Script Reference Guide

FortiADC 8.0.0

July 25, 2025
FortiADC 8.0.0 Script Reference Guide
00-540-000000-20200204

TABLE OF CONTENTS

Change Log

Date	Change Description
8/5/2025	Initial release.
9/9/2025	Formatting Corrections.

Introduction

FortiWeb supports Lua scripts to perform actions that are not currently supported by the built-in feature set. You can use Lua scripts to write simple, network aware pieces of code that will influence network traffic in a variety of ways. By using the scripts, you can customize FortiWeb's features by granularly controlling the traffic flow or even the contents of given sessions or packets.

In FortiWeb, the scripting language only supports HTTP and HTTPS policy.

Configuration overview

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 Policy** settings.

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

To configure a script:

1. Go to **Application Delivery > Scripting**.
2. Click **Create New** to display the configuration editor.
3. Complete the configuration as shown.

Settings	Guidelines
Name	Enter a unique name. No spaces or special characters. After you initially save the configuration, you cannot edit the name.
Input	Type or paste the script.

4. Click **OK** to Save the configuration.
5. You can also click **Import** to import a script file. It should be a ".txt" file.
6. When creating a server policy, in the **Scripting** section, enable **Scripting**, then select the scripts you want to run for this server policy.

Script Events

There are predefined scripts which specify the following events. When the events occur, it will trigger the system to take the actions defined in the script.

Type	Event	Description
RULE	RULE_INIT	When the server policy enables or reloads.
	RULE_EXIT	When the server policy disables or reloads.

Type	Event	Description
HTTP	HTTP_REQUEST	When the server policy has received the complete HTTP request header.
	HTTP_RESPONSE	When the server policy has received the complete HTTP response header.
	HTTP_DATA_REQUEST	When an `HTTP:collect` command finishes processing, after collecting the requested amount of data.
	HTTP_DATA_RESPONSE	When an `HTTP:collect` command finishes processing on the server side of a connection.
TCP	CLIENT_ACCEPTED	When the server policy has accepted a client connection.
	CLIENT_CLOSED	When the server policy has closed a client connection.
	SERVER_CONNECTED	When the server policy has connected to a server.
	SERVER_CLOSED	When the server policy has closed a server connection.
SSL	CLIENTSSL_HANDSHAKE	When a client-side SSL handshake is completed.
	SERVERSSL_HANDSHAKE	When a server-side SSL handshake is completed.
	SERVERSSL_CLIENTHELLO_SEND	When the system is about to send its SSL ClientHello message.
	CLIENTSSL_SERVERHELLO_SEND	When the system is about to send its SSL ServerHello message on the clientside connection.
	CLIENTSSL_RENEGOTIATE	When a client-side (Client to FortiWeb) SSL renegotiation is completed

Event priority

FortiWeb supports multiple scripts in one server policy. When a server policy with scripts is enabled, the system will load scripts one by one. If there are multiple same events defined in the scripts, the event running order is same as the loading order.

If you want to run a certain event first regardless of the script order, you can define its priority to prioritize its sequence. The default priority of events is 500. Lower value has higher priority.

For example:

```
when HTTP_REQUEST priority 499 {
...
}
```

Lua package compatibility

FortiWeb uses the lua version 5.4.

Package name	Compatible details
global	Supported, but: • Disable dofile() • Disable loadfile() • Modify print() to FortiWeb version, printing to debug log with level 1. (`diag debug proxyd scripting-user <1-7>`)
package	Disabled
coroutine	Disabled
table	Supported
io	Disabled
os	Disabled
string	Supported
math	Supported
utf8	Supported

Predefined packages and classes

- Global
- Core
- Policy
- IP

Global

debug(fmt, ..)

The function is the same as `print(string.format(fmt, ..))`.

The string will be printed to debug log with level 1.

Example

```
when HTTP_REQUEST {
    debug("This HTTP Request method is %s.\n", HTTP:method())
}
```

_id

This is the id of the proxyd worker running the lua stack.

Example

```
when HTTP_REQUEST {
    debug("id of the proxyd worker running the lua stack is %s.\n", _id)
}
```

_name

This is the name of the policy running the lua stack.

Example

```
when HTTP_REQUEST {
    debug("name of the proxyd worker running the lua stack is %s.\n", _name)
}
```

Core

core.debug(level, fmt, ..)

Similar to `debug()` but allows you to specify the debug log level.

Example:

```
when HTTP_REQUEST
       local host = HTTP:host()
       core.debug(6, "host = %s", host)
}
```

core.print(level, …)

Similar to `print()` but allows you to specify the debug log level.

Example:

```
when HTTP_REQUEST {
       local host = HTTP:host()
       core.print(6, "host = ", host)
}
```

Policy

This package is used for fetching the policy configurations.

policy.name()

Return the string of the policy name.

Example

```
when HTTP_REQUEST {
    debug("policy name is %s.\n", policy.name()
}
```

policy.http_ports()

Return a lua array with all HTTP ports. Port value is integer.

`{ 80, 8080 }`

Example

```
when HTTP_REQUEST {
    for k,v in pairs(policy.http_ports()) do
       debug("http port %s port is %s.\n", k, v)
    end
}
```

policy.https_ports()

Return a lua array with all HTTPS port. Port value is integer.

`{ 443, 8443 }`

Example

```
when HTTP_REQUEST {
    for k,v in pairs(policy.https_ports()) do
       debug("https port %s port is %s.\n", k, v)
    end
}
```

policy.crs()

Return lua array with all content routing names.

`{ “cr1”, “cr2”, “cr3” }`

Example

```
when HTTP_REQUEST {
    for k,v in pairs(policy.crs()) do
       debug("content routing name %s is %s.\n", k, v)
    end
}
```

policy.servers() / policy.servers(“cr-name”)

Return lua array with all servers. If the policy has content routing, the caller should pass the “cr-name” argument to fetch the servers of the specific content routing.

Example

```
when HTTP_REQUEST {
    for k,v in pairs(policy.servers()) do
       debug("server %s details are %s.\n", k, v)
    end
}
```

IP

This package contains IP related functions.

ip.addr(“ip-string”)

Generate an IP address class with an IP string.

ip.eq(ip_class_1, “ip-string”) / ip.eq(ip_class_1, ip_class_2)

Compare two IP addresses. The first one must be IP address class and the second one can be IP address class or IP string.

ip.reputation(“ip-string”) / ip.reputation(ip_class)

Check the reputation of a specific IP. Return Lua array with reputation categories. The reputation categories are:

`"Botnet", "Anonymous Proxy", "Phishing", "Spam", "Others", "Tor"`

If IP string is not a valid IP, return nil.

Return value example:

```
{ "Anonymous Proxy", "Phishing" }
```

ip.geo(“ip-string”) / ip.geo(ip_class)

Return GEO country name in string. If nothing is found or the IP string is not a valid IP, return nil.

ip.geo_code(“ip-string”) / ip.geo_code(ip_class)

Return GEO country code in string. If nothing is found or the IP string is not a valid IP, return nil.

IP address classes

__eq()

Support use “==” to compare two IP address classes.

__tostring()

Support use tostring(IP-class) to convert IP address class to IP string.

Example

```
when HTTP_REQUEST {
    local ip = tostring(IP:local_addr())
}
```

IP:local_addr()

Return IP address class, which is the local address of the connection.

Example

```
when HTTP_REQUEST {
    local ip = tostring(IP:local_addr())
    if ip == "10.10.10.10" then
```

```
        debug("local addr equals to 10.10.10.10")
    end
}
```

IP:remote_addr()

Return IP address class, which is the remote address of the connection.

Example

```
when HTTP_REQUEST {
    local ip = tostring(IP:remote_addr())
    if ip == "10.10.10.10" then
        debug("remote addr equals to 10.10.10.10")
    end
}
```

IP:client_addr()

Return IP address class, which is the client IP address of the stream.

Example

```
when HTTP_REQUEST {
    local ip = tostring(IP:client_addr())
    if ip == "10.10.10.10" then
        debug("client addr equals to 10.10.10.10")
    end
}
```

IP:server_addr()

Return IP address class, which is the server IP address of the stream. If server is not connected, return nil.

Example

```
when HTTP_REQUEST {
    local ip = tostring(IP:server_addr())
    if ip == "10.10.10.10" then
        debug("server addr equals to 10.10.10.10")
    end
}
```

IP:version()

Return the IP version of the connection, either 4 or 6.

Example

```
when HTTP_REQUEST {
    local version = IP:version()
    if version == 4 then
        debug("ip version is 4")
    end
}
```

Predefined commands

All commands are Lua classes but they only can be used inside scripting events. Some commands can only be used in specific events. For example, HTTP commands can only be used inside HTTP events (HTTP_REQUEST and HTTP_RESPONSE).

IP commands

IP commands can be used in HTTP and TCP events.

IP:local_addr()

Return IP address class, which is the local address of the connection.

Example

```
when HTTP_REQUEST {
    local ip = tostring(IP:local_addr())
    if ip == "10.10.10.10" then
        debug("local addr equals to 10.10.10.10")
    end
}
```

IP:remote_addr()

Return IP address class, which is the remote address of the connection.

Example

```
when HTTP_REQUEST {
    local ip = tostring(IP:remote_addr())
    if ip == "10.10.10.10" then
        debug("remote addr equals to 10.10.10.10")
    end
}
```

IP:client_addr()

Return IP address class, which is the client IP address of the stream.

Example

```
when HTTP_REQUEST {
    local ip = tostring(IP:client_addr())
    if ip == "10.10.10.10" then
        debug("client addr equals to 10.10.10.10")
    end
}
```

IP:server_addr()

Return IP address class, which is the server IP address of the stream. If server is not connected, return nil.

Example

```
when HTTP_REQUEST {
    local ip = tostring(IP:server_addr())
    if ip == "10.10.10.10" then
        debug("server addr equals to 10.10.10.10")
    end
}
```

IP:version()

Return the IP version of the connection.

Example

```
when HTTP_REQUEST {
    local version = IP:version()
    debug("ip version is %s", version)
}
```

TCP commands

TCP commands can be used in HTTP and TCP events.

TCP:local_port()

Return local TCP port of the connection. The value is integer.

Example

```
when HTTP_REQUEST {
    print_ips("HTTP_REQUEST", TCP, IP)
}
function print_ips(event, TCP, IP)
    debug("%s: version: %s, local: %s:%s, remote: %s:%s, client: %s:%s, server: %s:%s\n",
        event, IP:version(),
        IP:local_addr(), TCP:local_port(),
        IP:remote_addr(), TCP:remote_port(),
        IP:client_addr(), TCP:client_port(),
        IP:server_addr(), TCP:server_port())
end
```

TCP:remote_port()

Return remote TCP port of the connection. The value is integer.

Example

```
when HTTP_REQUEST {
    print_ips("HTTP_REQUEST", TCP, IP)
}
function print_ips(event, TCP, IP)
    debug("%s: version: %s, local: %s:%s, remote: %s:%s, client: %s:%s, server: %s:%s\n",
        event, IP:version(),
        IP:local_addr(), TCP:local_port(),
        IP:remote_addr(), TCP:remote_port(),
        IP:client_addr(), TCP:client_port(),
        IP:server_addr(), TCP:server_port())
end
```

TCP:client_port()

Return client TCP port of the connection. The value is integer.

Example

```
when HTTP_REQUEST {|
    print_ips("HTTP_REQUEST", TCP, IP)
}
```

```
function print_ips(event, TCP, IP)
    debug("%s: version: %s, local: %s:%s, remote: %s:%s, client: %s:%s, server: %s:%s\n",
        event, IP:version(),
        IP:local_addr(), TCP:local_port(),
        IP:remote_addr(), TCP:remote_port(),
        IP:client_addr(), TCP:client_port(),
        IP:server_addr(), TCP:server_port())
end
```

TCP:server_port()

Return server TCP port of the connection. The value is integer. If the server is not connected, return nil.

Example

```
when HTTP_REQUEST {
    print_ips("HTTP_REQUEST", TCP, IP)
}
function print_ips(event, TCP, IP)
    debug("%s: version: %s, local: %s:%s, remote: %s:%s, client: %s:%s, server: %s:%s\n",
        event, IP:version(),
        IP:local_addr(), TCP:local_port(),
        IP:remote_addr(), TCP:remote_port(),
        IP:client_addr(), TCP:client_port(),
        IP:server_addr(), TCP:server_port())
end
```

TCP:close()

Close current TCP connection and disable its TCP events. This function can only be used in event `SERVER_CONNECTED`.

Example

```
when SERVER_CONNECTED {
    debug("TCP_CLOSE")
    TCP:close()
}
```

LB commands

LB commands can be used in HTTP events.

LB:routing(“cr-name”)

Force current HTTP transaction to route to specific content routing.

Return value is Boolean. If the policy doesn't have content routing or cannot find the specific content routing, return false. If routing successes, return true.

Example

```
function startsWith(str, prefix)
    return string.sub(str, 1, #prefix) == prefix
end

when HTTP_REQUEST {
    local host = HTTP:host()
    if startsWith(host, "test1.com") then
        LB:routing("cr1")
    elseif startsWith(host, "test2.com") then
        LB:routing("cr2")
    end
}
```

LB:persist("key") / LB:persist("key", timeout)

Use the key string to do persistence. The type of the server pool's persistence must be set to scripting, otherwise the function has no effect.

Please note the following:

- If argument timeout doesn't exist, use the default timeout in the persistence of the server pool.
- If called in HTTP_REQUEST, the system will use the key to search the persistence table. If found, do persistence; If no found, insert key to the persistence table.
- If called in HTTP_RESPONSE, the system will insert the key string to the persistence table.

Examples

Do persistence in HTTP request header:

```
when HTTP_REQUEST {
    local jsession_id = HTTP:cookie("JSESSIONID")
    debug("jession_id=%s", jsession_id)
    if jsession_id then
        debug("jsession_id=%s", jsession_id)
        LB:persist(jsession_id)
    end
}
```

Do persistence in HTTP request body:

```
when HTTP_REQUEST {
    local uri = HTTP:url()
    debug("uri=%s", uri)
    if string.match(uri, "test_url") then
        HTTP:collect()
    end
}
when HTTP_DATA_REQUEST {
    local body_str = HTTP:body()
    local find_sessionID = body_str:find("persist=")

    if find_sessionID_2 then
        local start_pos = body_str:find("persist=")
        local sessionID = body_str:sub(start_pos + 8, start_pos + 8 + 3)
        debug("sessionID=%s", sessionID)
        LB:persist(sessionID)
    end
}
```

Do persistence in HTTP response header:

```
when HTTP_RESPONSE {
    local jsession_id = HTTP:cookie("JSESSIONID")
    local code, reason = HTTP:status()
    debug("code=%s", code)
    if jsession_id then
        -- if server respone has this cookie
        -- record the persistence to the persistence table
        debug("jsession_id=%s", jsession_id)
        LB:persist(jsession_id)
    end
}
```

Do persistence in HTTP response body:

```
when HTTP_RESPONSE {
    local code, reason = HTTP:status()
    debug("code=%s", code)
    if code == "302" then
        HTTP:collect()
    end
}
```

```
when HTTP_DATA_RESPONSE {
    local body_str = HTTP:body()
    local find_sessionID = body_str:find("persist=")
    if find_sessionID then
        local start_pos = body_str:find("persist=")
        local sessionID = body_str:sub(start_pos + 8, start_pos + 8 + 3)
        debug("sessionID=%s", sessionID)
        LB: persist(sessionID)
    end
}
```

SSL commands

SSL:sni()

Returns the SNI or false (if no).

This function should be used in script events `CLIENTSSL_HANDSHAKE` and `SERVERSSL_HANDSHAKE`.

Example

```
when CLIENTSSL_HANDSHAKE {
    local svr_name = SSL:sni()
    if svr_name then
        debug("client handshake sni: %s\n", svr_name)
    end
}
```

SSL: set_sni(svr_name)

Returns true if the server name indication extension has been set, otherwise false.

This function should be used in the script event `SEVERSSL_CLIENTHELLO_SEND`.

Example

```
when SERVERSSL_CLIENTHELLO_SEND {
    svr_name = "www.visa.com"
    debug("set Server Name Indication(SNI) in ClientHello = %s\n", svr_name)
    SSL:set_sni(svr_name)
}
```

SSL:cipher()

Returns the cipher in handshake (string type, in OPENSSL form). Please note that the name returned is in standard RFC format.

Example

```
when CLIENTSSL_HANDSHAKE {
    local cipher = SSL:cipher()
    if cipher then
       debug("cipher in client handshake =%s\n", cipher)
    end
}
```

SSL:version()

Returns the SSL version in handshake (string type).

This function should be used in script events `CLIENTSSL_HANDSHAKE` and `SERVERSSL_HANDSHAKE`.

Example

```
when CLIENTSSL_HANDSHAKE {
    local ssl_version = SSL:version()
    debug("client ssl version : %s\n", ssl_version)
}
```

SSL:alpn()

Returns the ALPN protocol selected in handshake (string type). Returns false if not presented or supported.

This function should be used in script events `CLIENTSSL_HANDSHAKE` and `SERVERSSL_HANDSHAKE`.

Example

```
when CLIENTSSL_HANDSHAKE {
    local alpn_protocol = SSL:alpn()
    if alpn_protocol then
        debug("alpn_protocol in client handshake =  %s\n", alpn_protocol)
    end
}
```

SSL:client_cert_verify()

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

This function should be ONLY used in script event `CLIENTSSL_HANDSHAKE`.

Example

```
when CLIENTSSL_HANDSHAKE {
    debug("status of client-certificate-verify = %s", SSL:client_cert_verify())
}
```

SSL: cert_count()

Returns the total number of certificates that the peer has offered, including the peer certificate and client certificate chains. (Integer)

This function should be ONLY used in script event `CLIENTSSL_HANDSHAKE`.

Example

```
when CLIENTSSL_HANDSHAKE {
    if SSL:client_cert_verify() then
        debug("client cert verify enabled\n")
        local cert_cnt = SSL:cert_count()
        debug("cert_cnt number %d\n", cert_cnt)
    end
}
```

SSL: get_peer_cert_by_idx(index_value)

Returns the issuer certificate of the index of the X509 SSL certificate in the peer certificate chain, where index is a value greater than or equal to zero.

A value of zero denotes the first certificate in the chain (aka leaf peer certificate);

A value of one denotes the next, and so on. If the input value is out of range, return nil.

Return type: A table including the information of a client certificate.

This function should be ONLY used in script event `CLIENTSSL_HANDSHAKE`.

Example

```
when CLIENTSSL_HANDSHAKE {
    if SSL:client_cert_verify() then
        debug("client cert verify enabled\n")
        local cert_cnt = SSL:cert_count()
        debug("cert_cnt number %d\n", cert_cnt)
        if cert_cnt >= 1 then
            local cert_table = SSL:get_peer_cert_by_idx(0)
            print_table(cert_table, 0)
        end
        debug("verify result: %d\n", SSL:verify_result())
    end
}
-- a function to print a table, i represents the number of \t for formatting purpose.
function print_table(table, indent)
    local space = string.rep('\t',indent)
    for key, value in pairs(table) do
        if(type(value)=='table') then
            debug("%s sub-table[%s]\n", space, key)
            print_table(value, indent+1)
        else
            debug("%s %s: %s\n", space, key, value)
        end
    end
end
```

SSL: verify_result()

Returns the result code from peer certificate verification. The returned code uses the same values as those of OpenSSL's X509 verify_result (X509_V_ERR_) definitions.

Returns type: Integer. Returns -1 if the verification code can not be retrieved

This function should be ONLY used in script event `CLIENTSSL_HANDSHAKE`.

Example

```
when CLIENTSSL_HANDSHAKE {
    if SSL:client_cert_verify() then
        debug("client cert verify enabled\n")
        debug("verify result: %d\n", SSL:verify_result())
    end
}
```

SSL Renegotiate

SSL_RENEGOTIATE()

When the system evaluates the command under a client-side context, the system immediately renegotiates a request for the associated client-side connection.

This function is temporarily ONLY available in HTTP_REQUEST event.

It returns `true` for success and `false` for failure.

This function does not support TLS1.3.

Example

In this sample script, when an HTTPS request with the prefix "autotest" is received, it triggers client certificate verification through SSL renegotiation.

Once the SSL renegotiation is completed, it checks the content-routing policy.

If the client certificate presented by the client meets certain conditions that matches a specific HTTP content routing policy, the traffic will be directed to a designated server pool.

The following is a function to print a table, i represents the number of \t for formatting purpose.

```
function print_table(table, indent)
    local space = string.rep('\t',indent)
    for key, value in pairs(table) do
        if(type(value)=='table') then
            debug("%s sub-table[%s]\n", space, key)
            print_table(value, indent+1)
        else
            debug("%s %s: %s\n", space, key, value)
        end
    end
end

when HTTP_REQUEST {
    local url = HTTP:url()
    if url:find("^/autotest") and HTTP:is_https() and SSL:client_cert_verify() then
        -- Trigger SSL renegotiate only when it's https request and SSL connection has
already been established
        -- Example URL-based certificate verify and then Content-Routing
        debug("url: %s match rule, need client certificate verify\n", url)
        local cert_count = SSL:cert_count()
        debug("cert_count = %s\n", cert_count)
        if cert_count and cert_count == 0 then
            SSL:renegotiate()
            debug("emit SSL renegotiation\n")
        end
    end
}


when CLIENTSSL_RENEGOTIATE {
    local cert_count = SSL:cert_count()
    debug("cert_count = %s\n", cert_count)
    if cert_count and cert_count > 0 then
        local cert_table = SSL:get_peer_cert_by_idx(0)
        print_table(cert_table, 0)
        local subject = cert_table["subject"]
        -- match CN value with regular expression
        local cn_value = subject:match("CN%s-=%s-([^,%s]+)")
        debug("CN value in X509 subject is: %s\n", cn_value)
        if cn_value and cn_value == "test1" then
            LB:routing("ctrt")
        end
    end
}
```

SSL:session(t) [TODO]

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

Input `t` is a table, with a key “operation”, and there will be three choices: “get_id” or “remove” or “reused”.

Return string for get operation, and boolean for remove or reused operation.

This function should be used in script events `CLIENTSSL_HANDSHAKE` and `SERVERSSL_HANDSHAKE`.

HTTP Commands

HTTP commands can be used in HTTP events.

Header fetch

HTTP:headers()

Fetch all HTTP request or response headers. When it is called in client side, it returns all HTTP request headers; When it is called in server side, it returns all HTTP response headers.

Return: lua table of arrays.

Example

```
when HTTP_REQUEST {
    for k, v in pairs(HTTP:headers()) do
        for i = 1, #v do
            debug("HEADER: %s[%d]: %s\n", k, i, v[i])
        end
    end
}
```

HTTP:header(“header-name”)

Fetch specific HTTP request or response header.

Return: lua array.

Example

```
when HTTP_RESPONSE {
    for i, v in ipairs(HTTP:header("set-cookie")) do
        debug("set-cookie[%d]: %s\n", i, v)
    end
}
```

HTTP:cookies()

Fetch all cookies. When it is called in client side, it fetches “Cookies”; When it is called in server side, it fetches “Set-Cookie”.

Return: lua table containing only keys and values.

Example

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

HTTP:cookie(“cookie-name”)

Fetch the value of specific cookies.

Return: string.

Example

```
when HTTP_REQUEST {
    persist = HTTP:cookie("persist")
}
```

HTTP:args()

Fetch all arguments of HTTP query.

Return: lua table containing key and value.

Example

```
when HTTP_REQUEST {
    for k, v in pairs(HTTP:args()) do
        debug("ARG: %s = %s\n", k, v)
    end
}
```

HTTP:arg(“arg-name”)

Fetch the value of specific arguments.

Return: string.

Example

```
when HTTP_REQUEST {
    v = HTTP:arg("ip")
}
```

HTTP:host()

Return the string of HTTP request host.

Example

Request : http://www.example.com/test.html?id=1234

```
when HTTP_REQUEST {
    local host = HTTP:host()
    if host == "www.example.com" then
        debug("host = %s", host)
    end
}
```

Output: `www.example.com`

HTTP:url()

Return the string of HTTP request URL. It is the full URL including path and query.

Example

For instance, request url: http://www.example.com/test.html?id=1234

```
when HTTP_REQUEST {
    local url = HTTP:url()
    if url == "/test.html?id=1234" then
        debug("url = %s", url)
    end
}
```

Output: `url = /test.html?id=1234`

HTTP:path()

Return the string of the HTTP request path.

Example

For instance, request url: http://www.example.com/test.html?id=1234

```
when HTTP_REQUEST {
    local path = HTTP:path()
    if path == "/test.html" then
        debug("path = %s", path)
    end
}
```

Output: `path = /test.html`

HTTP:method()

Return the string of HTTP request method.

Example

```
when HTTP_REQUEST {
    local method = HTTP:method()
    debug("method = %s", method)
    if method == "GET" then
       debug("method = %s", method)
    end
}
```

Output: `method = GET`

HTTP:version()

Return the string of HTTP request or response version.

Example

```
when HTTP_RESPONSE {
    debug("http version = %s", HTTP:version())
}
```

Output: `http version = 1.1`

HTTP:status()

Return two strings including HTTP response status code and reason.

`code, reason = HTTP:status()`

Example

```
when HTTP_RESPONSE {
    code, reason = HTTP:status()
    if code == "200" then
        debug("code = 200, reason = %s", reason)
    end
}
```

Output: `code = 200, reason = OK`

Header manipulate

HTTP:set_path("new-path")

Change the path in HTTP request header.

Return true for success and false for failure.

Example

```
when HTTP_REQUEST {
    HTTP:set_path("/new_path")
}
```

HTTP:set_query("new-query")

Change the query in HTTP request header.

Return true for success and false for failure.

Example

```
when HTTP_REQUEST {
    HTTP:set_query("test=1")
}
```

HTTP:set_url("new-url")

Change the whole URL, including the path and query.

Return true for success and false for failure.

Example

```
when HTTP_REQUEST {
    HTTP:set_url("/new_path?test=1")
}
```

HTTP:set_method("new-method")

Change the method in HTTP request header.

Return true for success and false for failure.

Example

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

HTTP:set_status(status-code) \ HTTP:set_status(status-code, "reason")

Change the status code and reason in HTTP response header. If reason does not exist, use default reason.

Return true for success and false for failure.

Example

```
when HTTP_RESPONSE {
    HTTP:set_status(200, "Other Reason")
}
```

HTTP:add_header("header-name", "header-value")

Add a header line to HTTP request or response header.

Return true for success and false for failure.

Example

```
function rewrite_request(HTTP, IP, args)
    debug("%s", IP:client_addr())
    client_ip = IP:client_addr()
    -- add/del/set header
    HTTP:add_header("X-COUNTRY-FMF", ip.geo(client_ip) or "unknown") -- add a new header
line
end
when HTTP_REQUEST{
    local path = HTTP:path()
    if path == "/rewrite_request" then
         rewrite_request(HTTP, IP, HTTP:args())
    end
}
```

HTTP:del_header("header-name")

Remove the header with name "header-name" from HTTP request or response.

Return true for success and false for failure.

Example

```
function rewrite_request(HTTP, IP, args)
    debug("%s", IP:client_addr())
    client_ip = IP:client_addr()
    -- add/del/set header
    HTTP:del_header("test")
end
when HTTP_REQUEST{
    local path = HTTP:path()
    if path == "/rewrite_request" then
        rewrite_request(HTTP, IP, HTTP:args())
    end
}
```

HTTP:set_header("header-name", header-value-array)

Remove the header with name “header-name” from HTTP request or response, and add this header with new value header-value-array. The argument header-value-array is a Lua array which is the value got from HTTP:header().

Return true for success and false for failure.

Example

```
function rewrite_request(HTTP, IP, args)
    debug("%s", IP:client_addr())
    client_ip = IP:client_addr()
    -- add/del/set header
    HTTP:set_header("test", { "line1", "line2", "line3" })
end
when HTTP_REQUEST{
    local path = HTTP:path()
    if path == "/rewrite_request" then
        rewrite_request(HTTP, IP, HTTP:args())
    end
}
```

HTTP:replace_header(“header-name”, “regex”, “replace”)

Match the regular expression in all occurrences of header field “header-name” according to “regex”, and replaces them with the “replace” argument. The replacement value can contain back references like 1,2, …

Return true for success and false for failure.

Example

```
function rewrite_request(HTTP, IP, args)
    debug("%s", IP:client_addr())
    client_ip = IP:client_addr()
    -- add/del/set header
    HTTP:replace_header("Set-Cookie", [[(.*)(Path=\/)(.*)]], [[\1\2api\3]])
end
when HTTP_REQUEST{
    local path = HTTP:path()
    if path == "/rewrite_request" then
        rewrite_request(HTTP, IP, HTTP:args())
    end
}
```

Custom reply

These functions only can be used in HTTP client side event (only HTTP_REQUEST now).

HTTP:redirect (“fmt”, …)

Reply to client with redirect response.

Example

```
when HTTP_REQUEST {
    HTTP:redirect(“https://%s”, HTTP:host())
}
```

HTTP:reply (response)

Reply to client with custom response.

Argument response is a lua array. It includes:

- status: Integer. Default is 200.
- reason: String. If not set, the system will use the default value of status code. For example, if the status code is 200, the default value of reason is “OK”.
- headers: Lua table. Each value of the table is a lua array. It contains all headers except “content-length”. “content-length” will be automatically set with the body size.
- Body: String.

To be specific:

```
HTTP:reply{
status = 400,
reason = “test reason”,
headers = {
["content-type"] = { "text/html" },
["cache-control"] = { "no-cache", "no-store" },
},
body = "<html><body><h1>invalid request<h1></body></html>",
}
```

Example

```
function reply_invalid(HTTP)
    HTTP:reply{
        status = 400,
        headers = {
            ["content-type"] = { "text/html" },
            ["cache-control"] = { "no-cache", "no-store" },
        },
        body = "<html><body><h1>Invalid API Request<h1></body></html>",
    }
end
function check_ip_reputation(HTTP)
    local v = HTTP:arg("ip")
    if v then v = ip.addr(v) end -- convert string to ip
    if not v then return reply_invalid(HTTP) end
    local r = ip.reputation(v)
    local body = string.format("<html><body><h1>Reputation of IP %s: %s<h1></body></html>",
        v, #r > 0 and table.concat(r, ', ') or "No Found")
    HTTP:reply{
        status = 200,
        headers = {
            ["content-type"] = { "text/html" },
            ["cache-control"] = { "no-cache", "no-store" },
        },
        body = body,
    }
end
function check_ip_geo(HTTP)
    local v = HTTP:arg("ip")
    if v then v = ip.addr(v) end -- convert string to ip
    if not v then return reply_invalid(HTTP) end
    local geo = ip.geo(v)
    local geo_code = ip.geo_code(v)
    local body = string.format("<html><body><h1>GEO of IP %s: %s, code:
%s<h1></body></html>",
        v, geo, geo_code)
HTTP:reply{
    status = 200,
    headers = {
        ["content-type"] = { "text/html" },
        ["cache-control"] = { "no-cache", "no-store" },
    },
    body = body,
    }
end
when RULE_INIT {
    actions = {}
    actions["reputation"] = check_ip_reputation
    actions["geo"] = check_ip_geo
    convert = {}
    convert["testing"] = "test"
    convert["debugging"] = "debug"
    whitelist = {}
    whitelist["test"] = true
    whitelist["debug"] = true
    whitelist["others"] = true
}
when HTTP_REQUEST {
    local path = HTTP:path()
    path = path:gsub("^/api/", "/api2/") -- convert /api/ to /api2/
    local api = path:match("^/api2/(.+)") -- get api string that is after /api2/
    -- check api
    if api then
        if actions[api] then -- if api is in table "actions", run function
            return actions[api](HTTP)
        end
        if convert[api] then -- if api is in table "convert", convert api
            api = convert[api] -- change to new api
        end
        if not whitelist[api] then -- if api is not in whitelist, reply invalid
            return reply_invalid(HTTP)
        end
        HTTP:set_path("/api2/" .. api) -- pass the api to server
        return
    end
    -- if path doesn't starts with /api or /api2, do nothing
}
```

Control

HTTP:close()

Close the current HTTP transaction and disable its HTTP events. This function can only be used in event HTTP_REQUEST.

Note the following:

- Close the current HTTP transaction and disable its HTTP events.
- This function can only be used in event HTTP_REQUEST.
- The code logic in the HTTP_REQUEST event will be executed and then close the http connection no matter where method HTTP:close() is.

Example

```
when HTTP_REQUEST {
    local path = HTTP:path()
    HTTP:close()
    local url = HTTP:url()
}
```

Protocol

HTTP:is_https()

Return true if the current transaction is in HTTPS connection.

Example:

```
when HTTP_REQUEST {
    debug("current transaction is HTTPS connections: %s", HTTP:is_https())
}
```

Transaction private data

In Lua, the local value can only be used in function and the global value is shared in whole Lua stack.

In FortiWeb, sometimes a private data is needed for HTTP transaction, and the value is shared in the same HTTP transaction.

HTTP:setpriv(object)

Store a lua object as the HTTP transaction private data. You can store a lua object in event HTTP_REQUEST and fetch it by calling HTTP:priv() in event HTTP_RESPONSE.

Example

```
when HTTP_REQUEST {
    store_data = "test"
    HTTP:setpriv(store_data)
}
when HTTP_RESPONSE {
    debug("stored_data = %s", HTTP:priv())
}
```

HTTP:priv()

Fetch the transaction private data that stored by HTTP:setpriv(). If no result is found, it will return an empty lua table.

```
when HTTP_REQUEST {
    store_data = "test"
    HTTP:setpriv(store_data)
}
when HTTP_RESPONSE {
    debug("stored_data = %s", HTTP:priv())
}
```

Data Collect

HTTP:collect()

`HTTP:collect()` function instructs FortiWeb to buffer and make available the HTTP request or response body for inspection in subsequent script events. This function can only be used in the `HTTP_REQUEST` and `HTTP_RESPONSE` events.

Syntax

```
HTTP:collect(size)
```

size	The number of bytes to collect from the HTTP body. • If **omitted** or set to -1, FortiWeb will collect up to the full length of the body, or until the maximum cached length is reached. • If a **positive integer** is specified, FortiWeb will collect that many bytes before triggering the HTTP_DATA_REQUEST or HTTP_DATA_RESPONSE event. • This enables **partial body inspection** and early triggering of body processing logic.

Availability

- **Events**: `HTTP_REQUEST`, `HTTP_RESPONSE`
- Applies to both request and response bodies depending on the context.

Example: Full Body Collection

```
when HTTP_REQUEST {
    if HTTP:header(“content-type”) == text/css
        HTTP::collect()
    end
}
when HTTP_DATA_REQUEST {
    local body_str = HTTP:body()
    debug("body = %s", body_str)
}
```

Example: Partial Body Collection

```
when HTTP_REQUEST {
    HTTP:collect(32)  -- collect first 32 bytes of the body
}

when HTTP_DATA_REQUEST {
    local body_sample = HTTP:body(0, 32)
    if body_sample then
        debug("partial collect body information, partial body = %s", body sample)
    end
}
```

Notes

- Specifying a partial size can help reduce latency or processing overhead when only a small portion of the body is needed to make decisions.
- When partial collection is used, the `HTTP_DATA_REQUEST` or `HTTP_DATA_RESPONSE` event is triggered as soon as the specified number of bytes is available.
- This function is often used in conjunction with:

- `HTTP:body(offset, length)`
- `debug()` logging

Body Rewrite

HTTP:body (offset, size)

Offset and size are optional.

If offset is missing, it will be set as zero.

If size is missing, it will be set as -1 which means the whole HTTP body.

Return string. The HTTP body will be returned.

Example:

```
when HTTP_REQUEST {
HTTP:collect()
}
    --This function will change "username:test" to "username:Test"
function username_first_char_uppercase(str)
       local str1 = str:sub(1, 9)
    local str2 = str:sub(10, 10)
    str2 = str2:upper()
    local str3 = str:sub(11, -1)
    return str1..str2..str3
    end
    when HTTP_DATA_REQUEST {
    local body_str = HTTP:body(0, 16)
    local body_new = body_str:gsub("username:[A-Za-z][A-Za-z0-9_]+", username_first_char_
uppercase)
    debug("body old = %s, body new = %s\n", body_str, body_new)
    HTTP:set_body(body_new, 0, 16)
}
```

HTTP:set_body("body_str", offset, size)

Offset and size are optional.

If offset is missing, it will be set as zero.

If size is missing, it will be set as -1 which means the whole `http` body.

The body_str is the HTTP body. Now only the string type body is supported.

Return boolean: `true/false`.

Example

```
when HTTP_REQUEST {
    HTTP:collect()
}
--This function will change "username:test" to "username:Test"
    function username_first_char_uppercase(str)
    local str1 = str:sub(1, 9)
    local str2 = str:sub(10, 10)
    str2 = str2:upper()
    local str3 = str:sub(11, -1)
    return str1..str2..str3
end
when HTTP_DATA_REQUEST {
    local body_str = HTTP:body(0, 16)
    local body_new = body_str:gsub("username:[A-Za-z][A-Za-z0-9_]+", username_first_char_
uppercase)
    debug("body old = %s, body new = %s\n", body_str, body_new)
    HTTP:set_body(body_new, 0, 16)
}
```

SSL Management

SSL:close()

Terminates the SSL/TLS connection during the handshake phase, allowing FortiWeb to enforce early-session security decisions based on SSL context. This function is particularly useful in scenarios where you need to prevent connections from proceeding beyond the SSL layer, such as when rejecting traffic based on the Server Name Indication (SNI) value or other handshake metadata before HTTP parsing or WAF processing occurs.

Internally, `SSL:close()` triggers a connection teardown by sending a TCP FIN or RST (depending on timing and state) without completing the handshake or generating application-level logs. Because the TLS handshake is aborted, this function minimizes resource usage and ensures the transaction is dropped silently from the client's perspective.

This function can only be used in the `CLIENTSSL_HANDSHAKE` or `SERVERSSL_HANDSHAKE` script events, where SSL-specific inspection (e.g., SNI retrieval via `SSL:sni()`) is supported.

Syntax

SSL:close()

Availability

- **Events**: `CLIENTSSL_HANDSHAKE`, `SERVERSSL_HANDSHAKE`

Behavior

- Terminates the SSL connection immediately, preventing further processing (including HTTP and WAF modules).
- Can be combined with functions like `SSL:sni()` to enforce domain-level access control.
- The connection is dropped silently without alerting the client (no TLS alerts or HTTP response).

Example: Block clients based on SNI

```
when CLIENTSSL_HANDSHAKE {
    local svr_name = SSL:sni()
    if svr_name == "www.blocked-site.com" then
        SSL:close()
        debug("Blocked connection with SNI: %s\n", svr_name)
    end
}
```

Example: Block server-side handshake for specific domain

```
when SERVERSSL_HANDSHAKE {
    local svr_name = SSL:sni()
    if svr_name == "internal-only.example.com" then
        SSL:close()
        debug("Terminating server handshake for internal domain: %s\n", svr_name)
    end
}
```

SSL_RENEGOTIATE()

When the system evaluates the command under a client-side context, the system immediately renegotiates a request for the associated client-side connection. This function is temporarily ONLY available in HTTP_REQUEST event.

Return true for success and false for failure.

Example

In this sample script, when an HTTPS request with the prefix "autotest" is received, it triggers client certificate verification through SSL renegotiation.

Once the SSL renegotiation is completed, it checks the content-routing policy.

If the client certificate presented by the client meets certain conditions that matches a specific HTTP content routing policy, the traffic will be directed to a designated server pool.

```
--
#a function to print a table, i represents the number of \t for formatting purpose.
function print_table(table, indent)
    local space = string.rep('\t',indent)
    for key, value in pairs(table) do
        if(type(value)=='table') then
             debug("%s sub-table[%s]\n", space, key)
             print_table(value, indent+1)
        else
            debug("%s %s: %s\n", space, key, value)
        end
    end
end

when HTTP_REQUEST {
    local url = HTTP:url()
    if url:find("^/autotest") and HTTP:is_https() and SSL:client_cert_verify() then
        -- Trigger SSL renegotiate only when it's https request and SSL connection has already
been established
        -- Example URL-based certificate verify and then Content-Routing
        debug("url: %s match rule, need client certificate verify\n", url)
        local cert_count = SSL:cert_count()
        debug("cert_count = %s\n", cert_count)
        if cert_count and cert_count == 0 then
            SSL:renegotiate()
            debug("emit SSL renegotiation\n")
        end
    end
}

when CLIENTSSL_RENEGOTIATE {
    local cert_count = SSL:cert_count()
    debug("cert_count = %s\n", cert_count)
    if cert_count and cert_count > 0 then
        local cert_table = SSL:get_peer_cert_by_idx(0)
        print_table(cert_table, 0)
        local subject = cert_table["subject"]
        -- match CN value with regular expression
        local cn_value = subject:match("CN%s-=%s-([^,%s]+)")
        debug("CN value in X509 subject is: %s\n", cn_value)
        if cn_value and cn_value == "test1" then
            LB:routing("ctrt")
        end
    end
}
```

Selective WAF Bypass

HTTP:skip_waf()

Use this Lua script to instruct FortiWeb to bypass WAF module inspection based on HTTP request or response content. This is especially useful when specific URL patterns, headers, or body contents are known to be safe but would otherwise be flagged by WAF rules.

Function:

HTTP:skip_waf()

Supported Events:

- **HTTP_REQUEST, HTTP_RESPONSE**: Skips all WAF modules(except for layer3 session level checked WAF modules)
- **HTTP_DATA_REQUEST, HTTP_DATA_RESPONSE (partial)**: Skips follow-up WAF modules if HTTP:collect(N) was used.
- **HTTP_DATA_* (full body)**: Skips remaining modules after LUA_body module follow-up processing.

Example:

```
when HTTP_REQUEST {
    HTTP:collect(32)
}

when HTTP_DATA_REQUEST {
    local url = HTTP:url()
    if url and url:find("^/webaccess/") then
        local body = HTTP:body(0, 32)
        if body and body:find("^MUXV2 / HTTP/1.0") then
            HTTP:skip_waf()
            debug("Bypassing WAF for MUXV2 protocol\n")
        end
    end
}
```

Example: Bypass for Tunnel Initialization Requests

In some deployment scenarios (e.g., VMware BLAST protocol), specific HTTP requests should bypass WAF inspection to avoid false positives and performance degradation. The following script skips WAF processing for POST requests to the /ice/tunnel endpoint if they use chunked transfer encoding:

```
when HTTP_REQUEST {
    local method = HTTP:method()
    local uri = HTTP:url()
    local transfer_enc = HTTP:header("Transfer-Encoding")
    if method == "POST"
        and uri:find("^/ice/tunnel")
        and transfer_enc == "chunked" then
        HTTP:skip_waf()
        debug("Bypassing WAF for tunnel request: %s\n", uri)
    end
}
```

This logic helps prevent misclassification of tunneled traffic and ensures compatibility with applications using streaming protocols.

Use Cases

Content routing by url

```
when HTTP_REQUEST {
    local url = HTTP:url()
    if url:find("^/sports") or url:find("^/news") or url:find("^/government") then
        LB:routing("cr1")
        debug("url %s starts with sports|news|government, routing to cr1\n", url)
    elseif url:find("^/finance") or url:find("^/technology") or url:find("^/shopping") then
        LB:routing("cr2")
        debug("url %s starts with finance|technology|shopping, routing to cr2\n", url)
    elseif url:find("^/game") or url:find("^/travel") then
        LB:routing("cr3")
        debug("url %s starts with game|travel, routing to cr3\n", url)
    else
        LB:routing("cr")
        debug("No match for uri: %s, routing to default cr\n", url)
    end
}
```

HTTP to HTTPS redirection for special ports

Only use the first port in HTTPS service.

```
when HTTP_REQUEST {
    if not HTTP:is_https() then
        local host = HTTP:header("host")[1]
        local https_port = policy.https_ports()[1] -- get the first port in HTTP service
        local newhost = host:gsub(":(%d+)", "") -- remove port from host if it has
        if https_port ~= 443 then
            -- if https port is not 443, add port to host
            newhost = newhost .. ":" .. tostring(https_port)
        end
        HTTP:redirect("https://%s%s", newhost, HTTP:url())
    end
```

HTTP connection will be closed if it finds any IP in IP reputation database during XFF headers full scanning

This function extracts all IPs from XFF headers and returns IP array.

```
function extract_xff(xff)
    local t = {}
    local k, v, s
    for k, v in ipairs(xff) do
        for s in v:gmatch("([^,]+)") do
            t[#t + 1] = s:gsub("%s+", "")
        end
    end
    return t
end
when HTTP_REQUEST {
    local ips = extract_xff(HTTP:header("X-Forwarded-For"))
    local r, i, v
    for i, v in ipairs(ips) do
        r = ip.reputation(v) -- check ip, will return an array
        if #r > 0 then -- Found IP in reputation database
            debug("Found bad IP %s in XFF headers, reputation: <%s>, GEO country: <%s>, GEO
country code: %s\n",
                v, table.concat(r, ', '),
                ip.geo(v) or "unknown", ip.geo_code(v) or "unknown")
            HTTP:close() -- force close this HTTP connection
            return -- Stop script and return
        end
    end
}
```

When traffic originates only from the specified IP addresses, the URL should be redirected from /test1 to /test2. All other IPs should not be redirected.

- 1.1.1.1
- 2.2.2.2

```
when HTTP_REQUEST {
    local ip_addresses = {"1.1.1.1", "2.2.2.2", "3.3.3.3"}
    local skipIPs = {}
    for _, ip in ipairs(ip_addresses) do
        skipIPs[ip] = true
    end
    local url = HTTP:url()
    local ip = tostring(IP:client_addr())
    debug("url = %s, ip = %s, contains = %s ", url, ip, skipIPs[ip])
    if skipIPs[ip] == nil and url == "/autotest/test1.html" then
        debug("redirect")
        HTTP:redirect("https://%s/autotest/upload/upload.html", HTTP:host())
    end
}
```

Decryption and Encryption

rand()

Generates a random number, returns an integer value between 0 and RAND_MAX(2^31-1).

Example

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

time()

Returns the current time as an integer, in Unix time format.

Example

```
when HTTP_REQUEST {
    local now = time()
    debug("time now = %d\n", now)
}
```

time_ms()

Returns the current time in million seconds, in Unix time format

Example

```
when HTTP_REQUEST {
    local now_ms = time_ms()
    debug("time now in million seconds = %d\n", now_ms)
}
```

ctime()

Returns the current time as a string, For instance `Thu Apr 15 09:01:46 2024 CST +0800`

Example

```
when HTTP_REQUEST {
    local now_str = ctime()
    debug("time now in string format:  %s\n", now_str)
}
```

md5(input_msg)

Calculates the MD5 hash of a given string input and returns the result as a string.

Example

The following is a helper function to convert byte string into hex representation.

```
function bytes2hex(bytestr)
    local hexString = ""
    for i = 1, string.len(bytestr) do
        hexString = hexString .. string.format("%02x", string.byte(bytestr, i))
    end
    return hexString
end
when HTTP_REQUEST {
    local md5_encrypted = md5_str("123")
    debug("length of md5_encrypted is %d \n", string.len(md5_encrypted))
    debug("encrypted md5 of string 123 is:  %s\n", bytes2hex(md5_encrypted))
}
```

md5_hex_str(input_msg)

Calculates the hex representation of the MD5 of a string, and returns the result as a string.

Example

```
when HTTP_REQUEST {
    local md5_encrypted_hex = md5_hex_str("123")
    debug("encrypted md5 of string 123 in hex representation is:  %s\n", md5_encrypted_hex)
}
```

sha1_str(input_msg)

Calculates the SHA1 of a string input, and returns the result as a string.

Example

The following is a helper function to convert byte string into hex representation.

```
function bytes2hex(bytestr)
    local hexString = ""
    for i = 1, string.len(bytestr) do
        hexString = hexString .. string.format("%02x", string.byte(bytestr, i))
    end
    return hexString
end
when HTTP_REQUEST {
    local sha1_123 = sha1_str("123")
    debug("length of sha1_123 is %d \n", string.len(sha1_123))
    debug("encrypted sha1 of string 123  is:  %s\n", bytes2hex(sha1_123))
}
```

sha1_123_hex(input_msg)

Calculates the hex representation of SHA1 of a string input, and returns the result as a string.

Example

```
when HTTP_REQUEST {
    local sha1_123_hex = sha1_hex_str("123")
    debug("encrypted sha1 of string 123 in hex representation is:  %s\n", sha1_123_hex)
}
```

sha256_str(input_msg)

Calculates the SHA256 of a string input, and returns the result as a string.

Example

The following is a helper function to convert byte string into hex representation.

```
function bytes2hex(bytestr)
    local hexString = ""
    for i = 1, string.len(bytestr) do
        hexString = hexString .. string.format("%02x", string.byte(bytestr, i))
    end
    return hexString
end
when HTTP_REQUEST {
    local sha256_123 = sha256_str("123")
    debug("length of sha256_123 is %d \n", string.len(sha256_123))
    debug("encrypted sha256 of string 123  is:  %s\n", bytes2hex(sha256_123))
}
```

sha256_hex_str(input_msg)

Calculates the hex representation of SHA1 of a string input, and return the result as a string.

Example

```
when HTTP_REQUEST {
    local sha256_123_hex = sha256_hex_str("123")
    debug("encrypted sha256 of string 123 in hex representation is:  %s\n", sha256_123_hex)
}
```

sha512_123(input_msg)

Calculates the SHA512 of a string input, and returns the result as a string.

Example

The following is a helper function to convert byte string into hex representation.

```
function bytes2hex(bytestr)
    local hexString = ""
    for i = 1, string.len(bytestr) do
        hexString = hexString .. string.format("%02x", string.byte(bytestr, i))
    end
    return hexString
end
when HTTP_REQUEST {
    local sha512_123 = sha512_str("123")
    debug("length of sha512_123 is %d \n", string.len(sha512_123))
    debug("encrypted sha512 of string 123  is:  %s\n", bytes2hex(sha512_123))
}
```

sha512_123_hex(input_msg)

Calculates the hex representation of SHA1 of a string input, and returns the result in string representation.

Example

```
when HTTP_REQUEST {
    local sha512_123_hex = sha512_hex_str("123")
    debug("encrypted sha512 of string 123 in hex representation is:  %s\n", sha512_123_hex)
}
```

base64_enc(input_msg)

Encodes a string input in base64 and outputs the results in string format.

Example

```
when HTTP_REQUEST {
    local b64_msg = base64_enc("https://www.base64encode.org/")
    debug("base64 encoded message is:  %s\n", b64_msg)
}
```

base64_dec(input_msg)

Decodes a base64 encoded string input and outputs the results in string format.

Example

```
when HTTP_REQUEST {
    local b64_dec_msg = base64_dec(b64_msg)
    debug("base64 decoded message is:  %s\n", b64_dec_msg)
}
```

base32_enc(input_msg)

Encodes a string input in base32 and outputs the results in string format.

Example

```
when HTTP_REQUEST {
    local b32_msg = base32_enc("https://www.base64encode.org/")
    debug("base32 encoded message is:  %s\n", b32_msg)
}
```

base32_dec(input_msg)

Decodes a base32 encoded string input and outputs the results in string format.

Example

```
when HTTP_REQUEST {
    local b32_dec_msg = base32_dec(b32_msg)
    debug("base32 decoded message is:  %s\n", b32_dec_msg)
}
```

htonl(input_msg)

Converts a long integer input into network byte order.

Example

```
when HTTP_REQUEST {
    local network_a  = htonl(32)
    debug("htonl of 32 is:  %s\n", network_a)
}
```

htons(input_msg)

Converts a short integer input into network byte order.

Example

```
when HTTP_REQUEST {
local network_a_short  = htons(32)
debug("htons of 32 is:  %s\n", network_a_short)
}
```

htons(input_msg)

Converts a long integer input into host byte order. Keep in mind, htonl(ntohl(x)) == x.

Example

```
when HTTP_REQUEST {
    local host_a = ntohl(network_a)
    debug("ntohl of network_a is:  %s\n", host_a)
}
```

host_a_short(input_msg)

Converts a short integer input into host byte order.

Example

```
when HTTP_REQUEST {
    local host_a_short = ntohs(network_a_short)
    debug("ntohs of network_a_short is:  %s\n", host_a_short)
}
```

to_hex(input_msg)

Converts a string to its hex representation.

Example

```
when HTTP_REQUEST {
    local hexit = to_hex("it")
    debug("hexit is:  %s\n", hexit)
}
```

crc32(input_msg)

Returns the crc32 check value of the string, return value is the crc32 code.

Example

```
when HTTP_REQUEST {
    local crc32_code = crc32("123456789")
    debug("CRC 32 code is:  %d\n", crc32_code)
}
```

key_gen(pass, salt, iter, key_len)

Derives an AES key from a password using a salt and iteration count as specified in RFC 2898 (Password-Based Key Derivation Function 2 with HMAC-SHA256).

Example

The following is a helper function to convert byte string into hex representation.

```
function bytes2hex(bytestr)
    local hexString = ""
    for i = 1, string.len(bytestr) do
       hexString = hexString .. string.format("%02x", string.byte(bytestr, i))
    end
    return hexString
end
when HTTP_REQUEST {
    local new_key = key_gen("pass", "salt", 32, 32)
    debug("new key is %s\n", bytes2hex(new_key))
}
```

aes_enc("your message", "paste your key here", Key Size)

Encrypts a string using AES algorithm.

Example

The following is a helper function to convert byte string into hex representation.

```
when HTTP_REQUEST {
    local aes_encrypted = aes_enc("your message", "paste your key here", 128)
    debug("encrypted in hex is %s, after b64 encoding %s\n", to_hex(aes_encrypted), base64_enc
(aes_encrypted))
}
```

aes_decrypted("your message", "paste your key here", Key Size)

Decrypt a string using AES algorithm.

Example

```
when HTTP_REQUEST {
    local aes_decrypted = aes_dec(aes_encrypted, "paste your key here", 128);
    debug("decrypted msg is %s\n", aes_decrypted)
}
```

EVP_Digest(alg, str)

EVP_Digest(alg, str) EVP_Digest for one-shot digest calculation.

Example

```
when HTTP_REQUEST {
    local evpd = EVP_Digest("MD5", "your data")
    debug("the digest in hex is %s\n", bytes2hex(evpd))
}
```

HMAC(alg, "your data", "paste your key here")

HMAC message authentication code.

Example

```
when HTTP_REQUEST {
    local hm = HMAC("SHA256", "your data", "paste your key here")
    debug("the HMAC in hex is %s\n", bytes2hex(hm))
}
```

HMAC_verify(alg, msg, key, verify)

Checks if the signature is same as the current digest.

Example

```
when HTTP_REQUEST {
    local is_same = HMAC_verify("SHA256", "your data", "paste your key here", hm)
    if is_same then
        debug("HMAC verified\n")
    else
        debug("HMAC not verified\n")
    end
}
```

rand_hex(input)

Generates a random number in HEX.

Example

```
when HTTP_REQUEST {
    local rand_h = rand_hex(16);
    debug("the random hex number  is %s\n", rand_h);
}
```

rand_alphanum(input)

Generates a random alphabet+number sequence.

Example

```
when HTTP_REQUEST {
    local alphanumber = rand_alphanum(16);
    debug("the alphabet+number sequence  is %s\n", alphanumber);
}
```

rand_seq(input)

Generates a random number sequence.

Example

```
when HTTP_REQUEST {
    local randseq = rand_seq(16);
    debug("the random sequence is %s\n", to_hex(randseq));
}
```

url_encode(input)

Encodes the target URL (Converts URL into a valid ASCII format, will not replace space by "+" sign).

Example

```
when HTTP_REQUEST {
    local encoded_url = url_encode("https://docs.fortinet.com/product/fortiweb/7.4");
    debug("the encoded url is %s\n", encoded_url);
}
```

url_decode(input)

Decodes the encoding-URL into its original URL.

Example

```
when HTTP_REQUEST {
    local decoded_url = url_decode(encoded_url);
    debug("the decoded url is %s\n", decoded_url);
}
```

www.fortinet.com