



## FortiADC - Script Reference Guide

**FORTINET DOCUMENT LIBRARY**

<https://docs.fortinet.com>

**FORTINET VIDEO GUIDE**

<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/support-and-training/training.html>

**NSE INSTITUTE**

<https://training.fortinet.com>

**FORTIGUARD CENTER**

<https://www.fortiguard.com>

**END USER LICENSE AGREEMENT**

<https://www.fortinet.com/doc/legal/EULA.pdf>

**FEEDBACK**

Email: [techdoc@fortinet.com](mailto:techdoc@fortinet.com)



February 4, 2020

FortiADC Script Reference Guide

---

# TABLE OF CONTENTS

<b>Change Log</b>	4
<b>Introduction</b>	5
<b>Definition of terms</b>	6
<b>Configuration overview</b>	7
<b>Script reference</b>	8
Control structures	8
Operators	8
String Library	9
Multiple Scripting Usage	10
Dynamic scripting configuration change	10
Function	10
Events	12
WAF events	13
Predefined commands	14
Management commands	15
IP commands	17
TCP commands	20
HTTP commands	24
HTTP DATA commands	45
Cookie commands	50
Authentication commands	52
SSL commands	57
GEO IP commands	64
RAM cache commands	65
PROXY commands	71
LB commands	72
Global commands	72
WAF commands	101
Examples of built-in predefined scripts	107

## Change Log

Date	Change Description
2019-07-19	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. After you initially save the configuration, you cannot edit the name.
Input	Type or paste the script.

# Script reference

- Control structures on page 8
- Operators on page 8
- String Library on page 9
- Multiple Scripting Usage on page 10
- Dynamic scripting configuration change on page 10
- Function on page 10
- Events on page 12
- Examples of built-in predefined scripts on page 107

## Control structures

The table below lists Lua control structures.

Type	Structure
if then else	<pre>if condition1 then ... elseif condition2 then ... else ... end</pre>
for	<p>Fetch all values of table 't'</p> <pre>for k, v in pairs(t) do ... end</pre>

## 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 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 using 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 wants 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.  Right now, allows the user to set tcp options, later can be used to config VS.  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.	V4.3
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.  TCP:sockopt() and management commands are available.  IP:client_port()/client_addr()/client_ip_ver() are available.	V5.2
SERVER_CONNECTED	When Htpproxy deems that the backend real server is connected.  TCP:sockopt() and management commands are available.	V5.2

Event Name	Description	Available Version
	Server-side IP functions are available.	
HTTP_RESPONSE	When a HTTP response comes from real server.	V4.3
HTTP_DATA_RESPONSE	Allows the user to manipulate http response data.	V4.8 and later
SERVER_CLOSED	When Htpproxy 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
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	<p>When authentication(HTML Form / HTTP-basic) is done. If auth event detect, it still trigger the AUTH_RESULT.</p> <p>LB:routing, ip commands, management commands and AUTH:commands can be used in AUTH_RESULT event.</p> <p>The following commands are support in AUTH_RESULT.</p> <pre>HTTP:"uri_get path_get method_get query_get" LB:"routing" 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" IP:"client_port local_port remote_port client_addr local_addr remote_addr client_ip_ver" MGM:"rand_id get_session_id disable_event enable_event set_event set_auto disable_auto enable_auto"</pre>	V5.2
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. If WAF function is not enabled on VS, then this will not be triggered.	when WAF_REQUEST_BEFORE_SCAN { debug("test WAF_REQUEST_BEFORE_SCAN\n") }
WAF_RESPONSE_BEFORE_SCAN	Before WAF_SCAN_STAGE_RES_HEADER start. If WAF function is not enabled on VS, then this will not be triggered.	when WAF_REQUEST_ATTACK_DETECTED { debug("test WAF_REQUEST_ATTACK_DETECTED\n") }
WAF_REQUEST_ATTACK_DETECTED	After all request stages when there are attacks detected (violation). If WAF function is not enabled on VS, then this will not be triggered. If WAF module does not detect any violations, then this will not be triggered.	when WAF_RESPONSE_BEFORE_SCAN { debug("test WAF_RESPONSE_BEFORE_SCAN\n") }
WAF_RESPONSE_ATTACK_DETECTED	After all response stages when there are attacks detected (violation). If WAF function is not enabled on VS, then this will not be triggered. If WAF module does not detect any violations, then this will not be triggered.	when WAF_RESPONSE_ATTACK_DETECTED { debug("test WAF_RESPONSE_ATTACK_DETECTED\n") }

## Predefined commands

- Management commands on page 15
- IP commands on page 17
- TCP commands on page 20
- HTTP commands on page 24
- HTTP DATA commands on page 45
- Cookie commands on page 50

- [Authentication commands on page 52](#)
- [SSL commands on page 57](#)
- [GEO IP commands on page 64](#)
- [RAM cache commands on page 65](#)
- [PROXY commands on page 71](#)
- [LB commands on page 72](#)
- [Global commands on page 72](#)

## Management commands

### MGM:get\_session\_id()

Returns the session id.

#### Syntax

MGM:get\_session\_id();

**Arguments:** N/A

#### Examples

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

FortiADC version: V5.0

Used in events:

Used in all events except VS\_LISTENER\_BIND.

### MGM:rand\_id()

Returns a 32-long string of HEX symbols.

#### Syntax

MGM:rand\_id();

**Arguments:** N/A

#### Examples

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

FortiADC version: V5.0

Used in events:

Used in all events 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.

### Examples

```
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 though they are disabled in 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.

### Examples

```
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:Client\_addr()

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

#### Syntax

cip=IP:client\_addr()

**Arguments: N/A**

**Examples:**

```
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 event except VS\_LISTENER\_BIND

### IP:server\_addr()

Returns the IP address of the server in backend.

#### Syntax

sip=IP:server\_addr()

**Arguments: N/A**

**Examples:**

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

### Syntax

sip=IP:local\_addr()

**Arguments:** N/A

**Examples:**

Please refer to IP:client\_addr() example.

FortiADC version: V5.0

Used in events:

All events 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 events except VS\_LISTENER\_BIND / SERVER\_BEFORE\_CONNECT

## IP:client\_port()

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

### Syntax

cp=IP:client\_port()

**Arguments:** N/A

**Examples:**

Please refer to IP:client\_addr() example.

FortiADC version: V5.0

Used in events:

All event except VS\_LISTENER\_BIND

## IP:server\_port()

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

### Syntax

sp=IP:server\_port()

**Arguments: N/A**

**Examples:**

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\_port()

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

### Syntax

sp=IP:local\_port()

**Arguments: N/A**

**Examples:**

Please refer to IP:client\_addr() example.

FortiADC version: V5.0

Used in events:

All events except VS\_LISTENER\_BIND / SERVER\_BEFORE\_CONNECT

## IP:remote\_port()

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

### Syntax

rp=IP:remote\_port()

**Arguments: N/A**

**Examples:**

Please refer to IP:client\_addr() example.

FortiADC version: V5.0

Used in events:

All events except VS\_LISTENER\_BIND / SERVER\_BEFORE\_CONNECT

### **IP:client\_ip\_ver()**

Returns the current client ip version number of the connection, either 4 or 6.

#### **Syntax**

cv=IP:client\_ip\_ver ()

**Arguments: N/A**

**Examples:**

Please refer to IP:client\_addr() example.

FortiADC version: V5.0

Used in events:

All events except VS\_LISTENER\_BIND

### **IP:server\_ip\_ver()**

Returns the current server ip version number of the connection, either 4 or 6.

#### **Syntax**

cv=IP:server\_ip\_ver ()

**Arguments: N/A**

**Examples:**

Please refer to IP:client\_addr() example.

FortiADC version: V5.0

Used in events:

server-side event

## **TCP commands**

### **TCP:reject()**

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

#### **Syntax**

TCP:reject();

**Arguments: N/A**

**Examples**

```

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

```

FortiADC version: V5.0

Used in events:

Used in 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 set\_snat\_ip() you need to make sure the flag SOURCE ADDRESS is selected in the HTTP or HTTPS type of profile.

### **Arguments**

Name	Description
str	A string which specifies the ip address.

### **Examples**

```

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: vs must config client-address enable in the profile

```

config load-balance profile
edit "http"
set type http
set client-address enable
next
end
FortiADC version: V5.2
Used in events:
Used in TCP_ACCEPTED / HTTP_REQUEST / HTTP_DATA_REQUEST / CLIENTSSL_HANDSHAKE

```

## TCP:clear\_snat\_ip()

Allows the user to clear whatever customized ip the user set with set\_snat\_ip().

### Syntax

TCP:clear\_snat\_ip();

### Arguments

Name	Description
str	A string which will be calculated.

### Examples

```
when HTTP_REQUEST {
if TCP:clear_snat_ip() then
debug("clear SNAT ip!\n")
}
FortiADC version: V5.0
Used in events: ALL
Used in 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. Right now, only supports snd\_buf and rcv\_buf buffer size.

For client-side events, it applies to 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.

### Examples

```
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.
```

## Script reference

---

```
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:

Used in client-side events, including TCP\_BIND, TCP\_ACCEPTED, HTTP\_REQUEST, HTTP\_DATA\_REQUEST

Used in server-side events, including HTTP\_RESPONSE, HTTP\_DATA\_RESPONSE, BEFORE\_CONNECT, SERVER\_CONNECTED.

## HTTP commands

### 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

#### Examples

```
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.

### Examples

```
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:

Used in 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.

### Examples

```
when HTTP_REQUEST {
host = HTTP:header_get_value("Host");
```

```
debug("host is %s\n", host);  
}
```

FortiADC version: V4.3

Used in events:

Used in 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.

### Examples

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

FortiADC version: V4.3

Used in events:

Used in 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

### Examples

```
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:

Used in **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>.

### Examples

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

FortiADC version: V4.3

Used in events:

Used in **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>.

### Examples

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

FortiADC version: V4.8

Used in events:

Used in 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

### **Examples**

```
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:

Used in HTTP\_REQUEST / HTTP\_RESPONSE

## **HTTP:header\_exists(header\_name)**

Returns true when the header <header\_name> exists and false when not exists

### **Syntax**

```
HTTP:header_exists(header_name);
```

### **Arguments**

Name	Description
Header_name	A string which specifies the header name.

### **Examples**

```
when HTTP_REQUEST {  
if HTTP:header_exists("Cookie") then  
...  
end  
}
```

FortiADC version: V4.3

Used in events:

Used in 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.

### **Examples**

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

FortiADC version: V4.3

Used in events:

Used in HTTP\_REQUEST / HTTP\_RESPONSE

## **HTTP:method\_get()**

Returns the string of the HTTP request method.

### **Syntax**

HTTP:method\_get();

Arguments: N/A

### **Examples**

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

FortiADC version: V4.3

Used in events:

Used in 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.

### Examples

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

FortiADC version: V4.3

Used in events:

Used in HTTP\_REQUEST

## HTTP:path\_get()

Returns the string of the HTTP request path.

### Syntax

```
HTTP:path_get();
```

### Arguments: N/A

### Examples

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

FortiADC version: V4.3

Used in events:

Used in HTTP\_REQUEST / AUTH\_RESULT

## HTTP:path\_set(value)

Sets HTTP request path to the strng <value>.

### Syntax

```
HTTP:path_set(value);
```

### Arguments

Name	Description
Value	A string which specifies the path.

**Examples**

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

FortiADC version: V4.3

Used in events:

Used in HTTP\_REQUEST

**HTTP:uri\_get()**

Returns the string of the HTTP request URI.

**Syntax**

```
HTTP:uri_get();
```

Arguments: N/A

**Examples**

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

FortiADC version: V4.3

Used in events:

Used in HTTP\_REQUEST / AUTH\_RESULT

**HTTP:uri\_set(value)**

Sets HTTP request URI to the string <value>.

**Syntax**

```
HTTP:uri_set(value);
```

Arguments

Name	Description
value	a string which specifies the uri.

**Examples**

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

FortiADC version: V4.3

Used in events:

Used in HTTP\_REQUEST

## HTTP:query\_get()

Returns the string of the HTTP request query.

### Syntax

HTTP:query\_get();

### Arguments: N/A

### Examples

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

FortiADC version: V4.3

Used in events:

Used in HTTP\_REQUEST / AUTH\_RESULT

## HTTP:query\_set(value)

Sets HTTP request query to the string <value>.

### Syntax

HTTP:query\_set(value);

### Arguments

Name	Description
value	A string which specifies the uri.

### Examples

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

FortiADC version: V4.3

Used in events:

Used in 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.

**Examples**

```
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.

Support 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.

**Examples**

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

**Note:**

: it was finally redirected to www.abc.com with the cookie "server=nginx"

FortiADC version: V4.8

Used in events:

HTTP\_REQUEST / HTTP\_DATA\_REQUEST / HTTP\_RESPONSE

Can not 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.

### Examples

```
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`

Can not use in `HTTP_DATA_RESPONSE`

## HTTP:version\_get()

Returns the HTTP version of the request or response.

### Syntax

```
HTTP:version_get();
```

Arguments: N/A

### Examples

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

FortiADC version: V5.8

Used in events:

Used in `HTTP_REQUEST / HTTP_RESPONSE`

```
HTTP:version_set(value)
```

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

**Syntax**

```
HTTP:version_set(value);
```

**Arguments**

Name	Description
value	A string which specifies the version.

**Examples**

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

FortiADC version: V4.8

Used in events:

Used in HTTP\_REQUEST / HTTP\_RESPONSE

**HTTP:status\_code\_get()**

Returns the response status code output as string.

**Syntax**

```
HTTP:status_code_get();
```

Arguments: N/A

**Examples**

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

FortiADC version: V4.8

Used in events:

Used in 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.

**Examples**

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

FortiADC version: V4.8

Used in events:

Used in HTTP\_RESPONSE

## HTTP:code\_get()

Returns the response status code, out put as integer.

### Syntax

```
HTTP:code_get();
```

Arguments: N/A

### Examples

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

FortiADC version: V4.8

Used in events:

Used in 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.

### Examples

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

FortiADC version: V4.8

Used in events:

Used in HTTP\_RESPONSE

## HTTP:reason\_get()

Returns the response reason.

**Syntax**

```
HTTP:reason_get();
```

**Arguments: N/A****Examples**

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

FortiADC version: V4.8

Used in events:

Used in 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.

**Examples**

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

FortiADC version: V4.8

Used in events:

Used in 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****Examples**

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

FortiADC version: V4.6

Used in events:

Used in HTTP\_REQUEST / HTTP\_RESPONSE

## HTTP:local\_addr()

For HTTP\_REQUEST, return the ip address of the virtual server the client is connected to;  
for HTTP\_RESPONSE, return the incoming interface ip address of the return packet.

### Syntax

HTTP:local\_addr();

### Arguments: N/A

### Examples

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

FortiADC version: V4.6

Used in events:

Used in HTTP\_REQUEST / HTTP\_RESPONSE

## HTTP:server\_addr()

Returns the ip address of the server in HTTP\_RESPONSE.

### Syntax

HTTP:server\_addr();

### Arguments: N/A

### Examples

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

FortiADC version: V4.6

Used in events:

Used in 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

## Examples

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

FortiADC version: V4.6

Used in events:

Used in HTTP\_REQUEST / HTTP\_RESPONSE

## HTTP:client\_port()

Returns real client port number in a string format.

### Syntax

```
HTTP:client_port();
```

**Arguments: N/A**

### Examples

```
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:

Used in 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 virtual server port.

In HTTP\_RESPONSE, local\_port is gateway's port which used to connect

### Syntax

HTTP:local\_port();

**Arguments:** N/A

**Examples**

```
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:

Used in 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 client port.

In HTTP\_RESPONSE, remote\_port is real server port.

**Syntax**

HTTP:remote\_port();

**Arguments:** N/A

**Examples**

```
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:

Used in HTTP\_REQUEST / HTTP\_RESPONSE / HTTP\_DATA\_REQUEST / HTTP\_DATA\_RESPONSE

## HTTP:server\_port()

Returns the server port number in a string format. It's real server port

**Syntax**

HTTP:server\_port();

**Arguments:** N/A

## Examples

```
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:

Only used in [HTTP\\_RESPONSE / HTTP\\_DATA\\_RESPONSE](#)

## [HTTP:client\\_ip\\_ver\(\)](#)

Returns the client ip version number, can use to get ipv4 or ipv6 version

### Syntax

`HTTP:client_ip_ver();`

**Arguments: N/A**

### Examples

```
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:

Used in [HTTP\\_REQUEST / HTTP\\_RESPONSE / HTTP\\_DATA\\_REQUEST / HTTP\\_DATA\\_RESPONSE](#)

## [HTTP:server\\_ip\\_ver\(\)](#)

Returns the server ip version number, can use to get ipv4 or ipv6 version

### Syntax

`HTTP:server_ip_ver();`

**Arguments: N/A**

### Examples

```
when HTTP_REQUEST {  
    string=HTTP:server_ip_ver()
```

```
debug("\nresult: %s \n",string)
}
```

FortiADC version: V4.8

Used in events:

Only used in HTTP\_RESPONSE / HTTP\_DATA\_RESPONSE

## HTTP:close()

Close an HTTP connection using code 503.

Can support multiple close call.

### Syntax

```
HTTP:close();
```

Arguments: N/A

### Examples

```
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:

Used in HTTP\_REQUEST / HTTP\_RESPONSE

## HTTP:respond(t)

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

### Syntax

```
HTTP:respond(t);
```

### Arguments

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

### Examples

```
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\nXXXXXX Test Page XXXXXXXX";  
status = HTTP:respond(tt);  
debug("HTTP_respond() status: %s\n", status);  
}
```

FortiADC version: V5.2

Used in events:

Used in HTTP\_REQUEST / HTTP\_RESPONSE / HTTP\_DATA\_REQUEST

## [\*\*HTTP:get\\_session\\_id\(\)\*\*](#)

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

With this unique id, the user can play with Lua script 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 info in the HTTP request event

### Syntax

```
HTTP:get_session_id();
```

Arguments: N/A

### Examples

```
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:

Used in HTTP\_REQUEST / HTTP\_RESPONSE / HTTP\_DATA\_REQUEST / HTTP\_DATA\_RESPONSE

## HTTP:rand\_id()

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

### Syntax

HTTP:rand\_id();

### Arguments: N/A

### Examples

```
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 enable or disable.

### Syntax

HTTP:set\_event(t);

### Arguments

Name	Description
t	A table when to specify enable/disable an event.

### Examples

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

### Note:

Event can be "req", "res", "data\_req", "data\_res"

And 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:

Used in 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 next transaction even if they have been disabled in the current transaction. User can disable/enable this automatic behavior using this facility.

### Syntax

```
HTTP:set_auto(t);
```

### Arguments

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

### Examples

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

### Note:

Event can be "req", "res", "data\_req", "data\_res"

And 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:

Used in HTTP\_REQUEST or HTTP\_RESPONSE

## HTTP DATA commands

### Note:

FortiADC buffer size is 1.25M, this is FortiADC harm limit. If http data exceed 1.25M, FortiADC only collect 1.25M data, the left will be forwarded directly.

## HTTP:collect()

Collects body data from 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.

**Examples**

```
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:**

Size: means httpoxy a block size, limited by FortiADC HARD LIMIT

FortiADC version: V4.8

Used in events:

HTTP\_REQUEST / HTTP\_RESPONSE

## HTTP:payload(size)

Returns the size of the buffered content. Returns value is an integer. Used in HTTP\_DATA\_REQUEST or HTTP\_DATA\_RESPONSE

**Syntax**

HTTP:payload(t);

**Arguments**

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

**Examples**

```
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(t);
```

### Arguments

Name	Description
str	A string which will be calculated.

### Examples

```
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 “offset” field is missing, zero is assumed.

If “size” field is missing, it will operate on the whole buffered data.

FortiADC version: V4.8

Used in events:

Used in HTTP\_DATA\_REQUEST or HTTP\_DATA\_RESPONSE

## HTTP:payload(set)

Insert the specified data at the specified location.

### Syntax

```
HTTP:pyaload(t);
```

### Arguments

Name	Description
t	A table which will give the parameter: offset, size, data to set.

### Examples

```
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 “offset” field is missing, zero is assumed.

If “size” field is missing, it will operate on the whole buffered data.

FortiADC version: V4.8

Used in events:

Used in HTTP\_DATA\_REQUEST or 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 will give the parameter: data, offset, size, scope.

**Examples**

```
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 “offset” field is missing, zero is assumed.
- If “size” field is missing, it will operate on the whole buffered data.
- “scope” field can be either “first” or “all”

FortiADC version: V4.8

Used in events:

Used in HTTP\_DATA\_REQUEST or 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 will give the parameter: data to remove, offset, size, scope.

### Examples

```
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:

Used in HTTP\_DATA\_REQUEST or 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 will give the parameter: data to replace, new_data, offset, size, scope.

### Examples

```
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:

Used in HTTP\_DATA\_REQUEST or HTTP\_DATA\_RESPONSE

## Cookie commands

### HTTP:cookie\_list()

Returns a list of cookies: their names and values.

#### Syntax

```
HTTP:cookie_list();
```

Arguments: N/A

#### Examples

```
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:

Used in 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.

**Examples**

```
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:

Used in 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.

**Examples**

```
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:

Used in HTTP\_REQUEST / HTTP\_RESPONSE

## Authentication commands

### AUTH:get\_baked\_cookie()

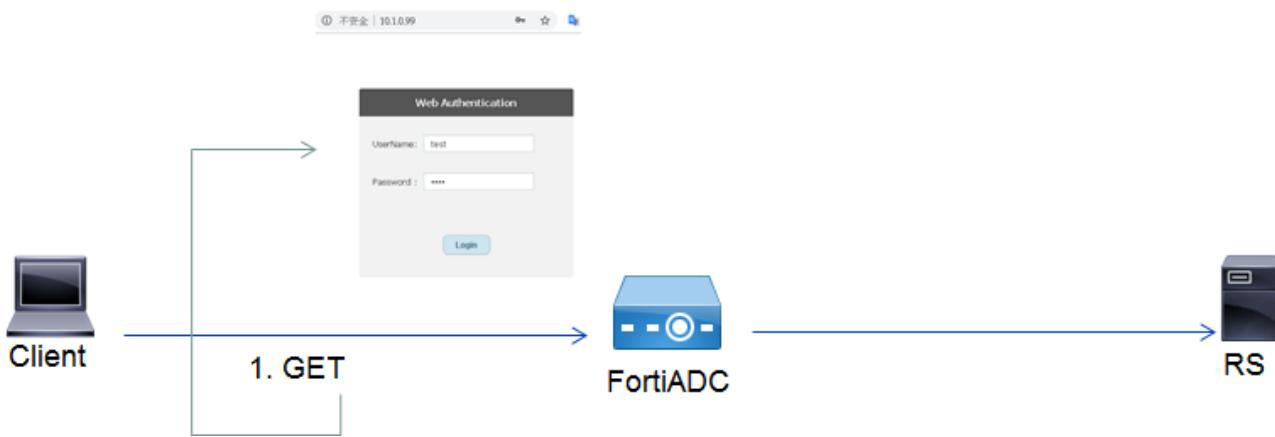
Allow you to retrieve the baked cookie.

#### Syntax

```
AUTH:get_baked_cookie();
```

**Arguments: N/A**

#### Examples



```
when COOKIE_BAKE {
cookie = AUTH:get_baked_cookie()
debug("Get cookie: %s\r\n", cookie)
}
Result:
```

```
Get cookie: Set-Cookie:
FortiADCauthSI=lfGnC2gs17xsbAg4JFs94e4CJfFXaP3U5z6QHvo7n08cCoT5MdtQog2LmcizPo3aRiBHY/R
ThhocqG+DdnvsCLFJh3nBUoLeuYjGK91Y5L4=|W86hXGg; expires=Tue 23 Oct 2018 04:19:45 GMT;
domain=10.1.0.99; path=/
```

FortiADC version: V5.2

Used in events:

Used in AUTH\_RESULT

## AUTH:set\_baked\_cookie(cookie)

Allow you to customize cookie attribute the baked cookie.

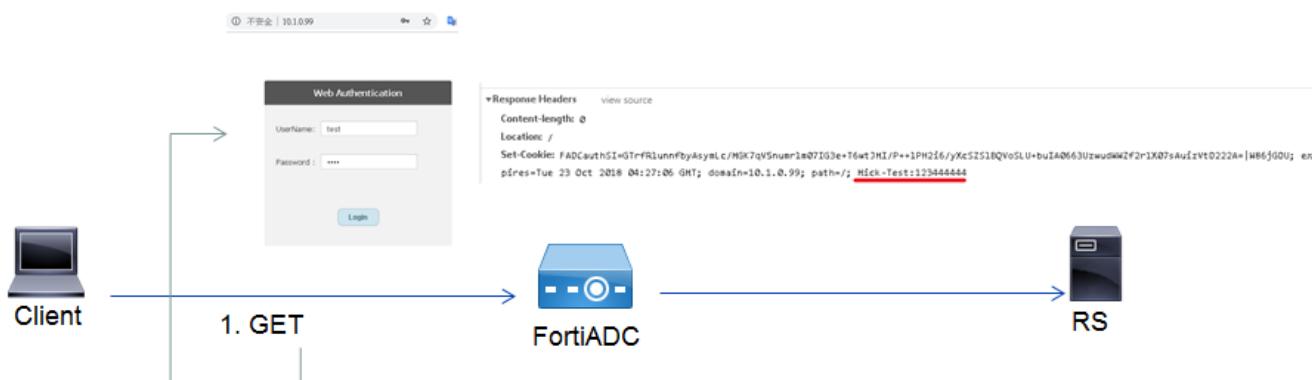
### Syntax

```
AUTH:set_baked_cookie(cookie);
```

### Arguments

Name	Description
cookie	A string which specifies the baked cookie.

### Examples



```
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:

Used in AUTH\_RESULT

## AUTH:on\_off()

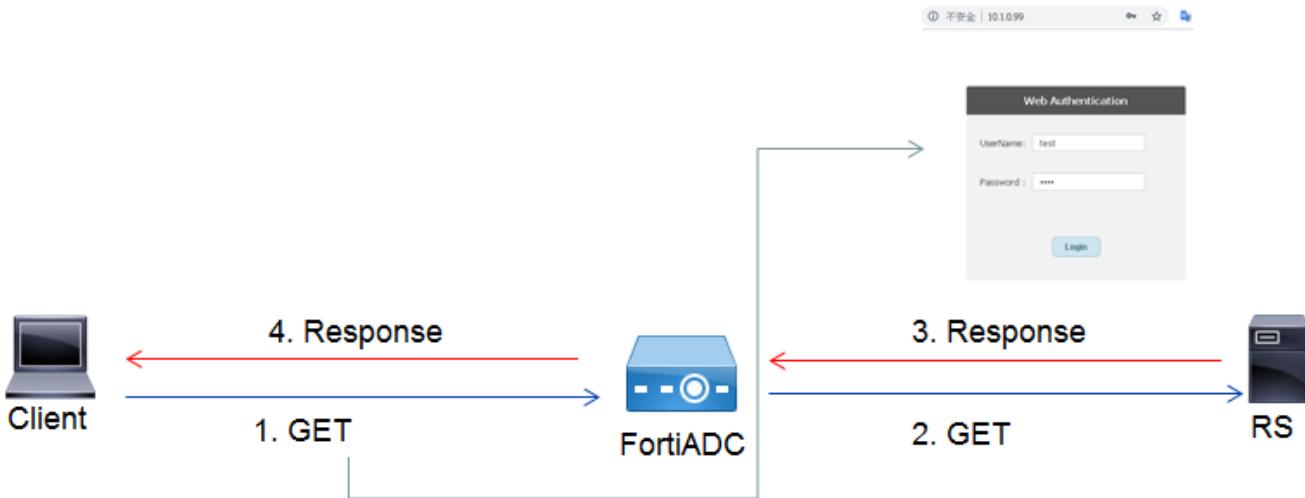
Returns the authentication whether required or not.

## Syntax

```
AUTH:on_off();
```

**Arguments:** N/A

## Examples



```
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:
authtication 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:

Used in AUTH\_RESULT / HTTP\_REQUEST / HTTP\_DATA\_REQUEST / HTTP\_RESPONSE / HTTP\_DATA\_RESPONSE

## AUTH:success()

Returns the authentication is successful or not

## Syntax

```
AUTH:success();
```

**Arguments: N/A**

**Examples**

Please refer to command AUTH:on\_off() example.

FortiADC version: V5.2

Used in events:

Used in AUTH\_RESULT / HTTP\_REQUEST / HTTP\_DATA\_REQUEST / HTTP\_RESPONSE / HTTP\_DATA\_RESPONSE

## **AUTH:form\_based()**

Returns the authentication is HTTP form based or not

**Syntax**

AUTH:form\_based();

**Arguments: N/A**

**Examples**

Please refer to command AUTH:on\_off() example.

FortiADC version: V5.2

Used in events:

Used in AUTH\_RESULT / HTTP\_REQUEST / HTTP\_DATA\_REQUEST / HTTP\_RESPONSE / HTTP\_DATA\_RESPONSE

AUTH:user()

Returns the user name in authentication

**Syntax**

AUTH:user();

**Arguments: N/A**

**Examples**

Please refer to command AUTH:on\_off() example.

FortiADC version: V5.2

Used in events:

Used in AUTH\_RESULT / HTTP\_REQUEST / HTTP\_DATA\_REQUEST / HTTP\_RESPONSE / HTTP\_DATA\_RESPONSE

AUTH:pass()

Returns the password in authentication

**Syntax**

AUTH:pass();

**Arguments: N/A**

### Examples

Please refer to command AUTH:on\_off() example.

FortiADC version: V5.2

Used in events:

Used in AUTH\_RESULT / HTTP\_REQUEST / HTTP\_DATA\_REQUEST / HTTP\_RESPONSE / HTTP\_DATA\_RESPONSE

## AUTH:usergroup()

Returns the usergroup which the user belong to

### Syntax

AUTH:usergroup();

**Arguments: N/A**

### Examples

Please refer to command AUTH:on\_off() example.

FortiADC version: V5.2

Used in events:

Used in AUTH\_RESULT / HTTP\_REQUEST / HTTP\_DATA\_REQUEST / HTTP\_RESPONSE / HTTP\_DATA\_RESPONSE

## AUTH:realm()

Returns the realm in authentication

### Syntax

AUTH:realm();

**Arguments: N/A.**

### Examples

Please refer to command AUTH:on\_off() example.

FortiADC version: V5.2

Used in events:

Used in 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.**

**Examples**

Please refer to command AUTH:on\_off() example.

FortiADC version: V5.2

Used in events:

Used in AUTH\_RESULT / HTTP\_REQUEST / HTTP\_DATA\_REQUEST / HTTP\_RESPONSE / HTTP\_DATA\_RESPONSE

## SSL commands

### SSL:cipher()

Returns the cipher in handshake..

**Syntax**

```
SSL:cipher();
```

**Arguments: N/A**

**Examples**

```
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:

Used in CLIENTSSL\_HANDSHAKE / SERVERSSL\_HANDSHAKE

### SSL:version()

Returns the SSL version in handshake.

**Syntax**

```
SSL:version();
```

**Arguments: N/A**

**Examples**

```
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:

Used in CLIENTSSL_HANDSHAKE / SERVERSSL_HANDSHAKE
```

## SSL:alg\_keysize()

Returns the SSL encryption keysize in handshake..

### Syntax

```
SSL:alg_keysize();
```

**Arguments:** N/A

### Examples

```
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:

```
Used in 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

### Examples

```
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:

Used in CLIENTSSL\_HANDSHAKE / SERVERSSL\_HANDSHAKE / CLIENTSSL\_RENEGOTIATE / SERVERSSL\_RENEGOTIATE

## SSL:sni()

Returns the SNI or false(if no).

### Syntax

```
SSL:sni();
```

### Arguments: N/A

#### Examples

```
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 send https request without sni  
[root@NxLinux certs]# openssl s_client -connect 5.1.1.100:443  
Debug output:  
Client handshake  
SNI: false  
2. client send 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:

Used in CLIENTSSL\_HANDSHAKE / SERVERSSL\_HANDSHAKE / CLIENTSSL\_RENEGOTIATE / SERVERSSL\_RENEGOTIATE

## SSL:npn()

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

#### Syntax

SSL:npn();

### Arguments: N/A

#### Examples

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

FortiADC version: V5.0

Used in events:

Used in CLIENTSSL\_HANDSHAKE / SERVERSSL\_HANDSHAKE / CLIENTSSL\_RENEGOTIATE / SERVERSSL\_RENEGOTIATE

## SSL:alpn()

Allow you to get the SSL ALPN extension.

### Syntax

```
SSL:alpn();
```

### Arguments: N/A

### Examples

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

FortiADC version: V5.0

Used in events:

Used in CLIENTSSL\_HANDSHAKE / SERVERSSL\_HANDSHAKE / CLIENTSSL\_RENEGOTIATE / SERVERSSL\_RENEGOTIATE

## SSL:session(t)

Allows you to get SSL session id / reused / remove from cache.

### Syntax

```
SSL:session(t);
```

### Arguments

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

### Examples

```
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:

Used in CLIENTSSL\_HANDSHAKE / SERVERSSL\_HANDSHAKE / CLIENTSSL\_RENEGOTIATE / SERVERSSL\_RENEGOTIATE

## SSL:cert(t)

Allow you to get the cert info between local or remote

### Syntax

```
SSL:cert(t);
```

### Arguments

Name	Description
t	A table which specifies the cert direction, and operation.

### Examples

```
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 return a table, it contains all info in the cert.

in the return, contain: 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:

Used in CLIENTSSL\_HANDSHAKE / SERVERSSL\_HANDSHAKE / CLIENTSSL\_RENEGOTIATE / SERVERSSL\_RENEGOTIATE

## SSL:cert\_der()

Retun the der certificate when client enable verify certificate.

**Syntax**

```
SSL:cert_der();
```

**Arguments: N/A****Examples**

```
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:

Used in 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 cert format.

**Examples**

```
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:

Used in CLIENTSSL\_HANDSHAKE / SERVERSSL\_HANDSHAKE / CLIENTSSL\_RENEGOTIATE / SERVERSSL\_RENEGOTIATE

## GEO IP commands

### 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.

#### Examples

```
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.

#### Examples

```
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

**Note:**

- FortiADC can be configured to do RAM caching.
- Certain URI can be excluded from caching.
- Further, FortiADC supports dynamic caching in which certain URIs are cached for a specified period of time and accessing some other URIs will invalidate their caching.
- Note that dynamic caching is identified by a configured ID.
- FortiADC script allows the user to control these behaviors and check the caching status.
- Allows you to disable caching (both cache hit and cache store).
- Allows you to disable exclude URI check.
- Allows you to disable dynamic cache check.
- Allows you to disable dynamic cache invalid check.
- Allows you to directly enable dynamic caching with a given ID and age.
- Allows you to invalidate a given dynamic cache indexed by its ID
- Allows you to check whether a response is from cache or not, if yes, regular cache or dynamic cache
- Allows you to check whether a response is been caching or not, if yes, regular cache or dynamic cache
- Allows you to replace the default key (the URI) with any customized key
- Allows you to drop an ongoing caching operation
- Allows you to check its cache status for a given URI key: its cache ID and hit counts .

Please make sure RAM caching configuration is selected in the HTTP or HTTPs profile.

### HTTP:exclude\_check\_disable()

Used to disable the exclude URI check.

**Syntax**

HTTP:exclude\_check\_disable();

**Arguments: N/A****Examples**

```
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:

Only used in HTTP\_REQUEST

### HTTP: cache\_disable()

Used to disable caching(both cache hit and cache store).

### Syntax

HTTP: cache\_disable();

### Arguments: N/A

### Examples

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

FortiADC version: V5.3

Used in events:

Only used in HTTP\_REQUEST

## [HTTP: dyn\\_check\\_disable\(\)](#)

Used to disable dynamic caching check.

### Syntax

HTTP: dyn\_check\_disable();

### Arguments: N/A

### Examples

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

FortiADC version: V5.3

Used in events:

Only used in HTTP\_REQUEST

## [HTTP: dyn\\_invalid\\_check\\_disable\(\)](#)

Used to disable dynamic invalid check caching.

### Syntax

HTTP: dyn\_invalid\_check\_disable();

### Arguments: N/A

### Examples

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

FortiADC version: V5.3

Used in events:

Only used in HTTP\_REQUEST

## HTTP: dyn\_cache\_enable(t)

Used to enable dynamic caching directly.

### 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:

Only used in HTTP\_REQUEST

## HTTP: cache\_user\_key(t)

Used to set customized key to replace the default user key.

### Syntax

```
HTTP:cache_user_key(t);
```

### Arguments

Name	Description
t	A table which specifies the caching uri.

### Examples

```
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:

Only used in HTTP\_REQUEST

## HTTP: dyn\_cache\_invalid(t)

Used to invalid a certain dynamic cache given by id.

### Syntax

```
HTTP: dyn_cache_invalid(t);
```

### Arguments

Name	Description
t	A table which specifies the cache id.

### Examples

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

FortiADC version: V5.3

Used in events:

Used in HTTP\_REQUEST / HTTP\_RESPONSE

## HTTP: cache\_check(t)

Used to check whether cached or not for a certain URI.

### Syntax

```
HTTP: cache_check(t);
```

### Arguments

Name	Description
t	A table which specifies the cache uri.

### Examples

```
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:

Used in HTTP\_REQUEST / HTTP\_RESPONSE

## HTTP: cache\_hits(t)

Used to check the cache hit count for a certain URI.

### Syntax

```
HTTP:cache_hits(t);
```

### Arguments

Name	Description
t	A table which specifies the cache uri.

### Examples

```
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:

Used in HTTP\_REQUEST / HTTP\_RESPONSE

## HTTP: res\_caching()

Used to check whether the response is been caching.

### Syntax

```
HTTP:res_caching();
```

**Arguments: N/A**

### Examples

```
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:

Only used in HTTP\_RESPONSE

## HTTP:res\_cached()

Used to check whether the response is from cache or not.

### Syntax

HTTP:res\_cached();

### Arguments: N/A

### Examples

```
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:

Only used in HTTP\_RESPONSE

## HTTP:caching\_drop()

Used to drop the ongoing caching.

### Syntax

HTTP:caching\_drop();

### Arguments: N/A

### Examples

```
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:

Only used in HTTP\_RESPONSE

## PROXY commands

### 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 authentication cookie.

#### Examples

```
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:

Used in VS\_LISTENER\_BIND / TCP\_BIND

### PROXY:clear\_auth\_key(value)

clear whatever customized authentication key you might have set before, then the default key is used.

#### Syntax

```
PROXY:clear_auth_key(value);
```

#### Arguments

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

#### Examples

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

FortiADC version: V5.2

Used in events:

Used in VS\_LISTENER\_BIND / TCP\_BIND

## LB commands

### LB:routing(value)

Route the request to the content routing server.

#### Syntax

```
LB:routing(value);
```

#### Arguments

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

#### Examples

```
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 enable both content-routing and scripting, then will perform cross check, check the content-routing used in scripting is used to the vs.

FortiADC version: V4.3

Used in events:

Used in HTTP\_REQUEST / HTTP\_DATA\_REQUEST

## Global commands

### 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	A string which will be calculated.

#### Examples

```
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)

Create 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.

### Examples

```
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)

encrypt the data using the previously-created AES key

### Syntax

```
Aes_enc(t);
```

### Arguments

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

### Examples

```
when HTTP_REQUEST {
```

```
t={};
t["message"] = "MICK-TEST";
t["key"] = "aaaaaaaaaaaaabbbbbbb" --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)**

decrypt the data using the previously-created AES key

**Syntax**

Aes\_dec(t);

**Arguments**

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

**Examples**

```
when HTTP_REQUEST {
t={};
t["message"] = "MICK-TEST";
t["key"] = "aaaaaaaaaaaaabbbbbbb"
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"] = "aaaaaaaaaaaaabbbbbbb"
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 oneshot 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.

### Examples

```
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.

### Examples

```
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)

Check if the signature is 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.

**Examples**

```
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.

**Examples**

```
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)

Used to match 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.

**Examples**

```
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 parameter, 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)

Used to search 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.

### Examples

```
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()

Used to match 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 For an IPv6 ip_addr/[mask], the mask can be a number between 0 and 128.

Name	Description
Addr_group	A group of IP address. addr_group = "192.168.1.0/24" --first network address addr_group = addr_group..",::ffff:172.30.1.0/120" --second network address

**Examples**

```
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)**

converted the url info a valid ASCII format

**Syntax**

url\_enc(str);

**Arguments**

Name	Description
str	A string which will be converted.

**Examples**

```
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)

converted the encoding-url into a original url.

### Syntax

```
url_dec(str);
```

### Arguments

Name	Description
str	A string which will be converted.

### Examples

```
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)

Parse a url, return a table contains 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.

### Examples

```
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)

Compare two url string, return true if they are the same.

### Syntax

```
url_compare(url1, url2);
```

### Arguments

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

### Examples

```
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()

Generate a random number. Returns is a integer number. After FAD reboot, the random number will be different.

### Syntax

```
rand();
```

Arguments: N/A

### Examples

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

FortiADC version: V5.2

Used in events: ALL

## srand(str)

Set the random seed.

### Syntax

```
srand(str);
```

### Arguments

Name	Description
str	A string which specifies the seed.

### Examples

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

FortiADC version: V5.2

Used in events: ALL

## Rand\_hex(int)

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

### Syntax

```
Rand_hex(int);
```

### Arguments

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

### Examples

```
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)

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

### Syntax

```
Random_alphanum(int);
```

### Arguments

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

**Examples**

```
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)**

Generate a random in sequence. Returns is a string, length is the <int>.

**Syntax**

Rand\_seq(int);

**Arguments**

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

**Examples**

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

FortiADC version: V5.2

Used in events: ALL

**Time()**

Returns the current time in a number which is the time since the Epoch measured in seconds.

**Syntax**

time();

**Arguments: N/A****Examples**

```
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 a string describing the current time like “Tue Jun 25 14:11:01 2019”.

### Syntax

```
ctime();
```

### Arguments: N/A

### Examples

```
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 like “Thu 27 Jun 2019 18:27:42 GMT”.

### Syntax

```
gmtime();
```

### Arguments: N/A

### Examples

```
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)

Return the MD5 calculated for the string provided.

### Syntax

```
Md5(str);
```

**Arguments**

Name	Description
str	A string which will be calculated.

**Examples**

```
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 to the string

**Syntax**

Md5\_hex(str);

**Arguments**

Name	Description
str	A string which will be calculated.

**Examples**

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

FortiADC version: V4.8

Used in events: ALL

**Md5\_str(str)**

Calculate 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	A string which will be calculated.

**Examples**

```
when HTTP_REQUEST {
    md5 = md5_hex_str(input); --input can be a cert in DER format
}
```

FortiADC version: V4.8

Used in events: ALL

**Md5\_hex\_str(str)**

Calculate 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.

**Examples**

```
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 sha1 calculated for the string provided.

**Syntax**

```
Sha1(str);
```

**Arguments**

Name	Description
str	A string which will be calculated.

**Examples**

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

FortiADC version: V4.8

Used in events: ALL

## Sha1\_hex(str)

Returns the sha1 calculated for the string in hex.

### Syntax

```
Sha1_hex(str);
```

### Arguments

Name	Description
str	A string which will be calculated.

### Examples

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

FortiADC version: V4.8

Used in events: ALL

## Sha1\_str(str)

Calculate the SHA1 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	A string which will be calculated.

### Examples

```
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)

Calculate the SHA1 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	A string which will be calculated.

### Examples

```
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 SHA256 of a string input and store the result in an intermediate variable.

### Syntax

```
Sha256(str);
```

### Arguments

Name	Description
str	A string which will be calculated.

### Examples

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

FortiADC version: V4.8

Used in events: ALL

## Sha256\_hex(str)

Calculate the SHA256 of a string input and output 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	A string which will be calculated.

**Examples**

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

FortiADC version: V4.8

Used in events: ALL

**Sha256\_str(str)**

Calculate the SHA256 of a string input and restore 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	A string which will be calculated.

**Examples**

```
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)**

Calculate the SHA256 of a string input and restore 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	A string which will be calculated.

**Examples**

```
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 SHA384 of a string input and store the result in an intermediate variable.

### Syntax

Sha384(str);

### Arguments

Name	Description
str	A string which will be calculated.

### Examples

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

FortiADC version: V4.8

Used in events: ALL

## Sha384\_hex(str)

Calculate the SHA384 of a string input and output 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	A string which will be calculated.

### Examples

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

FortiADC version: V4.8

Used in events: ALL

## Sha384\_str(str)

Calculate the SHA384 of a string input and restore 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	A string which will be calculated.

### Examples

```
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)

Calculate the SHA384 of a string input and restore 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	A string which will be calculated.

### Examples

```
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 SHA512 of a string input and store the result in an intermediate variable.

### Syntax

```
Sha512(str);
```

**Arguments**

Name	Description
str	A string which will be calculated.

**Examples**

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

FortiADC version: V4.8

Used in events: ALL

**Sha512\_hex(str)**

Calculate the SHA512 of a string input and output 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	A string which will be calculated.

**Examples**

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

FortiADC version: V4.8

Used in events: ALL

**Sha512\_str(str)**

Calculate the SHA512 of a string input and restore 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	A string which will be calculated.

## Examples

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

FortiADC version: V4.8

Used in events: ALL

## Sha512\_hex\_str(str)

Calculate the SHA512 of a string input and restore 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	A string which will be calculated.

## Examples

```
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)

encode a string input in base32 and output the result in string format.

### Syntax

```
B32_enc(str);
```

### Arguments

Name	Description
str	A string which will be calculated.

## Examples

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

FortiADC version: V5.2

Used in events: ALL

## B32\_enc\_str(str)

Encode a string input in base32 and output 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	A string which will be calculated.

### Examples

```
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)

Decode a base32 encoded string input and output the result in string format

### Syntax

B32\_dec(str);

### Arguments

Name	Description
str	A string which will be calculated.

### Examples

```
when HTTP_REQUEST {  
    str = "abc"  
    dec = b32_dec(str)  
}
```

FortiADC version: V5.2

Used in events: ALL

## B32\_dec\_str(str)

Decode a base32 encoded string input and output 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	A string which will be calculated.

**Examples**

```
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)**

Returns base64 encryption of string

**Syntax**

```
B64_enc(str);
```

**Arguments**

Name	Description
str	A string which will be calculated.

**Examples**

```
when HTTP_REQUEST {
result = b64_enc(input);
--Input can be general format:
str="test string"
a=12
en=b64_enc("%s, %d", str, a);
}
```

FortiADC version: V4.8

Used in events: ALL

**B64\_dec(str)**

Returns base64 decryption of string

**Syntax**

```
B64_dec(str);
```

**Arguments**

Name	Description
str	A string which will be calculated.

**Examples**

```
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

### Examples

```
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	A table which specifies the info.

### Examples

```
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)

Convert 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.

### Examples

```
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)

Convert 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.

### Examples

```
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)

Convert 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.

### Examples

```
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 receive some long integers in HTTP response from the network, convert 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.

### Examples

```
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.

### **Examples**

```
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 debug info when vs using scripting.

### **Syntax**

debug(str);

### **Arguments**

Name	Description
str	A string which will be printed.

### **Examples**

```
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 info 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.

**Examples**

```
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.

**Examples**

```
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()

**Examples**

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.

**Examples**

FortiADC version: V5.2

Used in events: ALL

**WAF commands****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. The deny code should be an integer from the following: 200, 202, 204, 205, 400, 403, 404, 405, 406, 408, 410, 500, 501, 502, 503, 504. <b>Note:</b> 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). <b>Note:</b> 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. 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. The string should be from the following list: <ul style="list-style-type: none"> <li>waf_web_attack_signature</li> </ul>

Name	Description
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:abandonViolation()**

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:abandonViolation();

**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:abandonViolation(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:raiseViolation(v);
}

```

**WAF:raiseViolation(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:overrideAction(string).

**Syntax**

```
WAF:raiseViolation(string);
```

**Arguments**

Name	Description
severity	Overrides the severity string that includes the values "low", "medium", and "high". <b>Note:</b> 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. <b>Note:</b> 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. <b>Note:</b> This argument must be specified.
action	The violation will show the defined action, including the values "pass", "deny", "block", "redirect", or "captcha". <b>Note:</b> 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. 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.

Name	Description
	<p>The deny code should be an integer from the following: 200, 202, 204, 205, 400, 403, 404, 405, 406, 408, 410, 500, 501, 502, 503, 504. If the deny code is not specified or it is an invalid integer, then it will be defaulted to 403.</p> <p>The return value is a bool value; when the operation is successful, it will return true, otherwise, it will return false.</p>
sub-category	<p>This string specifies the violation's sub-category. The string should be from the following list:</p> <ul style="list-style-type: none"> <li>• waf_web_attack_signature</li> <li>• waf_http_protocol_const</li> <li>• waf_heur_sqlxss_inject_detect</li> <li>• waf_url_protect,waf_bot_detection</li> <li>• waf_xml_check</li> <li>• waf_json_check</li> <li>• waf_web_scraping</li> <li>• waf_cookie_security</li> <li>• waf_csrf_protection</li> <li>• waf_html_input_validation</li> <li>• waf_brute_force,waf_data_leak_prevention</li> <li>• waf_credential_stuffing</li> <li>• waf_openapi_check</li> <li>• waf_api_gateway</li> </ul> <p><b>Note:</b> 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".</p>
owasp-top10	<p>The string will show the violation that is related to the OWASP TOP10 field name. <b>Note:</b> If this argument is not specified, then it will default to "unknown".</p>

## 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:abandonViolation(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:raiseViolation(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);
```

**Argument:** 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. 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-Forwarded-For header values and pool member names.
GENERAL_REDIRECT_DEMO	Redirects requests to a URL with the user-defined code and cookie. 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. 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.
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 "Location:http://www.example.com" to "Location:https://www.example.com"

Predefined script	Description
	<p>Note: You can use the script directly, without making any change</p>
REWRITE_HTTP_2_HTTPS_in_REFERER	<p>Rewrites HTTP referer to HTTPS, for example, rewrite “Referer: http://www.example.com” to “Referer: https://www.example.com”.</p> <p>Note: You can use the script directly, without making any change.</p>
REWRITE_HTTPS_2_HTTP_in_LOCATION	<p>Rewrites HTTPS location to HTTP, for example, rewrite “Location:https://www.example.com” to “Location:http://www.example.com”.</p> <p>Note: You can use the script directly, without making any change.</p>
REWRITE_HTTPS_2_HTTP_in_REFERER	<p>Rewrites HTTPS referer to HTTP, for example, rewrite “Referer: https://www.example.com” to “Referer: http://www.example.com”.</p> <p>Note: You can use the script directly, without making any change</p>
HTTP_DATA_FETCH_SET_DEMO	<p>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”.</p> <p>Note: Do NOT use this script "as is". Instead, copy it and manipulate the collected data.</p>
HTTP_DATA_FIND_REMOVE_REPLACE_DEMO	<p>Finds a specified string, removes a specified string, or replaces a specified string to new content in HTTP data.</p> <p>Note: Do NOT use this script "as is". Instead, copy it and manipulate the collected data.</p>
URL.Utility_Commands	Demonstrate how to use those url tools to encode/decode/parser/compare .
USE_REQUEST_HEADERS_in_OTHER_EVENTS	<p>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.</p> <p>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.</p>
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.

Predefined script	Description
	<p>Note: Before using this script, you must have the following four parameters configured in the client-ssl-profile:</p> <ul style="list-style-type: none"> <li>  client-certificate-verify—Set to the verify you'd like to use to verify the client certificate.</li> <li>  client-certificate-verify-option—Set to optional</li> <li>  ssl-session-cache-flag—Disable.</li> <li>  use-tls-tickets—Disable.</li> <li> </li> </ul>
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	<p>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.</p> <p>Note: Do NOT use this script "as is". Instead, copy it and customize the IP address and the IP address group.</p>
INSERT_RANDOM_MESSAGE_ID_DEMO	<p>Inserts a 32-bit hex string into the HTTP header with a parameter "Message-ID".</p> <p>Note: You can use the script directly, without making any change.</p>
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.
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	<p>Uses demo_1 and demo_2 script to show how multiple scripts work. Demo_1 with priority 12 has a higher priority.</p> <p>Note: You could enable or disable other events. Do NOT use this script "as is". Instead, copy it and customize the operation.</p>

Predefined script	Description
MULTIPLE_SCRIPT_CONTROL_DEMO_2	<p>Uses demo_1 and demo_2 script to show how multiple scripts work. Demo_2 with priority 24 has a lower priority.</p> <p>Note: You could enable or disable other events. Do NOT use this script "as is". Instead, copy it and customize the operation</p>



**FORTINET**



Copyright© 2021 Fortinet, Inc. All rights reserved. Fortinet®, FortiGate®, FortiCare® and FortiGuard®, and certain other marks are registered trademarks of Fortinet, Inc., in the U.S. and other jurisdictions, 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. In no event does Fortinet make any commitment related to future deliverables, features or development, and circumstances may change such that any forward-looking statements herein are not accurate. 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.