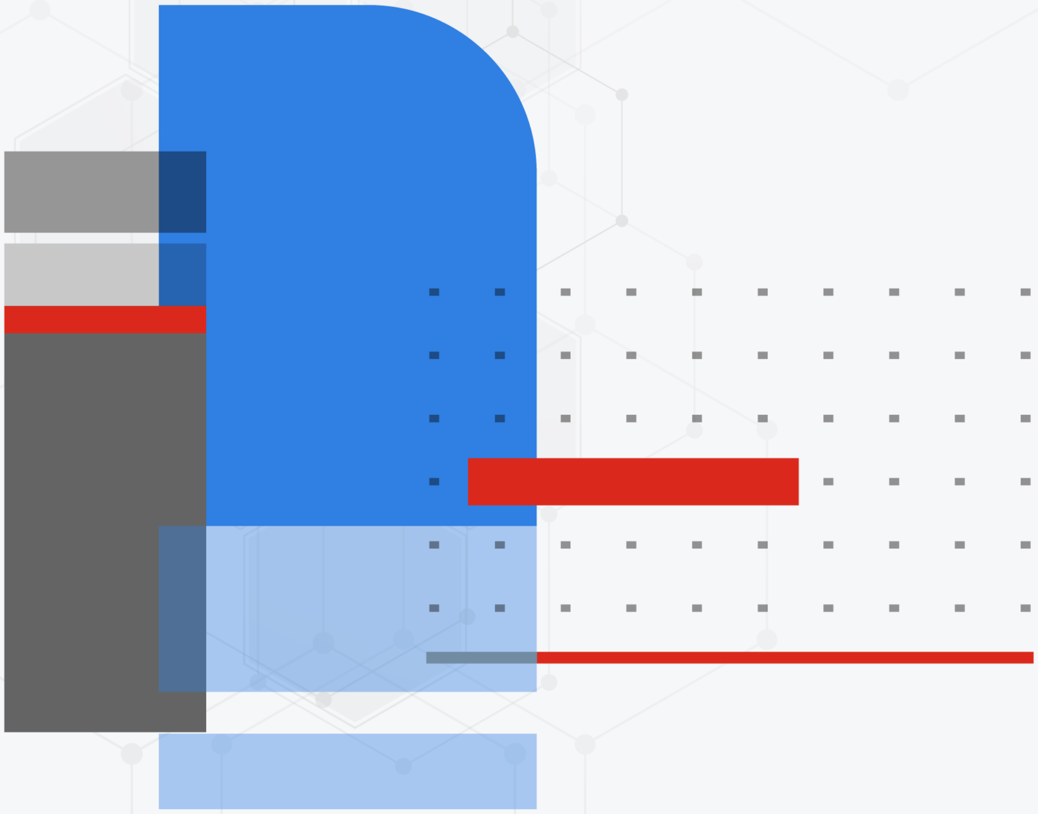
# Script Reference Guide

FortiADC 7.4.2



**FORTINET DOCUMENT LIBRARY**
https://docs.fortinet.com

**FORTINET VIDEO LIBRARY**
https://video.fortinet.com

**FORTINET BLOG**
https://blog.fortinet.com

**CUSTOMER SERVICE & SUPPORT**
https://support.fortinet.com

**FORTINET TRAINING & CERTIFICATION PROGRAM**
https://www.fortinet.com/training-certification

**FORTINET TRAINING INSTITUTE**
https://training.fortinet.com

**FORTIGUARD LABS**
https://www.fortiguard.com

**END USER LICENSE AGREEMENT**
https://www.fortinet.com/doc/legal/EULA.pdf

**FEEDBACK**
Email: techdoc@fortinet.com

February 7, 2024
FortiADC Script Reference Guide

# TABLE OF CONTENTS

**Change Log** .......... **8**
**Introduction** .......... **9**
**Definition of terms** .......... **10**
**Configuration overview** .......... **11**
**Script reference** .......... **12**
Control structures .......... 13
Operators .......... 14
String Library .......... 15
Multiple Scripting Usage .......... 16
Dynamic scripting configuration change .......... 17
Function .......... 18
**Events** .......... **20**
WAF events .......... 24
**Predefined commands** .......... **25**
Management commands .......... 26
MGM:get_session_id() .......... 26
MGM:rand_id() .......... 26
MGM:set_event(t) .......... 27
MGM:set_auto(t) .......... 27
IP commands .......... 29
IP:Client_addr() .......... 29
IP:server_addr() .......... 30
IP:local_addr() .......... 30
IP:remote_addr() .......... 31
IP:client_port() .......... 31
IP:server_port() .......... 31
IP:local_port() .......... 32
IP:remote_port() .......... 32
IP:client_ip_ver() .......... 33
IP:server_ip_ver() .......... 33
TCP commands .......... 34
TCP:reject() .......... 34
TCP:set_snat_ip(str) .......... 35
TCP:clear_snat_ip() .......... 35
TCP:sockopt(t) .......... 36
TCP:after_timer_set() .......... 38
TCP:after_timer_cancel() .......... 39
TCP:after_timer_get() .......... 40
TCP:close() .......... 41
HTTP commands .......... 43
HTTP:header_get_names() .......... 44
HTTP:header_get_values(header_name) .......... 45
HTTP:header_get_value(header_name) .......... 45

HTTP:header_remove(header_name) 46
HTTP:header_remove2(header_name, countid) 47
HTTP:header_insert(header_name, value) 47
HTTP:header_replace(header_name, value) 48
HTTP:header_replace2(header_name, value, countid) 48
HTTP:header_exists(header_name) 49
HTTP:header_count(header_name) 50
HTTP:method_get() 50
HTTP:method_set(value) 50
HTTP:path_get() 51
HTTP:path_set(value) 51
HTTP:uri_get() 52
HTTP:uri_set(value) 52
HTTP:query_get() 53
HTTP:query_set(value) 53
HTTP:redirect(“url”, …) 54
HTTP:redirect_with_cookie(url, cookie) 54
HTTP:redirect_t(t) 55
HTTP:version_get() 56
HTTP:version_set(value) 56
HTTP:status_code_get() 57
HTTP:status_code_set(value) 57
HTTP:code_get() 58
HTTP:code_set(integer) 58
HTTP:reason_get() 58
HTTP:reason_set(value) 59
HTTP:client_addr() 59
HTTP:local_addr() 60
HTTP:server_addr() 60
HTTP:remote_addr() 61
HTTP:client_port() 61
HTTP:local_port() 62
HTTP:remote_port() 63
HTTP:server_port() 63
HTTP:client_ip_ver() 64
HTTP:server_ip_ver() 64
HTTP:close() 65
HTTP:respond(t) 65
HTTP:get_session_id() 66
HTTP:rand_id() 67
HTTP:set_event(t) 67
HTTP:set_auto() 68
HTTP:get_unique_transaction_id() 69
HTTP:get_unique_session_id() 69
HTTP:get_unique_transaction_id() 70
HTTP DATA commands 71
HTTP:collect() 71
HTTP:payload(size) 72
HTTP:payload(content) 72

HTTP:payload(set) 73
HTTP:payload(find) 73
HTTP:payload(remove) 74
HTTP:payload(replace) 75
Cookie commands 76
HTTP:cookie_list() 76
HTTP:cookie(t) 76
HTTP:cookie_crypto(t) 77
Authentication commands 79
AUTH:get_baked_cookie() 79
AUTH:set_baked_cookie(cookie) 80
AUTH:on_off() 81
AUTH:success() 82
AUTH:form_based() 83
AUTH:user() 83
AUTH:pass() 84
AUTH:usergroup() 84
AUTH:realm() 84
AUTH:host() 85
AUTH:set_usergroup() 85
SSL commands 87
SSL:cipher() 87
SSL:version() 88
SSL:alg_keysize() 88
SSL:client_cert() 89
SSL:sni() 90
SSL:npn() 91
SSL:alpn() 91
SSL:session(t) 92
SSL:cert(t) 92
SSL:cert_der() 93
SSL:peer_cert(str) 94
GEO IP commands 96
ip2country_name(ip) 96
Ip2countryProv_name(ip) 96
RAM cache commands 98
HTTP:exclude_check_disable() 98
HTTP:cache_disable() 99
HTTP:dyn_check_disable() 99
HTTP:dyn_invalid_check_disable() 100
HTTP:dyn_cache_enable(t) 100
HTTP: cache_user_key(t) 101
HTTP:dyn_cache_invalid(t) 101
HTTP:cache_check(t) 102
HTTP:cache_hits(t) 102
HTTP:res_caching() 103
HTTP:res_cached() 103
HTTP:caching_drop() 104

PROXY commands ..... 105
PROXY:set_auth_key(value) ..... 105
PROXY:clear_auth_key(value) ..... 106
PROXY:shared_table_create() ..... 106
PROXY:shared_table_destroy() ..... 107
PROXY:shared_table_entry_count() ..... 108
PROXY:shared_table_memory_size() ..... 109
PROXY:shared_table_insert() ..... 110
PROXY:shared_table_lookup() ..... 111
PROXY:shared_table_delete() ..... 112
PROXY:shared_table_dump() ..... 113
LB commands ..... 115
LB:routing(value) ..... 115
LB:get_valid_routing() ..... 116
LB:get_current_routing() ..... 116
LB:method_assign_server() ..... 117
Persistence commands ..... 118
Global commands ..... 126
Crc32(str) ..... 128
Key_gen(str_pass, str_salt, iter_num, len_num) ..... 128
Aes_enc(t) ..... 129
Aes_dec(t) ..... 130
EVP_Digest(alg, str) ..... 131
HMAC(alg, str, key) ..... 131
HMAC_verify(alg, data, key, verify) ..... 132
G2F(alg, key) ..... 133
Class_match(str, method, list) ..... 133
Class_search(list, method, str) ..... 134
Cmp_addr() ..... 135
url_enc(str) ..... 136
url_dec(str) ..... 136
url_parser(str) ..... 137
url_compare(url1, url2) ..... 137
Rand() ..... 138
srand(str) ..... 139
Rand_hex(int) ..... 139
Rand_alphanum(int) ..... 140
Rand_seq(int) ..... 140
Time() ..... 141
Ctime() ..... 141
gmtime() ..... 142
Md5(str) ..... 142
Md5_hex(str) ..... 143
Md5_str(str) ..... 144
Md5_hex_str(str) ..... 144
Sha1(str) ..... 145
Sha1_hex(str) ..... 145
Sha1_str(str) ..... 146
Sha1_hex_str(str) ..... 146

Sha256(str) 147
Sha256_hex(str) 147
Sha256_str(str) 148
Sha256_hex_str(str) 148
Sha384(str) 149
Sha384_hex(str) 149
Sha384_str(str) 150
Sha384_hex_str(str) 150
Sha512(str) 151
Sha512_hex(str) 151
Sha512_str(str) 152
Sha512_hex_str(str) 152
B32_enc(str) 153
B32_enc_str(str) 153
B32_dec(str) 154
B32_dec_str(str) 154
B64_enc(str) 155
B64_dec(str) 155
Get_pid() 156
Table_to_string(t) 156
Htonl(int) 157
Ntohs(int) 157
Htons(int) 158
Ntohl(int) 159
To_HEX(str) 159
Debug(str) 160
Log(str) 160
File_open(path, str) 161
File_get(file, size) 161
File_close(file) 162
WAF commands 163
**Examples of built-in predefined scripts 171**

# Change Log

| Date | Change Description |
|---|---|
| February 7, 2024 | Initial release. |

# Introduction

FortiADC SLB supports Lua scripts to perform actions that are not currently supported by the built-in feature set. Scripts enable you to use predefined script commands and variables to manipulate the HTTP request/response or select a content route. The multi-script support feature enables you to use multiple scripts by setting their sequence of execution.

# Definition of terms

**Frontend, backend:**

When visiting the virtual server service, the client will create a connection with FortiADC. On the connection, source IP address is client address, and destination address is virtual server address. After FortiADC receives the request from the client, FortiADC will load balance the request back to the real servers. This time, FortiADC will create a connection with the real server; on this connection, source IP address is FortiADC’s out interface IP address, and destination address is the real server address.

The connection between the client and virtual server is considered frontend.

The connection between FortiADC and real server is considered backend.

# Configuration overview

You can use scripts to perform actions that are not currently supported by the built-in feature set. Scripts enable you to use predefined script commands and variables to manipulate the HTTP request/response or select a content route, or get SSL info.

You can type or paste the script content into the configuration page.

Before you begin:

- Create a script.
- You must have Read-Write permission for Server Load Balance settings.

After you have created a script configuration object, you can specify it in the virtual server configuration.

**To create a script configuration object:**

1. Go to Server Load Balance > Scripts.
2. Click Create New to display the configuration editor.
3. Complete the configuration as shown.
4. **Save** the configuration.

| Settings | Guidelines |
|---|---|
| Name | Unique group name. No spaces or special characters.<br>After you initially save the configuration, you cannot edit the name. |
| Input | Type or paste the script. |

# Script reference

- Control structures on page 13
- Operators on page 14
- String Library on page 15
- Multiple Scripting Usage on page 16
- Dynamic scripting configuration change on page 17
- Function on page 18

## Control structures

The table below lists Lua control structures.

| Type | Structure |
|---|---|
| if then else | `if condition1 then`<br>`…`<br>`elseif condition2 then`<br>`… break`<br>`else`<br>`… goto location1`<br>`end`<br><br>`::location1::` |
| for | Fetch all values of table 't'<br>`for k, v in pairs(t) do`<br>`…`<br>`end` |

## Operators

The table below lists Lua operators.

| Type | List |
|---|---|
| Relational operators | <> <= >= == ~= |
| Logic operators | not, and, or |

```
test=str:sub(2,5)
debug("return: %s \n", test)
log("record a log")
result: “bcde”
```

For a tutorial on scripting with the Lua string library, see http://lua-users.org/wiki/StringLibraryTutorial.

## String Library

The FortiADC OS supports only the Lua string library. All other libraries are disabled. The string library includes the following string-manipulation functions:

- String.byte(s, i)
- String.char(i1,i2…)
- String.dump(function)
- String.find(s, pattern)
- String.format
- String.gmatch
- String.gsub
- String.len
- String.lower
- String.match
- String.rep
- String.reverse
- String.sub
- String.upper
- String.starts_with
- String.end_with
- String.format

Like:

```
a=12
b=string.format("%s, pi=%.4f", a, 3.14);
```

- {s:byte(1,-1)}

Like:

```
str = "abc\1\2\0"
t={str:byte(1,-1)}
```

t is a table like an array to store the string.

t[1] is 97, t[2] is 98, t[3] is 99, t[4] is 1, t[5] is 2, t[6] is 0.

- {s:sub(1, 3)}

```
str="abcdefg"
```

## Multiple Scripting Usage

• FortiADC supports using multiple scriptings in one VS.

• Different scripts can contain the same event.

• Priority information can be specified for each event in each script file to control the order of each event if multiple scripts have been executed.

• The smaller the priority value, the higher the priority. Priority starts from digital number 1.

• Otherwise, the original order (created when the script was added to the VS script list) is used.

• When multiple script call function of redirect()/routing()/close(), the final one prevails.

• The the user can enable/disable events by using command set_event().

In the case of multiple scripts, the user might want to disable processing the rest of the scripts in certain cases, or might completely disable respond processing.

• The user can enable/disable event automatic by useing command set_auto().

Enabling behavior: in the case of http keep-alive mode, by default FortiADC will re-enable HTTP REQUEST and HTTP RESPONSE processing for the next transaction (even if the user disables this event in the current transaction using command set_event()). FortiADC allows the user to disable/enable this automatic enabling behavior.

## Dynamic scripting configuration change

When changing an in-use script, FortiADC supports dynamic reloading of the new scripting configuration.

## Function

FortiADC supports the basic commands. If the user want more functions, the user can implement functions to define more with the basic commands.

### Syntax

```
function function_name(parameter)
…
end
```

### Examples: cookie command usage

FortiADC supports two cookie commands: cookie_list() and cookie(t) with t as a table input

```
when HTTP_REQUEST {
ret=HTTP:cookie_list()
for k,v in pairs(ret) do
debug("-----cookie name %s, value %s-----\n", k,v);
end
```

GET value of cookie "test"

```
value = get_cookie_value(HTTP, "test") --the return value is either boolean false or its
     value if exists.
debug("-----get cookie value return %s-----\n", value);
```

GET attribute path of cookie "test", can be used to get other attributes too

```
case_flag = 0; -- or 1
ret = get_cookie_attribute(HTTP, "test", "path", case_flag);--return value is either boolean
     false or its value if exists
debug("-----get cookie path return %s-----\n", ret);
```

SET value of cookie "test"

```
ret = set_cookie_value(HTTP, "test", "newvalue")--return value is boolean
debug("-----set cookie value return %s-----\n", ret);
```

REMOVE a whole cookie

```
ret = remove_whole_cookie(HTTP, "test")--return value is boolean
debug("-----remove cookie return %s-----\n", ret);
```

INSERT a new cookie.

You need to make sure the cookie was not first created by the GET command. Otherwise, by design FortiADC shall use SET command to change its value or attributes.

In HTTP REQUEST, use $Path, $Domain, $Port, $Version; in HTTP RESPONSE, use Path, Domain, Port, Version, etc.

```
ret = insert_cookie(HTTP, "test", "abc.d; $Path=/; $Domain=www.example.com, ")--return value
     is boolean
debug("-----insert cookie return %s-----\n", ret);
}
```

```
function get_cookie_value(HTTP, cookiename)
local t={};
t["name"]=cookiename
t["parameter"]="value";
t["action"]="get"
return HTTP:cookie(t)
end
```

attrname can be path, domain, expires, secure, maxage, max-age, httponly, version, port.

case_flag: If you use zero, FortiADC looks for default attributes named Path, Domain, Expires, Secure, Max-Age, HttpOnly, Version, Port. By setting this to 1, you can specify the case sensitive attribute name to look for in t["parameter"] which could be PAth, DOmaIn, MAX-AGE, EXpires, secuRE, HTTPONLy, VerSion, Port, etc.

```
function get_cookie_attribute(HTTP, cookiename, attrname, case_flag)
local t={};
t["name"]=cookiename
t["parameter"]=attrname;
t["case_sensitive"] = case_flag;
t["action"]="get"
return HTTP:cookie(t)
end
function set_cookie_value(HTTP, cookiename, newvalue)
local t={};
t["name"]=cookiename
t["value"]=newvalue
t["parameter"]="value";
t["action"]="set"
return HTTP:cookie(t)
end
function remove_whole_cookie(HTTP, cookiename)
local t={};
t["name"]=cookiename
t["parameter"]="cookie";
t["action"]="remove"
return HTTP:cookie(t)
end
function insert_cookie(HTTP, cookiename, value)
local t={};
t["name"]=cookiename
t["value"]=value;
t["parameter"]="cookie";
t["action"]="insert"
return HTTP:cookie(t)
end
```

# Events

| Event Name | Description | Available Version |
|---|---|---|
| RULE_INIT | When initializing the script. | V5.2 and earlier |
| VS_LISTENER_BIND | When a VS tries to bind.<br>Right now, allows the user to set tcp options, later can be used to config VS.<br>TCP:sockopt() and MGM:set_event("vs_ listener_bind") are available. | V5.2 |
| TCP_ACCEPTED | When a TCP connection from a client is accepted. | V5.0 |
| TCP_CLOSED | When a TCP connection from a client is to be closed. | V5.0 |
| HTTP_REQUEST | When a HTTP request comes from a client.<br>HTTP: header_get_names, header_get_ values, header_get_value, header_remove, header_remove2, header_insert, header_ replace, header_replace2, header_exists, header_count, version_get, version_set, redirect_with_cookie, redirect_t, redirect, close, disable_event, enable_event, set_event, set_ auto, disable_auto, enable_auto, rand_id, get_ session_id, collect, cookie, cookie_list, cookie_ crypto, dyn_cache_invalid, cached_check, cache_hits, respond, method_get, method_set, uri_get, uri_set, path_get, path_set, query_get, query_set, cache_disable, exclude_check_ disable, dyn_check_disable, dyn_invalid_ check_disable, dyn_cache_enable, cache_ user_key, persist, client_port, local_port, remote_port, client_addr, local_addr, remote_ addr, client_ip_ver<br>PROXY: token_file_open, token_path, lua_ sync, tokeng_commit<br>LB: routing, method_assign_server, get_valid_ routing, get_current_routing | V4.3 |

| Event Name | Description | Available Version |
|---|---|---|
| | AUTH: result, success, gen_renew_cookie, flags, need_renew_cookie, clear_renew_cookie, on_off, clt_meth, form_based, method, auth_flags, author_type, sso_group, relay_type, sess_timeout, set_timeout, user, pass, realm, usergroup, host, uri, sso_domain, domain_prefix, logoff<br>IP: client_port, local_port, remote_port, client_addr, local_addr, remote_addr, client_ip_ver<br>SSL: renegotiate, cert_request, get_verify_depth, set_verify_depth, client_cert, peer_cert, cert<br>TCP: set_snat_ip, clear_snat_ip, sockopt<br>MGM: rand_id, get_session_id, disable_event, enable_event, set_event, set_auto, disable_auto, enable_auto | |
| HTTP_DATA_REQUEST | Allows the user to manipulate http request data. | V4.8 and later |
| SERVER_BEFORE_CONNECT | When connecting to the backend real server.<br>TCP:sockopt() and management commands are available.<br>IP:client_port()/client_addr()/client_ip_ver() are available. | V5.2 |
| SERVER_CONNECTED | When Httproxy deems that the backend real server is connected.<br>TCP:sockopt() and management commands are available.<br>Server-side IP functions are available. | V5.2 |
| HTTP_RESPONSE | When a HTTP response comes from real server. | V4.3 |
| HTTP_DATA_RESPONSE | Alllows the user to manipulate http response data. | V4.8 and later |
| SERVER_CLOSED | When Httproxy is going to terminate the backend real server connection. | V5.2 |
| CLIENTSSL_HANDSHAKE | When a client-side SSL handshake is completed. | V5.0 |
| CLIENTSSL_RENEGOTIATE | When a client-side SSL renegotiation is completed. It’s recommended not to use it as it’s not safe | V5.0 |
| SERVERSSL_HANDSHAKE | When a server-side SSL handshake is completed. | V5.0 |

| Event Name | Description | Available Version |
|---|---|---|
| SERVERSSL_RENEGOTIATE | When a server-side SSL renegotiation is completed. It's recommended not to use it as it's not safe. | V5.0 |
| AUTH_RESULT | When authentication(HTML Form / HTTP-basic) is done. If auth event detect, it still trigger the AUTH_RESULT.<br>LB:routing, ip commands, management commands and AUTH:commands can be used in AUTH_RESULT event.<br>The following commands are support in AUTH_RESULT.<br>HTTP:"uri_get path_get method_get query_get"<br>LB:"routing"<br>AUTH:"result success gen_renew_cookie flags need_renew_cookie clear_renew_cookie on_off clt_meth form_based method auth_flags author_type sso_group relay_type sess_timeout set_timeout the user pass realm the usergroup host uri sso_domain domain_prefix logoff"<br>IP:"client_port local_port remote_port client_addr local_addr remote_addr client_ip_ver"<br>MGM:"rand_id get_session_id disable_event enable_event set_event set_auto disable_auto enable_auto" | V5.2 |
| BEFORE_AUTH | The BEFORE_AUTH event triggers right before the authentication is performed to allow the user specified user group to be used instead. The new user group will override the authentication result of the original authentication policy.<br>HTTP: header_get_names header_get_values header_get_value header_remove header_remove2 header_insert header_replace header_replace2 header_exists header_count version_get version_set redirect_with_cookie redirect_t redirect close disable_event enable_event set_event set_auto disable_auto enable_auto rand_id get_session_id cookie cookie_list cookie_crypto respond method_get method_set uri_get uri_set path_get path_set query_get query_set client_port local_port remote_port client_addr local_addr remote_addr client_ip_ver<br>LB: routing get_valid_routing get_current_routing method_assign_server | V7.2 |

| Event Name | Description | Available Version |
|---|---|---|
| | AUTH: set_usergroup realm usergroup host<br>SSL: renegotiate cert_request get_verify_depth set_verify_depth client_cert peer_cert cert<br>IP: client_port local_port remote_port client_addr local_addr remote_addr client_ip_ver<br>MGM: rand_id get_session_id disable_event enable_event set_event set_auto disable_auto enable_auto | |
| COOKIE_BAKE | When FortiADC is done baking an authentication cookie. | V5.2 |

## WAF events

Use the WAF events to insert an action before or after a WAF scan.

In FortiADC, the WAF has six stages for when modules can scan for attacks:

- WAF_SCAN_STAGE_REQ_HEADER
- WAF_SCAN_STAGE_REQ_BODY (streaming stage)
- WAF_SCAN_STAGE_REQ_WHOLE_BODY
- WAF_SCAN_STAGE_RES_HEADER
- WAF_SCAN_STAGE_RES_BODY (streaming stage)
- WAF_SCAN_STAGE_RES_WHOLE_BODY

The WAF event may be applied to specific WAF stages depending on their hook point.

| Event | Hook point | Example |
|---|---|---|
| WAF_REQUEST_BEFORE_SCAN | Before WAF_SCAN_STAGE_REQ_HEADER start.<br>If WAF function is not enabled on VS, then this will not be triggered. | when WAF_REQUEST_BEFORE_SCAN {<br>debug("test WAF_REQUEST_BEFORE_SCAN\n")<br>} |
| WAF_RESPONSE_BEFORE_SCAN | Before WAF_SCAN_STAGE_RES_HEADER start.<br>If WAF function is not enabled on VS, then this will not be triggered. | when WAF_REQUEST_ATTACK_DETECTED {<br>debug("test WAF_REQUEST_ATTACK_DETECTED\n")<br>} |
| WAF_REQUEST_ATTACK_DETECTED | After all request stages when there are attacks detected (violation).<br>If WAF function is not enabled on VS, then this will not be triggered.<br>If WAF module does not detect any violations, then this will not be triggered. | when WAF_RESPONSE_BEFORE_SCAN {<br>debug("test WAF_RESPONSE_BEFORE_SCAN\n")<br>} |
| WAF_RESPONSE_ATTACK_DETECTED | After all response stages when there are attacks detected (violation).<br>If WAF function is not enabled on VS, then this will not be triggered.<br>If WAF module does not detect any violations, then this will not be triggered. | when WAF_RESPONSE_ATTACK_DETECTED {<br>debug("test WAF_RESPONSE_ATTACK_DETECTED\n")<br>} |

# Predefined commands

- Management commands on page 26
- IP commands on page 29
- TCP commands on page 34
- HTTP commands on page 43
- HTTP DATA commands on page 71
- Cookie commands on page 76
- Authentication commands on page 79
- SSL commands on page 87
- GEO IP commands on page 96
- RAM cache commands on page 98
- PROXY commands on page 105
- LB commands on page 115
- Persistence commands on page 118
- Global commands on page 126
- WAF commands on page 163

# Management commands

Management (MGM) commands script event management functions:

MGM:get_session_id() on page 26 — Returns the session ID.

MGM:rand_id() on page 26 — Returns a 32-character string of HEX symbols.

MGM:set_event(t) on page 27 — Allows the user to disable/enable the rest of the events from executing by disabling this event.

MGM:set_auto(t) on page 27 — In the case of keep-alive, all events will be re-enabled automatically even if they are disabled in the previous TRANSACTION using the HTTP:set_event(t) command. To disable this automatic re-enabling behavior, you can call HTTP:set_auto(t).

## MGM:get_session_id()

Returns the session ID.

### Syntax

MGM:get_session_id();

### Arguments

N/A

### Example

```
when HTTP_REQUEST {
sid = MGM:get_session_id()
debug("session id: %s\n", sid)
}
```

FortiADC version: V5.0

Used in events: all **except** VS_LISTENER_BIND

## MGM:rand_id()

Returns a 32-character string of HEX symbols.

### Syntax

MGM:rand_id();

### Arguments

N/A

### Example

```
when HTTP_REQUEST {
rid = MGM:rand_id()
debug("rand id is %s\n", rid)
}
```

FortiADC version: V5.0

Used in events: all **except** VS_LISTENER_BIND

## MGM:set_event(t)

Allows the user to disable/enable the rest of the events from executing by disabling this event.

### Syntax

MGM:set_event(t);

### Arguments

| Name | Description |
|---|---|
| t | A table which specifies the event and operation. |

### Example

```
when HTTP_REQUEST {
t={};
t["event"]="req"; -- can be "req", "res", "data_req", "data_res", "ssl_client", "ssl_
server", "tcp_accept", "tcp_close", "ssl_renego_client", "ssl_renego_server", "server_
connected", "server_close", "server_before_connect", "vs_listener_bind", "auth_result",
"cookie_bake"
t["operation"]="disable"; -- can be "enable", and "disable"
MGM:set_event(t);
debug("disable rest of the HTTP_REQUEST events\n");
}
```

FortiADC version: V5.0

Used in events: ALL

## MGM:set_auto(t)

In the case of keep-alive, all events will be re-enabled automatically even if they are disabled in the previous TRANSACTION using the HTTP:set_event(t) command. To disable this automatic re-enabling behavior, you can call

HTTP:set_auto(t).

## Syntax

MGM:set_auto(t);

## Arguments

| Name | Description |
|---|---|
| t | A table which specifies the event and operation to enable or disable. |

## Example

```
when HTTP_REQUEST {
t={};
t["event"]="req";
t["operation"]="disable";
MGM:set_auto(t);
debug("disable automatic re-enabling of the HTTP_REQUEST events\n");
}
```

**Note:** Event can be "req", "res", "data_req", "data_res", "ssl_server", "ssl_renego_server", "server_connected", "server_close", "server_before_connect." Operation can be "enable", and "disable."

FortiADC version: V5.0

Used in events: ALL

## IP commands

IP commands contain functions to obtain IP layer related information such as obtaining the IP address and port of the server and client:

IP:Client_addr() on page 29 — Returns the client IP address of a connection; for the frontend, it will return the source address, while for the backend, it will return the destination address.

IP:server_addr() on page 30 — Returns the IP address of the server in the backend.

IP:local_addr() on page 30 — For the frontend, it returns the IP address of the virtual server that the client is connected to. For the backend, it returns the incoming interface IP address of the return packet.

IP:remote_addr() on page 31 — Returns the IP address of the host on the far end of the connection.

IP:client_port() on page 31 — Returns the client port number.

IP:server_port() on page 31 — Returns the server port number. It is the real server port.

IP:local_port() on page 32 — Returns the local port number. In the frontend, the local port is the virtual server port. In the backend, the local port is the port used to connect to the gateway.

IP:remote_port() on page 32 — Returns the remote port number. In the frontend, the remote port is the client port. In the backend, remote port is the real server port.

IP:client_ip_ver() on page 33 — Returns the current client IP version number of the connection, either 4 (for IPv4) or 6 (for IPv6).

IP:server_ip_ver() on page 33 — Returns the current server IP version number of the connection, either 4 (for IPv4) or 6 (for IPv6).

## IP:Client_addr()

Returns the client IP address of a connection; for the frontend, it will return the source address, while for the backend, it will return the destination address.

### Syntax

cip=IP:client_addr()

### Arguments

N/A

### Example

```
when SERVERSSL_HANDSHAKE {
cip=IP:client_addr()
lip=IP:local_addr()
sip=IP:server_addr()
rip=IP:remote_addr()
cp=IP:client_port()
lp=IP:local_port()
sp=IP:server_port()
rp=IP:remote_port()
sipv=IP:server_ip_ver();
cipv=IP:client_ip_ver();
debug("in server ssl with remote addr %s:%s client %s:%s, local %s:%s, server %s:%s, ip
version %s:%s\n", rip, rp, cip, cp, lip,lp, sip, sp, sipv, cipv)
}}
```

FortiADC version: V5.0

Used in events: all **except** VS_LISTENER_BIND

## IP:server_addr()

Returns the IP address of the server in the backend.

### Syntax

sip=IP:server_addr()

### Arguments

N/A

### Example

Please refer to IP:client_addr() example.

FortiADC version: V5.0

Used in events: Server-side event (include HTTP_RESPONSE/HTTP_DATA_RESPONSE/…)

## IP:local_addr()

For the frontend, it returns the IP address of the virtual server that the client is connected to. For the backend, it returns the incoming interface IP address of the return packet.

### Syntax

sip=IP:local_addr()

### Arguments

N/A

### Example

Please refer to IP:client_addr() example.

FortiADC version: V5.0

Used in events: all **except** VS_LISTENER_BIND / SERVER_BEFORE_CONNECT

## IP:remote_addr()

Returns the IP address of the host on the far end of the connection.

### Syntax

sip=IP:remote_addr()

### Arguments

N/A

### Examples

Please refer to IP:client_addr() example.

FortiADC version: V5.0

Used in events: all **except** VS_LISTENER_BIND / SERVER_BEFORE_CONNECT

## IP:client_port()

Returns the client port number.

### Syntax

cp=IP:client_port()

### Arguments

N/A

### Example

Please refer to IP:client_addr() example.

FortiADC version: V5.0

Used in events: all **except** VS_LISTENER_BIND

## IP:server_port()

Returns the server port number. It is the real server port.

### Syntax

sp=IP:server_port()

### Arguments

N/A

### Example

Please refer to IP:client_addr() example.

FortiADC version: V5.0

Used in events: Server-side events (include HTTP_RESPONSE/HTTP_DATA_RESPONSE/…)

## IP:local_port()

Returns the local port number. In the frontend, the local port is the virtual server port. In the backend, the local port is the port used to connect to the gateway.

### Syntax

sp=IP:local_port()

### Arguments

N/A

### Example

Please refer to IP:client_addr() example.

FortiADC version: V5.0

Used in events: all **except** VS_LISTENER_BIND / SERVER_BEFORE_CONNECT

## IP:remote_port()

Returns the remote port number. In the frontend, the remote port is the client port. In the backend, remote port is the real server port.

### Syntax

rp=IP:remote_port()

### Arguments

N/A

### Example

Please refer to IP:client_addr() example.

FortiADC version: V5.0

Used in events: all **except** VS_LISTENER_BIND / SERVER_BEFORE_CONNECT

## IP:client_ip_ver()

Returns the current client IP version number of the connection, either 4 (for IPv4) or 6 (for IPv6).

### Syntax

cv=IP:client_ip_ver ()

### Arguments

N/A

### Example

Please refer to IP:client_addr() example.

FortiADC version: V5.0

Used in events: all **except** VS_LISTENER_BIND

## IP:server_ip_ver()

Returns the current server IP version number of the connection, either 4 (for IPv4) or 6 (for IPv6).

### Syntax

cv=IP:server_ip_ver ()

### Arguments

N/A

### Example

Please refer to IP:client_addr() example.

FortiADC version: V5.0

Used in events: Server-side events

# TCP commands

TCP commands contains functions to obtain and manipulate information related to the TCP layer, such as sockopt:

TCP:reject() on page 34 — Allows the user to reject a TCP connection from a client.

TCP:set_snat_ip(str) on page 35 — Allows the user to set the backend TCP connection's source address and port.

TCP:clear_snat_ip() on page 35 — Allows the user to clear any IP that was set using the set_snat_ip() command.

TCP:sockopt(t) on page 36 — Allows the user to customize the send buffer and receive buffer size. Can set or get various socket/IP/TCP operations, such as buffer size, timeout, MSS, etc. This currently only supports snd_buf and rcv_buf buffer sizes. For client-side events, this command applies to the client-side socket; for server-side events, it applies to server-side socket.

TCP:after_timer_set() on page 38 — Allows the user to create and schedule a timer with a callback function and timeout value. This allows you to create multiple timers each with a unique callback function name.

TCP:after_timer_cancel() on page 39 — Allows the user to cancel a scheduled timer. This function can only cancel a timer before it is triggered if it is not periodic.

TCP:after_timer_get() on page 40 — Allows the user to get the information about the scheduled timers.

TCP:close() on page 41 — Allows the user to close the TCP connection immediately.

## TCP:reject()

Allows the user to reject a TCP connection from a client.

### Syntax

TCP:reject();

### Arguments

N/A

### Example

```
when TCP_ACCEPTED {
--check if the st is true or false;
If st then
TCP:reject();
end
}
```

FortiADC version: V5.0

Used in events: TCP_ACCEPTED

## TCP:set_snat_ip(str)

Allows the user to set the backend TCP connection’s source address and port.

### Syntax

TCP:set_snat_ip(str);

**Note:** To use the set_snat_ip() command, you must ensure the SOURCE ADDRESS flag is selected in the HTTP or HTTPS profile type.

### Arguments

| Name | Description |
|---|---|
| str | A string which specifies the ip address. |

### Example

```
when TCP_ACCEPTED{
addr_group = “172.24.172.60/24”
client_ip = IP:client_addr()
matched = cmp_addr(client_ip, addr_group)
if matched then
if TCP:set_snat_ip(“10.106.3.124”) then
debug(“set SNAT ip to 10.106.3.124\n”)
end
end
}
```

**Note:** The VS must have the client address enabled in the profile, as shown in the example below.

```
config load-balance profile
   edit "http"
      set type http
      set client-address enable
   next
end
```

FortiADC version: V5.2

Used in events: TCP_ACCEPTED / HTTP_REQUEST / HTTP_DATA_REQUEST / CLIENTSSL_HANDSHAKE

## TCP:clear_snat_ip()

Allows the user to clear any IP that was set using the set_snat_ip() command.

### Syntax

TCP:clear_snat_ip();

### Arguments

N/A

### Example

```
when HTTP_REQUEST {
if TCP:clear_snat_ip() then
debug("clear SNAT ip!\n")
}
```

FortiADC version: V5.0

Used in events: TCP_ACCEPTED / HTTP_REQUEST / HTTP_DATA_REQUEST / CLIENTSSL_HANDSHAKE

## TCP:sockopt(t)

Allows the user to customize the send buffer and receive buffer size. Can set or get various socket/IP/TCP operations, such as buffer size, timeout, MSS, etc. This currently only supports snd_buf and rcv_buf buffer sizes. For client-side events, this command applies to the client-side socket; for server-side events, it applies to server-side socket.

### Syntax

TCP:sockopt(t);

### Arguments

| Name | Description |
|---|---|
| t | A table which specifies the event and operation, variable. |

### Example

```
when RULE_INIT {
debug(" ======== RULE_INIT ========\n");
-- access to https://notes.shichao.io/unp/ch7/ for more details.
tcp_message = {};
tcp_message[1]="snd_buf"; --int
tcp_message[2]="rcv_buf"; --int
setIntMsg = {};
setIntMsg[1]="snd_buf"; --int
setIntMsg[2]="rcv_buf"; --int
setIntValue = {};
setIntValue[1] = 111222;
setIntValue[2] = 111222;
}
```

```
when VS_LISTENER_BIND{
--when a VS tries to bind.
debug(" ======== VS_LISTENER_BIND ========\n");
for k,v in pairs(tcp_message) do
t = {};
t["op"] = "get"
t["message"]=v
if TCP:sockopt(t) then
debug("%s value is %d\n",v, TCP:sockopt(t));
else
debug("get %s status     %s\n",v,TCP:sockopt(t));
end
end
debug("        ==== set ==== \n");
for k,v in pairs(setIntMsg) do
s = {};
s["op"] = "set"; --or "set"
s["message"] = v
s["value"] = setIntValue[k]; -- for integer value
result = TCP:sockopt(s);
debug("setting %s to %s return %s\n",v,setIntValue[k], result);
end
debug("        ==== End set ==== \n");
for k,v in pairs(tcp_message) do
t = {};
t["op"] = "get"
t["message"]=v
if TCP:sockopt(t) then
debug("%s value is %d\n",v, TCP:sockopt(t));
else
debug("get %s status     %s\n",v,TCP:sockopt(t));
end
end
}

when HTTP_RESPONSE {
debug(" ======== HTTP_RESPONSE ========\n");
t={}
t["size"] = 100;
HTTP:collect(t)
debug("        ==== set ==== \n");
for k,v in pairs(setIntMsg) do
s = {};
s["op"] = "set"; --or "set"
s["message"] = v
s["value"] = setIntValue[k]; -- for integer value
result = TCP:sockopt(s);
debug("setting %s to %s return %s\n",v,setIntValue[k], result);
end
debug("        ==== End set ==== \n");
for k,v in pairs(tcp_message) do
t = {};
t["op"] = "get"
t["message"]=v
if TCP:sockopt(t) then
debug("%s value is %d\n",v, TCP:sockopt(t));
else
debug("get %s status     %s\n",v,TCP:sockopt(t));
end
end
}
```

```
when HTTP_DATA_RESPONSE {
debug(" ======== HTTP_DATA_RESPONSE ========\n");
debug("        ==== set ==== \n");
for k,v in pairs(setIntMsg) do
s = {};
s["op"] = "set"; --or "set"
s["message"] = v
s["value"] = setIntValue[k]; -- for integer value
result = TCP:sockopt(s);
debug("setting %s to %s return %s\n",v,setIntValue[k], result);
end
debug("        ==== End set ==== \n");
for k,v in pairs(tcp_message) do
t = {};
t["op"] = "get"
t["message"]=v
if TCP:sockopt(t) then
debug("%s value is %d\n",v, TCP:sockopt(t));
else
debug("get %s status     %s\n",v,TCP:sockopt(t));
end
end
}
```

FortiADC version: V5.0

Used in events:

- In client-side events, including TCP_BIND, TCP_ACCEPTED, HTTP_REQUEST, HTTP_DATA_REQUEST
- In server-side events, including HTTP_RESPONSE, HTTP_DATA_RESPONSE, BEFORE_CONNECT, SERVER_CONNECTED.

## TCP:after_timer_set()

Allows the user to create and schedule a timer with a callback function and timeout value. This allows you to create multiple timers each with a unique callback function name. Periodic timers will be executed periodically until the associated session is closed or the after_timer is closed.

Returns Boolean true if successful, otherwise, returns Boolean false.

### Syntax

TCP:after_timer_set (timer_cb_name, timeout, periodic);

### Arguments

| Name | Description |
|---|---|
| timer_cb_name | A string of the callback function name. This is also the unique identification of a timer. This parameter is required. |
| timeout | An integer as the timeout value in milliseconds. This parameter is required. |
| periodic | A Boolean to indicate whether this timer is periodic. This parameter is required. |

### Example

```
when TCP_ACCEPTED {
    debug("[%s]------> TCP accepted begin:\n",ctime());
       local_count = 10
       cip = IP:client_addr();
       debug("[%s]------> client IP %s\n", ctime(),cip);
       cport = IP:client_port();
       debug("[%s]------> client Port %s\n", ctime(),cport);
       debug("[%s]------> local_count= %d\n", ctime(),local_count);
       debug("[%s]------> After function called here.\n",ctime());

       AFTER_TIMER_NAME = function ()
               debug("[%s]=======> After function call begin:\n",ctime())
               cport = IP:client_port();
               debug("[%s]=======> client Port %s\n",ctime(),cport);
               debug("[%s]=======> After function call end.\n",ctime())
       end

       TCP:after_timer_set("AFTER_TIMER_NAME", 5000, true);
}
```

When the client successfully creates a TCP connection, the script will be executed. The function will be executed every 5 seconds (5000 milliseconds) and print out the text.

FortiADC console debug output:

```
[Thu Jan 11 16:30:50 2024]------> TCP accepted begin:
[Thu Jan 11 16:30:50 2024]------> client IP 10.1.0.161
[Thu Jan 11 16:30:50 2024]------> client Port 37818
[Thu Jan 11 16:30:50 2024]------> local_count= 10
[Thu Jan 11 16:30:50 2024]------> After function called here.
[Thu Jan 11 16:30:55 2024]=======> After function call begin:
[Thu Jan 11 16:30:55 2024]=======> client Port 37818
[Thu Jan 11 16:30:55 2024]=======> After function call end.
[Thu Jan 11 16:31:00 2024]=======> After function call begin:
[Thu Jan 11 16:31:00 2024]=======> client Port 37818
[Thu Jan 11 16:31:00 2024]=======> After function call end.
```

FortiADC version: V7.4.1

Used in events:

- HTTP events: HTTP_REQUEST / HTTP_RESPONSE / HTTP_DATA_REQUEST / HTTP_DATA_RESPONSE
- TCP events: TCP_ACCEPTED / SERVER_CONNECTED / SERVER_BEFORE_CONNECT

## TCP:after_timer_cancel()

Allows the user to cancel a scheduled timer. This function can only cancel a timer before it is triggered if it is not periodic.

Returns Boolean true if successful, otherwise, returns Boolean false.

### Syntax

TCP:after_timer_cancel (timer_cb_name);

### Arguments

| Name | Description |
|---|---|
| timer_cb_name | A string to indicate the name of the timer to be canceled. This parameter is required. |

### Example

```
when TCP_ACCEPTED {
       AFTER_TIMER_NAME = function ()
       debug("[%s]====>After function call begin:\n",ctime());
       debug("[%s]====>After function call end.\n",ctime());
end
TCP:after_timer_set("AFTER_TIMER_NAME", 1000, true);
}
when HTTP_REQUEST{
    debug("[%s]------> Events: HTTP_REQUEST begin:\n", ctime());
    TCP:after_timer_cancel("AFTER_TIMER_NAME");
}
```

When the client successfully creates a TCP connection, the script will be executed. When the HTTP is requested, the AFTER_TIMER_NAME will be stopped.

FortiADC console debug output:

```
[Thu Oct  5 13:37:51 2023]====>After function call begin:
[Thu Oct  5 13:37:51 2023]====>After function call end.
[Thu Oct  5 13:37:52 2023]====>After function call begin:
[Thu Oct  5 13:37:52 2023]====>After function call end.
[Thu Oct  5 13:37:53 2023]------> Events: HTTP_REQUEST begin:
```

FortiADC version: V7.4.1

Used in events:

- HTTP events: HTTP_REQUEST / HTTP_RESPONSE / HTTP_DATA_REQUEST / HTTP_DATA_RESPONSE
- TCP events: TCP_ACCEPTED / SERVER_CONNECTED / SERVER_BEFORE_CONNECT

## TCP:after_timer_get()

Allows the user to get the information about the scheduled timers.

When successful, returns a string for one timer, a table of strings for multiple timers. For example, the returned string "'AFTER_TIMER_NAME':5000:periodic" shows the timer name, expiration in milliseconds and if it is periodic.

Returns nil for all failures.

### Syntax

TCP:after_timer_get ([timer_cb_name]);

### Arguments

| Name | Description |
|---|---|
| timer_cb_name | A string to indicate the name of the timer.<br>This parameter is optional. If this parameter is empty, then the function will get the information for all existing timers. |

### Example

```
when TCP_ACCEPTED {
       AFTER_TIMER_NAME = function ()
       debug("[%s]====>After function call begin:\n",ctime());
       debug("[%s]====>After function call end.\n",ctime());
end

TCP:after_timer_set("AFTER_TIMER_NAME", 1000, true);
ret = TCP:after_timer_get("AFTER_TIMER_NAME");
debug("after_timer_get success: %s\n", ret);
}
```

When the client successfully creates a TCP connection, the script will be executed. This function gets the **after_timer** information and prints it out on the first line.

FortiADC console debug output:

```
after_timer_get success: 'AFTER_TIMER_NAME':1000:periodic
[Thu Oct  5 12:36:34 2023]====>After function call begin:
[Thu Oct  5 12:36:34 2023]====>After function call end.
[Thu Oct  5 12:36:35 2023]====>After function call begin:
[Thu Oct  5 12:36:35 2023]====>After function call end.
```

FortiADC version: V7.4.1

Used in events:

- HTTP events: HTTP_REQUEST / HTTP_RESPONSE / HTTP_DATA_REQUEST / HTTP_DATA_RESPONSE
- TCP events: TCP_ACCEPTED / SERVER_CONNECTED / SERVER_BEFORE_CONNECT

## TCP:close()

Allows the user to close the TCP connection immediately. Once an associated session is closed, all the after_timers will be deleted.

Returns Boolean true if successful, otherwise, returns Boolean false.

### Syntax

TCP:close();

### Arguments

N/A

## Example

```
when TCP_ACCEPTED {
       AFTER_TIMER_NAME = function ()
       debug("[%s]====>After function call begin:\n",ctime());
       debug("[%s]====>After function call end.\n",ctime());
end

TCP:after_timer_set("AFTER_TIMER_NAME", 1000, true);
}
when HTTP_REQUEST{
    debug("[%s]------> Events: HTTP_REQUEST begin:\n", ctime());
    TCP:close();
}
```

When the client successfully creates a TCP connection, the script will be executed. When the HTTP is requested, the TCP connection will be closed.

Client side:

```
[root@Client1 ~]# curl http://10.1.0.15 -v
*   Trying 10.1.0.15:80...
* Connected to 10.1.0.15 (10.1.0.15) port 80 (#0)
> GET / HTTP/1.1
> Host: 10.1.0.15
> User-Agent: curl/8.0.1
> Accept: */*
>
* Empty reply from server
* Closing connection 0
curl: (52) Empty reply from server
```

FortiADC version: V7.4.1

Used in events:

- HTTP events: HTTP_REQUEST / HTTP_RESPONSE / HTTP_DATA_REQUEST / HTTP_DATA_RESPONSE
- TCP events: TCP_ACCEPTED / SERVER_CONNECTED / SERVER_BEFORE_CONNECT
- CLIENTSSL_HANDSHAKE

## HTTP commands

The HTTP scripting class includes basic functions for manipulating HTTP elements and other important functions:

HTTP:header_get_names() on page 44 — Returns a list of all the headers present in the request or response.

HTTP:header_get_values(header_name) on page 45 — Returns a list of value(s) of the HTTP header named <header_name>, with a count for each value.

HTTP:header_get_value(header_name) on page 45 — Returns the value of the HTTP header named <header_name>.

HTTP:header_remove(header_name) on page 46 — Removes all headers named with the name <header_name>.

HTTP:header_remove2(header_name, countid) on page 47 — Header_get_values() returns a count ID for each item.

HTTP:header_insert(header_name, value) on page 47 — Inserts the header <header_name> with value <value> into the end of the HTTP request or response.

HTTP:header_replace(header_name, value) on page 48 — Replaces the occurrence value of header <header_name> with value <value>.

HTTP:header_replace2(header_name, value, countid) on page 48 — Header_get_values() returns a count ID for each item.

HTTP:header_exists(header_name) on page 49 — Returns true when the header <header_name> exists and false when it does not exist.

HTTP:header_count(header_name) on page 50 — Returns the integer counter of the header <header_name>.

HTTP:method_get() on page 50 — Returns the string of the HTTP request method.

HTTP:method_set(value) on page 50 — Sets the HTTP request method to the string <value>.

HTTP:path_get() on page 51 — Returns the string of the HTTP request path.

HTTP:path_set(value) on page 51 — Sets HTTP request path to the string <value>.

HTTP:uri_get() on page 52 — Returns the string of the HTTP request URI.

HTTP:uri_set(value) on page 52 — Sets the HTTP request URI to the string <value>.

HTTP:query_get() on page 53 — Returns the string of the HTTP request query.

HTTP:query_set(value) on page 53 — Sets the HTTP request query to the string <value>.

HTTP:redirect(“url”, …) on page 54 — Redirects an HTTP request or response to the specified URL.

HTTP:redirect_with_cookie(url, cookie) on page 54 — Redirects an HTTP request or response to the specified URL with cookie.

HTTP:redirect_t(t) on page 55 — Redirects an HTTP request or response to the URL specified in the table.

HTTP:version_get() on page 56 — Returns the HTTP version of the request or response.

HTTP:version_set(value) on page 56 — Sets the HTTP request or response version to the string <value>.

HTTP:status_code_get() on page 57 — Returns the response status code output as string.

HTTP:status_code_set(value) on page 57 — Sets the HTTP response status code.

HTTP:code_get() on page 58 — Returns the response status code, output as integer.

HTTP:code_set(integer) on page 58 — Sets the response status code.

HTTP:reason_get() on page 58 — Returns the response reason.

HTTP:reason_set(value) on page 59 — Sets the response reason.

HTTP:client_addr() on page 59 — Returns the client IP address of a connection.

HTTP:local_addr() on page 60 — For HTTP_REQUEST, returns the IP address of the virtual server the client is connected to; for HTTP_RESPONSE, returns the incoming interface IP address of the return packet.

HTTP:server_addr() on page 60 — Returns the IP address of the server in HTTP_RESPONSE.

HTTP:remote_addr() on page 61 — Returns the IP address of the host on the far end of the connection.

HTTP:client_port() on page 61 — Returns real client port number in a string format.

HTTP:local_port() on page 62 — Returns the local port number in a string format.

HTTP:remote_port() on page 63 — Returns the remote port number in a string format.

HTTP:server_port() on page 63 — Returns the real server port number in a string format.

HTTP:client_ip_ver() on page 64 — Returns the client IP version number. This can be used to get IPv4 or IPv6 versions.

HTTP:server_ip_ver() on page 64 — Returns the server IP version number. This can be used to get IPv4 or IPv6 versions.

HTTP:close() on page 65 — Close an HTTP connection using code 503.

HTTP:respond(t) on page 65 — Allows you to return a customized page and send out an HTTP response directly from FortiADC.

HTTP:get_session_id() on page 66 — FortiADC will assign each session a unique ID and allow the user to get this unique ID through this function.

HTTP:rand_id() on page 67 — Returns a random string of 32 characters long in hex format.

HTTP:set_event(t) on page 67 — Sets a request or response event to enable or disable.

HTTP:set_auto() on page 68 — Sets an automatic request or response event.

HTTP:get_unique_session_id() on page 69 — Returns a unique ID per session in history. Each ID is a string consisting of 32 hex digits, for example "b6e49ca3c937baaa47f2d111f593be8b".

HTTP:get_unique_transaction_id() on page 70 — Returns a unique ID per transaction in history. Each ID is a string consisting of 32 hex digits, for example "b0b9fec0b4b28306a2bf4f63eb97520e".

## HTTP:header_get_names()

Returns a list of all the headers present in the request or response.

### Syntax

HTTP:header_get_names();

### Arguments

N/A

### Example

```
when HTTP_REQUEST {
--use header and value
headers = HTTP:header_get_names()
for k, v in pairs(headers) do
debug("The value of header %s is %s.\n", k, v)
end
--only use the header name
for name in pairs(headers) do
debug("The request/response includes header %s.\n",name)
end
}
```

FortiADC version: V4.3

Used in events: HTTP_REQUEST / HTTP_RESPONSE

## HTTP:header_get_values(header_name)

Returns a list of value(s) of the HTTP header named <header_name>, with a count for each value. Note that the command returns all the values in the headers as a list if there are multiple headers with the same name.

### Syntax

HTTP:header_get_values(header_name);

### Arguments

| Name | Description |
|---|---|
| Header_name | A string which specifies the header name. |

### Example

```
when HTTP_REQUEST {
cookies=HTTP:header_get_values("Cookie")
for k, cnt in pairs(cookies) do
debug("initially include cookie %s cnt %d\n", k, v)
end
}
```

FortiADC version: V4.3

Used in events: HTTP_REQUEST / HTTP_RESPONSE / AUTH_RESULT

## HTTP:header_get_value(header_name)

Returns the value of the HTTP header named <header_name>.

Returns false if the HTTP header named <header_name> does not exist. The command operates on the value of the last head if there are multiple headers with the same name.

### Syntax

HTTP:header_get_value(header_name);

### Arguments

| Name | Description |
|---|---|
| Header_name | A string which specifies the header name. |

### Example

```
when HTTP_REQUEST {
host = HTTP:header_get_value("Host");
debug("host is %s\n", host);
}
```

FortiADC version: V4.3

Used in events: HTTP_REQUEST / HTTP_RESPONSE

## HTTP:header_remove(header_name)

Removes all headers named with the name <header_name>.

### Syntax

HTTP:header_remove(header_name);

### Arguments

| Name | Description |
|---|---|
| Header_name | A string which specifies the header name. |

### Example

```
when HTTP_REQUEST {
HTTP:header_remove(“Cookie”);
}
```

FortiADC version: V4.3

Used in events: HTTP_REQUEST / HTTP_RESPONSE

## HTTP:header_remove2(header_name, countid)

Header_get_values() returns a count ID for each item. This count ID can be used in both header_remove2() and header_replace2() to remove and replace a certain header of a given name referenced by the count ID.

### Syntax

HTTP:header_remove2(header_name, countid);

### Arguments

| Name | Description |
|---|---|
| Header_name | A string which specifies the header name. |
| Countid | A integer which specifies the header_name serial number |

### Example

```
when HTTP_RESPONSE {
cookies=HTTP:header_get_values("Set-Cookie")
for k, v in pairs(cookies) do
debug("include cookie %s cnt %d\n", k, v)
end
if HTTP:header_remove2("Set-Cookie", 1) then
debug("remove 1st cookie\n")
end
}
```

FortiADC version: V4.8

Used in events: HTTP_REQUEST / HTTP_RESPONSE

## HTTP:header_insert(header_name, value)

Inserts the header <header_name> with value <value> into the end of the HTTP request or response.

### Syntax

HTTP:header_insert(header_name, value);

### Arguments

| Name | Description |
|---|---|
| Header_name | A string which specifies the header name. |
| Value | A string which specifies the value of the header<header_name>. |

### Example

```
when HTTP_REQUEST {
HTTP:header_insert("Cookie", "insert_cookie=server1")
}
```

FortiADC version: V4.3

Used in events: HTTP_REQUEST / HTTP_RESPONSE

## HTTP:header_replace(header_name, value)

Replaces the occurrence value of header <header_name> with value <value>.

### Syntax

HTTP:header_replace(header_name, value);

### Arguments

| Name | Description |
|---|---|
| Header_name | A string which specifies the header name. |
| Value | A string which specifies the value of the header<header_name>. |

### Example

```
when HTTP_REQUEST {
HTTP:header_replace("Host", "www.fortinet.com")
}
```

FortiADC version: V4.8

Used in events: HTTP_REQUEST / HTTP_RESPONSE

## HTTP:header_replace2(header_name, value, countid)

Header_get_values() returns a count ID for each item. This count ID can be used in both header_remove2() and header_replace2() to remove and replace a certain header of a given name referenced by the count ID.

### Syntax

HTTP:header_replace2(header_name, value, countid);

### Arguments

| Name | Description |
|---|---|
| Value | A string which specifies the value of the header<header_name>. |
| Header_name | A string which specifies the header name. |
| Countid | A integer which specifies the header_name serial number |

### Example

```
when HTTP_REQUEST {
cookies=HTTP:header_get_values("Cookie")
for k, v in pairs(cookies) do
debug("include cookie %s cnt %d\n", k, v)
end
if HTTP:header_replace2("Cookie", “new_value”, 1) then
debug("replace 1st cookie\n")
end
}
```

FortiADC version: V4.8

Used in events: HTTP_REQUEST / HTTP_RESPONSE

## HTTP:header_exists(header_name)

Returns true when the header <header_name> exists and false when it does not exist.

### Syntax

HTTP:header_exists(header_name);

### Arguments

| Name | Description |
|---|---|
| Header_name | A string which specifies the header name. |

### Example

```
when HTTP_REQUEST {
if HTTP:header_exists(“Cookie”) then
…
end
}
```

FortiADC version: V4.3

Used in events: HTTP_REQUEST / RESPONSE

## HTTP:header_count(header_name)

Returns the integer counter of the header <header_name>.

### Syntax

HTTP:header_count(header_name);

### Arguments

| Name | Description |
|---|---|
| Header_name | A string which specifies the header name. |

### Example

```
when HTTP_REQUEST {
count = HTTP:header_count("Cookie");
}
```

FortiADC version: V4.3

Used in events: HTTP_REQUEST / HTTP_RESPONSE

## HTTP:method_get()

Returns the string of the HTTP request method.

### Syntax

HTTP:method_get();

### Arguments

N/A

### Example

```
when HTTP_REQUEST {
method = HTTP:method_get();
}
```

FortiADC version: V4.3

Used in events: HTTP_REQUEST / AUTH_RESULT

## HTTP:method_set(value)

Sets the HTTP request method to the string <value>.

### Syntax

HTTP:method_set(value);

### Arguments

| Name | Description |
|---|---|
| str | A string which specifies the method. |

### Example

```
when HTTP_REQUEST {
HTTP:method_set("POST")
}
```

FortiADC version: V4.3

Used in events: HTTP_REQUEST

## HTTP:path_get()

Returns the string of the HTTP request path.

### Syntax

HTTP:path_get();

### Arguments

N/A

### Example

```
when HTTP_REQUEST {
path = HTTP:path_get();
}
```

FortiADC version: V4.3

Used in events: HTTP_REQUEST / AUTH_RESULT

## HTTP:path_set(value)

Sets the HTTP request path to the string <value>.

### Syntax

HTTP:path_set(value);

### Arguments

| Name | Description |
|---|---|
| Value | A string which specifies the path. |

### Example

```
when HTTP_REQUEST {
HTTP:path_set("/other.html");
}
```

FortiADC version: V4.3

Used in events: HTTP_REQUEST

## HTTP:uri_get()

Returns the string of the HTTP request URI.

### Syntax

HTTP:uri_get();

### Arguments

N/A

### Example

```
when HTTP_REQUEST {
uri = HTTP:uri_get();
}
```

FortiADC version: V4.3

Used in events: HTTP_REQUEST / AUTH_RESULT

## HTTP:uri_set(value)

Sets the HTTP request URI to the string <value>.

### Syntax

HTTP:uri_set(value);

### Arguments

| Name | Description |
|---|---|
| value | a string which specifices the uri. |

### Example

```
when HTTP_REQUEST {
HTTP:uri_set("/index.html?para=xxxx");
}
```

FortiADC version: V4.3

Used in events: HTTP_REQUEST

## HTTP:query_get()

Returns the string of the HTTP request query.

### Syntax

HTTP:query_get();

### Arguments

N/A

### Example

```
when HTTP_REQUEST {
query = HTTP:query_get();
}
```

FortiADC version: V4.3

Used in events: HTTP_REQUEST / AUTH_RESULT

## HTTP:query_set(value)

Sets the HTTP request query to the string <value>.

### Syntax

HTTP:query_set(value);

### Arguments

| Name | Description |
|---|---|
| value | A string which specifies the uri. |

### Example

```
when HTTP_REQUEST {
HTTP:query_set(“query1=value1”);
}
```

FortiADC version: V4.3

Used in events: HTTP_REQUEST / AUTH_RESULT

## HTTP:redirect(“url”, …)

Redirects an HTTP request or response to the specified URL.

### Syntax

HTTP:redirect(“url”, …);

### Arguments

| Name | Description |
|---|---|
| url | A string which specifies the redirect url. |

### Example

```
when HTTP_REQUEST {
Host = HTTP:header_get_value("host")
Path = HTTP:path_get()
HTTP:redirect("https://%s%s", Host, Path);
}
```

FortiADC version: V4.3

Used in events: HTTP_REQUEST / HTTP_DATA_REQUEST / HTTP_RESPONSE

Cannot use in HTTP_DATA_RESPONSE

## HTTP:redirect_with_cookie(url, cookie)

Redirects an HTTP request or response to the specified URL with cookie.

Supports multiple redirect

### Syntax

HTTP:redirect_with_cookie(url, cookie);

### Arguments

| Name | Description |
|---|---|
| url | A string which specifies the redirect url. |
| cookie | A string as cookie. |

### Example

```
HTTP:redirect_with_cookie("www.example.com", "server=nginx")
HTTP:redirect_with_cookie("www.abc.com", "server=nginx")
```

**Note:** The final redirection is to www.abc.com with the cookie "server=nginx".

FortiADC version: V4.8

Used in events: HTTP_REQUEST / HTTP_DATA_REQUEST / HTTP_RESPONSE

Cannot use in HTTP_DATA_RESPONSE

## HTTP:redirect_t(t)

Redirects an HTTP request or response to the URL specified in the table.

Supports multiple redirect, same as HTTP:redirect_with_cookie().

### Syntax

HTTP:redirect_t(t);

### Arguments

| Name | Description |
|---|---|
| t | A table that defines the code, redirect url, and cookie. |

### Example

```
when HTTP_RESPONSE{
a={}    --initialize a table
a["code"]=303;
a["url"]="www.example.com"
a["cookie"]="test:server"
HTTP:redirect_t(a)
debug("redirected\n")
}
```

**Note:**

if code not set, default code in http redirect response is 302

if URL is missing in the input table, then a log will be generated!

FortiADC version: V4.8

Used in events: HTTP_REQUEST / HTTP_DATA_REQUEST / HTTP_RESPONSE

Cannot use in HTTP_DATA_RESPONSE

## HTTP:version_get()

Returns the HTTP version of the request or response.

### Syntax

HTTP:version_get();

### Arguments

N/A

### Example

```
when HTTP_REQUEST {
v = HTTP:version_get();
}
```

FortiADC version: V5.8

Used in events: HTTP_REQUEST / HTTP_RESPONSE

## HTTP:version_set(value)

Sets the HTTP request or response version to the string <value>.

### Syntax

HTTP:version_set(value);

### Arguments

| Name | Description |
|---|---|
| value | A string which specifies the version. |

### Example

```
when HTTP_REQUEST {
HTTP:version_set(“HTTP2.0”);
}
```

FortiADC version: V4.8

Used in events: HTTP_REQUEST / HTTP_RESPONSE

## HTTP:status_code_get()

Returns the response status code output as string.

### Syntax

HTTP:status_code_get();

Arguments: N/A

### Example

```
when HTTP_RESPONSE {
code = HTTP:status_code_get();
}
```

FortiADC version: V4.8

Used in events: HTTP_RESPONSE

## HTTP:status_code_set(value)

Sets the HTTP response status code.

### Syntax

HTTP:status_code_set(value);

### Arguments

| Name | Description |
|---|---|
| value | A string which specifies the status code. |

### Example

```
when HTTP_RESPONSE{
HTTP:status_code_set("304");
}
```

FortiADC version: V4.8

Used in events: HTTP_RESPONSE

## HTTP:code_get()

Returns the response status code, output as integer.

### Syntax

HTTP:code_get();

### Arguments

N/A

### Example

```
when HTTP_REQUEST {
code = HTTP:code_get();
}
```

FortiADC version: V4.8

Used in events: HTTP_RESPONSE

## HTTP:code_set(integer)

Sets the response status code.

### Syntax

HTTP:code_set(integer);

### Arguments

| Name | Description |
|---|---|
| integer A | Integer which specifies the status code. |

### Example

```
when HTTP_REQUEST {
HTTP:coe_set(503);
}
```

FortiADC version: V4.8

Used in events: HTTP_RESPONSE

## HTTP:reason_get()

Returns the response reason.

### Syntax

HTTP:reason_get();

### Arguments

N/A

### Example

```
when HTTP_RESPONSE {
reason = HTTP:reason_get();
}
```

FortiADC version: V4.8

Used in events: HTTP_RESPONSE

## HTTP:reason_set(value)

Sets the response reason.

### Syntax

HTTP:reason_set(value);

### Arguments

| Name | Description |
|---|---|
| value | A string which specifies the response reason. |

### Example

```
when HTTP_RESPONSE {
HTTP:reason_set("Not exist !");
}
```

FortiADC version: V4.8

Used in events: HTTP_RESPONSE

## HTTP:client_addr()

Returns the client IP address of a connection.

For HTTP_REQUEST packet, it's source address; for HTTP_RESPONSE packet, it's destination address.

### Syntax

HTTP:client_addr();

### Arguments

N/A

### Example

```
when HTTP_REQUEST priority 100 {
cip=HTTP:client_addr()
}
```

FortiADC version: V4.6

Used in events: HTTP_REQUEST / HTTP_RESPONSE

## HTTP:local_addr()

For HTTP_REQUEST, returns the IP address of the virtual server the client is connected to; for HTTP_RESPONSE, returns the incoming interface IP address of the return packet.

### Syntax

HTTP:local_addr();

### Arguments

N/A

### Example

```
when HTTP_REQUEST {
lip = HTTP:local_addr()
}
```

FortiADC version: V4.6

Used in events: HTTP_REQUEST / HTTP_RESPONSE

## HTTP:server_addr()

Returns the IP address of the server in HTTP_RESPONSE.

### Syntax

HTTP:server_addr();

### Arguments

N/A

### Example

```
when HTTP_RESPONSE {
sip = HTTP:server_addr()
}
```

FortiADC version: V4.6

Used in events: HTTP_RESPONSE

## HTTP:remote_addr()

Returns the IP address of the host on the far end of the connection

### Syntax

HTTP:remote_addr();

### Arguments

N/A

### Example

```
when HTTP_REQUEST {
rip = HTTP:remote_addr()
}
```

FortiADC version: V4.6

Used in events: HTTP_REQUEST / HTTP_RESPONSE

## HTTP:client_port()

Returns the real client port number in a string format.

### Syntax

HTTP:client_port();

### Arguments

N/A

### Example

```
when HTTP_REQUEST {
string1=HTTP:client_port()
string2=HTTP:local_port()
string3=HTTP:remote_port()
debug("result_client_port: %s \n",string1)
debug("result_local_port: %s \n",string2)
debug("result_remote_port: %s \n",string3)
}
when HTTP_RESPONSE {
debug("SERVER_side: \n")
string4=HTTP:server_port()
debug("result_server_port: %s \n",string4)
string5=HTTP:client_port()
string6=HTTP:local_port()
string7=HTTP:remote_port()
debug("result_client_port: %s \n",string5)
debug("result_local_port: %s \n",string6)
debug("result_remote_port: %s \n",string7)
}
```

FortiADC version: V4.8

Used in events: HTTP_REQUEST / HTTP_RESPONSE / HTTP_DATA_REQUEST / HTTP_DATA_RESPONSE

## HTTP:local_port()

Returns the local port number in a string format.

In HTTP_REQUEST, local_port is the virtual server port.

In HTTP_RESPONSE, local_port is the port of the gateway that was used to connect.

### Syntax

HTTP:local_port();

### Arguments

N/A

### Example

```
when HTTP_REQUEST {
string1=HTTP:client_port()
string2=HTTP:local_port()
string3=HTTP:remote_port()
debug("result_client_port: %s \n",string1)
debug("result_local_port: %s \n",string2)
debug("result_remote_port: %s \n",string3)
}
```

FortiADC version: V4.8

Used in events: HTTP_REQUEST / HTTP_RESPONSE / HTTP_DATA_REQUEST / HTTP_DATA_RESPONSE

## HTTP:remote_port()

Returns the remote port number in a string format.

In HTTP_REQUEST, remote_port is the client port.

In HTTP_RESPONSE, remote_port is the real server port.

### Syntax

HTTP:remote_port();

### Arguments

N/A

### Example

```
when HTTP_REQUEST {
string1=HTTP:client_port()
string2=HTTP:local_port()
string3=HTTP:remote_port()
debug("result_client_port: %s \n",string1)
debug("result_local_port: %s \n",string2)
debug("result_remote_port: %s \n",string3)
}
```

FortiADC version: V4.8

Used in events: HTTP_REQUEST / HTTP_RESPONSE / HTTP_DATA_REQUEST / HTTP_DATA_RESPONSE

## HTTP:server_port()

Returns the real server port number in a string format.

### Syntax

HTTP:server_port();

### Arguments

N/A

### Example

```
when HTTP_RESPONSE {
debug("SERVER_side: \n")
string4=HTTP:server_port()
debug("result_server_port: %s \n",string4)
string5=HTTP:client_port()
string6=HTTP:local_port()
string7=HTTP:remote_port()
debug("result_client_port: %s \n",string5)
debug("result_local_port: %s \n",string6)
debug("result_remote_port: %s \n",string7)
}
```

FortiADC version: V4.8

Used in events: HTTP_RESPONSE / HTTP_DATA_RESPONSE

## HTTP:client_ip_ver()

Returns the client IP version number. This can be used to get IPv4 or IPv6 versions.

### Syntax

HTTP:client_ip_ver();

### Arguments

N/A

### Example

```
when HTTP_REQUEST {
string=HTTP:client_ip_ver()
debug("\nresult: %s \n",string)
}
when HTTP_RESPONSE{
string=HTTP:client_ip_ver()
debug("\nresult: %s \n",string)
}
```

FortiADC version: V4.8

Used in events: HTTP_REQUEST / HTTP_RESPONSE / HTTP_DATA_REQUEST / HTTP_DATA_RESPONSE

## HTTP:server_ip_ver()

Returns the server IP version number. This can be used to get IPv4 or IPv6 versions.

### Syntax

HTTP:server_ip_ver();

### Arguments

N/A

### Example

```
when HTTP_REQUEST {
string=HTTP:server_ip_ver()
debug("\nresult: %s \n",string)
}
```

FortiADC version: V4.8

Used in events: HTTP_RESPONSE / HTTP_DATA_RESPONSE

## HTTP:close()

Close an HTTP connection using code 503.

Can support multiple close calls.

### Syntax

HTTP:close();

### Arguments

N/A

### Example

```
when HTTP_REQUEST {
Example1:
HTTP:close in script 1
HTTP:close in script 2
ps: it will send the close message to client correctly
Example2:
HTTP:close()
HTTP:redirect_with_cookie("www.example.com","server=nginx")
ps:the client get the redirect message, the close message is overwritten
HTTP:close()                --close http connection using code 503
```

FortiADC version: V4.6

Used in events: HTTP_REQUEST / HTTP_RESPONSE

## HTTP:respond(t)

Allows you to return a customized page and send out an HTTP response directly from FortiADC.

### Syntax

HTTP:respond(t);

### Arguments

| Name | Description |
|---|---|
| t | A table which will give the response code and content. |

### Example

```
when HTTP_REQUEST {
tt={}
tt["code"] = 200;
tt["content"] = "HTTP/1.1 200 OK\r\nConnection: close\r\nContent-Type:
text/plain\r\n\r\nXXXXX Test Page XXXXXXX";
status = HTTP:respond(tt);
debug("HTTP_respond() status: %s\n", status);
}
```

FortiADC version: V5.2

Used in events:

- HTTP_REQUEST / HTTP_DATA_REQUEST
- HTTP_RESPONSE supported since V7.2.4 and V7.4.1

## HTTP:get_session_id()

FortiADC will assign each session a unique ID and allow the user to get this unique ID through this function.

With this unique ID, the user can use Lua scripts to capture request headers, store them into a global variable indexed by this unique ID, and then index a global variable using this unique ID to extract its own request header information in the HTTP request event.

### Syntax

HTTP:get_session_id();

### Arguments

N/A

### Example

```
When RULE_INIT{
Env={}
}
when HTTP_REQUEST{
id=HTTP:get_session_id()
debug("session id %d\n", id);
env[id]=nil
req={}
req["url"]=HTTP:uri_get()
req["method"]=HTTP:method_get()
env[id]=req
}
when HTTP_RESPONSE{
id=1
request=env[id]
if req then
debug("session id %d and url %s\n", id,request["url"]);
debug("session id %d and method %s\n", id,request["method"]);
end
}
Output:
session id 1
session id 1 and url /index.html
session id 1 and method GET
```

FortiADC version: V4.8

Used in events: HTTP_REQUEST / HTTP_RESPONSE / HTTP_DATA_REQUEST / HTTP_DATA_RESPONSE

## HTTP:rand_id()

Returns a random string of 32 characters long in hex format.

### Syntax

HTTP:rand_id();

### Arguments

N/A

### Example

```
when HTTP_REQUEST {
id = HTTP:rand_id();
debug("random id: %s\n", id)
}
```

FortiADC version: V4.8

Used in events: ALL

## HTTP:set_event(t)

Sets a request or response event to enable or disable.

### Syntax

HTTP:set_event(t);

### Arguments

| Name | Description |
|---|---|
| t | A table that specifies when to enable/disable an event. |

### Example

```
when HTTP_REQUEST {
t={};
t["event"] = "data_res";
t["operation"] = "disable";
HTTP:set_event(t);
}
```

**Note:**

The event can be "req", "res", "data_req", "data_res". And the operation can be "enable" and "disable". This command will generate a log if the event or operation is wrong.

FortiADC version: V4.8

Used in events: HTTP_REQUEST / HTTP_RESPONSE

## HTTP:set_auto()

Sets an automatic request or response event.

In HTTP keep-alive mode, by default, FortiADC will automatically re-enable both HTTP_REQUEST and HTTP_RESPONSE event processes for the next transaction even if they have been disabled in the current transaction. Users can disable/enable this automatic behavior using this function.

### Syntax

HTTP:set_auto(t);

### Arguments

| Name | Description |
|---|---|
| t | A table which specifies the event and operation. |

### Example

```
when HTTP_REQUEST {
t={};
t["event"] = "data_req";
t["operation"] = "enable";
HTTP:set_auto(t);
}
```

**Note:**

The event can be "req", "res", "data_req", "data_res". And the operation can be "enable" and "disable". This command will generate a log if the event or operation is wrong.

FortiADC version: V4.8

Used in events: HTTP_REQUEST/HTTP_RESPONSE/HTTP_DATA_REQUEST/HTTP_DATA_RESPONSE/BEFORE_AUTH/AUTH_RESULT/COOKIE_BAKE WAF_REQUEST_BEFORE_SCAN/WAF_RESPONSE_BEFORE_SCAN/WAF_REQUEST_ATTACK_DETECTED/WAF_RESPONSE_ATTACK_DETECTED

## HTTP:get_unique_transaction_id()

Returns a unique ID per transaction in history. Each ID is a string consisting of 32 hex digits, for example "b0b9fec0b4b28306a2bf4f63eb97520e".

### Syntax

HTTP:get_unique_session_id();

### Arguments

N/A

### Example

```
when HTTP_REQUEST {
   uid=HTTP:get_unique_session_id()
   debug("Unique session id %s\n", uid);
}
```

FortiADC version: V7.4

Used in events: HTTP_REQUEST / HTTP_RESPONSE / HTTP_DATA_REQUEST / HTTP_DATA_RESPONSE

## HTTP:get_unique_session_id()

Returns a unique ID per session in history. Each ID is a string consisting of 32 hex digits, for example "b6e49ca3c937baaa47f2d111f593be8b".

### Syntax

HTTP:get_unique_session_id();

### Arguments

N/A

### Example

```
when HTTP_REQUEST {
   uid=HTTP:get_unique_session_id()
   debug("Unique session id %s\n", uid);
}
```

FortiADC version: V7.4

Used in events:

- HTTP events — HTTP_REQUEST/HTTP_RESPONSE/HTTP_DATA_REQUEST/HTTP_DATA_RESPONSE/BEFORE_AUTH
- WAF events — WAF_REQUEST_BEFORE_SCAN/WAF_RESPONSE_BEFORE_SCAN/WAF_REQUEST_ATTACK_DETECTED/WAF_RESPONSE_ATTACK_DETECTED

## HTTP:get_unique_transaction_id()

Returns a unique ID per transaction in history. Each ID is a string consisting of 32 hex digits, for example "b0b9fec0b4b28306a2bf4f63eb97520e".

### Syntax

HTTP:get_unique_transaction_id();

### Arguments

N/A

### Example

```
when HTTP_REQUEST {
   tuid=HTTP:get_unique_transaction_id()
   debug("Unique transaction id %s\n", tuid);
}
```

FortiADC version: V7.4

Used in events:

- HTTP events — HTTP_REQUEST/HTTP_RESPONSE/HTTP_DATA_REQUEST/HTTP_DATA_RESPONSE/BEFORE_AUTH
- WAF events — WAF_REQUEST_BEFORE_SCAN/WAF_RESPONSE_BEFORE_SCAN/WAF_REQUEST_ATTACK_DETECTED/WAF_RESPONSE_ATTACK_DETECTED

## HTTP DATA commands

If the HTTP data exceeds the 1.25M buffer size limit, FortiADC will only collect up to the 1.25M limit of data and forward the excess data directly.

HTTP:collect() on page 71 — Collects body data from the HTTP request or response. You may specify a specific amount using the length argument.

HTTP:payload(size) on page 72 — Returns the size of the buffered content. The returned value is an integer.

HTTP:payload(content) on page 72 — Returns the buffered content in a string.

HTTP:payload(set) on page 73 — Inserts the specified data at the specified location.

HTTP:payload(find) on page 73 — Searches for a particular string or a regular expression on the buffered data.

HTTP:payload(remove) on page 74 — Removes a particular string or a regular expression from the buffered data.

HTTP:payload(replace) on page 75 — Replaces a particular string or regular expression with a new string.

### HTTP:collect()

Collects body data from the HTTP request or response. You may specify a specific amount using the length argument.

#### Syntax

HTTP:collect(t);

#### Arguments

| Name | Description |
|---|---|
| t | A table which specifies the data size to collect. |

#### Example

```
when HTTP_RESPONSE{
debug("Start\n")
t={}
t["size"]=10
HTTP:collect(t)
debug("Done\n")
}
when HTTP_DATA_RESPONSE{
table={}
table["operation"]="size"
ret=HTTP:payload(table)
debug("Size: %d \n",ret)
}
```

**Note:** The "size" refers to the httproxy block size, which is limited by the FortiADC HARD LIMIT.

FortiADC version: V4.8

Used in events: HTTP_REQUEST / HTTP_RESPONSE

## HTTP:payload(size)

Returns the size of the buffered content. The returned value is an integer.

### Syntax

HTTP:payload(t);

### Arguments

| Name | Description |
|---|---|
| t | A table which specifies the operation-size of the request/response data. |

### Example

```
when HTTP_DATA_RESPONSE{
t1={}
t1["operation"]="size"
sz=HTTP:payload(t1)
debug("----response data size: %d-----\n", sz)}
```

FortiADC version: V4.8

Used in events: HTTP_DATA_REQUEST / HTTP_DATA_RESPONSE

## HTTP:payload(content)

Returns the buffered content in a string.

### Syntax

HTTP:payload(str);

### Arguments

| Name | Description |
|---|---|
| str | A string which will be calculated. |

### Example

```
when HTTP_DATA_REQUEST {
t={};
t[“operation”]=”content”;   --return the buffered content
t[“offset”]=12;
t[“size”]=20;
ct = HTTP:payload(t);   --return value is a string containing the buffered content
}
```

**Note:** The “offset” and “size” fields are optional. If the “offset” field is missing, zero is assumed. If the “size” field is missing, it will operate on the whole buffered data.

FortiADC version: V4.8

Used in events: HTTP_DATA_REQUEST / HTTP_DATA_RESPONSE

## HTTP:payload(set)

Inserts the specified data at the specified location.

### Syntax

HTTP:payload(t);

### Arguments

| Name | Description |
|---|---|
| t | A table which specifies the parameters: offset, size, data to set. |

### Example

```
when HTTP_DATA_REQUEST {
t={};
t[“operation”]=”set” --replace the buffered content by new data
t[“offset”]=12;
t[“size”]=20;
t[“data”]= ”new data to insert”;
ret = HTTP:payload(t); --return value is boolean: false if fail, true if succeed
}
```

**Note:** The “offset” and “size” fields are optional. If the “offset” field is missing, zero is assumed. If the “size” field is missing, it will operate on the whole buffered data.

FortiADC version: V4.8

Used in events: HTTP_DATA_REQUEST / HTTP_DATA_RESPONSE

## HTTP:payload(find)

Searches for a particular string or a regular expression on the buffered data.

### Syntax

HTTP:payload(t);

### Arguments

| Name | Description |
|---|---|
| t | A table which specifies the parameters: data, offset, size, scope. |

### Example

```
when HTTP_DATA_REQUEST {
t={};
t["operation"]="find"
t["data"]="sth"; -- can be a regular expression, like (s.h)
t["offset"]=12;
t["size"]=20;
t["scope"]="first" -- the scope field can be either "first" or "all"
ct = HTTP:payload(t);  --return value is a boolean false if operation fail or the number of
occurrences found;
}
```

**Note:** The “offset” and “size” fields are optional. If the “offset” field is missing, zero is assumed. If the “size” field is missing, it will operate on the whole buffered data. The “scope” field can be either be “first” or “all”.

FortiADC version: V4.8

Used in events: HTTP_DATA_REQUEST / HTTP_DATA_RESPONSE

## HTTP:payload(remove)

Removes a particular string or a regular expression from the buffered data.

### Syntax

HTTP:payload(t);

### Arguments

| Name | Description |
|---|---|
| t | A table which specifies the parameters: data to remove, offset, size, scope. |

### Example

```
when HTTP_DATA_REQUEST {
t={};
t["operation"]="remove"
t["data"]="sth"; -- can be a regular expression, like (s.h)
t["offset"]=12;
t[“size”]=20;
t[“scope”]=”first” --"first" or “all”
ct = HTTP:payload(t);  --return value is a boolean false if operation fail or the number of
occurrences removed
}
```

FortiADC version: V4.8

Used in events: HTTP_DATA_REQUEST / HTTP_DATA_RESPONSE

## HTTP:payload(replace)

Replaces a particular string or regular expression with a new string.

### Syntax

HTTP:payload(t);

### Arguments

| Name | Description |
|---|---|
| t | A table which specifies the parameters: data to replace, new_data, offset, size, scope. |

### Example

```
when HTTP_DATA_REQUEST {
t={};
t[“operation”]=”replace”
t[“data”]=”sth”; -- can be a regular expression, like (s.h)
t[“new_data”]=”sth new”; --“new_data” field is needed for the “replace” operation.
t[“offset”]=12;
t[“size”]=20;
t[“scope”]=”first” -- or “all”
ct = HTTP:payload(t);  --return value is a boolean false if operation fail or the number of
occurrences replaced
}
```

FortiADC version: V4.8

Used in events: HTTP_DATA_REQUEST / HTTP_DATA_RESPONSE

# Cookie commands

Part of the HTTP class, Cookie commands manipulate cookie operation:

HTTP:cookie_list() on page 76 — Returns a list of cookies: their names and values.

HTTP:cookie(t) on page 76 — Allows you to get/set cookie value and cookie attribute, remove a whole cookie, get the whole cookie in HTTP_RESPONSE, and insert a new cookie.

HTTP:cookie_crypto(t) on page 77 — The provided function response_encrypt_cookie can be used to perform cookie encryption in HTTP_RESPONSE and request_decrypt_cookie can be used to perform cookie decryption in HTTP_REQUEST.

## HTTP:cookie_list()

Returns a list of cookies: their names and values.

### Syntax

HTTP:cookie_list();

### Arguments

N/A

### Example

```
when HTTP_REQUEST {
ret=HTTP:cookie_list()
for k,v in pairs(ret) do
debug("cookie name %s, value %s\n", k,v);
end
}
```

FortiADC version: before V5.0

Used in events: HTTP_REQUEST / HTTP_RESPONSE

## HTTP:cookie(t)

Allows you to get/set cookie value and cookie attribute, remove a whole cookie, get the whole cookie in HTTP_RESPONSE, and insert a new cookie.

### Syntax

HTTP:cookie(t);

t = {}

t[“name”] = “name”

t[“parameter”] = {can be value, cookie, path, domain, expires, secure, maxage, max-age, httponly, version, port, attrname }

t[“value”] = value

t[“case_sensitive”] = {0 or 1}

t[“action”] = {can be get, set, remove, insert}

ret = HTTP:cookie(t) --return is true if succeed or false if failed

### Arguments

| Name | Description |
|---|---|
| t | A table which specifies cookie name, parameter, action. |

### Example

```
when HTTP_REQUEST {
t={};
t["name"]="test"
t["parameter"]="value";--value, cookie, path, domain, expires,  secure, maxage, max-age,
httponly,  version, port, attrname,
t["action"]="get"--get, set, remove, insert
ret = HTTP:cookie(t)
if ret then
debug("get cookie value succeed %s\n",ret);
else
debug("get cookie value failed\n");
end
}
```

**Note:**

- name: specify cookie name
- parameter: specify cookie value and attribute, including value, cookie, path, domain, expires, secure, maxage, max-age, httponly, version, port
- action: can be get, set, remove, insert

FortiADC version: before V5.0

Used in events: HTTP_REQUEST / HTTP_RESPONSE

## HTTP:cookie_crypto(t)

The provided function response_encrypt_cookie can be used to perform cookie encryption in HTTP_RESPONSE and request_decrypt_cookie can be used to perform cookie decryption in HTTP_REQUEST.

### Syntax

HTTP:cookie_crypto(t);

## Arguments

| Name | Description |
|---|---|
| t | A table which specifies cookie name, parameter, action. |

## Example

```
when HTTP_REQUEST {
--encrypt cookie "test" in HTTP REQUEST before forwarding to real servers
local t={};
t["name"]="cookiename"
t["action"]="encrypt"      --encrypt, or decrypt
t["key"]="0123456789ABCDEF";
t["prefix"]="XXXX";
t["size"]=size-- 128, 192, or 256, the corresponding key length is 16, 24, and 32
if  HTTP:cookie_crypto(t) then
debug("Encrypt cookie succeed\n");
else
debug("Encrypt cookie failed\n");
end
}
```

**Note:**

- name: specify cookie name
- action: can be encrypt, or decrypt
- size: can be 128, 192, or 256, the corresponding key length is 16, 24, and 32

FortiADC version: before V5.2

Used in events: HTTP_REQUEST / HTTP_RESPONSE

## Authentication commands

Authentication (AUTH) commands contain functions related to authentication and login:

AUTH:get_baked_cookie() on page 79 — Allows you to retrieve the baked cookie.

AUTH:set_baked_cookie(cookie) on page 80 — Allows you to customize the cookie attribute of the baked cookie.

AUTH:on_off() on page 81 — Returns whether authentication is required or not.

AUTH:success() on page 82 — Returns whether authentication is successful or not.

AUTH:form_based() on page 83 — Returns whether the authentication is HTTP form based or not.

AUTH:user() on page 83 — Returns the user name in the authentication.

AUTH:pass() on page 84 — Returns the password in the authentication.

AUTH:usergroup() on page 84 — Returns the usergroup which the user belong to.

AUTH:realm() on page 84 — Returns the realm in the authentication.

AUTH:host() on page 85 — Returns the host in the authentication.

AUTH:set_usergroup() on page 85 — Sets a new user group that is configured in the current authentication policy.

## AUTH:get_baked_cookie()

Allows you to retrieve the baked cookie.

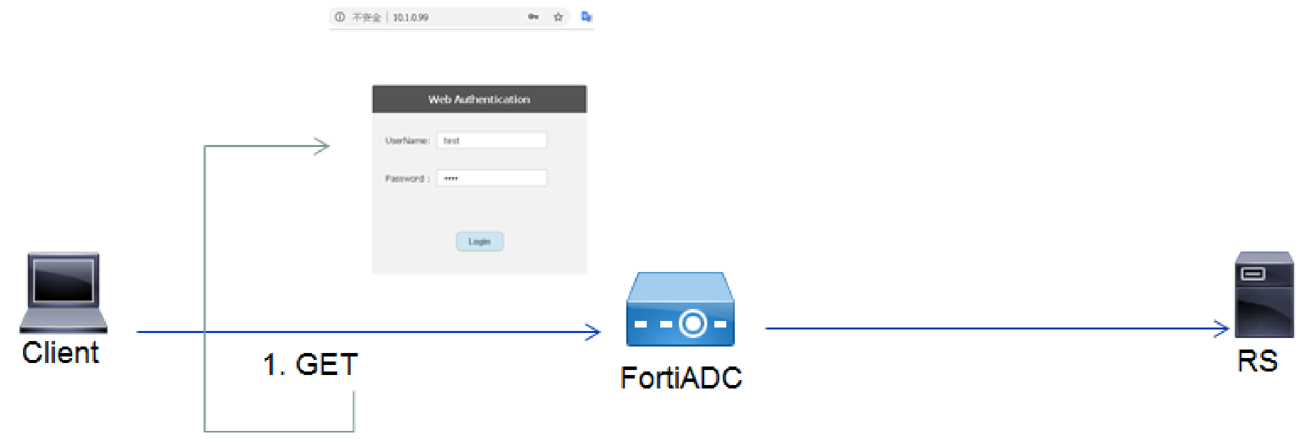
### Syntax

AUTH:get_baked_cookie();

### Arguments

N/A

### Example



```
when COOKIE_BAKE {
cookie = AUTH:get_baked_cookie()
debug("Get cookie: %s\r\n", cookie)
}
Result:
Get cookie: Set-Cookie:
FortiADCauthSI=lfGnC2gsl7xsbAg4JFs94e4CJfFXaP3U5z6QHvo7n08cCoT5MdtQog2LmcizPo3aRiBHY/RThhocq
G+DdnvsCLFJh3nBUoLeuYjGK9lY5L4=|W86hXGg; expires=Tue 23 Oct 2018 04:19:45 GMT;
domain=10.1.0.99; path=/
```

FortiADC version: V5.2

Used in events: AUTH_RESULT

## AUTH:set_baked_cookie(cookie)

Allows you to customize the cookie attribute of the baked cookie.

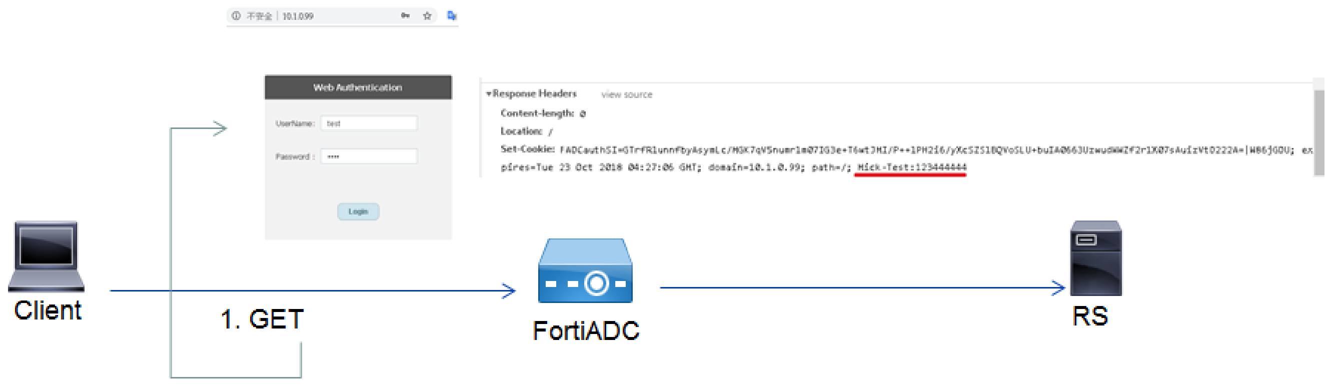
### Syntax

AUTH:set_baked_cookie(cookie);

### Arguments

| Name | Description |
|---|---|
| cookie | A string which specifies the baked cookie. |

### Example



```
▾Response Headers    view source
   Content-length: 0
   Location: /
   Set-Cookie: FADCauthSI=GTrfR1unnfbyAsymLc/MGK7qV5numr1m07IG3e+T6wtJHI/P++1PH2i6/yXcSZS1BQVoSLU+buIA0663UzwudWWZf2r1X07sAuizVtD222A=|W86jGDU; ex
   pires=Tue 23 Oct 2018 04:27:06 GMT; domain=10.1.0.99; path=/; Mick-Test:123444444
```

```
when COOKIE_BAED {
cookie = AUTH:get_baked_cookie()
new_cookie = cookie..”; Mick-Test:123444444”
status = AUTH:set_baked_cookie(new_cookie)
debug(“Set baked cookie, status: %s\n”, status)
}
Result:
Set baked cookie, status: true
```

FortiADC version: V5.2

Used in events: AUTH_RESULT

## AUTH:on_off()

Returns whether authentication is required or not.

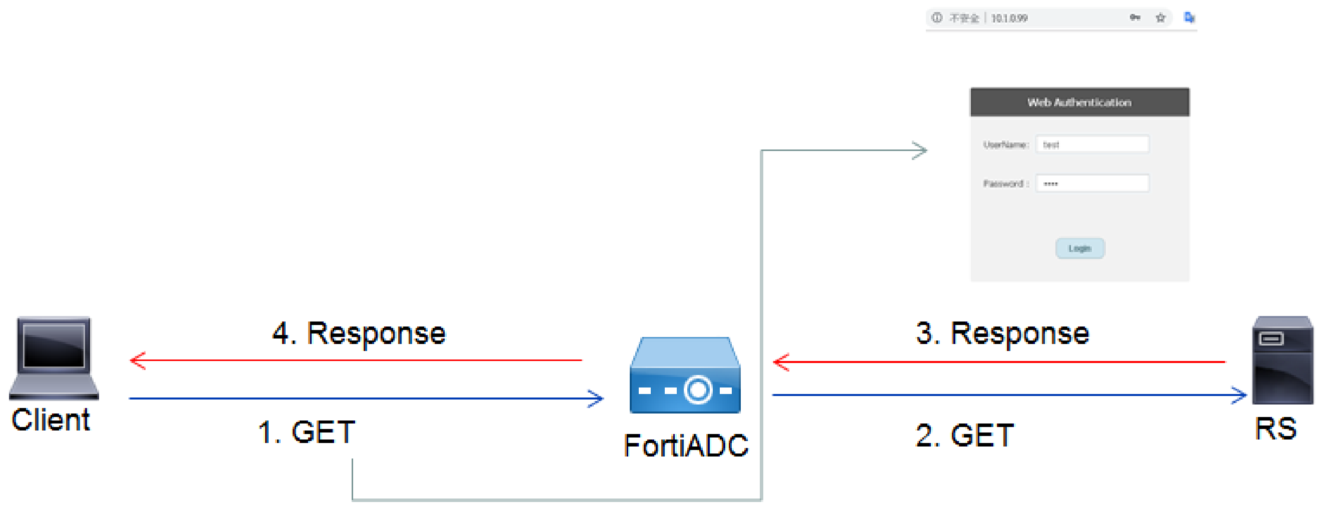
### Syntax

AUTH:on_off();

### Arguments

N/A

### Example



```
when AUTH_RESULT {
on_off = AUTH:on_off()
succ = AUTH:success()
fm = AUTH:form_based()
user = AUTH:user()
pass = AUTH:pass()
userg = AUTH:usergroup()
realm = AUTH:realm()
host = AUTH:host()
debug("authentication form based %s, on_off %s, success %s, the user %s, pass %s, realm %s,
the usergroup %s, host %s\n", fm, on_off, succ, the user, pass, realm, the userg, host)
}
Result:
authentication form based true, on_off true, success true, the user test, pass test, realm
Form333333, the userg test, host 10.1.0.99
```

FortiADC version: V5.2

Used in events: AUTH_RESULT / HTTP_REQUEST / HTTP_DATA_REQUEST / HTTP_RESPONSE / HTTP_DATA_RESPONSE

## AUTH:success()

Returns whether authentication is successful or not.

### Syntax

AUTH:success();

### Arguments

N/A

### Example

Please refer to command AUTH:on_off() example.

FortiADC version: V5.2

Used in events: AUTH_RESULT / HTTP_REQUEST / HTTP_DATA_REQUEST / HTTP_RESPONSE / HTTP_DATA_RESPONSE

## AUTH:form_based()

Returns whether the authentication is HTTP form based or not.

### Syntax

AUTH:form_based();

### Arguments

N/A

### Example

Please refer to command AUTH:on_off() example.

FortiADC version: V5.2

Used in events: AUTH_RESULT / HTTP_REQUEST / HTTP_DATA_REQUEST / HTTP_RESPONSE / HTTP_DATA_RESPONSE

## AUTH:user()

Returns the user name in the authentication.

### Syntax

AUTH:user();

### Arguments

N/A

### Example

Please refer to command AUTH:on_off() example.

FortiADC version: V5.2

Used in events: AUTH_RESULT / HTTP_REQUEST / HTTP_DATA_REQUEST / HTTP_RESPONSE / HTTP_DATA_RESPONSE

## AUTH:pass()

Returns the password in the authentication.

### Syntax

AUTH:pass();

### Arguments

N/A

### Example

Please refer to command AUTH:on_off() example.

FortiADC version: V5.2

Used in events: AUTH_RESULT / HTTP_REQUEST / HTTP_DATA_REQUEST / HTTP_RESPONSE / HTTP_DATA_RESPONSE

## AUTH:usergroup()

Returns the user group which the user belong to.

### Syntax

AUTH:usergroup();

### Arguments

N/A

### Example

Please refer to command AUTH:on_off() example.

FortiADC version: V5.2

Used in events: AUTH_RESULT / HTTP_REQUEST / HTTP_DATA_REQUEST / HTTP_RESPONSE / HTTP_DATA_RESPONSE

## AUTH:realm()

Returns the realm in the authentication.

### Syntax

AUTH:realm();

### Arguments

N/A

### Example

Please refer to command AUTH:on_off() example.

FortiADC version: V5.2

Used in events: AUTH_RESULT / HTTP_REQUEST / HTTP_DATA_REQUEST / HTTP_RESPONSE / HTTP_DATA_RESPONSE

## AUTH:host()

Returns the host in the authentication.

### Syntax

AUTH:host();

### Arguments

N/A

### Example

Please refer to command AUTH:on_off() example.

FortiADC version: V5.2

Used in events: AUTH_RESULT / HTTP_REQUEST / HTTP_DATA_REQUEST / HTTP_RESPONSE / HTTP_DATA_RESPONSE

## AUTH:set_usergroup()

Sets a new user group that is configured in the current authentication policy. A new realm can also be set at the same time. It returns true if successful, otherwise, false. A realm name and a user group name are needed as input parameters.

The user group specified by the function must be in the authentication policy referenced by the VS. The result specified by the new user group will override the authentication result of the original authentication policy.

### Syntax

AUTH:set_usergroup(“RealmName”, "UserGroupName");

## Arguments

| Name | Description |
|---|---|
| RealmName | The name of the new realm to be set. (Lua string with maximum length of 63). |
| UserGroupName | The name of the user group to be set. (Lua string with maximum length of 63, must also comply with original definition of user group). |

## Example

```
when BEFORE_AUTH {
    r = AUTH:set_usergroup(“Realm02”, "UserGroup02");
   debug("set_usergroup successfully? %s\n", tostring(r));
}
```

FortiADC version: V7.2

Used in events: BEFORE_AUTH

# SSL commands

SSL commands contain functions for obtaining SSL related information, such as obtaining certificates and SNI:

SSL:cipher() on page 87 — Returns the cipher in the handshake.

SSL:version() on page 88 — Returns the SSL version in the handshake.

SSL:alg_keysize() on page 88 — Returns the SSL encryption key size in the handshake.

SSL:client_cert() on page 89 — Returns the status of client-certificate-verify, whether or not it is enabled.

SSL:sni() on page 90 — Returns the SNI or false (if no SNI).

SSL:npn() on page 91 — Returns the next protocol negotiation string or false (if no NPN).

SSL:alpn() on page 91 — Allows you to get the SSL ALPN extension.

SSL:session(t) on page 92 — Allows you to get SSL session ID, reuse the session, or remove it from the cache.

SSL:cert(t) on page 92 — Allows you to get the certificate information between local or remote.

SSL:cert_der() on page 93 — Returns the DER certificate when the client enables verify certificate.

SSL:peer_cert(str) on page 94 — Returns the peer certificate.

## SSL:cipher()

Returns the cipher in the handshake.

### Syntax

SSL:cipher();

### Arguments

N/A

### Example

```
when CLIENTSSL_HANDSHAKE{
debug("client_handshake\n")
ci=SSL:cipher();
debug("Cipher: %s \n",ci);
}
Result: (if client send https request with cipher ECDHE-RSA-DES-CBC3-SHA)
Cipher: ECDHE-RSA-DES-CBC3-SHA
```

FortiADC version: V5.0

Used in events: CLIENTSSL_HANDSHAKE / SERVERSSL_HANDSHAKE

## SSL:version()

Returns the SSL version in the handshake.

### Syntax

SSL:version();

### Arguments

N/A

### Example

```
when CLIENTSSL_HANDSHAKE{
debug("client handshake\n")
ver=SSL:version();
debug("SSL Version: %s \n",ver);
}
Result: (client send https request with various version)
client handshake
SSL Version: TLSv1
or
client handshake
SSL Version: TLSv1.1
or
client handshake
SSL Version: TLSv1.2
or
client handshake
SSL Version: SSLv3
```

FortiADC version: V5.0

Used in events: CLIENTSSL_HANDSHAKE / SERVERSSL_HANDSHAKE

## SSL:alg_keysize()

Returns the SSL encryption key size in the handshake.

### Syntax

SSL:alg_keysize();

### Arguments

N/A

### Example

```
when CLIENTSSL_HANDSHAKE{
debug("client handshake\n")
ci=SSL:cipher();
key=SSL:alg_keysize();
debug("Cipher: %s\n",ci)
debug("Alg key size: %s \n",key);
}
Result: (client send https request with various ciphers)
client handshake
Cipher: ECDHE-RSA-RC4-SHA
Alg key size: 128
or
client handshake
Cipher: ECDHE-RSA-DES-CBC3-SHA
Alg key size: 168
or
client handshake
Cipher: EDH-RSA-DES-CBC-SHA
Alg key size: 56
or
client handshake
Cipher: ECDHE-RSA-AES256-GCM-SHA384
Alg key size: 256
```

FortiADC version: V5.0

Used in events: CLIENTSSL_HANDSHAKE / SERVERSSL_HANDSHAKE

## SSL:client_cert()

Returns the status of client-certificate-verify, whether or not it is enabled.

### Syntax

SSL:client_cert();

### Arguments

N/A

### Example

```
when CLIENTSSL_HANDSHAKE{
debug("client handshake\n")
cc=SSL:client_cert();
debug("Client cert: %s \n",cc);
}
```

**Result:**

1. If not verify certificate is not set:
   ```
   Debug output:
   client handshake
   Client cert: false
   ```
2. If enabled verify in client-ssl-profile:
   ```
   config system certificate certificate_verify
     edit "verify"
       config  group_member
         edit 2
           set ca-certificate ca6
         next
       end
     next
   end
   config load-balance client-ssl-profile
     edit "csp"
       set client-certificate-verify verify
     next
   end
   debug output:
   client handshake
   Client cert: true
   ```

FortiADC version: V5.0

Used in events: CLIENTSSL_HANDSHAKE / SERVERSSL_HANDSHAKE / CLIENTSSL_RENEGOTIATE / SERVERSSL_RENEGOTIATE

## SSL:sni()

Returns the SNI or false (if no SNI).

### Syntax

SSL:sni();

### Arguments

N/A

### Example

```
when CLIENTSSL_HANDSHAKE {
debug("client handshake\n")
cc=SSL:sni();
debug("SNI: %s \n",cc);
}
```

**Result:**

Enable sni in client-ssl-profile

```
config load-balance client-ssl-profile
  edit "csp"
```

```
      set client-sni-required enable
   next
end
```

1. Client sends HTTPS request without SNI:

   ```
   [root@NxLinux certs]# openssl s_client -connect 5.1.1.100:443
   Debug output:
   Client handshake
   SNI: false
   ```

2. Client sends HTTPS request with SNI:

   ```
   openssl s_client -connect 5.1.1.100:443 -servername 4096-rootca-rsa-server1
   debug output :
   client handshake
   SNI: 4096-rootca-rsa-server1
   ```

FortiADC version: V5.0

Used in events: CLIENTSSL_HANDSHAKE / SERVERSSL_HANDSHAKE / CLIENTSSL_RENEGOTIATE / SERVERSSL_RENEGOTIATE

## SSL:npn()

Returns the next protocol negotiation string or false (if no NPN).

### Syntax

SSL:npn();

### Arguments

N/A

### Example

```
when CLIENTSSL_HANDSHAKE {
npn = SSL:npn()
}
```

FortiADC version: V5.0

Used in events: CLIENTSSL_HANDSHAKE / SERVERSSL_HANDSHAKE / CLIENTSSL_RENEGOTIATE / SERVERSSL_RENEGOTIATE

## SSL:alpn()

Allows you to get the SSL ALPN extension.

### Syntax

SSL:alpn();

### Arguments

N/A

### Example

```
when CLIENTSSL_HANDSHAKE {
alpn = SSL:alpn()
}
```

FortiADC version: V5.0

Used in events: CLIENTSSL_HANDSHAKE / SERVERSSL_HANDSHAKE / CLIENTSSL_RENEGOTIATE / SERVERSSL_RENEGOTIATE

## SSL:session(t)

Allows you to get SSL session ID, reuse the session, or remove it from the cache.

### Syntax

SSL:session(t);

### Arguments

| Name | Description |
|---|---|
| t | A table which specifies the operation to the session. |

### Example

```
when CLIENTSSL_HANDSHAKE {
t={}
t["operation"] = "get_id";  --can be "get_id" or "remove" or "reused"
sess_Id = SSL:session(t)
if sess_id then
id = to_HEX(sess_id)
debug("client sess id %s\n", id)
else
sess_id = "FALSE"
end
}
```

FortiADC version: V5.0

Used in events: CLIENTSSL_HANDSHAKE / SERVERSSL_HANDSHAKE / CLIENTSSL_RENEGOTIATE / SERVERSSL_RENEGOTIATE

## SSL:cert(t)

Allows you to get the certificate information between local or remote.

### Syntax

SSL:cert(t);

### Arguments

| Name | Description |
|---|---|
| t | A table which specifies the certificate direction, and operation. |

### Example

```
when CLIENTSSL_HANDSHAKE{
debug("client handshake\n")
t={}
t["direction"]="remote";
t["operation"]="index";
t["idx"]=0;
t["type"]="info";
cert=SSL:cert(t)
if cert then
debug("client has cert\n")
end
for k,v in pairs(cert) do
if k=="serial_number" or k=="digest" then
debug("cert info name %s, value in HEX %s\n", k, to_HEX(v));
else
debug("cert info name %s, value %s\n", k, v);
end
end
}
```

**Note:**

- direction: local and remote. In CLIENTSSL_HANDSHAKE, local means FortiADC's cert, remote means client's cert.
- operation: index, count, issuer
- type: info, der, (pem)

This command returns a table that contains all the information in the certificate.
In the return, it contains: key_algorithm, hash, serial_number, not Before, not After, signature_algorithm, version, digest, issuer_name, subject_name, old_hash, pin-sha256, finger_print.

FortiADC version: V5.0

Used in events: CLIENTSSL_HANDSHAKE / SERVERSSL_HANDSHAKE / CLIENTSSL_RENEGOTIATE / SERVERSSL_RENEGOTIATE

## SSL:cert_der()

Returns the DER certificate when the client enables verify certificate.

### Syntax

SSL:cert_der();

### Arguments

N/A

### Example

```
when CLIENTSSL_HANDSHAKE{
debug("client handshake\n")
cder=SSL:cert_der();
--debug("cder in HEX %s\n", to_HEX(cder));
if cder then
cder_hex=b64_enc_str(cder);
debug("whole cert : %s\n", cder_hex);
end
}
```

FortiADC version: V5.0

Used in events: CLIENTSSL_HANDSHAKE / CLIENTSSL_RENEGOTIATE

## SSL:peer_cert(str)

Returns the peer certificate.

### Syntax

SSL:peer_cert(str);

### Arguments

| Name | Description |
|---|---|
| str | A string which specifies the certificate format. |

**Example**

```
when CLIENTSSL_HANDSHAKE {
cder = SSL:peer_cert(“der”);   --for remote leaf certificate, the input parameter can be
“info” or “der” or “pem”
if cder then
hash = sha1_hex_str(cder)
debug(“whole cert sha1 hash is: %s\n”, hash)
end
}
```

FortiADC version: V5.0

Used in events: CLIENTSSL_HANDSHAKE / SERVERSSL_HANDSHAKE / CLIENTSSL_RENEGOTIATE / SERVERSSL_RENEGOTIATE

## GEO IP commands

ip2country_name(ip) on page 96 — Returns the GEO information (country name) of an IP address.

Ip2countryProv_name(ip) on page 96 — Returns the GEO information (country name + possible province name) of an IP address.

## ip2country_name(ip)

Returns the GEO information (country name) of an IP address.

### Syntax

Ip2country_name(ip);

### Arguments

| Name | Description |
|---|---|
| ip | A string which specifies the IP address. |

### Example

```
when HTTP_REQUEST {
cip = IP:client_addr()
cnm = ip2country_name(cip)
debug("cname %s\n", cnm)
}
```

FortiADC version: V5.2

Used in events: ALL

## Ip2countryProv_name(ip)

Returns the GEO information (country name + possible province name) of an IP address.

### Syntax

ip2countryProv_name(ip);

### Arguments

| Name | Description |
|---|---|
| ip | A string which specifies the IP address. |

## Example

```
when HTTP_REQUEST {
cip = IP:client_addr()
cnm = ip2countryProv_name(cip)
debug("cname %s\n", cnm)
}
```

FortiADC version: V5.2

Used in events: ALL

## RAM cache commands

Before you begin, ensure RAM caching configuration is selected in the HTTP or HTTPs profile.

HTTP:exclude_check_disable() on page 98 — Disables the exclude URI check.

HTTP:cache_disable() on page 99 — Disables caching (both cache hit and cache store).

HTTP:dyn_check_disable() on page 99 — Disables dynamic caching check.

HTTP:dyn_invalid_check_disable() on page 100 — Disables dynamic invalid caching check.

HTTP:dyn_cache_enable(t) on page 100 — Directly enables dynamic caching with a given ID and age.

HTTP: cache_user_key(t) on page 101 — Replaces the default key (the URI) with any customized key.

HTTP:dyn_cache_invalid(t) on page 101 — Invalidates a given dynamic cache indexed by its ID.

HTTP:cache_check(t) on page 102 — Checks whether a URI has been cached or not.

HTTP:cache_hits(t) on page 102 — Checks the cache hit count for the specified URI.

HTTP:res_caching() on page 103 — Checks whether or not the response has been caching. If yes, then it checks whether it is regular cache or dynamic cache.

HTTP:res_cached() on page 103 — Check whether or not a response is from cache. If yes, then it checks whether it is regular cache or dynamic cache.

HTTP:caching_drop() on page 104 — Drops an ongoing caching operation.

## HTTP:exclude_check_disable()

Disables the exclude URI check.

### Syntax

HTTP:exclude_check_disable();

### Arguments

N/A

### Example

```
when HTTP_REQUEST {
if HTTP:exclude_check_disable() then
debug("set HTTP:exclude_check_disable: true\n");
else
debug("set HTTP:exclude_check_disable: Fail");
end
}
```

FortiADC version: V5.3

Used in events: HTTP_REQUEST

## HTTP:cache_disable()

Disables caching (both cache hit and cache store).

### Syntax

HTTP: cache_disable();

### Arguments

N/A

### Example

```
when HTTP_REQUEST {
HTTP:cache_disable()
}
```

FortiADC version: V5.3

Used in events: HTTP_REQUEST

## HTTP:dyn_check_disable()

Disables dynamic caching check.

### Syntax

HTTP: dyn_check_disable();

### Arguments

N/A

### Example

```
when HTTP_REQUEST {
HTTP:dyn_check_disable
}
```

FortiADC version: V5.3

Used in events: HTTP_REQUEST

## HTTP:dyn_invalid_check_disable()

Disables dynamic invalid caching check.

### Syntax

HTTP:dyn_invalid_check_disable();

### Arguments

N/A

### Example

```
when HTTP_REQUEST {
HTTP:dyn_invalid_check_disable()
}
```

FortiADC version: V5.3

Used in events: HTTP_REQUEST

## HTTP:dyn_cache_enable(t)

Directly enables dynamic caching with a given ID and age.

### Syntax

HTTP:dyn_cache_enable(t);

### Arguments

| Name | Description |
|---|---|
| t | A table which specifies the ID and age of caching. |

### Examples

```
when HTTP_REQUEST {
t={}
t["id"] = 1;
t["age"] = 20;      --in seconds
ret=HTTP:dyn_cache_enable(t)
}
```

FortiADC version: V5.3

Used in events: HTTP_REQUEST

## HTTP: cache_user_key(t)

Replaces the default key (the URI) with any customized key.

### Syntax

HTTP:cache_user_key(t);

### Arguments

| Name | Description |
|---|---|
| t | A table which specifies the caching URI. |

### Example

```
when HTTP_REQUEST {
url = HTTP:uri_get()
new_url = url.."external";
t={};
t["uri"] = new_url
HTTP:cache_user_key(t)
}
```

FortiADC version: V5.3

Used in events: HTTP_REQUEST

## HTTP:dyn_cache_invalid(t)

Invalidates a given dynamic cache indexed by its ID.

### Syntax

HTTP:dyn_cache_invalid(t);

### Arguments

| Name | Description |
|---|---|
| t | A table which specifies the cache ID. |

### Example

```
when HTTP_REQUEST {
t={}
t["id"] = 1       --between 1 and 1023
ret = HTTP:dyn_cache_invalid(t);
}
```

FortiADC version: V5.3

Used in events: HTTP_REQUEST / HTTP_RESPONSE

## HTTP:cache_check(t)

Checks whether a URI has been cached or not.

### Syntax

HTTP:cache_check(t);

### Arguments

| Name | Description |
|---|---|
| t | A table which specifies the cached URI. |

### Example

```
when HTTP_REQUEST {
t={}
t["uri"] = "/3.htm";
ret=HTTP:cached_check(t)
if ret then
debug("cached with id %s\n", ret);
else
debug("not cached\n");
end
}
```

FortiADC version: V5.3

Used in events: HTTP_REQUEST / HTTP_RESPONSE

## HTTP:cache_hits(t)

Checks the cache hit count for the specified URI.

### Syntax

HTTP:cache_hits(t);

### Arguments

| Name | Description |
|---|---|
| t | A table which specifies the cached URI. |

### Example

```
when HTTP_REQUEST {
t={}
t["uri"] = "/3.htm";
ret=HTTP:cache_hits(t)
if ret then
debug("cache hit count %s\n", ret);
else
debug("not cached\n");
end
}
```

FortiADC version: V5.3

Used in events: HTTP_REQUEST / HTTP_RESPONSE

## HTTP:res_caching()

Checks whether or not the response has been caching. If yes, then it checks whether it is regular cache or dynamic cache.

### Syntax

HTTP:res_caching();

### Arguments

N/A

### Example

```
when HTTP_RESPONSE {
id = HTTP:res_caching();
if  id then
debug("HTTP:res_caching() response caching with id %s !!!!\n", id);
else
debug("HTTP:res_caching() response NOT caching\n");
end
}
```

FortiADC version: V5.3

Used in events: HTTP_RESPONSE

## HTTP:res_cached()

Check whether or not a response is from cache. If yes, then it checks whether it is regular cache or dynamic cache.

### Syntax

HTTP:res_cached();

### Arguments

N/A

### Example

```
when HTTP_RESPONSE {
ret = HTTP:res_cached();
if  ret then
debug("HTTP:res_cached() response from cache !!!!\n");
else
debug("HTTP:res_cached() response NOT from cache\n");
end
}
```

FortiADC version: V5.3

Used in events: HTTP_RESPONSE

## HTTP:caching_drop()

Drops an ongoing caching operation.

### Syntax

HTTP:caching_drop();

### Arguments

N/A

### Example

```
when HTTP_RESPONSE {
ret=HTTP:caching_drop()
if ret then
debug("script dropping caching: True\n")
else
debug("script dropping caching: False\n");
end
}
```

FortiADC version: V5.3

Used in events: HTTP_RESPONSE

## PROXY commands

PROXY:set_auth_key(value) on page 105 — Customize the crypto key FortiADC used for encrypt/decrypt authentication cookie name "FortiADCauthSI". This will increase your FortiADC's security so that others cannot forge this authentication cookie.

PROXY:clear_auth_key(value) on page 106 — Clears the customized authentication key that was previously set to use the default key instead.

PROXY:shared_table_create() on page 106 — Creates a shared table if there is no existing shared table with the specified name.

PROXY:shared_table_destroy() on page 107 — Destroys the specified shared table and all the data entries if the calling process is the only one attached to the shared table. In case there is more than one process attached to this table, this function will only detach both the data entries and the shared table for the calling process.

PROXY:shared_table_entry_count() on page 108 — Returns the current count of entries in a shared table. Returns -1 if the table does not exist.

PROXY:shared_table_memory_size() on page 109 — Returns the current memory usage of a shared table. Returns -1 if the table does not exist.

PROXY:shared_table_insert() on page 110 — Inserts a pair of <key, value> as an entry into the shared table. If the key already exists in the table or if the table is full, the function will do nothing. The key can be a Lua string or integer while the value can be a Lua string, integer, or table.

PROXY:shared_table_lookup() on page 111 — Looks up whether a key exists in the shared table. If the key exists, returns the corresponding value.

PROXY:shared_table_delete() on page 112 — Deletes an entry specified by a key from the shared table. If the key does not exist, the function will do nothing. If there is more than one process attached to the data entry, this function only detaches the calling process.

PROXY:shared_table_dump() on page 113 — Prints the current contents of the shared table for debugging purposes. This works similar to an iterator for a shared table.

## PROXY:set_auth_key(value)

Customize the crypto key FortiADC used for encrypt/decrypt authentication cookie name "FortiADCauthSI". This will increase your FortiADC's security so that others cannot forge this authentication cookie.

### Syntax

PROXY:set_auth_key(value);

### Arguments

| Name | Description |
|---|---|
| value | A string which will be used to encrypt/decrypt the authentication cookie. |

### Example

```
when VS_LISTENER_BIND {
AUTH_KEY = "0123456789ABCDEF0123456789ABCDEF"
result = PROXY:set_auth_key(AUTH_KEY)
If result then
Debug("set auth key succeed\n")
end
}
```

FortiADC version: V5.2

Used in events: VS_LISTENER_BIND / TCP_BIND

## PROXY:clear_auth_key(value)

Clears the customized authentication key that was previously set to use the default key instead.

### Syntax

PROXY:clear_auth_key(value);

### Arguments

| Name | Description |
|---|---|
| value | A string which will be used to encrypt/decrypt the authentication cookie. |

### Example

```
when TCP_BIND {
result = PROXY:clear_auth_key()
}
```

FortiADC version: V5.2

Used in events: VS_LISTENER_BIND / TCP_BIND

## PROXY:shared_table_create()

Creates a shared table if there is no existing shared table with the specified name. Multiple shared tables can be created with different names. Each table operation must specify a table name.

Before creating the table, FortiADC will check whether there is already a table with the same name. If a table with the same name already exists, it will either be attached or will return an error. Typically, the first calling process creates the table, then all the other processes will attach to it with the same API call.

For any table, this API must be called first before other operations can be performed.

Returns Boolean true if successful, otherwise, returns Boolean false.

### Syntax

PROXY:shared_table_create(table_name, [entry_size], [memory_limit]);

### Arguments

| Name | Description |
|---|---|
| table_name | A Lua string as the name of the shared table. This is the unique identification of a shared table. This parameter is mandatory.<br>The maximum length of this table name is 255. |
| entry_size | The maximum number of entries this table can hold. This parameter is optional. The default value is 2048. Must not exceed 2048. |
| memory_limit | The maximum amount of memory that can be allocated for a shared table and its data entries. This parameter is optional. The default value is 4 G. |

### Example

```
when HTTP_REQUEST {
       table_name = "TableDemo1"
       ret = PROXY:shared_table_create(table_name, 2048, 20971520)
       if ret then
          debug("===>>shared_table_create success: [%s]\n", table_name)
       else
          debug("===>>shared_table_create failed: [%s]\n", table_name)
       end
}
```

FortiADC version: V7.4.2

Used in events:

- RULE_INIT
- HTTP events: HTTP_REQUEST, HTTP_RESPONSE, HTTP_DATA_REQUEST, HTTP_DATA_RESPONSE, BEFORE_AUTH, AUTH_RESULT, COOKIE_BAKE
- SSL events: CLIENTSSL_HANDSHAKE, SERVERSSL_HANDSHAKE, CLIENTSSL_RENEGOTIATE, SERVERSSL_RENEGOTIATE
- TCP events: TCP_ACCEPTED, TCP_CLOSED, SERVER_CONNECTED, SERVER_CLOSED, VS_LISTENER_BIND, SERVER_BEFORE_CONNECT
- WAF events: WAF_REQUEST_BEFORE_SCAN, WAF_RESPONSE_BEFORE_SCAN, WAF_REQUEST_ATTACK_DETECTED, WAF_RESPONSE_ATTACK_DETECTED

## PROXY:shared_table_destroy()

Destroys the specified shared table and all the data entries if the calling process is the only one attached to the shared table. In case there is more than one process attached to this table, this function will only detach both the data entries and the shared table for the calling process.
Returns Boolean true if successful, otherwise, returns Boolean false.

### Syntax

PROXY:shared_table_destroy(table_name);

### Arguments

| Name | Description |
|---|---|
| table_name | A Lua string as the name of the shared table. This is the unique identification of a shared table. This parameter is mandatory.<br>The maximum length of this table name is 255. |

### Example

```
when HTTP_REQUEST {
       table_name = "TableDemo1"
       ret = PROXY:shared_table_destroy(table_name)
       if ret then
          debug("===>>shared_table_destroy success: [%s]\n", table_name)
       else
          debug("===>>shared_table_destroy failed: [%s]\n", table_name)
       end
}
```

FortiADC version: V7.4.2

Used in events:

- RULE_INIT
- HTTP events: HTTP_REQUEST, HTTP_RESPONSE, HTTP_DATA_REQUEST, HTTP_DATA_RESPONSE, BEFORE_AUTH, AUTH_RESULT, COOKIE_BAKE
- SSL events: CLIENTSSL_HANDSHAKE, SERVERSSL_HANDSHAKE, CLIENTSSL_RENEGOTIATE, SERVERSSL_RENEGOTIATE
- TCP events: TCP_ACCEPTED, TCP_CLOSED, SERVER_CONNECTED, SERVER_CLOSED, VS_LISTENER_BIND, SERVER_BEFORE_CONNECT
- WAF events: WAF_REQUEST_BEFORE_SCAN, WAF_RESPONSE_BEFORE_SCAN, WAF_REQUEST_ATTACK_DETECTED, WAF_RESPONSE_ATTACK_DETECTED

## PROXY:shared_table_entry_count()

Returns the current count of entries in a shared table. Returns -1 if the table does not exist.

### Syntax

PROXY:shared_table_entry_count(table_name);

### Arguments

| Name | Description |
|---|---|
| table_name | A Lua string as the name of the shared table. This is the unique identification of a shared table. This parameter is mandatory.<br>The maximum length of this table name is 255. |

### Example

```
when HTTP_REQUEST {
        table_name = "TableDemo1"
        ret = PROXY:shared_table_entry_count(table_name)
        debug("===>>shared_table_entry_count: [%s]=[%d]\n", table_name, ret)
}
```

FortiADC version: V7.4.2

Used in events:

- RULE_INIT
- HTTP events: HTTP_REQUEST, HTTP_RESPONSE, HTTP_DATA_REQUEST, HTTP_DATA_RESPONSE, BEFORE_AUTH, AUTH_RESULT, COOKIE_BAKE
- SSL events: CLIENTSSL_HANDSHAKE, SERVERSSL_HANDSHAKE, CLIENTSSL_RENEGOTIATE, SERVERSSL_RENEGOTIATE
- TCP events: TCP_ACCEPTED, TCP_CLOSED, SERVER_CONNECTED, SERVER_CLOSED, VS_LISTENER_BIND, SERVER_BEFORE_CONNECT
- WAF events: WAF_REQUEST_BEFORE_SCAN, WAF_RESPONSE_BEFORE_SCAN, WAF_REQUEST_ATTACK_DETECTED, WAF_RESPONSE_ATTACK_DETECTED

## PROXY:shared_table_memory_size()

Returns the current memory usage of a shared table. Returns -1 if the table does not exist.

### Syntax

PROXY:shared_table_memory_size(table_name);

### Arguments

| Name | Description |
|---|---|
| table_name | A Lua string as the name of the shared table. This is the unique identification of a shared table. This parameter is mandatory.<br>The maximum length of this table name is 255. |

### Example

```
when HTTP_REQUEST {
        table_name = "TableDemo1"
        ret = PROXY:shared_table_memory_size(table_name)
        debug("===>>shared_table_memory_size: [%s]=[%d]\n", table_name, ret)
}
```

FortiADC version: V7.4.2

Used in events:

- RULE_INIT
- HTTP events: HTTP_REQUEST, HTTP_RESPONSE, HTTP_DATA_REQUEST, HTTP_DATA_RESPONSE, BEFORE_AUTH, AUTH_RESULT, COOKIE_BAKE
- SSL events: CLIENTSSL_HANDSHAKE, SERVERSSL_HANDSHAKE, CLIENTSSL_RENEGOTIATE, SERVERSSL_RENEGOTIATE
- TCP events: TCP_ACCEPTED, TCP_CLOSED, SERVER_CONNECTED, SERVER_CLOSED, VS_LISTENER_BIND, SERVER_BEFORE_CONNECT
- WAF events: WAF_REQUEST_BEFORE_SCAN, WAF_RESPONSE_BEFORE_SCAN, WAF_REQUEST_ATTACK_DETECTED, WAF_RESPONSE_ATTACK_DETECTED

## PROXY:shared_table_insert()

Inserts a pair of <key, value> as an entry into the shared table. If the key already exists in the table or if the table is full, the function will do nothing. The key can be a Lua string or integer while the value can be a Lua string, integer, or table.

This function will fail if the table has not been created yet; it will not trigger the creation of a new table.

Returns Boolean true if successful, otherwise, returns Boolean false.

### Syntax

PROXY:shared_table_insert(table_name, key, value);

### Arguments

| Name | Description |
|---|---|
| table_name | A Lua string as the name of the shared table. This is the unique identification of a shared table. This parameter is mandatory.<br>The maximum length of this table name is 255. |
| key | The key can be a Lua string or integer. This parameter is mandatory.<br>The maximum length for the string is 255. |
| value | The value can be a Lua string, integer, or table. The table will be serialized before storing into the shared table. This parameter is mandatory.<br>The maximum length of any value is 64K. |

### Example

```
when HTTP_REQUEST {
        table_name = "TableDemo1"
        key1="keyString101"
        val1="valString101"
        ret = RPXOY:shared_table_insert(table_name, key1, val1)
        if ret then
           debug("===>>shared_table_insert success:[Table:%s] [%s]=[%s]\n",table_name, key1, val1)
        else
           debug("===>>shared_table_insert failed:[Table:%s] [%s]=[%s]\n",table_name, key1, val1)
        end
}
```

FortiADC version: V7.4.2

Used in events:

- RULE_INIT
- HTTP events: HTTP_REQUEST, HTTP_RESPONSE, HTTP_DATA_REQUEST, HTTP_DATA_RESPONSE, BEFORE_AUTH, AUTH_RESULT, COOKIE_BAKE
- SSL events: CLIENTSSL_HANDSHAKE, SERVERSSL_HANDSHAKE, CLIENTSSL_RENEGOTIATE, SERVERSSL_RENEGOTIATE
- TCP events: TCP_ACCEPTED, TCP_CLOSED, SERVER_CONNECTED, SERVER_CLOSED, VS_LISTENER_BIND, SERVER_BEFORE_CONNECT
- WAF events: WAF_REQUEST_BEFORE_SCAN, WAF_RESPONSE_BEFORE_SCAN, WAF_REQUEST_ATTACK_DETECTED, WAF_RESPONSE_ATTACK_DETECTED

## PROXY:shared_table_lookup()

Looks up whether a key exists in the shared table. If the key exists, returns the corresponding value.

This function will fail if the table has not been created yet; it will not trigger the creation of a new table.

Returns the stored value if successful, otherwise, returns Boolean false. The returned value can be an Lua string, integer, or table. The table will be deserialized before returning, so the API will receive a Lua table.

### Syntax

PROXY:shared_table_lookup(table_name, key);

### Arguments

| Name | Description |
|---|---|
| table_name | A Lua string as the name of the shared table. This is the unique identification of a shared table. This parameter is mandatory.<br>The maximum length of this table name is 255. |
| key | The key can be a Lua string or integer. This parameter is mandatory.<br>The maximum length for the string is 255. |

### Example

```
when HTTP_REQUEST {
       table_name = "TableDemo1"
       key1="keyString101"
       ret = PROXY:shared_table_lookup(table_name,key1)
       if ret then
          debug("===>>shared_table_lookup success: [Table:%s]  [%s]=[%s]\n", table_name, key1,
ret)
       else
          debug("===>>shared_table_lookup failed for key: [Table:%s] [%s]\n", table_name, key1)
       end
}
```

FortiADC version: V7.4.2

Used in events:

- RULE_INIT
- HTTP events: HTTP_REQUEST, HTTP_RESPONSE, HTTP_DATA_REQUEST, HTTP_DATA_RESPONSE, BEFORE_AUTH, AUTH_RESULT, COOKIE_BAKE
- SSL events: CLIENTSSL_HANDSHAKE, SERVERSSL_HANDSHAKE, CLIENTSSL_RENEGOTIATE, SERVERSSL_RENEGOTIATE
- TCP events: TCP_ACCEPTED, TCP_CLOSED, SERVER_CONNECTED, SERVER_CLOSED, VS_LISTENER_BIND, SERVER_BEFORE_CONNECT
- WAF events: WAF_REQUEST_BEFORE_SCAN, WAF_RESPONSE_BEFORE_SCAN, WAF_REQUEST_ATTACK_DETECTED, WAF_RESPONSE_ATTACK_DETECTED

## PROXY:shared_table_delete()

Deletes an entry specified by a key from the shared table. If the key does not exist, the function will do nothing. If there is more than one process attached to the data entry, this function only detaches the calling process.

This function will fail if the table has not been created yet; it will not trigger the creation of a new table.

Returns Boolean true if successful, otherwise, returns Boolean false.

### Syntax

PROXY:shared_table_delete(table_name, key);

### Arguments

| Name | Description |
|---|---|
| table_name | A Lua string as the name of the shared table. This is the unique identification of a shared table. This parameter is mandatory.<br>The maximum length of this table name is 255. |
| key | The key can be a Lua string or integer. This parameter is mandatory.<br>The maximum length for the string is 255. |

### Example

```
when HTTP_REQUEST {
       table_name = "TableDemo1"
       key3 ="keyString103"
       ret = PROXY:shared_table_delete(table_name,key3)
       if ret then
          debug("===>>shared_table_delete success for key: [Table:%s]  [%d]\n", table_name, key3)
       else
          debug("===>>shared_table_delete failed for key: [Table:%s]  [%d]\n", table_name, key3)
       end
}
```

FortiADC version: V7.4.2

Used in events:

- RULE_INIT
- HTTP events: HTTP_REQUEST, HTTP_RESPONSE, HTTP_DATA_REQUEST, HTTP_DATA_RESPONSE, BEFORE_AUTH, AUTH_RESULT, COOKIE_BAKE
- SSL events: CLIENTSSL_HANDSHAKE, SERVERSSL_HANDSHAKE, CLIENTSSL_RENEGOTIATE, SERVERSSL_RENEGOTIATE
- TCP events: TCP_ACCEPTED, TCP_CLOSED, SERVER_CONNECTED, SERVER_CLOSED, VS_LISTENER_BIND, SERVER_BEFORE_CONNECT
- WAF events: WAF_REQUEST_BEFORE_SCAN, WAF_RESPONSE_BEFORE_SCAN, WAF_REQUEST_ATTACK_DETECTED, WAF_RESPONSE_ATTACK_DETECTED

## PROXY:shared_table_dump()

Prints the current contents of the shared table for debugging purposes. This works similar to an iterator for a shared table.

Returns a pair table, which can be traversed with for [k, v]. All keys and values will be converted into strings. Returns NIL if there is any error or the table is empty.

### Syntax

PROXY:shared_table_dump(table_name, [index], [count]);

### Arguments

| Name | Description |
|---|---|
| table_name | A Lua string as the name of the shared table. This is the unique identification of a shared table. This parameter is mandatory.<br>The maximum length of this table name is 255. |
| index | A Lua integer is used as the print index. This will indicate the order of printing for all the items. |

| Name | Description |
|---|---|
| | This parameter is optional. The default value is 1. The valid range is from 1 to the current entry count of the table. FortiADC will check the validity of this index and report errors for invalid entries.<br>**Note**: The item order in the hash table is not important, so there is no need to try to match the index to the item. |
| count | A Lua integer is used to indicate the number of items to print.<br>This parameter is optional. The default value is 1000 or the current entry count (whichever is smaller). The valid range is from 1 to the current entry count of the table. FortiADC will check the validity of this count and report errors for invalid entries.<br>The index is always processed before the count, so if only one parameter exists, then it is assumed as the index.<br>The actual returned count may be less than the specified value if we run out of items or if there is any error. |

## Example

```
when HTTP_REQUEST {
       table_name = "TableDemo1"
       index = 1
       count = PROXY:shared_table_entry_count(table_name)
       ret = PROXY:shared_table_dump(table_name, index, count)
       --Or simply:
       --ret = PROXY:shared_table_dump(table_name)
       if ret then
          debug("===>>PROXY-shared_table_dump success [Table-%s]\n", table_name)
          for k, v in pairs(ret) do
             debug("===>>Key: %s Value: %s\n", k, v)
          end
       else
          debug("===>>PROXY-shared_table_dump failed [Table-%s]\n", table_name)
       end
}
```

FortiADC version: V7.4.2

Used in events:

- RULE_INIT
- HTTP events: HTTP_REQUEST, HTTP_RESPONSE, HTTP_DATA_REQUEST, HTTP_DATA_RESPONSE, BEFORE_AUTH, AUTH_RESULT, COOKIE_BAKE
- SSL events: CLIENTSSL_HANDSHAKE, SERVERSSL_HANDSHAKE, CLIENTSSL_RENEGOTIATE, SERVERSSL_RENEGOTIATE
- TCP events: TCP_ACCEPTED, TCP_CLOSED, SERVER_CONNECTED, SERVER_CLOSED, VS_LISTENER_BIND, SERVER_BEFORE_CONNECT
- WAF events: WAF_REQUEST_BEFORE_SCAN, WAF_RESPONSE_BEFORE_SCAN, WAF_REQUEST_ATTACK_DETECTED, WAF_RESPONSE_ATTACK_DETECTED

## LB commands

LB commands contain load balancing manipulating functions, with the most important function being LB:routing which allows you to select the backend:

LB:routing(value) on page 115 — Routes the request to the content routing server.

LB:get_valid_routing() on page 116 — Returns a list of backend names configured on the current Virtual Server in the form of a Lua table (number as key, string as value).

LB:get_current_routing() on page 116 — Returns the currently allocated backend for this request.

LB:method_assign_server() on page 117 — Returns the server through the current load balance method configured on current the Virtual Server.

### LB:routing(value)

Routes the request to the content routing server.

#### Syntax

LB:routing(value);

#### Arguments

| Name | Description |
|---|---|
| value | A string which specifies the content routing to route. |

#### Example

```
when HTTP_REQUEST {
LB:routing("content_routing1");
}
```

--supports multiple routing

```
LB:routing("cr1")
LB:routing("cr2")
```

--It will be routed to cr2; the final one prevails.

**Note:**

When a VS enables both content-routing and scripting, then this function will perform a cross-check to check the content-routing used in scripting is applied to the VS.

FortiADC version: V4.3

Used in events: HTTP_REQUEST / HTTP_DATA_REQUEST

## LB:get_valid_routing()

Returns a list of backend names configured on the current Virtual Server in the form of a Lua table (number as key, string as value).

Since there must be at least one backend, the returned value cannot be nil or empty. In case of a single routing with zero or one content routing configured, the pool name will be returned.

### Syntax

LB:get_valid_routing();

### Arguments

N/A

### Example

```
when HTTP_REQUEST {
t=LB:get_valid_routing()
for k, v in pairs(t) do
debug("Key: %d Value: %s\n", k, v)
end
}
```

FortiADC version: V7.2

Used in events: HTTP_REQUEST / HTTP_DATA_REQUEST / AUTH_RESULT

## LB:get_current_routing()

Returns the currently allocated backend for this request.

If there is only one backend configured, this function returns the backend name in a string format. If there are multiple backends available, and they are already set via LB:routing(), then this function returns the backend name as a string. Otherwise, this returns an empty string if no backend is configured.

In case of a single routing with zero or one content routing configured, the pool name will be returned.

### Syntax

LB:get_current_routing();

### Arguments

N/A

### Example

```
when HTTP_REQUEST {
t=LB:get_current_routing()
if (s == nil or s == '') then
       s = "No backend returned."
       end
      debug("current routing: %s\n", s).
}
```

FortiADC version: V7.2

Used in events: HTTP_REQUEST / HTTP_DATA_REQUEST / AUTH_RESULT

## LB:method_assign_server()

Returns the server through the current load balance method configured on current the Virtual Server.

The implementation will first check whether the backend is already set. If the backend is set, then it runs the load balance algorithm to assign a server and returns the server name as a string. If the backend is not set, the function returns an empty string, and then FortiADC will set the backend only if there is only one backend available. If there is no server available in the corresponding pool or all the servers are down in the pool, an empty string will also be returned.

### Syntax

LB:method_assign_server();

### Arguments

N/A

### Example

```
when HTTP_REQUEST {
    s=LB:method_assign_server()
    if (s == nil or s == '') then
    s = "No Server returned."
    end
   debug("assign server: '%s'\n", s)
}
```

**Note**:

The result may be overwritten by functions in PERSISTENCE events, which happen after HTTP_REQUEST. For example, this function returns server01. But the PERSISTENCE event specifies to use server02. The result from the PERSISTENCE event (server02) will always supercede results from non-PERSISTENCE events (server01).

FortiADC version: V7.2

Used in events: HTTP_REQUEST / HTTP_DATA_REQUEST / AUTH_RESULT

## Persistence commands

HTTP:persist(save_tbl) on page 118 — Saves the entry to the stick table.

HTTP:persist(read_tbl) on page 119 — Reads the stick table content according to the hash_value.

HTTP:persist(dump_tbl) on page 120 — Dumps the stick table content.

HTTP:persist(get_valid_server) on page 121 — Gets the list of usable real servers and statuses.

HTTP:persist(cal_server_from_hash) on page 121 — Calculates the real server from the hash.

HTTP:lookup_tbl(t) on page 122 — Use this hash value to lookup the stick table, and then persist the session.

HTTP:persist(get_current_assigned_server) on page 123 — Gets the real server currently assigned to this session.

PROXY:init_stick_tbl_timeout(int) on page 124 — Sets the timeout of the stick table.

### HTTP:persist(save_tbl)

Saves the entry to the stick table.

#### Syntax

HTTP:persist(t);

#### Arguments

| Name | Description |
|---|---|
| t | A table specifies the operation, hash value, and server_name. |

#### Example

```
when PERSISTENCE{
    cip = HTTP:client_addr()
    hash_str_cip = sha512_hex(cip)

    debug("-----save_tbl-----\n")
    t={}
    t["operation"] = "save_tbl"
    t["hash_value"] = hash_str_cip
    t["srv_name"] = "pool1-2"
    ret = HTTP:persist(t)
    if ret then
           debug("hash save table success\n");
       else
           debug("save table failed\n");
       end

       t={};
       t["operation"] = "save_tbl";
```

```
        t["hash_value"] = "246810";
        t["srv_name"] = "pool1-3";
        ret = HTTP: persist(t);
        if ret then
            debug("===server add success\n");
    else
        debug("===server add fail\n");
    end
}
Output:
true: success, false: failed
```

**Note**:

Due to limitations in the stick table, it only supports 16 characters in the hash value. Otherwise, we will hash it to obtain 16 bytes as index.

FortiADC version: V5.4, V7.2 (extended function to HTTP_REQUEST events)

Used in events: PERSISTENCE / POST_PERSIST / HTTP_REQUEST

## HTTP:persist(read_tbl)

Reads the stick table content according to the hash_value.

### Syntax

HTTP:persist(t);

### Arguments

| Name | Description |
|---|---|
| t | A table specifies the operation and hash value. |

### Example

```
when PERSISTENCE{
        t={};
        t["operation"] = "save_tbl";
        t["hash_value"] = "246810";
        t["srv_name"] = "pool1-3";
        ret = HTTP: persist(t);
        if ret then
            debug("===server add success\n");
    else
        debug("===server add fail\n");
    end

    t={}
    t["operation"] = "read_tbl"
    t["hash_value"] = "246810"
    ret_tbl = HTTP:persist(t)
    if ret_tbl then
        debug("246810-server: %s\n",ret_tbl)
    else
        debug("246810-server read fail\n")
end
}
Output:
       Return server name of the entry, or false if no entry found
```

FortiADC version: V5.4, V7.2 (extended function to HTTP_REQUEST events)

Used in events: PERSISTENCE / HTTP_REQUEST

## HTTP:persist(dump_tbl)

Dumps the stick table content.

### Syntax

HTTP:persist(t);

### Arguments

| Name | Description |
|---|---|
| t | A table specifies the operation. |

### Example

```
when PERSISTENCE{
       t={};
       t["operation"] = "save_tbl";
       t["hash_value"] = "246810";
       t["srv_name"] = "pool1-3";
       ret = HTTP: persist(t);
       if ret then
           debug("===server add success\n");
    else
        debug("===server add fail\n");
    end

    t={}
    t["operation"] = "dump_tbl"
    t[“index”] = 1
    t[“count”] = 15
    ret_tbl = HTTP:persist(t)
    if ret_tbl then
        for hash, srv in pairs(ret_tbl) do
            debug("tbl hash %s srv %s\n",hash,srv)
        end
    end
}
       “index”:
      “count”:
Output:
       Return A table include hash and server name
```

FortiADC version: V5.4, V7.2 (extended function to HTTP_REQUEST events)

Used in events: PERSISTENCE / HTTP_REQUEST

## HTTP:persist(get_valid_server)

Gets the list of usable real servers and statuses.

### Syntax

HTTP:persist(t);

### Arguments

| Name | Description |
|---|---|
| t | A table specifies the operation. |

### Example

```
when PERSISTENCE{
    debug("-----get valid server-----\n")
    t={}
    t["operation"] = "get_valid_server"
    ret = HTTP:persist(t)
    if ret then
        for srv,stat in pairs(ret) do
            debug("server %s, status %s\n",srv,stat)
        end
    end
}
Output:
       Return the table of usable real server and server state(enable, backup)
```

FortiADC version: V5.4, V7.2 (extended function to HTTP_REQUEST events)

Used in events: PERSISTENCE / HTTP_REQUEST

## HTTP:persist(cal_server_from_hash)

Calculates the real server from the hash.

### Syntax

HTTP:persist(t);

### Arguments

| Name | Description |
|---|---|
| t | A table specifies the operation and hash value. |

### Example

```
when PERSISTENCE{
    debug("-----cal_server_from_hash-----\n")
    t={}
    t["operation"] = "cal_server_from_hash"
    t["hash_value"] = "246810"
    ret = HTTP:persist(t)
    if ret then
        debug("hash 246810, server %s\n",ret)
    end
}
Output:
       Return the real server name according to the hash value using our algorithm or False if
failed
```

FortiADC version: V5.4, V7.2 (extended function to HTTP_REQUEST events)

Used in events: PERSISTENCE / HTTP_REQUEST

## HTTP:lookup_tbl(t)

Use this hash value to lookup the stick table, and then persist the session.

### Syntax

HTTP:lookup_tbl(t);

### Arguments

| Name | Description |
|---|---|
| t | A table specifies the operation and hash value. |

### Example

```
when PERSISTENCE{
        cip = HTTP:client_addr()
    hash_str_cip = sha512_hex(cip)

    t={}
    t["operation"] = "save_tbl"
    t["hash_value"] = hash_str_cip
    t["srv_name"] = "pool1-3"
    ret = HTTP:persist(t)
    if ret then
           debug("hash save table success\n");
       else
           debug("save table failed\n");
       end

    t={}
    t["hash_value"]=hash_str_cip
    ret = HTTP:lookup_tbl(t)
    if ret then
        debug("hash LOOKUP success\n")
    else
        debug("hash lookup failed\n")
    end
}
Output:
       Return True: lookup success, False, lookup failed and use the org. LB method
```

FortiADC version: V5.4

Used in events: PERSISTENCE

## HTTP:persist(get_current_assigned_server)

Gets the real server currently assigned to this session.

### Syntax

HTTP:persist(t);

### Arguments

| Name | Description |
|---|---|
| t | A table specifies the operation. |

### Example

```
when PERSISTENCE{
        cip = HTTP:client_addr()
    hash_str_cip = sha512_hex(cip)

    t={}
    t["operation"] = "save_tbl"
    t["hash_value"] = hash_str_cip
    t["srv_name"] = "pool1-3"
    ret = HTTP:persist(t)
    if ret then
           debug("hash save table success\n");
       else
           debug("save table failed\n");
       end

    t={}
    t["hash_value"]=hash_str_cip
    ret = HTTP:lookup_tbl(t)
    if ret then
        debug("hash LOOKUP success\n")
    else
        debug("hash lookup failed\n")
    end
}
```

```
when POST_PERSIST{
debug("-----event POST_PERSIST-----\n")

    debug("-----get current assigned server-----\n")
    t={}
    t["operation"]="get_current_assigned_server"
    ret=HTTP:persist(t)
    debug("current assigned server: %s\n",ret)
}
Output:
       Return the real server name which is assigned to current session or False if no server is 
assigned right now
```

FortiADC version: V5.4, V7.2 (extended function to HTTP_REQUEST events)

Used in events: POST_PERSIST / HTTP_REQUEST

## PROXY:init_stick_tbl_timeout(int)

Sets the timeout of the stick table.

### Syntax

PROXY:init_stick_tbl_timeout(init);

### Arguments

| Name | Description |
|---|---|
| int | A positive integer that specifies the timeout. |

### Example

```
when RULE_INIT{
    env={}
    PROXY:init_stick_tbl_timeout(500)
}
when PERSISTENCE{
    cip = HTTP:client_addr()
    hash_str_cip = sha512_hex(cip)

  debug("-----save_tbl-----\n")
    t={}
    t["operation"] = "save_tbl"
    t["hash_value"] = hash_str_cip
    t["srv_name"] = "pool1-3"
    ret = HTTP:persist(t)
    if ret then
           debug("hash save table success\n");
       else
           debug("save table failed\n");
       end
}
```

```
Output:
       Return True: success, False: failed
```

FortiADC version: V5.4

Used in events: RULE_INIT

## Global commands

Crc32(str) on page 128 — Returns the crc32 check value of the string, or 0 if it is an empty string.

Key_gen(str_pass, str_salt, iter_num, len_num) on page 128 — Creates an AES key to encrypt/decrypt data, either generated by password or user defined.

Aes_enc(t) on page 129 — Encrypts the data using the previously-created AES key.

Aes_dec(t) on page 130 — Decrypts the data using the previously-created AES key.

EVP_Digest(alg, str) on page 131 — EVP_Digest for one-shot digest calculation.

HMAC(alg, str, key) on page 131 — HMAC message authentication code.

HMAC_verify(alg, data, key, verify) on page 132 — Checks if the signature is the same as the current digest.

G2F(alg, key) on page 133 — Returns a G2F random value.

Class_match(str, method, list) on page 133 — Matches the string against an element.

Class_search(list, method, str) on page 134 — Searches an element in the list against a string.

Cmp_addr() on page 135 — Matches one IP address against a group of IP addresses. It can automatically detect IPv4 and IPv6 and can be used to compare IPv4 addresses with IPv6 addresses.

url_enc(str) on page 136 — Converts the URL information into valid ASCII format.

url_dec(str) on page 136 — Converts the encoding-URL into the original URL.

url_parser(str) on page 137 — Parses a URL, returns a table containing host, port, path, query, fragment, the username, password, etc., from the URL.

url_compare(url1, url2) on page 137 — Compares two URL strings, returns true if they are the same.

Rand() on page 138 — Generates a random number. Returns an integer number. After FortiADC reboots, the random number will be different.

srand(str) on page 139 — Sets the random seed.

Rand_hex(int) on page 139 — Generates a random number in HEX. Returns a string, length is the <int>.

Rand_alphanum(int) on page 140 — Generates a random alphabet + number sequence. Returns a string, length is the <int>.

Rand_seq(int) on page 140 — Generates a random number sequence. Returns a string, length is the <int>.

Time() on page 141 — Returns the current time as a number in seconds. This is the time since the Epoch was measured.

Ctime() on page 141 — Returns the current time as a string, for example, “Tue Jun 25 14:11:01 2019”.

gmtime() on page 142 — Returns the GMT time as a string, for example, “Thu 27 Jun 2019 18:27:42 GMT”.

Md5(str) on page 142 — Returns the MD5 calculated for the specified string.

Md5_hex(str) on page 143 — Returns the MD5 value in hex as a string.

Md5_str(str) on page 144 — Calculates the MD5 of a string input and stores the results in an intermediate variable, in some cases you need a version to deal with it.

Md5_hex_str(str) on page 144 — Calculates the MD5 of a string input of a string input and outputs the results in HEX format, in some cases you need a version to deal with it.

Sha1(str) on page 145 — Returns the SHA-1 calculated for the specified string.

Sha1_hex(str) on page 145 — Returns the SHA-1 calculated for the string in hex.

Sha1_str(str) on page 146 — Calculates the SHA-1 of a string input and stores the results in an intermediate variable, in some cases you need a version to deal with it.

Sha1_hex_str(str) on page 146 — Calculates the SHA-1 of a string input and output the results in HEX format, in some cases you need a version to deal with it.

Sha256(str) on page 147 — Calculates the SHA-256 of a string input and stores the result in an intermediate variable.

Sha256_hex(str) on page 147 — Calculates the SHA-256 of a string input and outputs the result in an intermediate variable. In some cases you need a version to deal with it.

Sha256_str(str) on page 148 — Calculates the SHA-256 of a string input and stores the result in an intermediate variable. In some cases you need a version to deal with it.

Sha256_hex_str(str) on page 148 — Calculates the SHA-256 of a string input and stores the result in an intermediate variable. In some case you need a version to deal with it.

Sha384(str) on page 149 — Calculates the SHA-384 of a string input and stores the result in an intermediate variable.

Sha384_hex(str) on page 149 — Calculates the SHA-384 of a string input and outputs the result in an intermediate variable. In some cases you need a version to deal with it.

Sha384_str(str) on page 150 — Calculates the SHA-384 of a string input and stores the result in an intermediate variable. In some cases you need a version to deal with it.

Sha384_hex_str(str) on page 150 — Calculates the SHA-384 of a string input and stores the result in an intermediate variable. In some case you need a version to deal with it.

Sha512(str) on page 151 — Calculates the SHA-512 of a string input and stores the result in an intermediate variable.

Sha512_hex(str) on page 151 — Calculates the SHA-512 of a string input and outputs the result in an intermediate variable. In some cases you need a version to deal with it.

Sha512_str(str) on page 152 — Calculates the SHA-512 of a string input and stores the result in an intermediate variable. In some cases you need a version to deal with it.

Sha512_hex_str(str) on page 152 — Calculates the SHA-512 of a string input and stores the result in an intermediate variable. In some case you need a version to deal with it.

B32_enc(str) on page 153 — Encodes a string input in Base32 and outputs the result in string format.

B32_enc_str(str) on page 153 — Encodes a string input in Base32 and outputs the result in string format. In some cases you need a version to deal with it.

B32_dec(str) on page 154 — Decodes a Base32 encoded string input and outputs the result in string format.

B32_dec_str(str) on page 154 — Decodes a Base32 encoded string input and outputs the result in string format. In some cases you need a version to deal with it.

B64_enc(str) on page 155 — Encodes a string input in Base64 and outputs the result in string format.

B64_dec(str) on page 155 — Decodes a Base64 encoded string input and outputs the result in string format.

Get_pid() on page 156 — Returns the PID value of the VS process.

Table_to_string(t) on page 156 — Returns the table in a string.

Htonl(int) on page 157 — Converts a 32 bit long integer from host byte order to network byte order.

Ntohs(int) on page 157 — Converts a 16 bit short integer from network byte order to host byte order.

Htons(int) on page 158 — Converts a 16 bit short integer from host byte order to network byte order.

Ntohl(int) on page 159 — When receiving long integers in HTTP response from the network, this command converts a 32 bit long integer from network byte order to host byte order.

To_HEX(str) on page 159 — Returns the HEX calculate of the string.

Debug(str) on page 160 — Prints the debug information when VS using scripting.

Log(str) on page 160 — Prints the scripting running information in log format. When using this command, you should enable scripting log.

File_open(path, str) on page 161 — Opens a file, returns a file object.

File_get(file, size) on page 161 — Returns the file content.

File_close(file) on page 162 — Closes a file.

## Crc32(str)

Returns the crc32 check value of the string, or 0 if it is an empty string.

### Syntax

crc32(str);

### Arguments

| Name | Description |
|---|---|
| str | The string which will be calculated. |

### Example

```
when HTTP_REQUEST {
str = "any string for crc32 calculation"
crc = crc32(str);
debug("crc is %d\n", crc);
}
```

FortiADC version: V5.2

Used in events: ALL

## Key_gen(str_pass, str_salt, iter_num, len_num)

Creates an AES key to encrypt/decrypt data, either generated by password or user defined.

### Syntax

key_gen(str_pass, str_salt, iter_num, len_num);

### Arguments

| Name | Description |
|---|---|
| str_pass | The password string. |
| str_salt | The salt string. |
| iter_num | The number of iterations. |
| len_num | The key length. |

### Example

```
when HTTP_REQUEST {
new_key = key_gen("pass", "salt", 32, 32);     -- first parameter is the password string,
second the salt string, third the number of iterations, fourth the key length
debug("new key in hex is %s\n", to_HEX(new_key));
}
```

FortiADC version: V5.2

Used in events: ALL

## Aes_enc(t)

Encrypts the data using the previously-created AES key.

### Syntax

Aes_enc(t);

### Arguments

| Name | Description |
|---|---|
| t | A table which specifies the message, key, and key size that was used to encrypt. |

### Example

```
when HTTP_REQUEST {
t={};
t["message"]  = "MICK-TEST";
t["key"]  = "aaaaaaaaaabbbbbb"            --16bit
t["size"]= 128        -- 128, 192, or 256, the corresponding key length is 16, 24, and 32
enc = aes_enc(t)
debug("The aes_enc output to HEX\n %s\n",to_HEX(enc));
}
```

**Note:**

- Message: a string which will be encrypted
- Key: a string to encrypt str
- Size: must be 128, 192, or 256, the corresponding key length is 16, 24, and 32

FortiADC version: V5.2

Used in events: ALL

## Aes_dec(t)

Decrypts the data using the previously-created AES key.

### Syntax

Aes_dec(t);

### Arguments

| Name | Description |
|---|---|
| t | A table which specifies the message, key, and key size that was used to decrypt. |

### Example

```
when HTTP_REQUEST {
t={};
t["message"]  = "MICK-TEST";
t["key"]  = "aaaaaaaaaabbbbbb"
t["size"]= 128 -- 128, 192, or 256, the corresponding key length is 16, 24, and 32
enc = aes_enc(t)
--aes decryption
a={};
a["message"]  = enc;
a["key"]  = "aaaaaaaaaabbbbbb"
a["size"]= 128;
dec = aes_dec(a);
debug("key length %s decrypted is %s\n",”128” ,dec);
}
```

**Note:**

- Message: a string which will be decrypted
- Key: a string to decrypt str
- Size: must be 128, 192, or 256, the corresponding key length is 16, 24, and 32

FortiADC version: V5.2

Used in events: ALL

## EVP_Digest(alg, str)

EVP_Digest for one-shot digest calculation.

### Syntax

EVP_Digest(alg, str);

### Arguments

| Name | Description |
|---|---|
| alg | A string which specifies the algorithm. |
| str | A string which will be EVP_Digested. |

### Example

```
when HTTP_REQUEST {
alg = "MD5"; -- or "SHA1", "SHA256", "SHA384", "SHA512"
data = "your data"
re = EVP_Digest(alg, data);
debug("the digest in hex is %s\n", to_HEX(re));
}
```

**Note:**

Alg: type of hashing algorithms to use, must be MD5, SHA1, SHA256, SHA384, SHA512

FortiADC version: V5.2

Used in events: ALL

## HMAC(alg, str, key)

HMAC message authentication code.

### Syntax

HMAC(alg, str, key);

### Arguments

| Name | Description |
|---|---|
| alg | A string which specifies the algorithm. |
| str | A string which will be calculated. |
| key | A string which is a secret key. |

### Example

```
when HTTP_REQUEST {
alg = "MD5"; -- or "SHA1", "SHA256", "SHA384", "SHA512"
data = "your data"
key  = "123456789ABCDEF0123456789ABCDEF\121"; -- or you can generate a key using key_gen
re = HMAC(alg, data, key);
debug("the HMAC in hex is %s\n", to_HEX(re));
}
```

**Note:**

Alg: type of hashing algorithms to use, must be MD5, SHA1, SHA256, SHA384, SHA512

FortiADC version: V5.2

Used in events: ALL

## HMAC_verify(alg, data, key, verify)

Checks if the signature is the same as the current digest.

### Syntax

HMAC_verify(alg, data, key verify);

### Arguments

| Name | Description |
|---|---|
| alg | A string which specifies the algorithm. |
| key | A string which is a secret key. |
| data | A string which will be calculated. |
| verify | A signature to compare the current digest against. |

### Example

```
when HTTP_REQUEST {
alg = "MD5"; -- or "SHA1", "SHA256", "SHA384", "SHA512"
data = "your data"
verify = "your result to compare"
key  = "123456789ABCDEF0123456789ABCDEF\121"; -- or you can generate a key using key_gen
re = HMAC_verify(alg, data, key, verify);
if re then
debug("verified\n")
else
debug("not verified\n")
end
}
```

**Note:**

Alg: type of hashing algorithms to use, must be MD5, SHA1, SHA256, SHA384, SHA512

FortiADC version: V5.2

Used in events: ALL

## G2F(alg, key)

Returns a G2F random value.

### Syntax

G2F(alg, key);

### Arguments

| Name | Description |
|---|---|
| alg | A string which specifies the algorithm. |
| key | A string which is a secret key. |

### Example

```
when HTTP_REQUEST {
alg = "MD5"; -- or "SHA1", "SHA256", "SHA384", "SHA512"
key  = "123456789ABCDEF0123456789ABCDEF\121"; -- or you can generate a key using key_gen
re = G2F(alg, key);
debug("the G2F value is %d\n", re);
}
```

**Note:**

Alg: type of hashing algorithms to use, must be MD5, SHA1, SHA256, SHA384, SHA512

FortiADC version: V5.2

Used in events: ALL

## Class_match(str, method, list)

Matches the string against an element.

### Syntax

Class_match(str, method, list);

### Arguments

| Name | Description |
|---|---|
| str | A string which will be matched. |
| method | A string which specifies the match method |
| list | A list which specifies the match target. |

### Example

```
when HTTP_REQUEST {
url_list = ""
url = HTTP:uri_get()
status, count, t = class_match(url, "starts_with", url_list);    --or "ends_with", "equals",
"contains"
debug("status %s, count %s\n", status, count);
for k,v in pairs(t) do
debug("index %s, value %s\n", k,v);
end
}
```

**Note:**

Method: must be “starts_with”, “equals”, “contains”, “end_with”

This command return three parameters, first “status”: true or false means if match or not; second “count”: return the number of times matches; third “t”: return matched index and matched value in the list.

FortiADC version: V5.2

Used in events: ALL

## Class_search(list, method, str)

Searches an element in the list against a string.

### Syntax

Class_search(list, method, str);

### Arguments

| Name | Description |
|---|---|
| str | A string which will be calculated. |
| list | A string which will be matched. |
| method | A string which specifies the match method. |

### Example

```
when HTTP_REQUEST {
status, count, t = class_search(url_list, "starts_with", url);     --or "ends_with",
"equals", "contains"
for k,v in pairs(t) do
debug("index %s, value %s\n", k,v);
end
}
```

**Note:**

Method: , must be “starts_with”, “equals”, “contains”, “end_with”

FortiADC version: V5.2

Used in events: ALL

## Cmp_addr()

Matches one IP address against a group of IP addresses. It can automatically detect IPv4 and IPv6 and can be used to compare IPv4 addresses with IPv6 addresses.

### Syntax

Cmp_addr(client_Ip, addr_group );

### Arguments

| Name | Description |
|---|---|
| Client_ip | For an IPv4 ip_addr/[mask], the mask can be a number between 0 and 32 or a dotted format like 255.255.255.0<br>For an IPv6 ip_addr/[mask], the mask can be a number between 0 and 128. |
| Addr_group | A group of IP address.<br>addr_group = "192.168.1.0/24" --first network address<br>addr_group = addr_group..",::ffff:172.30.1.0/120" --second network address |

### Example

```
when RULE_INIT{
```

--initialize the address group here

--for IPv4 address, mask can be a number between 0 to 32 or a dotted format

--support both IPv4 and IPv6, for IPv6, the mask is a number between 0 and 128

```
addr_group = "192.168.1.0/24"
addr_group = addr_group..",172.30.1.0/255.255.0.0"
addr_group = addr_group..",::ffff:172.40.1.0/120"
```

```
}
when HTTP_REQUEST{
client_ip = HTTP:client_addr()
matched = cmp_addr(client_ip, addr_group)
if matched then
debug("client ip found in address group\n");
else
debug("client ip not in address group\n");
end
}
```

FortiADC version: V4.8

Used in events: ALL

## url_enc(str)

Converts the URL information into valid ASCII format.

### Syntax

url_enc(str);

### Arguments

| Name | Description |
|---|---|
| str | A string which will be converted. |

### Example

```
when HTTP_REQUEST {
url_list =https://abc.www.123.bbEEE.com/?5331=212&qe1=222
debug("Ori= %s \nencodeed= %s\n", url_list,url_enc(url_list));
}
```

FortiADC version: V5.2

Used in events: ALL

## url_dec(str)

Converts the encoding-URL into the original URL.

### Syntax

url_dec(str);

### Arguments

| Name | Description |
|---|---|
| str | A string which will be converted. |

### Example

```
when HTTP_REQUEST {
str = "http%3A%2F%2Fwww.example.com%3A890%2Furl%2Fpath%2Fdata%3Fname%3Dforest%23nose"
debug("String= %s\ndecoded= %s\n", str,url_dec(str));
}
```

FortiADC version: V5.2

Used in events: ALL

## url_parser(str)

Parses a URL, returns a table containing host, port, path, query, fragment, the username, password, etc., from the URL.

### Syntax

url_parser(str);

### Arguments

| Name | Description |
|---|---|
| str | A url which will be parser. |

### Example

```
when HTTP_REQUEST {
url_list="http://foo:bar@w1.superman.com/very/long/path.html?p1=v1&p2=v2#more-details"
purl = url_parser(url_list);
debug("parsed url scheme %s host %s\n port %s path %s query %s\n fragment %s, the username
%s\n passowrd %s\n", purl["scheme"], purl["host"], purl["port"],purl["path"], purl["query"],
purl["fragment"], purl["username"], purl["password"]);
}
```

FortiADC version: V5.2

Used in events: ALL

## url_compare(url1, url2)

Compares two URL strings, returns true if they are the same.

### Syntax

url_compare(url1, url2);

### Arguments

| Name | Description |
|---|---|
| url1, url2 | Two urls which will be compared. |

### Example

```
when HTTP_REQUEST {
url_list={};
url_list[1]="http://10.10.10.10:80/"
url_list[2]="http://10.10.10.10/"
url_list[3]="https://5.5.5.5:443/"
url_list[4]="https://5.5.5.5/"
url_list[5]="http://[2001::1]:80"
url_list[6]="http://[2001::1]"
url_list[7]="https://[2001:99:1]:443"
url_list[8]="https://[2001:99:1]"
for i = 1,8,2 do
if url_compare(url_list[i],url_list[i+1]) then
debug("URL_List %d %d Match !\n",i,i+1);
else
debug("URL_List %d %d NOT Match !\n",i,i+1);
end
end
}
```

FortiADC version: V5.2

Used in events: ALL

## Rand()

Generates a random number. Returns an integer number. After FortiADC reboots, the random number will be different.

### Syntax

rand();

### Arguments

N/A

### Example

```
when HTTP_REQUEST {
a = rand()
debug(“a = %d\n”, a)
}
```

FortiADC version: V5.2

Used in events: ALL

## srand(str)

Sets the random seed.

### Syntax

srand(str);

### Arguments

| Name | Description |
|---|---|
| str | A string which specifies the seed. |

### Example

```
when HTTP_REQUEST {
srand(1111)
a = rand()
debug(“a = %d\n”, a)
}
```

FortiADC version: V5.2

Used in events: ALL

## Rand_hex(int)

Generates a random number in HEX. Returns a string, length is the <int>.

### Syntax

Rand_hex(int);

### Arguments

| Name | Description |
|---|---|
| Int | An integer which specifies the length of the returned string. |

### Example

```
when HTTP_REQUEST {
b = rand_hex(15)
debug("-----rand_hex b = %s-----\n", b)
}
Result:
-----rand_hex b = 43474FB47A8A8C4-----
```

FortiADC version: V5.2

Used in events: ALL

## Rand_alphanum(int)

Generates a random alphabet + number sequence. Returns a string, length is the <int>.

### Syntax

Random_alphanum(int);

### Arguments

| Name | Description |
|---|---|
| Int | An integer which specifies the length of the returned string. |

### Example

```
when HTTP_REQUEST {
c = rand_alphanum(17)
debug("-----rand_alphanum c = %s-----\n", c)
}
Result:
-----rand_alphanum c = XTHQpb6ngabMqH7nx-----
```

FortiADC version: V5.2

Used in events: ALL

## Rand_seq(int)

Generates a random number sequence. Returns a string, length is the <int>.

### Syntax

Rand_seq(int);

### Arguments

| Name | Description |
|---|---|
| Int | An integer which specifies the length of the returned string. |

### Example

```
when HTTP_REQUEST {
d = rand_seq(18)
debug("-----rand_seq d = %s-----\n", d)
}
Result:

-----rand_seq = 329514876985314568-----
```

FortiADC version: V5.2

Used in events: ALL

## Time()

Returns the current time as a number in seconds. This is the time since the Epoch was measured.

### Syntax

time();

### Arguments

N/A

### Example

```
when HTTP_REQUEST {
t = time()
debug(“-----time: t %s-----\n”, t)
}
Result:
-----time: t 1561424783-----
```

FortiADC version: V4.8

Used in events: ALL

## Ctime()

Returns the current time as a string, for example, “Tue Jun 25 14:11:01 2019”.

### Syntax

ctime();

### Arguments

N/A

### Example

```
when HTTP_REQUEST {
ct = ctime()
debug("-----ctime: ct %s-----\n", ct)
}
Result:
-----ctime: ct Mon Jun 24 18:06:23 2019-----
```

FortiADC version: V4.8

Used in events: ALL

## gmtime()

Returns the GMT time as a string, for example, “Thu 27 Jun 2019 18:27:42 GMT”.

### Syntax

gmtime();

### Arguments

N/A

### Example

```
when HTTP_REQUEST {
gt = gmtime()
debug("-----gmtime: gt %s-----\n", gt)
}
Result:
-----gmtime: gt Thu 27 Jun 2019 18:27:42 GMT -----
```

FortiADC version: V5.3

Used in events: ALL

## Md5(str)

Returns the MD5 calculated for the specified string.

### Syntax

Md5(str);

### Arguments

| Name | Description |
|---|---|
| str | The string which will be calculated. |

### Example

```
when HTTP_REQUEST {
str1 = "abc"
md5 = md5(str1)
str = "test string"
a=12
md = md5("%s,123%d",str,a)
}
```

FortiADC version: V4.8

Used in events: ALL

## Md5_hex(str)

Returns the MD5 value in hex as a string.

### Syntax

Md5_hex(str);

### Arguments

| Name | Description |
|---|---|
| str | The string which will be calculated. |

### Example

```
when HTTP_REQUEST {
str1 = "abc"
str2 = md5_hex(str1)
}
```

FortiADC version: V4.8

Used in events: ALL

## Md5_str(str)

Calculates the MD5 of a string input and stores the results in an intermediate variable, in some cases you need a version to deal with it.

### Syntax

Md5_str(str);

### Arguments

| Name | Description |
|---|---|
| str | The string which will be calculated. |

### Example

```
when HTTP_REQUEST {
md5 = md5_str(input);     --input can be a cert in DER format
}
```

FortiADC version: V4.8

Used in events: ALL

## Md5_hex_str(str)

Calculates the MD5 of a string input of a string input and outputs the results in HEX format, in some cases you need a version to deal with it.

### Syntax

Md5_hex_str(str);

### Arguments

| Name | Description |
|---|---|
| str | A string which will be calculated. |

### Example

```
when HTTP_REQUEST {
md = md5_hex_str(input);   --input  can be a cert in DER format
}
```

FortiADC version: V4.8

Used in events: ALL

## Sha1(str)

Returns the SHA-1 calculated for the specified string.

### Syntax

Sha1(str);

### Arguments

| Name | Description |
|---|---|
| str | A string which will be calculated. |

### Example

```
when HTTP_REQUEST {
result = sha1(input)
}
```

FortiADC version: V4.8

Used in events: ALL

## Sha1_hex(str)

Returns the SHA-1 calculated for the string in hex.

### Syntax

Sha1_hex(str);

### Arguments

| Name | Description |
|---|---|
| str | A string which will be calculated. |

### Example

```
when HTTP_REQUEST {
str1 = "123456789"
sha1 = sha1_hex(str1)
}
```

FortiADC version: V4.8

Used in events: ALL

## Sha1_str(str)

Calculates the SHA-1 of a string input and stores the results in an intermediate variable, in some cases you need a version to deal with it.

### Syntax

Sha1_str(str);

### Arguments

| Name | Description |
|---|---|
| str | The string which will be calculated. |

### Example

```
when HTTP_REQUEST {
result = sha1_str(input);   --input can be a cert in DER format
}
```

FortiADC version: V4.8

Used in events: ALL

## Sha1_hex_str(str)

Calculates the SHA-1 of a string input and output the results in HEX format, in some cases you need a version to deal with it.

### Syntax

Sha1_hex_str(str);

### Arguments

| Name | Description |
|---|---|
| str | The string which will be calculated. |

### Example

```
when HTTP_REQUEST {
result = sha1_hex_str(input);  -- input can be a cert in DER format
}
```

FortiADC version: V4.8

Used in events: ALL

## Sha256(str)

Calculates the SHA-256 of a string input and stores the result in an intermediate variable.

### Syntax

Sha256(str);

### Arguments

| Name | Description |
|---|---|
| str | The string which will be calculated. |

### Example

```
when HTTP_REQUEST {
str1 = “abc”
str2 = sha256(str1)
}
```

FortiADC version: V4.8

Used in events: ALL

## Sha256_hex(str)

Calculates the SHA-256 of a string input and outputs the result in an intermediate variable. In some cases you need a version to deal with it.

### Syntax

Sha256_hex(str);

### Arguments

| Name | Description |
|---|---|
| str | The string which will be calculated. |

### Example

```
when HTTP_REQUEST {
str1 = “abc”
sha256 = sha256_hex(str)
}
```

FortiADC version: V4.8

Used in events: ALL

## Sha256_str(str)

Calculates the SHA-256 of a string input and stores the result in an intermediate variable. In some cases you need a version to deal with it.

### Syntax

Sha256_str(str);

### Arguments

| Name | Description |
|---|---|
| str | The string which will be calculated. |

### Example

```
when HTTP_REQUEST {
result = sha256_str(input);   --input can be a cert in DER format
}
```

FortiADC version: V4.8

Used in events: ALL

## Sha256_hex_str(str)

Calculates the SHA-256 of a string input and stores the result in an intermediate variable. In some case you need a version to deal with it.

### Syntax

Sha256_hex_str(str);

### Arguments

| Name | Description |
|---|---|
| str | The string which will be calculated. |

### Example

```
when HTTP_REQUEST {
result = sha256_hex_str(input); --input can be a cert in DER format
}
```

FortiADC version: V4.8

Used in events: ALL

## Sha384(str)

Calculates the SHA-384 of a string input and stores the result in an intermediate variable.

### Syntax

Sha384(str);

### Arguments

| Name | Description |
|---|---|
| str | The string which will be calculated. |

### Example

```
when HTTP_REQUEST {
str1 = "abc"
str2 = sha384(str1)
}
```

FortiADC version: V4.8

Used in events: ALL

## Sha384_hex(str)

Calculates the SHA-384 of a string input and outputs the result in an intermediate variable. In some cases you need a version to deal with it.

### Syntax

Sha384_hex(str);

### Arguments

| Name | Description |
|---|---|
| str | The string which will be calculated. |

### Example

```
when HTTP_REQUEST {
str1 = "abc"
sha384 = sha384_hex(str)
}
```

FortiADC version: V4.8

Used in events: ALL

## Sha384_str(str)

Calculates the SHA-384 of a string input and stores the result in an intermediate variable. In some cases you need a version to deal with it.

### Syntax

Sha384_str(str);

### Arguments

| Name | Description |
|---|---|
| str | The string which will be calculated. |

### Example

```
when HTTP_REQUEST {
result = sha384_str(input);   --input can be a cert in DER format
}
```

FortiADC version: V4.8

Used in events: ALL

## Sha384_hex_str(str)

Calculates the SHA-384 of a string input and stores the result in an intermediate variable. In some case you need a version to deal with it.

### Syntax

Sha384_hex_str(str);

### Arguments

| Name | Description |
|---|---|
| str | The string which will be calculated. |

### Example

```
when HTTP_REQUEST {
result = sha384_hex_str(input); --input can be a cert in DER format
}
```

FortiADC version: V4.8

Used in events: ALL

## Sha512(str)

Calculates the SHA-512 of a string input and stores the result in an intermediate variable.

### Syntax

Sha512(str);

### Arguments

| Name | Description |
|---|---|
| str | The string which will be calculated. |

### Example

```
when HTTP_REQUEST {
str1 = "abc"
str2 = sha512(str1)
}
```

FortiADC version: V4.8

Used in events: ALL

## Sha512_hex(str)

Calculates the SHA-512 of a string input and outputs the result in an intermediate variable. In some cases you need a version to deal with it.

### Syntax

Sha512_hex(str);

### Arguments

| Name | Description |
|---|---|
| str | The string which will be calculated. |

### Example

```
when HTTP_REQUEST {
str1 = "abc"
sha512 = sha512_hex(str)
}
```

FortiADC version: V4.8

Used in events: ALL

## Sha512_str(str)

Calculates the SHA-512 of a string input and stores the result in an intermediate variable. In some cases you need a version to deal with it.

### Syntax

Sha512_str(str);

### Arguments

| Name | Description |
|---|---|
| str | The string which will be calculated. |

### Example

```
when HTTP_REQUEST {
result = sha512_str(input);   --input can be a cert in DER format
}
```

FortiADC version: V4.8

Used in events: ALL

## Sha512_hex_str(str)

Calculates the SHA-512 of a string input and stores the result in an intermediate variable. In some case you need a version to deal with it.

### Syntax

Sha512_hex_str(str);

### Arguments

| Name | Description |
|---|---|
| str | The string which will be calculated. |

### Example

```
when HTTP_REQUEST {
result = sha512_hex_str(input); --input can be a cert in DER format
}
```

FortiADC version: V4.8

Used in events: ALL

## B32_enc(str)

Encodes a string input in Base32 and outputs the result in string format.

### Syntax

B32_enc(str);

### Arguments

| Name | Description |
|---|---|
| str | The string which will be calculated. |

### Example

```
when HTTP_REQUEST {
str = “abc”
en = b32_enc(str)
}
```

FortiADC version: V5.2

Used in events: ALL

## B32_enc_str(str)

Encodes a string input in Base32 and outputs the result in string format. In some cases you need a version to deal with it.

### Syntax

B32_enc_str(str);

### Arguments

| Name | Description |
|---|---|
| str | The string which will be calculated. |

### Example

```
when HTTP_REQUEST {
result = b32_enc_str(input);    --input can be a cert in DER format
}
```

FortiADC version: V5.2

Used in events: ALL

## B32_dec(str)

Decodes a Base32 encoded string input and outputs the result in string format.

### Syntax

B32_dec(str);

### Arguments

| Name | Description |
|---|---|
| str | The string which will be calculated. |

### Example

```
when HTTP_REQUEST {
str = "abc"
dec = b32_dec(str)
}
```

FortiADC version: V5.2

Used in events: ALL

## B32_dec_str(str)

Decodes a Base32 encoded string input and outputs the result in string format. In some cases you need a version to deal with it.

### Syntax

B32_dec_str(str);

### Arguments

| Name | Description |
|---|---|
| str | The string which will be calculated. |

### Example

```
when HTTP_REQUEST {
result = b32_dec_str(input);    --input can be a cert in DER format
}
```

FortiADC version: V5.2

Used in events: ALL

## B64_enc(str)

Encodes a string input in Base64 and outputs the result in string format.

### Syntax

B64_enc(str);

### Arguments

| Name | Description |
|---|---|
| str | The string which will be calculated. |

### Example

```
when HTTP_REQUEST {
result = b64_enc(input);
--Input can be general format:
str=”test string”
a=12
en=b64_enc(“%s, 123 %d”, str, a);
}
```

FortiADC version: V4.8

Used in events: ALL

## B64_dec(str)

Decodes a Base64 encoded string input and outputs the result in string format.

### Syntax

B64_dec(str);

### Arguments

| Name | Description |
|---|---|
| str | The string which will be calculated. |

### Example

```
when HTTP_REQUEST {
result = b64_dec(input);
str=”test string”
a=12
de=b64_dec("%s, 123 %d", str, a);
}
```

FortiADC version: V4.8

Used in events: ALL

## Get_pid()

Returns the PID value of the VS process.

### Syntax

Get_pid();

### Arguments

N/A

### Example

```
when HTTP_REQUEST {
pid = get_pid();
debug("VS PID is : %d\n", pid)
}
```

FortiADC version: V5.2

Used in events: ALL

## Table_to_string(t)

Returns the table in a string.

### Syntax

Table_to_string(t);

### Arguments

| Name | Description |
|---|---|
| t | The table which specifies the information. |

### Example

```
when HTTP_REQUEST {
t={};
t[1]=97;
t[2]=98;
t[3]=99;
t[4]=1;
str = table_to_string(t);
debug("str is %s\n", str)
}
Result:
str is abc
```

FortiADC version: V4.8

Used in events: ALL

## Htonl(int)

Converts a 32 bit long integer from host byte order to network byte order.

### Syntax

htonl(int);

### Arguments

| Name | Description |
|---|---|
| int | An integer which will be calculated. |

### Example

```
when HTTP_REQUEST {
str="0x12345678"
test=htonl(str)
debug("return : %x \n", test)
}
Result:
return: 78563412
```

FortiADC version: V4.8

Used in events: ALL

## Ntohs(int)

Converts a 16 bit short integer from network byte order to host byte order.

### Syntax

ntohs(int);

### Arguments

| Name | Description |
|---|---|
| int | An integer which will be calculated. |

### Example

```
when HTTP_REQUEST {
str="0x12345678"
test=ntohs(str)
debug("return : %x \n", test)
}
Result:
Return: 7856
```

FortiADC version: V4.8

Used in events: ALL

## Htons(int)

Converts a 16 bit short integer from host byte order to network byte order.

### Syntax

htons(int);

### Arguments

| Name | Description |
|---|---|
| int | An integer which will be calculated. |

### Example

```
when HTTP_REQUEST {
str="0x12345678"
test=htons(str)
debug("return : %x \n", test)
}
Result
Return: 7856
```

FortiADC version: V4.8

Used in events: ALL

## Ntohl(int)

When receiving long integers in HTTP response from the network, this command converts a 32 bit long integer from network byte order to host byte order.

### Syntax

ntohl(int);

### Arguments

| Name | Description |
|---|---|
| int | An integer which will be calculated. |

### Example

```
when HTTP_REQUEST {
str="0x12345678"
test=ntohl(str)
debug("return : %x \n", test)
log("record a log: %x \n", test)
}
Result:
return: 78563412
```

FortiADC version: V4.8

Used in events: ALL

## To_HEX(str)

Returns the HEX calculate of the string.

### Syntax

To_HEX(str);

### Arguments

| Name | Description |
|---|---|
| str | A string which will be calculated. |

### Example

```
when HTTP_REQUEST {
str = "\0\123\3"
hex = to_HEX(str)
debug("this str in hex is: %s\n", hex)
}
```

FortiADC version: V4.8

Used in events: ALL

## Debug(str)

Prints the debug information when VS using scripting.

### Syntax

debug(str);

### Arguments

| Name | Description |
|---|---|
| str | A string which will be printed. |

### Example

```
when HTTP_REQUEST {
debug("http request method is %s.\n", HTTP:method_get())
}
```

FortiADC version: V4.3

Used in events: ALL

## Log(str)

Prints the scripting running information in log format. When using this command, you should enable scripting log.

### Syntax

log(str);

### Arguments

| Name | Description |
|---|---|
| str | A string which will be logged. |

### Example

```
when HTTP_REQUEST {
log("http request method is %s.\n", HTTP:method_get())
}
```

FortiADC version: V4.8

Used in events: ALL

## File_open(path, str)

Opens a file, returns a file object.

### Syntax

File_open(path, str);

### Arguments

| Name | Description |
|---|---|
| str | A string which specifies the method to open the file. |
| path | A string which specifies the file path. |

### Example

```
when HTTP_REQUEST {
filepath = "/etc/resolv.conf";
fp = file_open(filepath,"r");
if not fp then
debug("file open failed\n");
end
repeat
line = file_gets(fp, 256);
if line then
debug("line %s", line);
end
until not line
file_close(fp);
}
```

FortiADC version: V5.2

Used in events: ALL

## File_get(file, size)

Returns the file content.

### Syntax

File_get(file, size);

### Arguments

| Name | Description |
|---|---|
| file | A file object that get from file_open() |

FortiADC version: V5.2

Used in events: ALL

## File_close(file)

Closes a file.

### Syntax

File_close(file);

### Arguments

| Name | Description |
|---|---|
| file | A file object which will be closed. |

FortiADC version: V5.2

Used in events: ALL

# WAF commands

WAF commands contain functions for obtaining and manipulating WAF related result information:

WAF:enable() on page 163 — Enables the current session's WAF scan function.

WAF:disable() on page 164 — Disables the current session's WAF scan function.

WAF:status() on page 164 — Returns a status string to specify the current status of WAF detection. The status may be "enable" or "disable".

WAF:action() on page 164 — Returns the current session's WAF action. This can only be called in an ATTACK_ DETECTED event.

WAF:override_action(string) on page 165 — Overrides the current stage's detected action to the specified.

WAF:violations() on page 166 — Returns a table that includes all the violations detected by the current WAF stage as string values.

WAF:abandon_violation() on page 167 — Removes a violation by the specified signature ID. The signature ID should be a valid integer that is already in violations, otherwise, you can list the violations by calling WAF:violations. If the signature ID is not valid, then it will return "false", otherwise, it will return "true".

WAF:raise_violation(string) on page 167 — Raises a violation immediately. This function will send a log by the input arguments. If the signature ID is already raised by the WAF then this command will override it.

WAF:abandon_all() on page 169 — Abandons all of the results detected by the WAF module, including all of the violations, and resets the action to "pass".

WAF:block(integer) on page 170 — Blocks the current session's client IP. Specify the period of the block in seconds as an integer (Range: 1-2147483647, default = 3600).

WAF:unblock() on page 170 — Unblocks the client IP of the current session if it is already blocked.

## WAF:enable()

Enables the current session's WAF scan function.

### Syntax

WAF:enable();

### Arguments

N/A

### Example

```
when WAF_REQUEST_ATTACK_DETECTED {
local s = WAF:status()
debug("test WAF_REQUEST_ATTACK_DETECTED, status %s\n", s)
WAF:enable()
}
```

## WAF:disable()

Disables the current session's WAF scan function.

### Syntax

WAF:disable();

### Arguments

N/A

### Example

```
when WAF_REQUEST_ATTACK_DETECTED {
local s = WAF:status()
debug("test WAF_REQUEST_ATTACK_DETECTED, status %s\n", s)
WAF:disable()
}
```

## WAF:status()

Returns a status string to specify the current status of WAF detection. The status may be "enable" or "disable".

### Syntax

WAF:status();

### Arguments

N/A

### Example

```
when WAF_REQUEST_ATTACK_DETECTED {
local s = WAF:status()
debug("test WAF_REQUEST_ATTACK_DETECTED, status %s\n", s)
WAF:disable()
}
```

## WAF:action()

Returns the current session's WAF action. This can only be called in an ATTACK_DETECTED event.

The return value is a string, which may include the following values:

- "pass"
- "deny"
- "block"
- "redirect"
- "captcha"

### Syntax

WAF:action();

### Arguments

N/A

### Example

```
when WAF_REQUEST_ATTACK_DETECTED {
local s = WAF:action()
debug("test WAF_REQUEST_ATTACK_DETECTED, action %s\n", s)
WAF:override_action("deny", 501);
}
```

## WAF:override_action(string)

Overrides the current stage's detected action to the specified.

### Syntax

WAF:override_action(string);

### Arguments

| Name | Description |
|---|---|
| deny | Requires a second argument specifying the deny code.<br>The deny code should be an integer from the following:<br>200, 202, 204, 205, 400, 403, 404, 405, 406, 408, 410, 500, 501, 502, 503, 504.<br>**Note**: If the deny code is not specified or it is an invalid integer, then it will be defaulted to 403. |
| pass | The WAF stage's detected action may be allowed to pass. |
| captcha | Requires the client to successfully fulfill the CAPTCHA request. |
| block | Requires a second argument specifying the period of the block as an integer (Range: 1-2147483647, default = 3600).<br>**Note**: If the period is not specified, then it will be defaulted to 3600. |
| redirect | Requires a second argument specifying the redirect URL, and it should be a valid string. The redirect URL must be specified, otherwise this function will fail.<br>The return value is a bool value; when the function fails, it will return false, otherwise, it will return true. |

### Example

```
when WAF_REQUEST_ATTACK_DETECTED {
local s = WAF:action()
debug("test WAF_REQUEST_ATTACK_DETECTED, action %s\n", s)
WAF:override_action("deny", 501);
}
```

### WAF:violations()

Returns a table that includes all the violations detected by the current WAF stage as string values.

The table fields include the following:

| Name | Description |
|---|---|
| severity | Includes the values "low", "medium", and "high". |
| information | The information that the WAF module defined when the specific attack was detected. |
| signature | An integer ID that is defined by the WAF module for every different attack. |
| action | The defined action is a violation, including the values "pass", "deny", "block", "redirect", or "captcha". |
| sub-category | The violation is related to a WAF sub-category field name.<br>The string should be from the following list:<br>• waf_web_attack_signature<br>• waf_http_protocol_const<br>• waf_heur_sqlxss_inject_detect<br>• waf_url_protect,waf_bot_detection<br>• waf_xml_check<br>• waf_json_check<br>• waf_web_scraping<br>• waf_cookie_security<br>• waf_csrf_protection<br>• waf_html_input_validation<br>• waf_brute_force,waf_data_leak_prevention<br>• waf_credential_stuffing<br>• waf_openapi_check<br>• waf_api_gateway |
| owasp-top10 | The violation is related to the OWASP TOP10 field name. |

#### Syntax

WAF:violations();

#### Arguments

N/A

### Example

```
when WAF_REQUEST_ATTACK_DETECTED {
debug("test WAF_REQUEST_ATTACK_DETECTED\n")
local vl = WAF:violations();
for k, v in pairs(vl) do
debug("%d. Violation: signature %d, severity %s, information %s, action %s, sub-category %s,
owasp-top10 %s.\n", k, v["signature"], v["severity"], v["information"], v["action"], v["sub-
category"], v["owasp-top10"]);
}
```

## WAF:abandon_violation()

Removes a violation by the specified signature ID. The signature ID should be a valid integer that is already in violations, otherwise, you can list the violations by calling WAF:violations. If the signature ID is not valid, then it will return "false", otherwise, it will return "true".

This command can only be called in the ATTACK_DETECTED event.

### Syntax

WAF:abandon_violation();

### Arguments

N/A

### Example

```
when WAF_REQUEST_ATTACK_DETECTED {
debug("test WAF_REQUEST_ATTACK_DETECTED\n")

local vl = WAF:violations();
for k, v in pairs(vl) do
debug("%d. Violation: signature %d.\n", k, v["signature"]);
WAF:abandon_violation(v["signature"]);
end
v = {};
v["signature-id"] = 100010000;
v["severity"] = "high";
v["information"] = "waf raise violation test";
v["action"] = "deny";
v["sub-category"] = "waf_url_protect";
v["owasp-top10"] = "test-owasp10";
WAF:raise_violation(v);
}
```

## WAF:raise_violation(string)

Raises a violation immediately. This function will send a log by the input arguments. If the signature ID is already raised by the WAF then this command will override it.

This function will prevent the WAF action from executing as specified. To override the WAF action, call WAF:override_action(string).

### Syntax

WAF:raise_violation(string);

### Arguments

| Name | Description |
|---|---|
| severity | Overrides the severity string that includes the values "low", "medium", and "high".<br>**Note**: If the value is not specified, then "low" will be used as the severity level for the violation. |
| information | The violation will show the information that the WAF module defined when the specific attack was detected.<br>**Note:** If this is not specified, then it will show "N/A" as the violation's information. |
| signature | The attack signature string ID that WAF detected. Users can specify this if the signature ID already exists in the violation, which will override the related field of the violation by this function.<br>**Note**: This argument must be specified. |
| action | The violation will show the defined action, including the values "pass", "deny", "block", "redirect", or "captcha".<br>**Note**: If this is not specified, then the violation's action will take "pass" as default. |
| block-period | If the action is "block", then this argument must be specified. Otherwise, this will be defaulted to 3600.<br>This argument should be an integer and range from 1-2147483647. |
| redirect-url | If the action is "redirect", then this argument must be specified. Otherwise, the "redirect" action will be ignored and will take a "deny" action instead. |
| deny-code | If the action is "deny", then this argument must be specified.<br>The deny code should be an integer from the following:<br>200, 202, 204, 205, 400, 403, 404, 405, 406, 408, 410, 500, 501, 502, 503, 504.<br>If the deny code is not specified or it is an invalid integer, then it will be defaulted to 403.<br>The return value is a bool value; when the operation is successful, it will return true, otherwise, it will return false. |
| sub-category | This string specifies the violation's sub-category.<br>The string should be from the following list:<br>• waf_web_attack_signature<br>• waf_http_protocol_const<br>• waf_heur_sqlxss_inject_detect<br>• waf_url_protect,waf_bot_detection<br>• waf_xml_check |

| Name | Description |
|---|---|
| | • waf_json_check<br>• waf_web_scraping<br>• waf_cookie_security<br>• waf_csrf_protection<br>• waf_html_input_validation<br>• waf_brute_force,waf_data_leak_prevention<br>• waf_credential_stuffing<br>• waf_openapi_check<br>• waf_api_gateway<br>**Note**: This argument is not required to be specified. But if this argument is not specified or if the string is not a valid sub-category, then it will default to "waf_web_attack_signature". |
| owasp-top10 | The string will show the violation that is related to the OWASP TOP10 field name.<br>**Note**: If this argument is not specified, then it will default to "unknown". |

### Example

```
when WAF_REQUEST_ATTACK_DETECTED {
debug("test WAF_REQUEST_ATTACK_DETECTED\n")
local vl = WAF:violations();
for k, v in pairs(vl) do
debug("%d. Violation: signature %d.\n", k, v["signature"]);
WAF:abandon_violation(v["signature"]);
end
v = {};
v["signature-id"] = 100010000;
v["severity"] = "high";
v["information"] = "waf raise violation test";
v["action"] = "deny";
v["sub-category"] = "waf_url_protect";
v["owasp-top10"] = "test-owasp10";
WAF:raise_violation(v);
}
```

## WAF:abandon_all()

Abandons all of the results detected by the WAF module, including all of the violations, and resets the action to "pass".

This command can only be called in the ATTACK_DETECTED event.

### Syntax

WAF:abandon_all();

### Arguments

N/A

### Example

```
when WAF_REQUEST_ATTACK_DETECTED {
debug("test WAF_REQUEST_ATTACK_DETECTED\n")
WAF:abandon_all()
}
```

## WAF:block(integer)

Blocks the current session's client IP. Specify the period of the block in seconds as an integer (Range: 1-2147483647, default = 3600).

### Syntax

WAF:block(integer);

### Arguments

| Name | Description |
|---|---|
| int | An integer ranging from 1-2147483647. |

### Example

```
when WAF_REQUEST_ATTACK_DETECTED {
debug("test WAF_REQUEST_ATTACK_DETECTED\n")
WAF:block(3600)
}
```

## WAF:unblock()

Unblocks the client IP of the current session if it is already blocked.

### Syntax

WAF:unblock();

### Argument

N/A

### Example

```
when WAF_REQUEST_BEFORE_SCAN {
local s = WAF:status()
debug("test WAF_REQUEST_BEFORE_SCAN, status %s\n", s)
WAF:unblock()
}
```

# Examples of built-in predefined scripts

As of V5.3.0, FortiADC has the following built-in scripts; the user can refer to these examples to finish their scripting as needed.

| Predefined script | Description |
|---|---|
| IP_COMMANDS | Used to get various types IP Address and port number between client and server side. |
| SNAT_COMMANDS | Allows you to overwrite client source address to a specific IP for certain clients, also support IPv4toIPv6 or IPv6toIPv4 type.<br>Note: Make sure the flag SOURCE ADDRESS is selected in the HTTP or HTTPS type of profile. |
| SOCKOPT_COMMAND_ USAGE | Allows the user to customize the TCP_send buffer and TCP_receive buffer size. |
| TCP_EVENTS_n_COMMANDS | Demonstrates how to reject a TCP connection from a client in TCP_ACCEPTED event. |
| GEOIP_UTILITY | Used to fetch the GEO information country and possible province name of an IP address. |
| CONTENT_ROUTING_by_URI | Routes to a pool member based on URI string matches. You should not use this script as is. Instead, copy it and customize the URI string matches and pool member names. |
| CONTENT_ROUTING_by_X_ FORWARDED_FOR | Routes to a pool member based on IP address in the X-Forwarded-For header. You should not use this script as is. Instead, copy it and customize the X-Fowarded-For header values and pool member names. |
| GENERAL_REDIRECT_DEMO | Redirects requests to a URL with the user-defined code and cookie.<br>Note: Do NOT use this script "as is". Instead, copy and customize the code, URL, and cookie. |
| HTTP_2_HTTPS_ REDIRECTION | Redirects requests to the HTTPS site. You can use this script without changes |
| HTTP_2_HTTPS_ REDIRECTION_FULL_URL | Redirects requests to the specified HTTPS URL.<br>Note: This script can be used directly, without making any change. |
| REDIRECTION_by_STATUS_ CODE | Redirects requests based on the status code of server HTTP response (for example, a redirect to the mobile version of a site). Do NOT use this script "as is". Instead, copy it and customize the condition in the server HTTP response status code and the URL values. |

| Predefined script | Description |
|---|---|
| REDIRECTION_by_USER_ AGENT | Redirects requests based on User Agent (for example, a redirect to the mobile version of a site). You should not use this script as is. Instead, copy it and customize the User Agent and URL values |
| REWRITE_HOST_n_PATH | Rewrites the host and path in the HTTP request, for example, if the site is reorganized. You should not use this script as is. Instead, copy |
| REWRITE_HTTP_2_HTTPS_in_ LOCATION | Rewrites HTTP location to HTTPS, for example, rewrite<br>“Location:http://www.example.com” to<br>“Location:https://www.example.com”<br>Note: You can use the script directly, without making any change |
| REWRITE_HTTP_2_HTTPS_in_ REFERER | Rewrites HTTP referer to HTTPS, for example, rewrite<br>“Referer: http://www.example.com” to<br>“Referer: https://www.example.com”.<br>Note: You can use the script directly, without making any change. |
| REWRITE_HTTPS_2_HTTP_in_ LOCATION | Rewrites HTTPS location to HTTP, for example, rewrite<br>“Location:https://www.example.com” to<br>“Location:http://www.example.com”.<br>Note: You can use the script directly, without making any change. |
| REWRITE_HTTPS_2_HTTP_in_ REFERER | Rewrites HTTPS referer to HTTP, for example, rewrite<br>“Referer: https://www.example.com” to<br>“Referer: http://www.example.com”.<br>Note: You can use the script directly, without making any change |
| HTTP_DATA_FETCH_SET_ DEMO | Collects data in HTTP request body or HTTP response body. In HTTP_ REQUEST or HTTP_RESPONSE, you could collect specified size data with “size” in collect().In HTTP_DATA_REQUEST or HTTP_DATA_RESPONSE. You could print the data use “content”, calculate data length with “size”, and rewrite the data with “set”.<br>Note: Do NOT use this script "as is". Instead, copy it and manipulate the collected data. |
| HTTP_DATA_FIND_REMOVE_ REPLACE_DEMO | Finds a specified string, removes a specified string, or replaces a specified string to new content in HTTP data.<br>Note: Do NOT use this script "as is". Instead, copy it and manipulate the collected data. |
| URL_UTILITY_COMMANDS | Demonstrate how to use those url tools to encode/decode/parser/compare . |
| USE_REQUEST_HEADERS_ in_OTHER_EVENTS | Stores a request header value in an event and uses it in other events. For example, you can store a URL in a request event, and use it in a response event.<br>Note: Do NOT use this script "as is". Instead, copy it and customize the content you want to store, use collect() in HTTP_REQUEST to trigger HTTP_DATA_ REQUEST,or use collect() in HTTP_ RESPONSE to trigger HTTP_DATA_ RESPONSE. |

| Predefined script | Description |
|---|---|
| SSL_EVENTS_n_COMMANDS | Demonstrate how to fetch the SSL certificate information and some of the SSL connection parameters between server and client side. |
| AUTH_COOKIE_BAKE | Allows you to retrieve the baked cookie and edit the cookie content. |
| AUTH_EVENTS_n_ COMMANDS | Used to get the information from authentication process. |
| OPTIONAL_CLIENT_ AUTHENTICATION | Performs optional client authentication.<br>Note: Before using this script, you must have the following four parameters configured in the client-ssl-profile:<br>l client-certificate-verify—Set to the verify you'd like to use to verify the client certificate.<br>l client-certificate-verify-option—Set to optional<br>l ssl-session-cache-flag—Disable.<br>l use-tls-tickets—Disable.<br>l |
| CUSTOMIZE_AUTH_KEY | Demonstrate how to customize the crypto key for authentication cookie. |
| COOKIE_COMMANDS | Demonstrate the cookie command to get the whole cookie in a table and how to remove/insert/set the cookie attribute. |
| COOKIE_COMMANDS_USAGE | Demonstrate the sub-function to handle the cookie attribute "SameSite" and others. |
| COOKIE_CRYPTO_ COMMANDS | Used to perform cookie encryption/decryption on behalf of the real server. |
| AES_DIGEST_SIGN_2F_ COMMANDS | Demonstrate how to use AES to encryption/decryption data and some tools to generate the digest. |
| CLASS_SEARCH_n_MATCH | Demonstrates how to use the class_match and class_search utility function. |
| COMPARE_IP_ADDR_2_ ADDR_GROUP_DEMO | Compares an IP address to an address group to determine if the IP address is included in the specified IP group. For example ,192.168.1.2 is included 192.168.1.0/24.<br>Note: Do NOT use this script "as is". Instead, copy it and customize the IP address and the IP address group. |
| INSERT_RANDOM_ MESSAGE_ID_DEMO | Inserts a 32-bit hex string into the HTTP header with a parameter “Message-ID”.<br>Note: You can use the script directly, without making any change. |
| MANAGEMENT_COMMANDS | Allow you to disable/enable rest of the events from executing. |
| UTILITY_FUNCTIONS_DEMO | Demonstrates how to use the basic string operations and random number/alphabet, time, MD5, SHA1, SHA2, BASE64, BASE32, table to string conversion, network to host conversion utility function. |

| Predefined script | Description |
|---|---|
| SPECIAL_CHARACTERS_ HANDLING_DEMO | Shows how to use those "magic characters" which have special meanings when used in a certain pattern. The magic characters are ( ) . % + - * ? [ ] ^ $ |
| MULTIPLE_SCRIPT_ CONTROL_DEMO_1 | Uses demo_1 and demo_2 script to show how multiple scripts work. Demo_1 with priority 12 has a higher priority.<br>Note: You could enable or disable other events. Do NOT use this script "as is". Instead, copy it and customize the operation. |
| MULTIPLE_SCRIPT_ CONTROL_DEMO_2 | Uses demo_1 and demo_2 script to show how multiple scripts work. Demo_2 with priority 24 has a lower priority.<br>Note: You could enable or disable other events. Do NOT use this script "as is". Instead, copy it and customize the operation |

www.fortinet.com

Copyright© 2024 Fortinet, Inc. All rights reserved. Fortinet®, FortiGate®, FortiCare® and FortiGuard®, and certain other marks are registered trademarks of Fortinet, Inc., and other Fortinet names herein may also be registered and/or common law trademarks of Fortinet. All other product or company names may be trademarks of their respective owners. Performance and other metrics contained herein were attained in internal lab tests under ideal conditions, and actual performance and other results may vary. Network variables, different network environments and other conditions may affect performance results. Nothing herein represents any binding commitment by Fortinet, and Fortinet disclaims all warranties, whether express or implied, except to the extent Fortinet enters a binding written contract, signed by Fortinet's General Counsel, with a purchaser that expressly warrants that the identified product will perform according to certain expressly-identified performance metrics and, in such event, only the specific performance metrics expressly identified in such binding written contract shall be binding on Fortinet. For absolute clarity, any such warranty will be limited to performance in the same ideal conditions as in Fortinet's internal lab tests. Fortinet disclaims in full any covenants, representations, and guarantees pursuant hereto, whether express or implied. Fortinet reserves the right to change, modify, transfer, or otherwise revise this publication without notice, and the most current version of the publication shall be applicable.