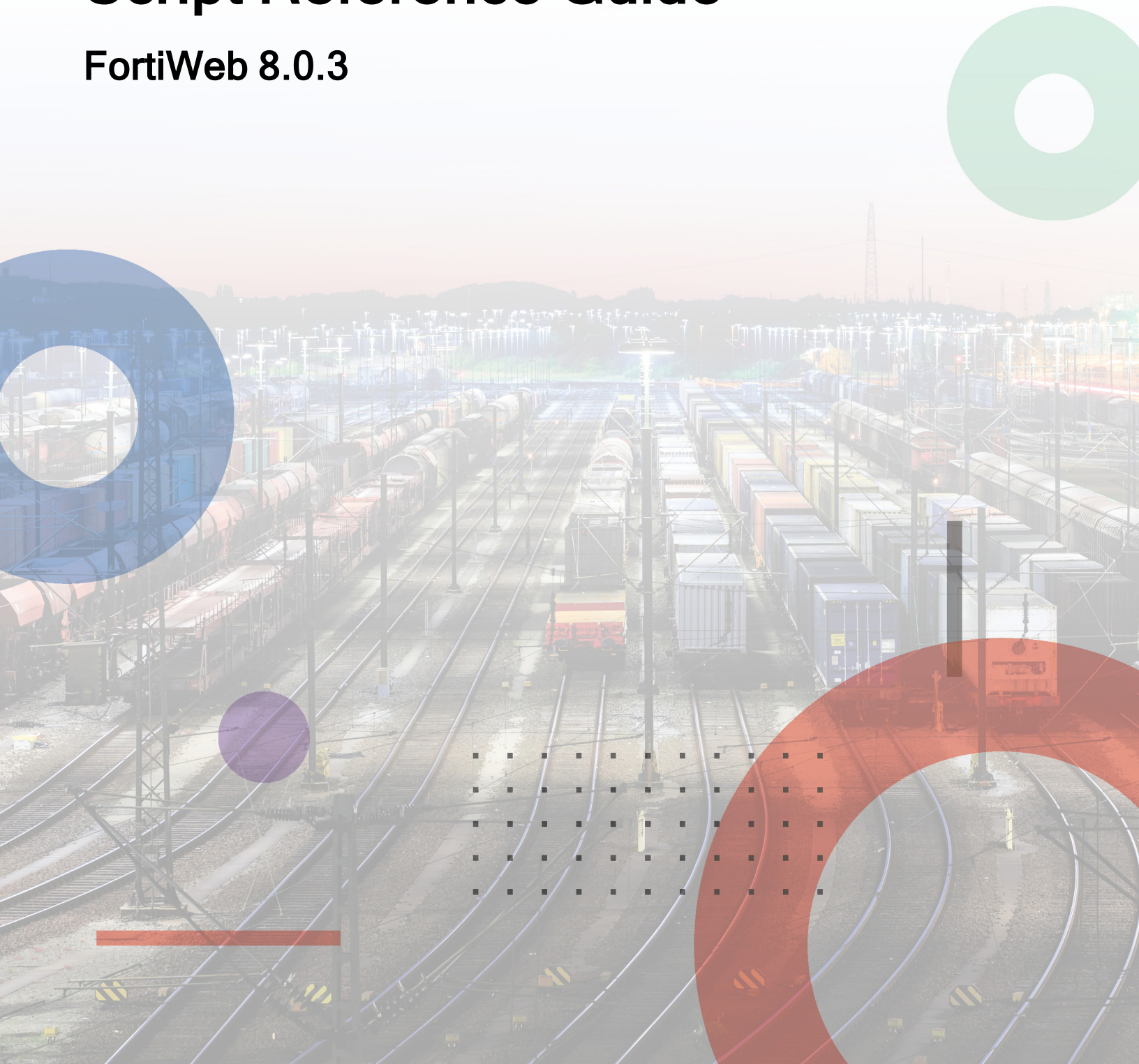
FORTINET

Script Reference Guide

FortiWeb 8.0.3



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

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

FORTINET BLOG
https://blog.fortinet.com

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

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

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

FORTIGUARD LABS
https://www.fortiguard.com

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

FEEDBACK
Email: techdoc@fortinet.com

December 12, 2025
FortiWeb 8.0.3 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.
12/23/2025	Add key concepts and new commands.

Introduction

FortiWeb supports Lua scripting to customize the behavior of your application delivery controller, including defining custom content rules, manipulating HTTP requests and responses, and performing advanced traffic handling.

The following provides an overview of how Lua scripting works in FortiWeb:

Integration with FortiWeb

Lua scripting is integrated into the FortiWeb platform and can be executed at various stages of the request-response lifecycle.

Event Triggers

Lua scripts can be triggered at specific points in the HTTP and TCP request-response flow, such as when a request is received, before it is forwarded to a backend server, or after a response is received.

Access to Request and Response Data

Lua scripts can access request and response data. For HTTP scripting, this includes headers, cookies, and payloads, enabling inspection and modification as needed.

Custom Logic

Lua scripts allow you to implement custom logic, such as routing decisions, content manipulation, header enrichment, authentication, and logging.

Integration with FortiWeb Features

Lua scripting extends FortiWeb functionality by integrating with features such as load balancing, SSL offloading, caching, and traffic management policies.

How to Configure

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 creating 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.

When you disable **Scripting** with loaded scripts, please note the current scripting list will be cleared and must be re-added if scripting is enabled again.

Script Events

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

Type	Event	Trigger event
RULE	RULE_INIT	When the server policy enables or reloads.
	RULE_EXIT	When the server policy disables or reloads.
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 {
...
}
```

Key concepts and features

This section covers key concepts and features you can reference when implementing Lua scripting in FortiWeb:

Lua package compatibility

FortiWeb uses 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

Definition of terms

Frontend

The connection established between the client and FortiWeb when accessing a virtual server service.

Backend

The connection between FortiWeb and the real server after FortiWeb receives and load-balances the client request.

Source IP address (frontend)

The client IP address on the frontend connection.

Destination address (frontend)

The virtual server IP address on the frontend connection.

Source IP address (backend)

The IP address of FortiWeb's outgoing interface on the backend connection.

Destination address (backend)

The real server IP address on the backend connection.

Control structures

The following table lists the Lua control structures.

Type	Structure

if then else	`if condition1 then`
	`  ...`
	`elseif condition2 then`
	`  ...`
	`  break`
	`else`
	`  ...`
	`  goto location1`
	`end`
	
	`::location1::`

for	The following iterates through all key-value pairs in table 't':
	
	`for k, v in pairs(t) do`
	`...`
	`end`

Operators

Operators are symbols used to perform specific mathematical or logical manipulations.

The Lua language includes the following categories of built-in operators:

For more details, see the Lua Reference Manual.

Arithmetic operators

The arithmetic operators listed below operate on real numbers. In the **Example** column, assume that variable **A** is 10 and variable **B** is 20.

Operator	Description	Example
+	Adds two operands.	A + B results in 30.
-	Subtracts the second operand from the first.	A - B results in -10.
*	Multiplies both operands.	A * B results in 200.
/	Divides the first operand by the second operand	B / A results in 2.
//	Floor division divides two operands and rounds the result down toward negative infinity, returning the largest integer less than or equal to the exact division result.	B // A results in 2.
%	Modulo returns the remainder of an integer division where the quotient is rounded toward negative infinity (floor division).	B % A results in 0.
^	Raises the first operand to the power of the second.	A^2 results in 100
-	Unary minus acts as negation.	-A results in -10

Bitwise operators

Bitwise operators convert its operands to integers, operate on all bits of those integers, and result in an integer.

In the **Example** column, assume we have two integer variables with binary representations:

a = 1100; // (12 in decimal)

b = 1010; // (10 in decimal)

Operator	Description	Example
&	Bitwise AND: compares each bit of the first operand with the corresponding bit of the second operand. If both bits are 1, the corresponding result bit is set to 1. Otherwise, the result bit is set to 0.	c = a & b; // (8 in decimal) c = 1000;
|	Bitwise OR: compares each bit of the first operand to the corresponding bit of the second operand. If either of the bits is 1, the corresponding result bit is set to 1. Otherwise, the result bit is set to 0.	c = a | b; // (14 in decimal) c = 1110;
^	Bitwise exclusive OR (XOR): compares each bit of the first operand to the corresponding bit of the second operand. If the bits are different, the corresponding result bit is set to 1. Otherwise, the result bit is set to 0.	c = a ^ b; // (6 in decimal) c = 0110;

>>	Bitwise Right Shift: shifts the bits of the given number to the right by the specified positions.
	When a number is right shifted, 0s are added to the left side of the number, and the rightmost bits are discarded.
		Using a variable "n" and a shift count "s", the left shift operation can be represented as:
		c = n >> s;
		a = 1100; (12 in decimal)
		If we right shift "a" by 2 positions, the resulting value is:
		c = a >> 2; // (3 in decimal)
		c = 11;

<<	Bitwise Left Shift: shifts the bits of the given number to the left by the specified positions. When a number is left shifted, 0s are added to the right side of the number. The leftmost bits are discarded.	Using a variable "n" and a shift count "s", the left shift operation can be represented as:
		c = n << s;
		a = 1010; (10 in decimal)
		If we left shift "a" by 2 positions, the resulting value is:
		c = a << 2; // (40 in decimal)
		c = 101000;

Operator	Description	Example
~	Unary bitwise NOT: used to perform bitwise negation on a single operand. It flips all the bits of the operand, changing each - bit to 1 and each 1 bit to 0.	Using the variable "a": a = 1010; (10 in decimal) c = ~a c = 11111111111111111111111111110101; (-11 in decimal) Applying the unary bitwise NOT operator to "10" flips all the bits, resulting in the Lua integer "-11".

Relational operators

Lua provides the Relational operators listed below. This type of operator always results in true or false.

In the **Example** column, assume variable A is 10 and variable B is 20.

Operator	Description	Example
==	Checks if the value of two operands are equal or not. If yes, then the condition is true.	(A == B) is false.
~=	Checks if the value of two operands are equal or not. If the values are not equal, then the condition is true.	(A ~= B) is true.
>	Checks if the value of the left operand is greater than the value of the right operand. If yes, then the condition is true.	(A > B) is not false.
<	Checks if the value of the left operand is less than the value of the right operand. If yes, then the condition is true.	(A < B) is true.
>=	Checks if the value of the left operand is greater than or equal to the value of the right operand. If yes, then the condition is true.	(A >= B) is false.
<=	Checks if the value of the left operand is less than or equal to the value of the right operand. If yes, then the condition is true.	(A <= B) is true.

Logical operators

Lua provides the Logical operators listed below. This type of operator considers false and nil both as false, and anything else as true.

In the **Example** column, assume variable A is 0 and variable B is 10.

Operator	Description	Example
and	Performs a logical "and" comparison between two values. If both the operands are non-zero then the condition is true. The operator and returns its first argument if it is false; otherwise, it returns its second argument.	(A and B) is false.
or	Performs a logical "or" comparison between two values. If any of the two operands is non-zero, then the condition is true. The operator or returns its first argument if it is not false; otherwise, it returns its second argument	(A or B) is true.
!	Performs a logical "not" on a value. Used to reverse the logical state of its operand. If a condition is true, then the logical not operator will make it false.	!(A and B) is true.

Miscellaneous operators

Miscellaneous operators supported by Lua include concatenation and length.

Operator	Description	Example
..	The string concatenation operator in Lua is denoted by two dots ('..'). If both operands are strings or numbers, then they are converted to a string, similarly to __concat.	a = "Hello " b = "World" a..b returns "Hello World".
#	The length operator returns the length of a string or an array-like table. The length of a string is its number of bytes (that is, the usual meaning of string length when each character is one byte). The length of a table t is defined to be any integer index n such that t[n] is not nil and t[n+1] is nil; moreover, if t[1] is nil, n can be zero.	a = "FortiWeb" t = {1, 2, 3, 4} #a returns 8 #t returns 4

Functions

FortiWeb supports basic lua commands. Additional capabilities can be implemented by creating custom functions using these commands.

Syntax

```
function function_name(parameter)
…
End
```

For example, 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
}
```

String library

The FortiWeb 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.ends_with

- If you want to use regular expression match, you can use string.match with Lua patterns.
- All relational operators >, <, >=, <=, ~=, == can be applied to strings. In particular, == can be used to test if one string equals to another string.
- string.find can be used to test whether one string contains another string.

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

Special characters

When written in a string, these characters look like this (between double quotes): "~!@#$^&*()_+{}[].?"

Note: The back slash (\) and the percent (%) signs are handled in a unique way in log and debug scripts. To print out %, you must use %%; to print out \, you must use \\.

Character	Name
~	Tilde
!	Exclamation
@	At sign
#	Number sign (hash)
$	Dollar sign
^	Caret
&	Ampersand
*	Asterisk
(	Left parenthesis
)	Right parenthesis
_	Underscore
+	Plus

Character	Name
{	Left brace
}	Right brace
[	Left bracket
]	Right bracket
.	Full stop
?	Question mark
\	Backslash

HTTP data body commands

HTTP data body commands support regular expressions, where special characters such as $ ^ ? * + . | ( ) [ ] { } \ " have special meanings. You must escape these characters to treat them as literal values.

To escape a special character, prefix it with two backslashes (\\).

For example, to use the literal caret (^) in an HTTP data body command, enter \^.

For grouping characters such as parentheses (), square brackets [], or braces {}, prefix each character with two backslashes.

For example, to use the {and}, enter \\{and\\}.

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).

Global commands

rand()

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

Syntax

```
rand()
```

Arguments

N/A

Events

Applicable in all events.

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.

Syntax

```
time()
```

Arguments

N/A

Events

Applicable in all events.

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

Syntax

```
time_ms()
```

Arguments

N/A

Events

Applicable in all events.

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

Syntax

```
ctime()
```

Arguments

N/A

Events

Applicable in all events.

Example

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

md5(msg)

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

Syntax

```
md5(msg)
```

Arguments

Name	Description
msg	String type message.

Events

Applicable in all events

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(msg)

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

Syntax

```
md5_hex_str(msg)
```

Arguments

Name	Description
msg	String type message.

Events

Applicable in all events

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(msg)

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

Syntax

```
sha1_str(msg)
```

Arguments

Name	Description
msg	String type message.

Events

Applicable in all events.

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_hex_str(msg)

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

Syntax

```
sha1_hex_str(msg)
```

Arguments

Name	Description
msg	String type message.

Events

Applicable in all events.

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(msg)

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

Syntax

```
Sha256_str(msg)
```

Arguments

Name	Description
msg	String type message.

Events

Applicable in all events.

Examples

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(msg)

Calculates the hex representation of SHA256 of a string input, and return the result as a string.

Syntax

```
Sha256_hex_str(msg);
```

Arguments

Name	Description
msg	String type message

Events

Applicable in all events.

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_str(msg)

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

Syntax

```
sha512_str(msg)
```

Arguments

Name	Description
msg	String type message

Events

Applicable in all events.

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_hex_str(msg)

Calculates the hex representation of SHA512 of a string input and returns the result in string representation.

Syntax

```
sha512_hex_str(msg)
```

Arguments

Name	Description
msg	String type message.

Events

Applicable in all events.

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(msg)

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

Syntax

```
base64_enc(msg)
```

Arguments

Name	Description
msg	String type message.

Events

Applicable in all events.

Example

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

base64_dec(msg)

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

Syntax

```
base64_dec(msg)
```

Arguments

Name	Description
msg	String type message.

Events

Applicable in all events.

Example

```
when HTTP_REQUEST {
    local b64_msg = base64_enc("https://www.base64encode.org/")
    debug("base64 encoded message is:  %s\n", b64_msg)
    local b64_dec_msg = base64_dec(b64_msg)
    debug("base64 decoded message is:  %s\n", b64_dec_msg)
}
```

base32_enc(msg)

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

Syntax

```
base32_enc(msg)
```

Arguments

Name	Description
msg	String type message.

Events

Applicable in all events.

Example

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

base32_dec(msg)

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

Syntax

```
base32_dec(msg)
```

Arguments

Name	Description
msg	String type message.

Events

Applicable in all events.

Example

```
when HTTP_REQUEST {
    local b32_msg = base32_enc("https://www.base64encode.org/")
    debug("base32 encoded message is:  %s\n", b32_msg)
    local b32_dec_msg = base32_dec(b32_msg)
    debug("base32 decoded message is:  %s\n", b32_dec_msg)
}
```

htonl(msg)

Converts a long integer input into network byte order.

Syntax

```
htonl(msg)
```

Arguments

Name	Description
msg	Long integer.

Events

Applicable in all events.

Example

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

htons(msg)

Converts a short integer input into network byte order.

Syntax

```
htons(input_msg)
```

Arguments

Name	Description
msg	Short integer.

Events

Applicable in all events.

Example

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

ntohl(msg)

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

Syntax

```
ntohl(msg)
```

Arguments

Name	Description
msg	Long integer.

Events

Applicable in all events.

Example

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

ntohs(msg)

Converts a short integer input into host byte order.

Syntax

```
ntohs(msg)
```

Arguments

Name	Description
msg	Short integer message.

Events

Applicable in all events.

Example

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

to_hex(msg)

Converts a string to its hex representation.

Syntax

```
to_hex(msg)
```

Arguments

Name	Description
msg	Short integer message.

Events

Applicable in all events.

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.

Syntax

```
crc32(input_msg)
```

Arguments

Name	Description
input_msg	Short integer.

Events

Applicable in all events.

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).

Syntax

```
key_gen(pass, salt, iter, key_len)
```

Arguments

Name	Description
pass	A string type password.
salt	A string type salt.
iter	Integer type iteration count.
key_len	Integer type key length.

Events

Applicable in all events.

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(msg, key, key_size)

Encrypts a string using AES algorithm.

Syntax

```
aes_enc(msg, key, key_size)
```

Arguments

Name	Description
msg	A string type message.
key	A string type key.
key_size	Integer type key size.

Events

Applicable in all events.

Example

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

```
when HTTP_REQUEST {
    local aes_encrypted = aes_enc("msg", "key", 128)
```

```
    debug("encrypted in hex is %s, after b64 encoding %s\n", to_hex(aes_encrypted), base64_
enc(aes_encrypted))
}
```

aes_dec(msg, key, key_size)

Decrypt a string using AES algorithm.

Syntax

```
aes_dec (msg, key, key_size)
```

Arguments

Name	Description
msg	A string type message.
key	A string type key.
key_size	Integer type key size.

Events

Applicable in all events.

Example

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

EVP_Digest(alg, msg)

EVP_Digest for one-shot digest calculation.

Syntax

```
EVP_Digest(alg, msg)
```

Arguments

Name	Description
alg	A string type algorithm. For example, "MD5".
msg	A string type message.

Events

Applicable in all events.

Example

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

HMAC(alg, msg, key)

HMAC message authentication code.

Syntax

```
HMAC(alg, msg, key)
```

Arguments

Name	Description
alg	A string type algorithm. For example, "SHA256".
msg	A string type message.
key	A string type key.

Events

Applicable in all events.

Example

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

HMAC_verify(alg, data, key, digest)

Checks if the signature is same as the current digest.

Syntax

```
HMAC_verify(alg, data, key, digest)
```

Arguments

Name	Description
alg	A string type algorithm. For example, "SHA256".
data	A string type data.
key	A string type key.
digest	A string type digest.

Events

Applicable in all events.

Example

```
when HTTP_REQUEST {
    local hm = HMAC("SHA256", "msg", "key")
    local is_same = HMAC_verify("SHA256", "msg", "key", 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.

Syntax

```
rand_hex (input)
```

Arguments

Name	Description
input	an integer type

Events

Applicable in all events.

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.

Syntax

```
rand_alphanum(input)
```

Arguments

Name	Description
input	an integer type

Events

Applicable in all events.

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.

Syntax

```
rand_seq(input)
```

Arguments

Name	Description
input	an integer type

Events

Applicable in all events.

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).

Syntax

```
url_encode(input)
```

Arguments

Name	Description
input	A string type URL.

Events

Applicable in all events.

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.

Syntax

```
url_decode(input)
```

Arguments

Name	Description
input	A string type URL.

Events

Applicable in all events.

Example

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

debug(fmt, ..)

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

Syntax

```
debug(fmt, ..)
```

Arguments

Name	Description
fmt	A string type input format.

Events

Applicable in all events.

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.

Syntax

```
_id
```

Arguments

N/A

Events

Applicable in all events.

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.

Syntax

```
_name
```

Arguments

N/A

Events

Applicable in all events.

Example

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

Return the string of the policy name.

Syntax

policy.name()

Arguments

N/A

Events

Applicable in all events.

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 }

Syntax

```
policy.http_ports()
```

Arguments

N/A

Events

Applicable in all events.

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 }

Syntax

```
policy.https_ports()
```

Arguments

N/A

Events

Applicable in all events.

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" }

Syntax

```
policy.crs()
```

Arguments

N/A

Events

Applicable in all events.

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.

Syntax

```
policy.servers() / policy.servers("cr-name")
```

Arguments

Name	Description
cr-name	Optional, string type CR name. If cr-name is missing, all servers will be returned.

Events

Applicable in all events.

Example

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

core.debug(level, fmt, ..)

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

Syntax

```
core.debug(level, fmt, ..)
```

Arguments

Name	Description
level	Debug log level
fmt	String type input format

Events

Applicable in all events.

Example

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

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.

Syntax

```
LB:routing(cr-name)
```

Arguments

Name	Description
cr_name	String type content routing name.

Events

Applicable in HTTP_REQUEST.

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.

Syntax

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

Arguments

Name	Description
key	String type persistence key
timeout	optional, integer type timeout, from 10-86400

Events

Applicable in HTTP_REQUEST, HTTP_RESPONSE, HTTP_DATA_REQUEST and HTTP_DATA_RESPONSE.

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
}
```

LB:persist_remove(key)

Remove the persistence node based on the key

Syntax

```
LB:persist_remove(key)
```

Arguments

Name	Description
key	String type persistence key

Events

Applicable in HTTP_REQUEST, HTTP_RESPONSE, HTTP_DATA_REQUEST and HTTP_DATA_RESPONSE.

Example

```
when HTTP_REQUEST {
    local uri = HTTP:path()
    -- Extract the first folder from the URI using pattern matching
    -- For example: /api/v1/resource -> "api"
    local sysName = uri:match("^/([^/]+)/")
    local sysnamelower = sysName:lower()
    if sysnamelower then
```

```
        LB:persist(sysnamelower)
        debug("persist sys name: %s \n", sysnamelower)
    end
}

when HTTP_RESPONSE {
    local uri = HTTP:path()
    local status = HTTP:status()
    -- Extract the first folder from the URI using pattern matching
    -- For example: /api/v1/resource -> "api"
    local sysName = uri:match("^/([^/]+)/")

    if sysName then
        local sysnamelower = sysName:lower()
        if status == "503" then
            LB:persist_remove(sysnamelower)
            debug("status 503, remove %s persistence\n", sysnamelower)
        end
    end
}
```

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.

Syntax

```
TCP:local_port()
```

Arguments

N/A

Events

Applicable in events HTTP_REQUEST, HTTP_RESPONSE, HTTP_DATA_REQUEST, HTTP_DATA_RESPONSE, CLIENT_ACCEPTED, CLIENT_CLOSED, SERVER_CONNECTED and SERVER_CLOSED.

Example

```
when HTTP_REQUEST {
    debug("local port%s”, TCP:local_port())
}
```

TCP:remote_port()

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

Syntax

```
TCP:remote_port()
```

Arguments

N/A

Events

Applicable in events HTTP_REQUEST, HTTP_RESPONSE, HTTP_DATA_REQUEST, HTTP_DATA_RESPONSE, CLIENT_ACCEPTED, CLIENT_CLOSED, SERVER_CONNECTED and SERVER_CLOSED.

Example

```
when HTTP_REQUEST {
    debug("remote port%s”, TCP:remote_port())
}
```

TCP:client_port()

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

Syntax

```
TCP:client_port()
```

Arguments

N/A

Events

Applicable in events HTTP_REQUEST, HTTP_RESPONSE, HTTP_DATA_REQUEST, HTTP_DATA_RESPONSE, CLIENT_ACCEPTED, CLIENT_CLOSED, SERVER_CONNECTED and SERVER_CLOSED.

Example

```
when HTTP_REQUEST {
    debug("client port%s”, TCP:client_port())
}
```

TCP:server_port()

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

Syntax

```
TCP:server_port()
```

Arguments

N/A

Events

Applicable in events HTTP_REQUEST, HTTP_RESPONSE, HTTP_DATA_REQUEST, HTTP_DATA_RESPONSE, CLIENT_ACCEPTED, CLIENT_CLOSED, SERVER_CONNECTED and SERVER_CLOSED.

Example

```
when HTTP_REQUEST {
    debug("server port%s”, TCP:server_port())
}
```

TCP:close()

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

Syntax

```
TCP:close()
```

Arguments

N/A

Events

Applicable in event CLIENT_ACCEPTED

Example

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

SSL commands

SSL:sni()

Returns the SNI or false (if no).

Syntax

```
SSL:sni()
```

Arguments

N/A

Events

Applicable in CLIENTSSL_HANDSHAKE, SERVERSSL_HANDSHAKE, HTTP_REQUEST, and HTTP_DATA_REQUEST

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.

Syntax

```
SSL:set_sni(svr_name)
```

Arguments

Name	Description
svr_name	String type server name indication extension.

Events

Applicable in 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.

Syntax

```
SSL:cipher()
```

Arguments

N/A

Events

Applicable in event CLIENTSSL_HANDSHAKE and SERVERSSL_HANDSHAKE. HTTP_REQUEST and HTTP_DATA_REQUEST

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).

Syntax

```
SSL:version()
```

Arguments

N/A

Events

Applicable in event CLIENTSSL_HANDSHAKE and SERVERSSL_HANDSHAKE. HTTP_REQUEST and HTTP_DATA_REQUEST

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.

Syntax

```
SSL:alpn()
```

Arguments

N/A

Events

Applicable in event CLIENTSSL_HANDSHAKE and SERVERSSL_HANDSHAKE, HTTP_REQUEST and HTTP_DATA_REQUEST

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.

Syntax

```
SSL:client_cert_verify()
```

Arguments

N/A

Events

Applicable in 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)

Syntax

```
SSL:cert_count()
```

Arguments

N/A

Events

Applicable in 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.

Syntax

```
SSL:get_peer_cert_by_idx(index_value)
```

Arguments

N/A

Events

Applicable in 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

Syntax

```
SSL:verify_result()
```

Arguments

N/A

Events

Applicable in 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()

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.

Syntax

```
SSL:renegotiate()
```

Arguments

N/A

Events

Applicable in event HTTP_REQUEST.

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 representing the number of \t for formatting purposes.

```
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: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.

- 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).

Syntax

```
SSL:close()
```

Arguments

N/A

Events

Applicable in event CLIENTSSL_HANDSHAKE, SERVERSSL_HANDSHAKE.

Examples

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
}
```

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
}
```

HTTP Commands

HTTP:redirect(fmt, ...)

Reply to client with redirect response.

Syntax

```
HTTP:redirect(fmt, …)
```

Arguments

Name	Description
fmt	String Type Format.

Events

Applicable in HTTP_REQUEST.

Example

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

HTTP:reply(response)

Reply to client with custom response.

Syntax

```
HTTP:reply(response)
```

Arguments

Name	Description

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>",
	}

Events

Applicable in HTTP_REQUEST.

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
}
```

HTTP:close()

Close the current HTTP transaction and disable its HTTP events.

Syntax

```
HTTP:close()
```

Arguments

N/A

Events

Applicable in HTTP_REQUEST.

Example

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

```
    local url = HTTP:url()
}
```

HTTP:is_https()

Return true if the current transaction is an HTTPS connection.

Syntax

```
HTTP:is_https()
```

Arguments

N/A

Events

Applicable in HTTP_REQUEST, HTTP_RESPONSE, HTTP_DATA_REQUEST, and HTTP_DATA_RESPONSE

Example

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

HTTP:setpriv(object)

Store a lua object as the HTTP transaction private data.

Syntax

```
HTTP:setpriv(object)
```

Arguments

Name	Description
object	Lua object

Events

Applicable in HTTP_REQUEST.

Example

```
when HTTP_REQUEST {
    store_data = "test"
    HTTP:setpriv(store_data)
}
```

HTTP:priv()

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

Syntax

```
HTTP:priv()
```

Arguments

N/A

Events

Applicable in HTTP_RESPONSE.

Example

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

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.

When called in HTTP_REQUEST or HTTP_REQUEST event, skips all WAF modules(except for layer3 session level checked WAF modules).

When called in HTTP_DATA_REQUEST or HTTP_DATA_RESPONSE with partial data collect, skips follow-up WAF modules.

When called in HTTP_DATA_REQUEST or HTTP_DATA_RESPONSE with full data collect, skips remaining modules after LUA_body module follow-up processing.

Syntax

```
HTTP:skip_waf()
```

Arguments

N/A

Events

Applicable in HTTP_REQUEST, HTTP_RESPONSE, HTTP_DATA_REQUEST, HTTP_DATA_RESPONSE.

Example

Skip WAF with partial data collect

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

when HTTP_DATA_REQUEST {
    HTTP:skip_waf()
}
```

Skip WAF with full data collect

```
when HTTP_REQUEST {
    HTTP:collect()
}
when HTTP_DATA_REQUEST {
    HTTP:skip_waf()
}
```

Skip WAF with header

```
when HTTP_REQUEST {
    local method = HTTP:method()
if method == "POST" then
    HTTP:skip_waf()
        debug("Bypassing WAF for POST method: %s\n", uri)
    end
}
```

HTTP 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.

Syntax

```
HTTP:headers()
```

Arguments

N/A

Events

Applicable in HTTP_REQUEST and HTTP_RESPONSE, HTTP_DATA_REQUEST and HTTP_DATA_RESPONSE

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.

Syntax

```
HTTP:header("header-name")
```

Arguments

Name	Description
header-name	String type header name

Events

Applicable in HTTP_REQUEST and HTTP_RESPONSE events.

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”.

Syntax

```
HTTP:cookies()
```

Arguments

N/A

Events

Applicable in HTTP_REQUEST, HTTP_RESPONSE, HTTP_DATA_REQUEST and HTTP_DATA_RESPONSE

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.

Syntax

```
HTTP:cookie(cookie-name)
```

Arguments

Name	Description
cookie-name	String type cookie name

Events

Applicable in HTTP_REQUEST, HTTP_RESPONSE, HTTP_DATA_REQUEST and HTTP_DATA_RESPONSE

Example

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

HTTP:args()

Fetch all arguments of HTTP query.

Syntax

```
HTTP:args()
```

Arguments

N/A

Events

Applicable in HTTP_REQUEST, HTTP_RESPONSE and HTTP_DATA_REQUEST

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.

Syntax

HTTP:arg(arg-name)

Arguments

Name	Description
arg-name	String type argument name

Events

Applicable in HTTP_REQUEST, HTTP_RESPONSE and HTTP_DATA_REQUEST

Example

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

HTTP:host()

Return the string of HTTP request host.

Syntax

```
HTTP:host()
```

Arguments

N/A

Events

Applicable in HTTP_REQUEST and HTTP_RESPONSE, HTTP_DATA_REQUEST and HTTP_DATA_RESPONSE

Example

```
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.

Syntax

```
HTTP:url()
```

Arguments

N/A

Events

Applicable in HTTP_REQUEST, HTTP_RESPOSNE, HTTP_DATA_REQUEST and HTTP_DATA_RESPONSE

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.

Syntax

```
HTTP:path()
```

Arguments

N/A

Events

Applicable in HTTP_REQUEST, HTTP_RESPOSNE, HTTP_DATA_REQUEST and HTTP_DATA_RESPONSE

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.

Syntax

```
HTTP:method()
```

Arguments

N/A

Events

Applicable in HTTP_REQUEST, HTTP_RESPOSNE, HTTP_DATA_REQUEST and HTTP_DATA_RESPONSE

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:status()

Return two strings including HTTP response status code and reason.

`code, reason = HTTP:status()`

Syntax

```
HTTP:status()
```

Arguments

N/A

Events

Applicable in HTTP_RESPONSE and HTTP_DATA_RESPONSE

Example

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

Output: `code = 200, reason = OK`

HTTP Header manipulate

HTTP:set_path(“new-path”)

Change the path in HTTP request header.

Return true for success and false for failure.

Syntax

```
HTTP:set_path(path)
```

Arguments

Name	Description
path	String type path name.

Events

Applicable in HTTP_REQUEST and HTTP_DATA_REQUEST

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.

Syntax

```
HTTP:set_query(query)
```

Arguments

Name	Description
query	String type query name.

Events

Applicable in HTTP_REQUEST and HTTP_DATA_REQUEST

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.

Syntax

```
HTTP:set_url("new-url")
```

Arguments

Name	Description
query	String type query name.

Events

Applicable in HTTP_REQUEST and HTTP_DATA_REQUEST

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.

Syntax

```
HTTP:set_method(method)
```

Arguments

Name	Description
method	String type http method

Events

Applicable in HTTP_REQUEST and HTTP_DATA_REQUEST

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.

Syntax

```
HTTP:set_status(status_code) \ HTTP:set_status(status_code, reason)
```

Arguments

Name	Description
status_code	Integer status code
reason	Optional, string type reason.

Events

Applicable in HTTP_RESPONSE and HTTP_DATA_RESPONSE

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.

Syntax

```
HTTP:add_header(header_name, header_value)
```

Arguments

Name	Description
header_name	String type header_name
header_value	String type header_value

Events

Applicable in HTTP_REQUEST and HTTP_RESPONSE

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.

Syntax

```
HTTP:del_header(header_name)
```

Arguments

Name	Description
header_name	String type header_name

Events

Applicable in HTTP_REQUEST and HTTP_RESPONSE events.

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.

Syntax

```
HTTP:set_header(header_name, header_value_array)
```

Arguments

Name	Description
header_name	String type header_name
header_value_array	Lua array

Events

Applicable in HTTP_REQUEST and HTTP_RESPONSE events.

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.

Syntax

```
HTTP:replace_header(header_name, regex, replace)
```

Arguments

Name	Description
header_name	String type header_name
regex	Match what to be replaced.
replace	Content to replace argument with.

Events

Applicable in HTTP_REQUEST and HTTP_RESPONSE events.

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
}
```

HTTP Data

HTTP:collect()/HTTP:collect(size)

Instructs FortiWeb to buffer and make available the HTTP request or response body for inspection in subsequent script events. 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.

Syntax

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

Arguments

Name	Description
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.

Events

Applicable in HTTP_REQUEST, HTTP_RESPONSE.

Examples

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)
}
```

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
}
```

- 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, size)`
  - `debug()` logging

HTTP:body(offset, size)

The HTTP body will be returned.

Syntax

HTTP:body(offset, size)

Arguments

Name	Description
offset	Optional; if offset is missing, it will be set as zero.

Name	Description
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.

Events

Applicable in HTTP_DATA_REQUEST, HTTP_DATA_RESPONSE.

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, offset, size)

Set body at target place, return true/false based on the result

Syntax

```
HTTP:set_body(body, offset, size)
```

Arguments

Name	Description
body	String type HTTP body.
offset	Optional; if offset is missing, it will be set as zero.
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.

Events

Applicable in HTTP_DATA_REQUEST, HTTP_DATA_RESPONSE.

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)
}
```

IP commands

IP commands can be used in HTTP and TCP events.

ip.addr("ip-string")

Generate an IP address class with an IP string.

Syntax

```
ip.addr("ip-string")
```

Arguments

Name	Description
ip-string	A string which specifies the IP address or IP class

Events

Applicable in all events except RULE_INIT and RULE_EXIT.

Example

```
when HTTP_REQUEST {
    local ip_test = ip.addr("1.1.1.1")
}
```

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.

Syntax

```
ip.eq(ip_class_1, "ip-string") / ip.eq(ip_class_1, ip_class_2)
```

Arguments

Name	Description
ip_class_1	IP class
"ip-string" or ip_class_2	A string which specifies the IP address or IP class

Events

Applicable in all events except RULE_INIT and RULE_EXIT.

Example

```
when HTTP_REQUEST {
    local ip_1 = ip.addr("1.1.1.1")
    local ip_2 = ip.addr("1.1.1.2")
    debug("are two ips the same %s", ip.eq(ip_1, ip_2))
}
```

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" }`

Syntax

```
ip.reputation("ip-string") / ip.reputation(ip_class)
```

Arguments

Name	Description
"ip_string" or ip_class	A string which specifies the IP address or IP class

Events

Applicable in all events except RULE_INIT and RULE_EXIT.

Example

```
when HTTP_REQUEST {
    debug("check ip reputation %s", ip.reputation("1.1.1.1"))
}
```

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.

Syntax

```
ip.geo("ip-string") / ip.geo(ip_class)
```

Arguments

Name	Description
"ip_string" or ip_class	A string which specifies the IP address or IP class

Events

Applicable in all events except RULE_INIT and RULE_EXIT.

Example

```
when HTTP_REQUEST {
    debug("geo of ip %s", ip.geo("1.1.1.1"))
}
```

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.

Syntax

```
ip.geo_code("ip-string") / ip.geo_code(ip_class)
```

Arguments

Name	Description
"ip_string" or ip_class	A string which specifies the IP address or IP class

Events

Applicable in all events except RULE_INIT and RULE_EXIT.

Example

```
when HTTP_REQUEST {
    debug("geo code of ip %s", ip.geo_code ("1.1.1.1"))
}
```

IP:local_addr()

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

Syntax

```
IP:local_addr()
```

Arguments

N/A

Events

Applicable in all events except RULE_INIT and RULE_EXIT.

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.

Syntax

```
IP:remote_addr()
```

Arguments

N/A

Events

Applicable in all events except RULE_INIT and RULE_EXIT.

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.

Syntax

```
IP:client_addr()z
```

Arguments

N/A

Events

Applicable in all events except RULE_INIT and RULE_EXIT.

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.

Syntax

```
IP:server_addr()
```

Arguments

N/A

Events

Applicable in all events, except RULE_INIT and RULE_EXIT.

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.

Syntax

```
IP:version()
```

Arguments

N/A

Events

Applicable in all events, except RULE_INIT and RULE_EXIT.

Example

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

tostring(ip_class)

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

Syntax

```
tostring(ip_class)
```

Arguments

Name	Description
ip_class	IP class

Events

Applicable in all events, except RULE_INIT and RULE_EXIT.

Example

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

Global Key-Value Table

Fortiweb LUA supports a feature (Key-Value Table) that supports different transactions within same server policy to share global data. Transactions of different server policies cannot visit other's global data.

SVRPOLICY:shared_table_create(table)

The `share_table_create` function is used to create a shared table if one with the specified name does not already exist. Each operation on a shared table must specify a `table_name`. The optional parameters include `entry_size` and `memory_limit`. If a table with the given `table_name` already exists, the function will return true. Typically, the first thread that calls this function will create the table. Returns Boolean true if the table is successfully created or has already been created. Otherwise, it returns false.

Syntax

```
SVRPOLICY:shared_table_create(table)
```

Arguments

Name	Description
table	A LUA table including: • **table_name** Required, length 1-127 characters • **entry_count** Optional, default 4096, 1-FWB_MAX_WEB_CACHE_HASH_SIZE • **memory_limit** Optional, default 1024*1024*2(2MB), 1-1024*1024*8(8MB)

Events

Applicable in the RUIT_INIT event.

Example

```
local table_name = "test_table"

when RULE_INIT {
local table_1 = {
    table_name = table_name,
    entry_count = 3,
    memory_limit = 1024 * 1024
}
local success = SVRPOLICY:shared_table_create(table_1)
debug("create table result %s", success)
}
```

SVRPOLICY:shared_table_destroy(table_name)

The `shared_table_destroy` function is used to destroy the specified shared table and all the entries associated. Returns Boolean true if the destroy action is successful. Otherwise, it returns false.

Syntax

```
SVRPOLICY:shared_table_destroy(table_name)
```

Arguments

Name	Description
table_name	String type, length 1-127 characters

Events

Applicable in the RULE_EXIT event.

Example

```
local table_name = "test_table"

when RULE_EXIT {
local success = SVRPOLICY:shared_table_destroy(table_name)
debug("destroy table result %s", success)
}
```

SVRPOLICY:shared_table_insert(table_name, key, value, expire_seconds)

The `shared_table_insert` function is used to insert a pair of <key, value> as the entry into the shared table. Returns true if successful, otherwise return false.

Syntax

```
SVRPOLICY:shared_table_insert(table_name, key, value, expire_seconds)
```

Arguments

Name	Description
table_name	String type, length 1-127 characters.
key	Required, maximum length 255 characters.
value	Required, can be a lua string, number or table.
expire_seconds	Optional, integer type, 10-86400.

Events

Applicable in all events.

Example

```
local table_name = "test_table"

when HTTP_REQUEST {
local user = "Alice"
local user_info = {
    id = 456,
    name = "Alice Tian",
    roles = {"admin", "editor"}
}
local success = SVRPOLICY:shared_table_insert(table_name, user, user_info, 1200)
debug("insert key-value result %s", success)
}
```

SVRPOLICY:shared_table_lookup(table_name, key)

The `shared_table_lookup` function is used to look up whether a key exists in the shared table. Return the stored value if successful, otherwise, return nil.

Syntax

```
SVRPOLICY:shared_table_lookup(table_name, key)
```

Arguments

Name	Description
table_name	String type, length 1-127 characters.
key	Required, maximum length 255 characters.

Events

Applicable in all events.

Example

```
local table_name = "test_table"

when HTTP_REQUEST {
local key = "Alice"
local value = SVRPOLICY:shared_table_lookup(table_name, key)
if value then
    debug("Found key %s value = %s \n", key, value)
else
    debug("cannot find target node")
end
}
```

SVRPOLICY:shared_table_delete(table_name, key)

The `shared_table_delete` function is used to delete an entry specified by a key from the shared table. If the key does not exist or the table with `table_name` does not exist, the function will do nothing. Returns **True** if successful, otherwise, returns **False**.

Syntax

```
SVRPOLICY:shared_table_delete(table_name, key)
```

Arguments

Name	Description
table_name	String type, length 1-127 characters.
key	Required, maximum length 255 characters.

Events

Applicable in all events.

Example

```
local table_name = "test_table"

when HTTP_REQUEST {
local key = "Alice"
local success = SVRPOLICY:shared_table_delete(table_name, key)
if success then
    debug("key = %s successfully deleted from table %s\n", key, table_name)
else
    debug("Failed to delete key = %s", key)
end
}
```

SVRPOLICY:shared_table_dump(table_name, index, count)

The `shared_table_dump` function is used to print the current contents of the shared table for debugging purposes. Returns a pair table, which can be traversed with for [k, v]. All keys and values will be converted into strings. Returns NIL if there is any error or the table is empty.

Syntax

```
SVRPOLICY:shared_table_dump(table_name, index, count)
```

Arguments

Name	Description
table_name	String type, length 1-127 characters.
index	Optional. Default value: 1. Range: (1 - current entry count of the table).
count	Optional. Default value: current entry count of the table. Range: (1-current entry count of the table).

Events

Applicable in all events.

Example

```
local table_name = "test_table"

when HTTP_REQUEST {
local dump_result = SVRPOLICY:shared_table_dump(table_name)
if dump_result then
    for k,v in pairs(dump_result) do
        debug("====>> Key = %s Value %s\n", k, v)
    end
else
    debug("Failed to dump of table %s", table_name)
end
}
```

SVRPOLICY:shared_table_entry_count(table_name)

The `shared_table_count` function is used to retrieve the current count of entries in a shared table. Return the entries count, or returns -1 if the table does not exist.

Syntax

```
SVRPOLICY:shared_table_entry_count(table_name)
```

Arguments

Name	Description
table_name	String type, length 1-127 characters.

Events

Applicable in all events.

Example

```
local table_name = "test_table"

when HTTP_REQUEST {
local count_result = SVRPOLICY:shared_table_entry_count(table_name)
if count_result then
        debug(" there are %d entry \n", count_result)
else
    debug("Failed to count or no such table found")
end
}
```

Use Cases

HTTP to HTTPS redirection

```
-- HTTP to HTTPS redirection for default HTTP and HTTPS ports
when HTTP_REQUEST {
       if not HTTP:is_https() then
              HTTP:redirect("https://%s%s", HTTP:header("host")[1], HTTP: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 get commands

```
when HTTP_REQUEST {
    debug("============= Dump HTTP request header =============\n")
    debug("host: %s, path: %s, url: %s, method: %s, version: %s, content type: %s\n",
           HTTP:host(), HTTP:path(), HTTP:url(), HTTP:method(), HTTP:version(),
HTTP:content_type())
    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
    for k, v in pairs(HTTP:cookies()) do
        debug("Cookie: %s = %s\n", k, v)
    end
    for k, v in pairs(HTTP:args()) do
        debug("ARGS: %s = %s\n", k, v)
    end
    debug("========== Dump HTTP request header done ==========\n")
}

when HTTP_RESPONSE {
    debug("============= Dump HTTP response header =============\n")
    debug("version:%s, content-type: %s\n", HTTP:version(), HTTP:content_type())
    debug("status code: %s reason: %s\n", HTTP:status())
    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
    for k, v in pairs(HTTP:cookies()) do
        debug("Cookie: %s = %s\n", k, v)
    end
    debug("========== Dump HTTP response header done ==========\n")
}
```

IP_PKG example

```
when CLIENT_ACCEPTED {
    local client_ip = IP:client_addr()
    local r = ip.reputation(client_ip)
    debug("Client IP: %s, reputation: <%s>, GEO country: <%s>, GEO country code: %s\n",
            client_ip,
            #r > 0 and table.concat(r, ', ') or "No Found",
            ip.geo(client_ip) or "unknown",
            ip.geo_code(client_ip) or "unknown")
}
```

TCPIP commands

```
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

when CLIENT_ACCEPTED {
    print_ips("CLIENT_ACCEPTED", TCP, IP)
}

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

when HTTP_RESPONSE {
    print_ips("HTTP_RESPONSE", TCP, IP)
}

when SERVER_CONNECTED {
    print_ips("SERVER_CONNECTED", TCP, IP)
}

when SERVER_CLOSED {
    print_ips("SERVER_CLOSED", TCP, IP)
}

when CLIENT_CLOSED {
    print_ips("CLIENT_CLOSED", TCP, IP)
}
```

Content routing by URL

```
-- The policy should have four content routing configuration: cr, cr1, cr2, cr3
-- Don't need set any rule in the content routing, all rules will be ignored
-- after calling LB:routing("content-routing-name")
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
}
```

Full scan for XFF

```
-- This script will full scan the XFF headers, if it found any IP
-- in IP reputation database, it will close the HTTP connection

-- 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 coun
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
}
```

Persist by cookie

```
-- Example of LB persistence
-- the type of persistence of the server pool must be "scripting"
-- LB:persist(string, timeout)
--   string: persist on this string, can use any string
--   timeout: persist on this timeout, unit is second,
--                       if doesn't exist, use the default value in the persistence configuration

cookie_name = "anyone"

when HTTP_REQUEST {
       local value = HTTP:cookie(cookie_name)
       if value then
               -- persist on value and the timeout is default
               -- will record the persistence when doesn't find the persistence
           LB:persist(value)
       end
}

when HTTP_RESPONSE {
       local value = HTTP:cookie(cookie_name)
       if value then
               -- if server respone has this cookie
               -- record the persistence to the persistence table
               LB:persist(value)
       end
}
```

HTTP rewrite headers

```
-- Example of rewriting HTTP headers
-- GET /rewrite_request to rewrite the request with arguments
-- GET /rewrite_response to rewrite the response with arguments

function rewrite_request(HTTP, args)
    local v

    -- rewrite method
    v = args["method"]
    if v then HTTP:set_method(v) end

    -- rewrite path
    v = args["path"]
    if v then HTTP:set_path(v) end
```

```
    -- add/del/set header
    HTTP:del_header("cookie") -- remove header "cookie"
    HTTP:add_header("Accept", "*/*") -- add a new header line
    HTTP:set_header("Test", { "line1", "line2", "line3" }) -- add new header
end

function rewrite_response(HTTP, args)
    local v

    -- rewrite status, includes code and reason
    v = args["code"]
    if v then v = v:match("%d+") end -- get first number from code string
    if v then
        local code = tonumber(v) -- convert code string to number
        local reason = args["reason"]

        if reason then
            HTTP:set_status(code, reason) -- if reason exists, set code and reason
        else
            HTTP:set_status(code) -- set code and use default reason
        end
    end
end
when HTTP_REQUEST {
    local path = HTTP:path()

    if path == "/rewrite_request" then
        rewrite_request(HTTP, HTTP:args())
    elseif path == "/rewrite_response" then
        -- use the HTTP transaction private data to pass data to response
        -- Do not use the global variable, it is shared with all TCP connections and HTTP
transactions
        local t = HTTP:priv() -- get the HTTP transaction private table, it is an empty
table on first read
        t["rewrite_response"] = true -- set rewrite_response to true in the HTTP
transaction private table
        t["args"] = HTTP:args() -- store request arguments to the private table
    end

    HTTP:set_query("test=1")
}

when HTTP_RESPONSE {
    local t = HTTP:priv()

    if t["rewrite_response"] then
        return rewrite_response(HTTP, t["args"])
    end
}
```

HTTP custom reply

```
-- Example of HTTP custom reply

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
}
```

SSL commands

```
-- Example of SSL
-- SET SNI when the fwb is about to send client hello to pservers based on the host name of
```

```
client2FWB request


-- the variable to record the host of client2fortiweb request
local host_name

-- set the host name
when SERVERSSL_CLIENTHELLO_SEND {
    local name = host_name
    if name then
        debug("set Server Name Indication(SNI) in ClientHello = %s\n", name)
        SSL:set_sni(name)
    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



when CLIENTSSL_HANDSHAKE {
    -- get server name extension.
    local svr_name = SSL:sni()
    if svr_name then
        debug("client handshake sni: %s\n", svr_name)
    end

    if svr_name == "www.xyz.com" then
        SSL:close()
        debug("client handshake sni: %s, close the connection\n", svr_name)
    end

    -- get ssl_version
    local ssl_version = SSL:version()
    debug("client ssl version : %s\n", ssl_version)


    -- get alpn_protocol
    local alpn_protocol = SSL:alpn()
    if alpn_protocol then
        debug("alpn_protocol in client handshake =  %s\n", alpn_protocol)
```

```
    end


    -- get cipher
    local cipher = SSL:cipher()
    if cipher then
        debug("cipher in client handshake =%s\n", cipher)
    end

    -- get client ceritificate information
    if SSL:client_cert_verify() then
        debug("client cert verify enabled\n")
        local cert_cnt = SSL:cert_count()
        debug("number of certificates %d\n", cert_cnt)
        -- get the client leaf cert information
        if cert_cnt >= 1 then
            for i = 0, cert_cnt-1 do
              -- code to be executed
              debug("cert index %d\n", i)
              local cert_table = SSL:get_peer_cert_by_idx(i)
              print_table(cert_table, 0)
            end
        end
        debug("verify result: %d\n", SSL: verify_result())
    end
}

when HTTP_REQUEST {
    local host = HTTP:header("host")[1]
    if host then
        host_name = host
        debug("when http request, set the global host_name = %s\n", host)
    end
}


when SERVERSSL_HANDSHAKE {

    debug("when server ssl handshake \n")
    -- get server name extension.
    local svr_name = SSL:sni()
    if svr_name then
        debug(" handshake sni: %s\n", svr_name)
    end

    if svr_name == "www.abc.com" then
        SSL:close()
        debug("client handshake sni: %s, close the connection\n", svr_name)
    end

    -- get ssl_version
    local ssl_version = SSL:version()
```

```
    debug(" ssl version : %s\n", ssl_version)


    -- get alpn_protocol
    local alpn_protocol = SSL:alpn()
    if alpn_protocol then
        debug("alpn_protocol in handshake =  %s\n", alpn_protocol)
    end

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

URL certificate verify by SSL renegotiation

```
-- 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, it matches a
specific HTTP content routing policy,
-- directing traffic 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
}
```

Utility functions demo

```
-- This is a demo for utilities functions that are available in all types of events.
-- For more details, please refer to FortiWeb Administration Guide and Script Reference
Guide.


-- 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 {

    -- Generates a random number
    -- returns an integer value between 0 and RAND_MAX(2^31-1)
    local rand_num = rand()
    debug("rand_num=%d\n",rand_num)

    -- time(), return the current time as an integer, in Unix time format
    local now = time()
```

```
    debug("time now = %d\n", now)

    -- time_ms(), return the current time in million seconds, in Unix time format
    local now_ms = time_ms()
    debug("time now in million seconds = %d\n", now_ms)

    -- ctime(), return the current time as a string, For instance Thu Apr 15 09:01:46 2024
CST +0800
    local now_str = ctime()
    debug("time now in string format:  %s\n", now_str)

    -- md5(input_msg), return the Calculate the MD5 of a string input and return the result
in string representation
    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), Calculate the hex representation of the MD5 of a string, and
return the result in string representation
    local md5_encrypted_hex = md5_hex_str("123")
    debug("encrypted md5 of string 123 in hex representation is:  %s\n", md5_encrypted_hex)

    -- Calculates the SHA1 of a string input, and return the result in string
representation
    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))

    -- Calculates the hex representation of SHA1 of a string input, and return the result
in string representation
    local sha1_123_hex = sha1_hex_str("123")
    debug("encrypted sha1 of string 123 in hex representation is:  %s\n", sha1_123_hex)

    -- Calculates the SHA256 of a string input, and return the result in string
representation
    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))

    -- Calculates the hex representation of SHA1 of a string input, and return the result
in string representation
    local sha256_123_hex = sha256_hex_str("123")
    debug("encrypted sha256 of string 123 in hex representation is:  %s\n", sha256_123_hex)

    -- Calculates the SHA512 of a string input, and return the result in string
representation
    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))

    -- Calculates the hex representation of SHA1 of a string input, and return the result
in string representation
```

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

    -- Encodes a string input in base64 and outputs the results in string format.
    local b64_msg = base64_enc("https://www.base64encode.org/")
    debug("base64 encoded message is:  %s\n", b64_msg)

    -- Decodes a base64 encoded string input and outputs the results in string format.
    local b64_dec_msg = base64_dec(b64_msg)
    debug("base64 decoded message is:  %s\n", b64_dec_msg)

    -- Encodes a string input in base32 and outputs the results in string format.
    local b32_msg = base32_enc("https://www.base64encode.org/")
    debug("base32 encoded message is:  %s\n", b32_msg)

    -- Decodes a base32 encoded string input and outputs the results in string format.
    local b32_dec_msg = base32_dec(b32_msg)
    debug("base32 decoded message is:  %s\n", b32_dec_msg)

    -- Converts a long integer input into network byte order
    local network_a  = htonl(32)
    debug("htonl of 32 is:  %s\n", network_a)

    -- Converts a short integer input into network byte order
    local network_a_short  = htons(32)
    debug("htons of 32 is:  %s\n", network_a_short)

    -- Remember htonl(ntohl(x)) == x
    -- Converts a long integer input into host byte order
    local host_a = ntohl(network_a)
    debug("ntohl of network_a is:  %s\n", host_a)

    -- Converts a short integer input into host byte order
    local host_a_short = ntohs(network_a_short)
    debug("ntohs of network_a_short is:  %s\n", host_a_short)

    --Convert a string to its hex representation
    local hexit = to_hex("it")
    debug("hexit is:  %s\n", hexit)

    -- Returns the crc32 check value of the string, return value is the crc32 code
    local crc32_code = crc32("123456789")
    debug("CRC 32 code is:  %d\n", crc32_code)

    -- key_gen(pass, salt, iter, key_len);
    -- This function 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).
    local new_key = key_gen("pass", "salt", 32, 32)
    debug("new key is %s\n", bytes2hex(new_key))

    -- Encrypts a string using AES algorithm
    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))

    -- Decrypt a string using AES algorithm
    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 for one-shot digest calculation.
    local evpd = EVP_Digest("MD5", "your data")
    debug("the digest in hex is %s\n", bytes2hex(evpd))

    -- HMAC message authentication code.
    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) Check if the signature is same as the current
digest.
    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

    -- Generate a random number in HEX
    local rand_h = rand_hex(16);
    debug("the random hex number  is %s\n", rand_h);

    -- Generate a random alphabet+number sequence:
    local alphanumber = rand_alphanum(16);
    debug("the alphabet+number sequence  is %s\n", alphanumber);

    -- Generate a random number sequence:
    local randseq = rand_seq(16);
    debug("the random sequence is %s\n", to_hex(randseq));

    -- Encode the target url (Converted the url into a valid ASCII format, will not replace
space by "+" sign)
    local encoded_url = url_encode("https://docs.fortinet.com/product/fortiweb/7.4");
    debug("the encoded url is %s\n", encoded_url);

    -- Decode the encoding-url into its original url
    local decoded_url = url_decode(encoded_url);
    debug("the decoded url is %s\n", decoded_url);
}
```

www.fortinet.com