proxy does not hand off the transaction to another proxy agent, but tunnels the SOCKS transaction.
- `auto`—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.

The `socks.accelerated=` condition can be used to test which agent was selected for handoff. The `tunneled=` condition 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, the `socks=` condition can be used to test for transactions use SOCKS for client communication.

Layer and Transaction Notes

- Use in `<Proxy>` layers.
- Applies to SOCKS proxy transactions.

See Also

- Properties: `socks_gateway( )`, `socks.authenticate( )`, `socks.authenticate.force( )`
- Conditions: `socks=`, `socks.accelerated=`, `socks.method=`, `socks.tunneled=`, `socks.version=`

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.

The regular `authenticate( )` property 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( )` 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 `user=`, `group=`, and `realm=` 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)`.

Syntax

```
socks.authenticate(realmname)
```

where:

- *realmname*—One of the already-configured realms.
- Consider that `socks.authenticate()` depends exclusively on a limited number of triggers:
  - `proxy.address=`
  - `proxy.card=`
  - `proxy.port=`
  - `client.address=`
  - `socks.version=`

Date and time triggers, while available, are not recommended.

Layer and Transaction Notes

- Use in `<Proxy>` layers.
- Applies to SOCKS proxy transactions.

See Also

- Properties: `authenticate()`, `socks_gateway()`, `socks.accelerate()`, `socks.authenticate.force()`
- Conditions: `socks=`, `socks.method=`, `socks.tunneled=`, `socks.version=`

socks.authenticate.force( )

This property controls the relation between SOCKS authentication and denial.

Syntax

```
socks.authenticate.force(yes|no)
```

The default value is `no`.

where:

- `yes`—Makes `socks.authenticate( )` higher priority than `deny( )` or `exception( )`. Use `yes` to ensure that user ID's are available for access logging, even of denied requests.
- `no`—`deny( )` and `exception( )` have a higher priority than `socks.authenticate( )`. 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.

Note: This does not affect regular `authenticate( )`.

Layer and Transaction Notes

- Use in `<Proxy>` layers.
- Applies to SOCKS proxy transactions.

See Also

- Properties: `socks.authenticate( )`, `socks_gateway( )`, `socks.accelerate( )`
- Conditions: `socks.method=`, `socks.tunneled=`, `socks.version=`

socks_gateway( )

Controls whether or not the request associated with the current transaction is sent through a SOCKS gateway.

There is a box-wide configuration setting (`config > socks-gateways > 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 box-wide configuration setting (`config> socks-gateways > failure-mode`) for the default SOCKS gateway failure mode. The `socks_gateway.fail_open( )` property overrides the configured default.

Syntax

`socks_gateway(`*`alias_list`*`|no)`

The default value is `no`.

where:

- *`alias_list`*—Send this request through the specified alias list. The ProxySG 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`—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

- Use in `<Forward>` layers.
- Applies to all except administrator transactions.

See Also

- Properties: `direct( )`, `forward( )`, `socks.accelerate( )`, `socks.authenticate( )`, `socks.authenticate.force( )`
- Conditions: `socks.method=`, `socks.tunneled=`, `socks.version=`

socks_gateway.fail_open( )

Controls whether the ProxySG 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 box-wide configuration setting (`config > socks-gateways > failure-mode`) for the default SOCKS gateway failure mode. The `socks_gateway.fail_open( )` property overrides the configured default.

Syntax

```
socks_gateway.fail_open(yes|no)
```

The default value is `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`—Terminates the request if the specified SOCKS gateway or any designated backup or default cannot be contacted.

Layer and Transaction Notes

- Use in `<Forward>` layers.
- Applies to all except administrator transactions.

See Also

- Properties: `socks.accelerate( )`, `socks.authenticate( )`, `socks.authenticate.force( )`, `socks_gateway( )`
- Conditions: `socks.method=`, `socks.tunneled=`, `socks.version=`

ssl.forward_proxy( )

Determines whether SSL connections should be intercepted.

The default value is `https`.

Syntax

```
ssl.forward_proxy(no|yes|sips|stunnel|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
- `sips` recognizes Secure Session Initial Protocol (SIPS)
- `stunnel` tunnels intercepted SSL traffic; if secure ADN is enabled, traffic is accelerated via byte caching
- `https` intercepts SSL connections using the HTTPS Forward Proxy

The above elements are also governed by an optional secondary argument that determines when or if interception is to take place.

- `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` will only intercept SSL traffic if policy execution results in an exception, (typically a deny). Otherwise, the SSL traffic will be tunneled.

Layer and Transaction Notes

- Use in `<SSL-Intercept>` layers.
- Applies to SSL Intercept transactions.

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.

The default value is "". The default behavior is to use the hostname from the original server certificate.

Syntax

```
ssl.forward_proxy.hostname(String)
```

Layer and Transaction Notes

- Use in `<SSL-Intercept>` layers.
- Applies to SSL Intercept transactions.

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.

The default value is auto. The default behavior is to use the keyring specified in configuration by the `SGOS#(config ssl) intercept` CLI command.

Syntax

```
ssl.forward_proxy.issuer_keyring (auto|KeyringId|hsm-keyring())
```

Layer and Transaction Notes

- Use in `<SSL-Intercept>` layers.
- Applies to SSL Intercept transactions.

Examples

A. 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)
```

B. Use the following policy to set the SSL Proxy to use the HSM keyring `test-hsmkeyring1`:

```
ssl.forward_proxy.issuer_keyring(test-hsmkeyring1)
```

ssl.forward_proxy.preserve_untrusted

When an OCS presents a certificate to the ProxySG 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. A warning message is displayed to the user, and they can decide to ignore the warning and visit the Website or cancel the request.

The default value is `auto`.

Syntax

```
ssl.forward_proxy.preserve_untrusted(auto|yes|no)
```

where:

- `auto` - Uses the "preserve-untrusted" configuration setting on the appliance to determine whether untrusted certificate issuer should be preserved for a connection. This is the default.
- `yes` - Preserve untrusted certificate issuer is enabled for the connection.
- `no` - Preserve untrusted certificate issuer is disabled for the connection.

Layer and Transaction Notes

- Use in `<SSL-Intercept>` layers.
- Applies to SSL Intercept transactions.

Example

```
<ssl-intercept>
  ssl.forward_proxy.preserve_untrusted(auto)
```

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|KeyringId)
```

Layer and Transaction Notes

- Use in `<SSL-Intercept>` layers.
- Applies to SSL Intercept transactions.

Example

```
<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. .

The default value is "". The string argument is limited to 200 printable characters.

Syntax

```
ssl.forward_proxy.splash_text(String)
```

Layer and Transaction Notes

- Use in `<SSL-Intercept>` layers.
- Applies to SSL Intercept transactions.

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.

The default value is "".

Syntax

```
ssl.forward_proxy.splash_url(""|Url)
```

Layer and Transaction Notes

- Use in `<SSL-Intercept>` layers.
- Applies to SSL Intercept transactions.

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 WM Client.

Effectively, setting `yes` enables the WM 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 WM Client to stream the content at the bitrate speed of the content.

Syntax

```
streaming.fast_cache(yes|no)
```

Default value is enabled (`yes`)

where:

- `yes`—SG will advertise the com.microsoft.wm.fastcache token in the Supported header for non-local responses returned to a WM Client if that token is included in the response from the streaming server. For local responses, the SG will advertise the com.microsoft.wm.fastcache token in the Supported header.
- `no`—SG will not advertise the com.microsoft.wm.fastcache token in the Supported header for both local and non-local responses returned to a WM Client.

Layer and Transaction Notes

- Use in `<Proxy>` layers.
- Applies to Proxy transactions.

Example

This property allows the SG to disable fast-caching on the WM Client. This property is applicable only in the `<Proxy>` layer. The following example shows the property used to 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

- Use in `<Proxy>` layer.
- Applies to Flash transactions.

Example

```
<Proxy>
    url.address=10.12.13.14 streaming.rtmp.tunnel_encrypted(yes)
```

In this example, RTMPE or RTMPTE connections to 10.12.13.14 will be tunneled (not accelerated).

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.

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

- Use in `<Forward>` layers.
- Applies to streaming transactions.
- 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.

See Also

- Conditions: `bitrate=`, `live=`, `streaming.client=`, `streaming.content=`

terminate_connection( )

The `terminate_connection( )` property is used in an `<Exception>` layer to 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)
```

The default is `no`.

Layer and Transaction Notes

- Use in `<Exception>` layers.
- Applies to HTTP transactions.

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* is, by default, `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

- Use in any layer.
- Applies to all transactions.

Example

```
; Change directory location of trace output file to
; https://appliance_IP_address:8081/Policy/Trace/test/default_trace.html
trace.destination(test/)

; Change trace output file location to
; https://appliance_IP_address:8081/Policy/Trace/test/phase_2.html
trace.destination(test/phase_2.html)
```

See Also

- Properties: `trace.request()`
- *ProxySG Log Fields and CPL Substitutions Reference*

trace.header( )

Specifies whether unlimited, full header trace output is generated for the current request. The default value is `no`, which limits header trace output to 2k. Header data beyond 2k is truncated.

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( )` property.

Syntax

```
trace.header(yes|no)
```

The default value is `no`.

Layer and Transaction Notes

- Use in any layer.
- Applies to all transactions.

Example

```
; Generate full header trace details when a specific URL is requested.
url=//www.example.com/help trace.header(yes)
```

See Also

- Properties: `trace.destination(),trace.request()`

trace.request( )

Determines whether detailed trace output is generated for the current request. The default value is `no`, which produces no output. 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=` or `client.address=`.

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( )` property.

Note: Tracing is best used temporarily, such as for troubleshooting; the `log_message( )` action is best for on-going monitoring.

Syntax

`trace.request(yes|no)`

The default value is `no`.

Layer and Transaction Notes

- Use in any layer.
- Applies to all transactions.

Example

```
; Generate trace details when a specific URL is requested.
url=//www.example.com/confidential trace.request(yes)
```

See Also

- Properties: `trace.destination()`

transform.data_type()

(Introduced in SGOS 6.5.4.1) This policy gesture 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 url_rewrite`, `define active_content`,.

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.

`url_rewrite` is different, as it is not restricted to a specific file type. Rather, this transformation type can work in two ways:

- When a `url_rewrite` 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.
- `url_rewrite` 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()` 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

- Use in `<Proxy>` layers.
- Applies to HTTP response data affected by a transform 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 policy will not modify the content.

Example

```
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

- Actions: `transform`
- Conditions: `.`*`header_name`*`=`, `.`*`header_name`*`.address=`, `request.x_header.`*`header_name`*`=`, `request.x_header.`*`header_name`*`.address=`, `response.header.`*`header_name`*`=`, `response.x_header.`*`header_name`*`=`, `server_url=`
- Definitions: `javascript, active_content, url_rewrite`

trust_destination_ip()

This property allows the ProxySG 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 (i.e. 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.

By default it is configured as enabled.

Syntax

```
trust_destination_ip(yes|no)
```

The default value is taken from a global configuration setting.

Layer and Transaction Notes

- Valid layers: Proxy
- Applies to: Proxy transactions

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 ProxySG appliance's freshness algorithm determines the time-to-live value.

Note: `advertisement(yes)` overrides any `ttl( )` value.

Syntax

`ttl(`*`seconds`*`)`

where *`seconds`* is 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

- Use in `<Cache>` layers.

Example

```
; Delete the specified cached objects after 30 seconds.
url=//www.example.com/dyn_images ttl(30)
```

See Also

- Properties: `advertisement( ),cache( )`

ua_sensitive( )

Used to modify 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. Set to `yes` to specify this behavior.

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 value is `no`.

Layer and Transaction Notes

- Use in `<Cache>` layers.
- Applies to proxy transactions, which execute both `<Cache>` and `<Proxy>` layers. Does not apply to FTP over HTTP transactions.

See Also

- Properties: `advertisement( ),always_verify( ),bypass_cache( ),cache( ), cookie_sensitive( ),delete_on_abandonment( ),direct( ),dynamic_bypass( ), force_cache( ),pipeline( ),refresh( ),ttl( )`

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 value is `yes`.

Layer and Transaction Notes

- Valid layers: Proxy, Admin
- Applies to: Proxy transactions, Administrative transactions

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 value is `yes`.

Layer and Transaction Notes

- Valid layers: Proxy, Admin
- Applies to: Proxy transactions, Administrative transactions

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)
```

webpulse.categorize.mode( )

Determines how dynamic categorization will be performed.

Syntax

```
webpulse.categorize.mode(none|realtime|background|default)
```

where:

- `none`: suppresses dynamic categorization for this request
- `realtime`: performs dynamic categorization in real-time; the request waits until the dynamic category is available from the service
- `background`: performs 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`: restores the setting to the configuration-specified default (to undo the effect of a previous policy layer)

The default value is set via configuration.

Layer and Transaction Notes

- Use in `<Cache>` and `<Exception>` layers.
- Applies to all transactions.

Example

This example illustrates how the property is used to control dynamic categorization.

Do not dynamically categorize this domain

```
<Cache>
  url.domain=bluecoat.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_list)
webpulse.categorize.send_headers.header_name(yes|no)
webpulse.categorize.send_headers[header_name_list](yes|no)
```

where:

- `header_name_list`: is a comma separated list of header_names
- `header_name`: is a header name (e.g. Referer, User-agent)
- `auto`: determines that property will follow `send-request-info` configuration value

The default value is set via configuration. If `send-request-info` is enabled, send all headers. If `send-request-info` is disabled, send no headers

Layer and Transaction Notes

- Valid layers: Cache, Exception
- Applies to: All transactions

Example

This example illustrates how the property is used to control headers sent to WebPulse

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 via configuration. If `send-request-info` is enabled, default is `full`. If `send-request-info` is disabled, default is `path`.

Layer and Transaction Notes

- Valid layers: Cache, Exception
- Applies to: All transactions

Example

This example illustrates how the property is used to control information submitted to WebPulse

Send only hostname to WebPulse for this domain

```
<Cache>
  url.domain=bluecoat.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 ProxySG 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 BCWF 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`: (the default) sends a notification to WebPulse
- `no`: disables malware notification to WebPulse

Layer and Transaction Notes

- Valid layers: Cache, Exception
- Applies to: All transactions subject to URL categorization

Example

To disable malware notification to WebPulse:

```
<Cache>
    webpulse.notify.malware(no)
```

Chapter 5: *Action Reference*

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.

Topics in this Chapter

This chapter includes information about the following topics:

- "Argument Syntax" on page 445
- "Action Reference" on page 445

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. For example: `log_message("Access alert").`
- Enumeration—Actions such as `delete( )` 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 Appendix D: "Using Regular Expressions".
- 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( )`, `rewrite( )`, and `rewrite( )` actions, and the string argument in the `append( )`, `log_message( )`, 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, see the *ProxySG Log Fields and CPL Reference*.

Action Reference

The remainder of this chapter lists the actions and their accepted values. It also provides the context in which each action can be used and examples of how to use them.

append( )

Appends a new component to the specified header.

Syntax

```
append(header, string)
where:
```

- *header*—A header specified using the following form. For a list of recognized headers, including headers that support field repetition,
  - .*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*—A quoted string that can optionally include one or more variable substitutions.

Layer and Transaction Notes

- Use from <Proxy> or <Cache> layers.

See Also

- Actions: delete( ), delete_matching( ), rewrite(header, regex_pattern, replacement_component), set(header, string)
- Conditions: .header_name=, .header_name.address=, request.x_header.header_name=, request.x_header.header_name.address=, response.header.header_name=, response.x_header.header_name=
- *ProxySG Log Fields and CPL Substitutions Reference*

delete( )

Deletes all components of the specified header.

Syntax

```
delete(header)
```

where:

- *header*—A header specified using the following form. For a list of recognized headers, see Appendix C: "Recognized HTTP Headers".
  - 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

- Use with exception.*response.header.header_name* in <Proxy> or <Exception> layers.
- Use with request or response headers in <Proxy> or <Cache> layers.
- Applies to HTTP transactions.

Example

```
; For the test.com domai
; 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

- Actions: `append( )`, `delete_matching( )`, `rewrite(`*`header`*`, `*`regex_pattern`*`, `*`replacement_component`*`)`, `set(`*`header`*`, `*`string`*`)`
- Conditions: `.`*`header_name`*`=`, `.`*`header_name`*`.address=`, `request.x_header.`*`header_name`*`=`, `request.x_header.`*`header_name`*`.address=`, `response.header.`*`header_name`*`=`, `response.x_header.`*`header_name`*`=`

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`*—A header specified using the following form. For a list of recognized headers, see Appendix C: "Recognized HTTP Headers".
  - `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`*—A quoted regular-expression pattern. For more information, see Appendix D: "Using Regular Expressions".

Layer and Transaction Notes

- Use in `<proxy>` and `<cache>` layers only.

See Also

- Actions: `append( )`, `delete( )`, `rewrite(`*`header`*`, `*`regex_pattern`*`, `*`replacement_component`*`)`, `set(`*`header`*`, `*`string`*`)`
- Conditions: `.`*`header_name`*`=`, `.`*`header_name`*`.address=`, `request.x_header.`*`header_name`*`=`, `request.x_header.`*`header_name`*`.address=`, `response.header.`*`header_name`*`=`, `response.x_header.`*`header_name`*`=`

diagnostic.stop(pcap)

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.

Syntax

```
diagnostic.stop(pcap)
```

Layer and Transaction Notes

- Use in all layer types.
- Start a packet capture with your desired filters, direction, and interface:
  - in the **Management Console** > **Maintenance** > **Service Information** > **Packet Captures**
  - The CLI with `pcap start`.
  - From the **Management Console** > **Statistics** > **Advanced** > **Packet Capture** > **Start Packet Capture.**
- 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>:8082/PCAP/bluecoat.cap
- There is no VPM command for this action.

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)
...
end
```

In the example above, `header` indicates a header that the ProxySG appliance recognizes, and which uses the specified form.

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

- "iterator=" on page 149
- "iterator.append( )" on page 452
- "iterator.delete( )" on page 453
- "log.rewrite.field-id( )" on page 365

iterator.append( )

Appends a new `iterator` value to the HTML header.

Syntax

`iterator.append(`*`string`*`)`

In the example above,

`string` indicates a quoted string that can optionally include one or more variable substitutions.

Layer and Transaction Notes

- Use from `<Proxy>` or `<Cache>` layers.
- 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. For example:

```
iterate(.Cookie)
      iterator.append("CookieName=CookieValue");
end
```

iterator.delete( )

Removes the `iterator` value from the HTML header.

Syntax

```
iterator.delete
```

Layer and Transaction Notes

- Use from `<Proxy>` or `<Cache>` layers.
- 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. For example:

```
iterate(.Cookie)
      iterator.delete();
end
```

iterator.rewrite( )

This action modifies cookie attributes inside the header being iterated over.

Syntax

```
iterator.rewrite(regex_pattern, replacement_string)
```

In the example above:

- `regex_pattern` indicates a 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` indicates a quoted string that includes one or more variable substitions. The string replaces the entire portion of the header that matches the `regex_pattern`. For more information, see the *ProxySG Log Fields and CPL Substitutions Reference*.

Layer and Transaction Notes

- Use from `<Proxy>` or `<Cache>` layers.

Example

The following example shows the `Secure` and `HttpOnly` attributes being added to a cookie header, and the expiration date of the cookie, if one exists, being set to midnight. Blue Coat 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

- "iterate()" on page 451

log_message( )

Writes the specified string to the ProxySG 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`* is a quoted string that can optionally include one or more variable substitutions.

Layer and Transaction Notes

- Can be referenced by any layer.

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

- Actions: `notify_email( ),notify_snmp( )`
- Properties: `access_log( ), log.rewrite( ), log.suppress( )`
- *ProxySG Log Fields and CPL Substitutions Reference*

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.

The e-mail is sent 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* and *body* are quoted strings that can optionally include one or more variable substitutions.

Layer and Transaction Notes

- Can be referenced by any layer except `<dns-proxy>`.

Example

```
define condition restricted_sites
url.domain=a_very_bad_site
...

end

<proxy>
condition=restricted_sites action.notify_restricted(yes)

define action notify_restricted
notify_email("restricted: ", \
               "$(client.address) accessed url: $(url)")
end
```

See Also

- Actions: `log_message( ),notify_snmp( )`
- *ProxySG Log Fields and CPL Substitutions Reference*

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`* is a quoted string that can optionally include one or more variable substitutions.

Layer and Transaction Notes

- Can be referenced by any layer.

See Also

- Actions: `log_message( ),notify_email( )`
- *ProxySG Log Fields and CPL Substitutions Reference*

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 3*xx* 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.

Important: 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`*—An HTTP redirect code used as the HTTP response code; supported codes are 301, 302, 305, and 307.
- *`regex_pattern`*—A 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 D: "Using Regular Expressions".
- `redirect_location`—A 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 the *ProxySG Log Fields and CPL Substitution Reference*.

Layer and Transaction Notes

- Use in `<Proxy>` or `<Cache>` layers.

See Also

- Actions: `rewrite(url.host, host_regex_pattern, replacement_host),rewrite(url, regex_pattern, redirect_location),set(url.port, port_number)`
- Conditions: `exception.id=`
- *ProxySG Log Fields and CPL Substitutions Reference*

request_redirect( )

Only use the `request_redirect` 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). Continue to use `redirect( )` for redirects to an OCS.

The following is an example of the typical policy that is known to cause the exception when other policy (such as ICAP scanning) is also involved:

```
<Proxy>
   ALLOW url.path.exact=/wpad.dat action.ReturnRedirect1(yes)
define action ReturnRedirect1
   redirect( 302, ".*", "http://proxy.company.com/accelerated_pac_base.pac" )
end
```

With this new gesture, rewrite the above policy as:

```
<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
```

Use of the `request_redirect` gesture prevents the **Request could not be handled** exception when a redirect is necessary in combination with policy that requires a response from an upstream server.

syntax

```
request_redirect (response_code, regex_pattern, redirect_location)
```

where:

- *`response_code`*—An HTTP redirect code used as the HTTP response code; supported codes are 301, 302, 305, and 307.
- *`regex_pattern`*—A 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 D: "Using Regular Expressions".
- `redirect_location`—A 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 the *ProxySG Log Fields and CPL Substitutions Reference*.

Layer and Transaction Notes

- Use in `<Proxy>` or `<Cache>` layers.

See Also

- Actions: `rewrite(url.host, host_regex_pattern, replacement_host),rewrite(url, regex_pattern, redirect_location),set(url.port, port_number)`
- Conditions: `exception.id=`
- *ProxySG Log Fields and CPL Substitutions Reference*

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 the `rewrite( )` 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

When planning to use the `rewrite( )` action for HTTPS traffic, the following points should be considered:

- To perform host rewrites for HTTPS requests, `rewrite(url.host)` rules need to be created in an `<SSL-Intercept>` layer.
- HTTPS interception requires that your ProxySG 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 SG 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, ...])
rewrite(url.host, regex_pattern, replacement_host[, URL_form1, ...])
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 Appendix C: "Recognized HTTP Headers".
  - `.`*`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*—A 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 Appendix D: "Using Regular Expressions".
- *redirect_location*—A 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, see the *ProxySG Log Fields and CPL Substitutions Reference*.
- *replacement_host*—A 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, see the *ProxySG Log Fields and CPL Substitution Reference*.
- *URL_form1, ...*—An optional 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*—A 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, see the *ProxySG Log Fields and CPL Substitutions Reference*.

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= 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 in `<Proxy>`, `<SSL-Intercept>` and `<Cache>` layers.
- URL and host rewrites apply to all transactions. Header rewrites apply to HTTP transactions.

Example

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
```

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

- Actions: `append( ), delete( ), delete_matching( ), redirect( ), set( ), transform`
- Conditions: `.`*`header_name`*`=, .`*`header_name`*`.address=, request.x_header.`*`header_name`*`=, request.x_header.`*`header_name`*`.address=, response.header.`*`header_name`*`=, response.x_header.`*`header_name`*`=, server_url=`
- Definitions: `transform url_rewrite`
- *ProxySG Log Fields and CPL Substitutions Reference*.

set( )

Sets the specified header to the specified string after deleting all components of the header.

HTTPS Limitation

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_form2, ...])
```

where:

- `header`—A header specified using the following form. For a list of recognized headers, see Appendix C: "Recognized HTTP Headers".
  - `.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`—A quoted string that can optionally include one or more variable substitutions, which replaces the specified header components once the substitutions are performed.
- `port_number`—The port number that the request URL is set to. The range is an integer between 1 and 65535.

- *URL_form1, URL_form2, ...*—An optional 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=` 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 in an `<SSL-Intercept>` layer, only set (url.port) may be used.
- When used with headers, applies to HTTP transactions.
- 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

- Actions: `append( )`, `delete( )`, `delete_matching( )`, `redirect( )`, `rewrite(url.host, regex_pattern, replacement_host)`, `rewrite(url, `*`regex_pattern`*`, redirect_location)`
- Conditions: `.`*`header_name`*`=`, `.`*`header_name`*`.address=`, `request.x_header.`*`header_name`*`=`, `request.x_header.`*`header_name`*`.address=`, `response.header.`*`header_name`*`=`, `response.x_header.`*`header_name`*`=`, `server_url=`
- *ProxySG Log Fields and CPL Substitutions Reference*.

transform( )

Invokes an `active_content`, javascript, or `URL_rewrite` transformer. The invoked transformer takes effect only if the `transform` action is used in a define action definition block, and that block is in turn enabled by an `action( )` 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`* is a user-defined identifier for a transformer definition block. This identifier is not case-sensitive.

Layer and Transaction Notes

- Use in `<Proxy>` or `<Cache>` layers.

Example

```
; 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( ), transform_data.type()`
- Definitions: `define action`, `transform active_content`, `transform url_rewrite`
-

Chapter 6: *Definition Reference*

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.

Topics in this Chapter

This chapter includes information about the following topics:

- "Definition Names" on page 471

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 chapter 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( )` 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 Chapter 5: "Action Reference" on page 445 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`*—A 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`*—A list of actions to be carried out in sequence. See Chapter 5: "Action Reference" on page 445 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

The following is a sample action given the name `scrub_private_info`, that clears the `From` and `Referer` headers (which normally could be used to identify the user and where they clicked from) in any request going to servers not in the internal domain.

```
<cache>
    url.domain=!my_internal_site.com action.scrub_private_info(yes)

define action scrub_private_info
    set( request.header.From, "" )
    set( request.header.Referer, "" )
end
```

Notice that the object on which the `set( )` 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( )`
- Definitions: `transform active_content`, `transform url_rewrite`

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` action in a `define action` definition block, and that block is in turn enabled an `action( )` 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.

Replaces: `transform active_content`

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`*—A user-defined identifier for a transformer definition block. Used to invoke the transformer using the `transform` action in a `define action` definition block.
- *`HTML_tag_name`*—The 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`*—A 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

- Applies to Proxy transactions.
- Only alphanumeric, underscore, dash, and slash characters can be used with the define action name.

Example

```
<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

- Actions: `transform`
- Definitions: `define action`, `define url_rewrite`
- Properties: `action( ), transform_data.type( )`
-

define category

Category definitions are used to extend vendor content categories or to create your own. The *category_name* definition can be used anywhere a content filter category name would normally be used, including in `category=` tests.

Definitions can include other definitions to create a hierarchy. For example, sports could include football by including `category=football` in the definition for sports. A defined category can have at most one parent category (multiple inheritance is not allowed).

Multiple definitions using the same *category_name* are coalesced together.

When policy tests a request URL to determine if it is in one of the categories specified by a trigger, all sub-categories are also checked (see Examples).

Syntax

```
define category category_name
    url_patterns
end
```

where:

- *category_name*—If *category_name* matches the name of an existing category from the configured content filtering service, this is used to extend the coverage of that category; otherwise it defines a new user defined category. *category_name* can be used anywhere a content filter category name would normally be used, including in `category=` tests.
- *url_patterns*—A list of URL patterns. A pattern can include the scheme, host, port, query string, and path components of the URL. If the pattern does not specify a component, the corresponding component of the URL is not tested and can have any value.

Layer and Transaction Notes

- Use in `<Proxy>` and `<Cache>` Layers.
- Applies to all transactions.

Examples

The following example illustrates some of the variations allowed in a category definition:

```
define category Grand_Canyon
    kaibab.org
    www2.nature.nps.gov/ard/parks/grca/
    nps.gov/grca/
    grandcanyon.org
end
```

The following definitions define the categories sports and football, and make football a sub-category of sports:

```
define category sports
    sports.com
    sportsworld.com
    category=football ; include subcategory
end

define category football
    nfl.com
    cfl.ca
end
```

The following policy needs only to refer to the sports category to also test the sub-category football:

```
<Proxy>
    deny category=sports ; includes subcategories
```

For more information on using `category=` tests, including examples, refer to the *SGOS Administration Guide*, Filtering Web Content chapter.

See Also

- Conditions: `category=`
- Properties: `action( )`

define condition

Binds a user-defined label to a set of conditions for use in a `condition=` 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`, `define url.domain condition`, and `define server_url.domain condition`.

Syntax

```
define condition label
    condition_expression ...

    ...
end
```

where:

- *`label`*—A user-defined identifier for a condition definition. Used to call the definition from an `action.action_label( )` 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.

The `condition=`*`condition`* 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=`, `condition=`
- Properties: `action.`*`action_label`*`( )`

define javascript

A javascript definition is used to 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*—A user-defined identifier for a transformer definition block. Used to invoke the transformer using the `transform` action in a `define action` definition block.
- A *javascript-statement* has the following syntax:

  javascript-statement ::= *section-type replacement*
  section-type ::= prolog | onload | epilog
  replacement ::= << *endmarker newline lines-of-text newline endmarker*

  This allows you to specify a block of javascript to be inserted at the beginning of the HTML page (prolog), to be inserted at the end of the HMTL page (epilog), and to be executed when parsing is complete and the page is loaded (onload). Each of the section types is optional.

Layer and Transaction Notes

- Applies to proxy transactions.

Example

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.

See Also

- Actions: `transform`
- Definitions: `define action`
- Properties: `action( ), transform_data.type( )`

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.

Syntax

```
define LayerType policy MacroName
Layer1
Layer2
...
end
```

For example, here is a policy macro of type *proxy*:

```
define proxy policy WebAccessPolicy
<proxy>
  DENY hour=9..17 category=NotBusinessRelated
  DENY category=IllegalOrOffensive
end
```

A policy macro is called from another layer using the syntax policy.*MacroName* within a rule. The calling layer must have the same type as the policy macro. For example:

```
<proxy> url.address=TheInternet
  group=Operator ALLOW
  group=Employee policy.WebAccessPolicy
  DENY
```

A policy macro call (policy.*MacroName*) 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.

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.

define server_url.domain condition

Binds a user-defined label to a set of domain-suffix patterns for use in a `condition=` expression. Using this definition block allows you to quickly test a large set of `server_url.domain=` conditions. 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.

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( )` 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`*—A user-defined identifier for a domain condition definition. Used in a `condition=` condition.
- *`domain_suffix_pattern`*—A URL pattern that includes a domain name (domain), as a minimum. See the `url=` condition reference for a complete description.
- *`condition_expression`* ...—An optional condition expression, using any of the conditions available in a rule, that are allowed in a `<Forward>` layer. For more information, see Chapter 3: "Condition Reference" on page 53.

The `condition=` condition 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 `server_url.domain` definition block can call another condition-related definition block, so that they are in effect *nested*. See the example in the `define condition` definition block topic. Any referenced condition must be valid in a `<Forward>` layer.

Layer and Transaction Notes

- Use in `<Forward>` layers.
- Applies to all transactions.

Example

```
define server_url.domain condition allowed
    inventory.example.com
    affinityclub.example.com
end

<Forward>
    condition=!allowed access_server(no)
```

See Also

Condition: `condition=`, `server_url.domain=`

Definitions: `define url.domain condition`

define string

Define a named, multi-line character string.

Syntax

```
define string StringName
>first line of text
>second line of text

;comments and blank lines ignored
>third line of text
end
```

Notes:

- 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.

Example

```
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)
```

The above CPL code returns 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.

See Also

Properties: exception( )

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=`, `proxy.address=`, `request.header.`*`header_name`*`.address=`, `request.x_header.`*`header_name`*`.address`, 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 | subnet } { ip_address | subnet } { ip_address_range}
    { ip_address_wildcards}...

end
```

where:

- *`label`*—A user-defined identifier for this subnet definition.
- *`ip_address`*—IP address; for example, `10.1.198.0`.
- *`subnet`*—Subnet specification; for example, `10.25.198.0/16`.
- *`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`

Example

```
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=`, `proxy.address=`, `request.header.`*`header_name`*`.address=`, `request.x_header.`*`header_name`*`.address`, and `server_url.address=`
- Information on wildcards: http://bluecoat.force.com/knowledgebase/articles/Solution/000010037
- Information on IP address ranges: http://bluecoat.force.com/knowledgebase/articles/Solution/000010950

define url condition

Binds a user-defined label to a set of URL prefix patterns for use in a `condition=` expression. Using this definition block allows you to quickly test a large set of `url=` 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=` 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.Please note that the `define url condition` definition block is not the same as the `url=` condition.

Syntax

```
define url condition label
    url_prefix_pattern [condition_expression ...]
    ...
end
```

where:

- *`label`*—A user-defined identifier for a prefix condition definition.
- *`url_prefix_pattern`* ... —A URL pattern that includes at least a portion of the following:

  `scheme://host:port/path`

  - *`scheme`*—A URL scheme (`http`, `https`, `ftp`, `mms`, or `rtsp`) followed by a colon (`:`).
  - *`host`*—A 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`*—A port number, between 1 and 65535.
  - *`path`*—A 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* ...—An optional condition expression, using any of the conditions available in a rule. For more information, see Chapter 3: "Condition Reference" on page 53. The layer and timing restrictions for the defined condition will depend on the layer and timing restrictions of the contained expressions.

The `condition=` condition 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 the `define condition` definition block topic.

Example

```
define url condition allowed
 http://www.inventory.example.com http.method=GET
 www.affinityclub.example.com/public ; any scheme allowed
end

<Proxy>
 condition=allowed allow
```

See Also

Conditions: `category=`, `condition=`, `url=`

Definitions: `define url.domain condition`

define url.domain condition

Binds a user-defined label to a set of domain-suffix patterns for use in a `condition=` 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 label
    domain_suffix_pattern [condition_expression ...]
    ...
end
```

where:

- *label*—A user-defined identifier for a domain condition definition. Used in a `condition=` condition.
- *domain_suffix_pattern*—A URL pattern suitable to the `url.domain=` condition, that includes a domain name (domain), as a minimum. See the `url=` condition reference for a complete description.
- *condition_expression* ...—An optional condition expression, using any of the conditions available in a rule. For more information, see Chapter 3: "Condition Reference" on page 53. The layer and timing restrictions for the defined condition will depend on the layer and timing restrictions of the contained expressions.

The `condition=` condition 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 the `define condition` definition block topic.

Layer and Transaction Notes

- Use in `<Proxy>`, `<Cache>`, `<ssl>`, `<ssl-intercept>` and `<Exception>` layers.
- Applies to all transactions.

Example

```
define url.domain condition allowed
    inventory.example.com method=GET
    affinityclub.example.com
end

<proxy>
    condition=allowed allow
```

See Also

- Condition: `condition=`, `server_url.domain=`
- Definitions: `define url condition`, `define server_url.domain condition`

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 a `define action` definition block, and that block is called from an `action( )` property. 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). See **How Multiple Rewrite Statements Affect Transformation**.

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"
  rewrite_statement "replacement" "match"
  ...
end
```

where:

- *`transformer_id`*—A 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"`*—A 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*.

Layer and Transaction Notes

- Applies to `<proxy>` transactions.

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://example" "http://example"
end
```

This policy makes the following transformations in a page containing `http://10.1.1.1` and `http://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://example`

The second `rewrite` statement does not transform the `http://example` resulting from the first `rewrite` statement.

Example

```
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`
- Definitions: `define action`, `define active_content`
- Properties: `action( )`, `transform_data.type( )`

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=`, `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
    restricted_domain_list
  except
    exempted_domain_list
end
```

where:

- *`restricted_domain_list`*—Domains for which DNS lookup is restricted.
- *`exempted_domain_list`*—Domains exempt from the DNS restriction. Policy is able to use DNS lookups when evaluating policy related to these domains.

Layer and Transaction Notes

- Applies to all layers and transactions.

Example

The following definition restricts DNS resolution to all but `mydomain.com`:

```
restrict dns
    . ; meaning "all"
  except
    mydomain.com
end
```

See Also

- Conditions: `category=`, `url=`, `server_url=`
- Definitions: `restrict rdns`

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 (introduced in SGOS 6.5.9.10).

If the requested URL specifies the host in IP form, no reverse DNS lookup is performed to match any `category=`, `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

- Applies to all layers and transactions.

Example

The following definition restricts 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=`, `url=`, `server_url=`
- Definitions: `restrict dns`
- Information on wildcards: http://bluecoat.force.com/knowledgebase/articles/Solution/000010037
- Information on IP address ranges: http://bluecoat.force.com/knowledgebase/articles/Solution/000010950

Appendix A: *Glossary*

actions	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 five 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 ProxySG 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 ProxySG appliance; allow permits full access to the appliance.
`<Cache>` layer	One of the five layer types allowed in a policy. Used to list policy rules that are evaluated during a cache or proxy transaction.
cache transaction	Encapsulation of a request, generated by the ProxySG 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.
`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 five 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 five 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 ProxySG 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. As a result, any following rules 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.
	Note: Using `OK` is a matter of preference; you can write conditions without explicit actions to achieve the same result. The following conditions have the same effect:
	`group=special OK`
	`group=special`

policy files	Any one of four files that contain CPL: Central, Local, VPM, or Forward. When the policy is installed, the contents of each of the files is concatenated according to the evaluation order.
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 five layer types allowed in a policy, used to list policy rules that control access to proxy services configured on the ProxySG. 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 ProxySG. 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.
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 ProxySG 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.
trigger	A named test of some aspect of a transaction. CPL triggers have the form *trigger_name=value*. Triggers are used in rules, and in condition definitions.
Visual Policy Manager file	A file created and stored on an individual ProxySG 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.

Appendix B: *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 ProxySG 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. See "Overview of Policy Tracing" on page 503.
- 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. See "Determining Which Policy Rules are Matched in Transactions" on page 508.

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( )` action is best for on-going monitoring. For more information about the `log_message( )` action, see "log_message( )" on page 455. 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( )`—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( )`—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 (A 'piece' of policy specifically means any place in CPL where a condition, layer, or filter is evaluated.) are matched. With this information, policy can be reordered for better performance or deleted if policy is not useful.

Enabling Request Tracing

Use the `trace.request( )` 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`—Enables tracing.

- `no`—Disables tracing.

Example

The following enables full tracing information for all transactions:

```
<cache>
  trace.request(yes)
```

Configuring the Path

Use the `trace.destination( )` 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* is a filename, directory path, or both. If you specify only a directory, the default trace filename is used.

You can view policy statistics through the Management Console: **Statistics > Advanced > Policy > List of policy URLs**.

Example

In the following 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 would be

```
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. The sample policy used is not intended as suitable for any particular purpose, other than to illustrate most aspects of policy trace output.

http://bluecoat.force.com/knowledgebase/articles/Solution/000030214Here are the relevant policy requirements to be expressed:

- 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.

The Sample Policy

```
; DNS lookups are restricted except for one site that is being hosted
  restrict dns
  .
  except
  my_site.com
  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()` 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`.

Note: 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()` is set to `yes`.
- Lines 3 to 4 and 10 to 11 show rule matches, and are included when `trace.request()` is set to `yes`.
- Lines 5 to 9 show rule misses, and are included when `trace.request()` 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(yes)`.
- 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 ProxySG 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 **View current policy coverage**). The Policy Coverage page opens.

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, search forBlue Coat Knowledge Base article 000009825 at https://bto.bluecoat.com/knowledgebase.

Appendix C: *Recognized HTTP Headers*

The tables provided in this appendix list all recognized HTTP 1.1 headers and indicate how the ProxySG 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( ),delete( ),rewrite( ),`or `set( )` actions can be used to change the header.

Recognized headers can be used with the `.header_name=` and `response.header.header_name=` conditions. Headers not shown in these tables must be tested with the `request.x_header.header_name=` and `response.x_header.header_name=` conditions. In addition, the following three header fields take address values, so they can be used with the condition `.header_name.address= Client-IP, Host, X-Forwarded-For.`

Blue Coat uses the ELFF *#Remark* directive to record the serial number and name of the appliance in an ELFF formatted access log.

Table C.1: HTTP Headers Recognized by the

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			

Table C.1: HTTP Headers Recognized by the (Continued)

Header Field	Request/Response Form	Allowed Actions		
		`rewrite( )` `set( )`	`append( )`	`delete( )`
Content-Location	Request/Response	X		X
Content-MD5	Request/Response			
Content-Range	Request/Response			
Content-Type	Request/Response	X		
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

Table C.1: HTTP Headers Recognized by the (Continued)

Header Field	Request/Response Form	Allowed Actions		
		`rewrite( )` `set( )`	`append( )`	`delete( )`
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
Warning	Request/Response	X	X	X
WWW-Authenticate	Response			

The following table lists custom headers that are recognized by the ProxySG.

Table C.2: Custom HTTP Headers Recognized by the ProxySG

Header Field	Request/Response Form	Allowed Actions
Authentication-Info	Response	`append( )`
Front-End-Https	Request/Response	`rewrite( ),set( ),delete( )`
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	`rewrite( ),set( ),delete( )`

Appendix D: *Using Regular Expressions*

Regular expressions can be used for complex pattern matching. The

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/.*$" you should write "url.domain=bluecoat.com".

The following Content Policy Language (CPL) conditions use regular-expression arguments:

- All triggers with the **`.regex`** qualifier (for example, `url.regex=`, `url.host.regex=`, `im.text.message.regex=`)
- Request and response header triggers (for example, `request.header.NAME=`, `request.x.header_NAME=`, `response.header.NAME=`, `response.x_header.NAME=`)

The following CPL actions include regular-expression arguments (refer to the *Blue Coat Systems Content Policy Language Guide* for more information):

- `delete_matching( )`
- `redirect( )`
- `rewrite( )`

The regular expression support in the ProxySG 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 The uses a Regular Expression Engine (RE ENGINE) to evaluate regular expressions.

This appendix 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 `courier` 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.

Table E.1: Metacharacters Used in Regular Expressions

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.
`$`	Matches the end of the string.
`*`	Causes the resulting RE to match zero (0) or more repetitions of the preceding RE, as many repetitions as are possible. ab* will match 'a', 'ab', or 'a' followed by any number of 'b's.
`+`	Causes the resulting RE to match one (1) or more repetitions of the preceding RE. ab+ will match 'a' followed by any non-zero number of 'b's; it will not match just 'a'.
`?`	Causes the resulting RE to match 0 or 1 repetitions of the preceding RE. ab? will match either 'a' or 'ab'.
`*?, +?, ??`	The *, +, and ? qualifiers are all greedy; they match as much text as possible. Sometimes this behavior isn't desired. If the RE /page1/.*/ is matched against /page1/heading/images/, it will match the entire string, and not just /page1/heading/. Adding ? after the qualifier makes it perform the match in non-greedy or minimal fashion; matching as few characters as possible. Using .*? in the previous expression will match only /page1/heading/.
`{m,n}`	Causes the resulting RE to match from m to n repetitions of the preceding RE, attempting to match as many repetitions as possible. For example, a{3,5} will match from 3 to 5 'a' characters.
`{m,n}?`	Causes the resulting RE to match from m to n repetitions of the preceding RE, attempting to match as few repetitions as possible. This is the non-greedy version of the previous qualifier. For example, on the 6-character string 'aaaaaa', a{3,5} will match 5 'a' characters, while a{3,5}? will only match 3 characters.
`\`	Either escapes special characters (permitting you to match characters like '*?+&$'), 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:

Table D.1: Metacharacters Used Outside Square Brackets

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:

Table D.2: Metacharacters Used in Square Brackets (Character Class)

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 ProxySG 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.