HTTP Inspect Inspector

HTTP Inspect Inspector Overview

Type	Inspector (service)
Usage	Inspect
Instance Type	Multiton
Other Inspectors Required	`stream_tcp`
Enabled	`true`

Hypertext Transfer Protocol (HTTP) is an application layer protocol that enables the exchange of hypermedia (audio, video, images, and text) between a client and server. HTTP is a stateless protocol that requires reliable message transmission. Communication between a client and a server is in the form of HTTP requests and responses.

An HTTP/1.1 server typically uses port 80 over TCP/IP. The secure version of HTTP (HTTP/TLS or HTTPS) uses port 443. HTTP defines access control and authentication mechanisms in the protocol.

HTTP/2 contains improvements to increase speed and push more information than the client requested, but operates over the same ports and protocols as HTTP/1.1. HTTP/2-specific rules are configured with `service:http2`.

HTTP/3 is connection-less, using the QUIC (Quick UDP Internet Connections) protocol rather than TCP, and can support more active streams with better loss recovery. HTTP/3 uses the same messaging as prior versions of HTTP. HTTP/3-specific rules are configured with `service:http3`.

The HTTP inspector supports all three versions of HTTP in an identical fashion.

The `http_inspect` inspector detects and analyzes the protocol data unit (PDU) of the HTTP message. `http_inspect` receives the TCP payload from the TCP stream and examines the encapsulated HTTP message.

The HTTP inspector can detect the following HTTP message sections:

- Request line
- Status line
- Headers
- Content-Length message body (message body defined by Content-length header)
- Chunked message body
- Previous message body (message body with no Content-Length header
- Trailers

The `http_inspect` inspector detects and normalizes all HTTP header fields and the components of the HTTP URI. The `http_inspect` inspector does not normalize the TCP port.

The `http_inspect` inspector can detect four types of URI:

- Asterisk (`*`): not normalized
- Authority: a URI used with the HTTP CONNECT method
- Origin: a URI that begins with a slash (no scheme or authority present)
- Absolute: a URI that includes a scheme, host, and an absolute path

An HTTP URI can include:

- Scheme (ftp, http, or https)
- Host (domain name of server)
- TCP port
- Path (directory and file)
- Query (request parameters)
- Fragment (part of the file)

You can configure the `http_inspect` inspector to alert on the sections of the HTTP message. For example:

- Specify the amount of bytes to read from the HTTP request or response body
- Enable JavaScript detection and normalization
- Handle various types of file decompression
- Customize the decoding of the HTTP URI

Note The `http_inspect` inspector can partially inspect the stream TCP payload.

Best Practices for Configuring the HTTP Inspect Inspector

Consider the following best practices when configuring the `http_inspect` inspector:

- Set the `request_depth` and `response_depth` parameters if your HTTP traffic includes large video files.
- Use the default settings for the HTTP URI inspection parameters:

```
"utf8": "true"
"plus_to_space": "true"
"percent_u": "true"
"utf8_bare_byte": "true"
"iis_unicode": "true"
"iis_double_decode": "true"
```

HTTP Inspect Inspector Parameters

HTTP service configuration

The `binder` inspector defines the HTTP service configuration. For more information, see the Binder Inspector Overview.

Example:

```
[
    {
        "when": {
             "service": "http",
             "role": any
        },
        "use": {
            "type": "http_inspect"
        }
    }
]
```

request_depth

Specifies the number of bytes to read from the HTTP message request body.

Specify `-1` to place no limit on the number of bytes to inspect. We recommend that you specify the `request_depth` and `response_depth` parameters to limit the amount of HTTP body data to analyze.

To inspect only the HTTP headers, set `request_depth` to `0`.

Type: integer

Valid range: `-1` to `9,007,199,254,740,992 (max53)`

Default Value: `-1`

response_depth

Specifies the number of bytes to read from the HTTP message response body.

Specify `-1` to place no limit on the number of bytes to inspect. We recommend that you specify the `request_depth` and `response_depth` parameters to limit the amount of HTTP body data to analyze.

To inspect only the HTTP headers, set `response_depth` to `0`.

Type: integer

Valid range: `-1` to `9,007,199,254,740,992 (max53)`

Default Value: `-1`

unzip

Specifies whether to decompress `gzip` files and deflate message bodies before inspecting them. When you turn off decompression, the HTTP inspector is unable to process all parts of the HTTP message body. The `http_inspect` inspector can process the HTTP headers.

Type: boolean

Valid values: `true, false`

Default value: `true`

maximum_host_length

Specifies the maximum number of bytes allowed in the `Host` HTTP header value.

Specify `-1` to place no limit on the header value length.

Type: integer

Valid range: `-1` to `9,007,199,254,740,992 (max53)`

Default value: `-1`

maximum_chunk_length

Specifies the maximum number of bytes allowed in an HTTP message body chunk.

Specify `-1` to place no limit on the number of bytes in an HTTP chunk.

Type: integer

Valid range: `-1` to `9,007,199,254,740,992 (max53)`

Default value: `-1`

normalize_utf

Specifies whether to normalize UTF encodings (UTF-8, UTF-7, UTF-16LE, UTF-16BE, UTF-32LE, and UTF-32BE) found in the HTTP response body. The `http_inspect` inspector determines the UTF character encoding from the HTTP `Content-Type` header.

Type: boolean

Valid values: `true, false`

Default value: `true`

decompress_pdf

Specifies whether to decompress the deflate-compatible compressed portions of the `application/pdf` (PDF) files found in the HTTP response body. The `http_inspect` inspector decompresses PDF files with the `/FlateDecode` stream filter.

Type: boolean

Valid values: `true, false`

Default value: `false`

decompress_swf

Specifies whether to decompress `application/vnd.adobe.flash-movie` (SWF) files found in the HTTP response body.

Note You can only decompress the compressed portions of files found in the HTTP GET responses.

Type: boolean

Valid values: `true, false`

Default value: `false`

decompress_vba

Specifies whether to decompress Microsoft Office Visual Basic for Applications macro files found in the HTTP response body.

Type: boolean

Valid values: `true, false`

Default value: `false`

decompress_zip

Specifies whether to decompress `application/zip` (ZIP) files found in the HTTP response body.

Type: boolean

Valid values: `true, false`

Default value: `false`

script_detection

Specifies whether to inspect the JavaScript content after detecting the script end element (<\script>). When `http_inspect` detects the end of a script, it immediately forwards the partially read message body for early detection. Script detection enables Snort to quickly block response messages that may contain malicious JavaScript.

Type: boolean

Valid values: `true, false`

Default value: `false`

normalize_javascript

Specifies whether to use the legacy mechanism to normalize JavaScript in the HTTP response body. This option configures the legacy JavaScript normalizer. The `http_inspect` inspector normalizes obfuscated JavaScript data including the `unescape` and `decodeURI` functions, and the `String.fromCharCode` method. The HTTP inspector normalizes encodings within the unescape, decodeURI, and decodeURIComponent functions: %XX, %uXXXX, XX, and uXXXXi.

The `http_inspect` inspector detects consecutive white spaces and normalizes them into a single space. When `normalize_javascript` is enabled, you can set `max_javascript_whitespaces` to limit the number of consecutive white spaces in the obfuscated JavaScript.

Type: boolean

Valid values: `true, false`

Default value: `false`

js_norm_bytes_depth

Specifies the number of input JavaScript bytes to normalize. This option is specific to the enhanced JavaScript normalizer.

Note If you use the enhanced JavaScript normalizer, the default settings from the Lightweight Security Package (LSP) and Snort 3 are used. JavaScript-specific configurations are blocked from the network analysis policy (NAP) user interface. To override the default settings and customize the normalizer settings, you can modify the `NAPOverride.lua` file located at `/ftd/app_data/Volume/root1/ngfw/var/cisco/deploy`.

The `http_inspect` inspector detects consecutive white spaces and normalizes them into a single space. The inspector keeps track of scripts in different PDUs where the start <script> is in one PDU and the end </script> is in another PDU to normalize the traffic effectively. A new buffer `js_data` was added to the Snort 3 IPS buffer that uses the Just in Time (JIT) approach to detect and normalize JavaScript code where the normalizer is called only when this option is used in the rule.

The `http_inspect` inspector normalizes the function name, variable name, and the label name associated with the JavaScript code. In addition, the inspector normalizes JavaScript code transferred in the form of external script using the application/javascript or similar MIME type. The normalizer performs automatic semicolon insertion where the JavaScript functionality is not altered from its original input from the client side.

The `http_inspect` inspector normalizes obfuscated JavaScript data including the `unescape`, `decodeURI,` and `decodeURIComponent` functions, and the `String.fromCharCode` and `String.fromCodePoint` methods. The HTTP inspector normalizes encodings within the unescape, decodeURI, and decodeURIComponent functions: %XX, %uXXXX, \uXX, \u{XXXX}\xXX, decimal code point, and hexadecimal code point.

The `http_inspect` inspector also normalizes the Javascript plus (+) operator and concatenates strings using the operator.

Specify `-1` to place no limit on the number of JavaScript bytes.

Type: integer

Valid range: `-1` to `9,007,199,254,740,992 (max53)`

Default value: `-1`

js_norm_identifier_depth

Specifies the maximum number of unique JavaScript identifiers to normalize. This option is specific to the enhanced JavaScript normalizer.

Note If you use the enhanced JavaScript normalizer, the default settings from the Lightweight Security Package (LSP) and Snort 3 are used. JavaScript-specific configurations are blocked from the network analysis policy (NAP) user interface. To override the default settings and customize the normalizer settings, you can modify the `NAPOverride.lua` file located at `/ftd/app_data/Volume/root1/ngfw/var/cisco/deploy`.

Type: integer

Valid range: `0` to `65536`

Default value: `65536`

js_norm_max_bracket_depth

Specifies the maximum depth of JavaScript bracket nesting to normalize. This option is specific to the enhanced JavaScript normalizer.

Note If you use the enhanced JavaScript normalizer, the default settings from the Lightweight Security Package (LSP) and Snort 3 are used. JavaScript-specific configurations are blocked from the network analysis policy (NAP) user interface. To override the default settings and customize the normalizer settings, you can modify the `NAPOverride.lua` file located at `/ftd/app_data/Volume/root1/ngfw/var/cisco/deploy`.

Type: integer

Valid range: `1` to `65535`

Default value: `256`

js_norm_max_scope_depth

Specifies the maximum depth of JavaScript scope nesting to normalize. This option is specific to the enhanced JavaScript normalizer.

Note If you use the enhanced JavaScript normalizer, the default settings from the Lightweight Security Package (LSP) and Snort 3 are used. JavaScript-specific configurations are blocked from the network analysis policy (NAP) user interface. To override the default settings and customize the normalizer settings, you can modify the `NAPOverride.lua` file located at `/ftd/app_data/Volume/root1/ngfw/var/cisco/deploy`.

Type: integer

Valid range: `1` to `65535`

Default value: `256`

js_norm_max_tmpl_nest

Specifies the maximum depth of JavaScript template literal nesting to normalize. This option is specific to the enhanced JavaScript normalizer.

Note If you use the enhanced JavaScript normalizer, the default settings from the Lightweight Security Package (LSP) and Snort 3 are used. JavaScript-specific configurations are blocked from the network analysis policy (NAP) user interface. To override the default settings and customize the normalizer settings, you can modify the `NAPOverride.lua` file located at `/ftd/app_data/Volume/root1/ngfw/var/cisco/deploy`.

Type: integer

Valid range: `0` to `255`

Default value: `32`

max_javascript_whitespaces

Specifies the maximum consecutive whitespaces allowed within the JavaScript obfuscated data.

Type: integer

Valid range: `1` to `65535`

Default value: `200`

percent_u

Specifies whether to normalize the `%uNNNN` and `%UNNNN` encodings. The four `N` characters represent a hex-encoded value that correlates to a Microsoft internet information services (IIS) Unicode code point. As legitimate clients rarely use `%u` encodings, we recommend that you normalize the HTTP traffic encoded with `%u` encodings.

Type: boolean

Valid values: `true, false`

Default value: `false`

utf8

Specifies whether to normalize the standard UTF-8 Unicode sequences in the URI. The `http_inspect` inspector can normalize two or three byte UTF-8 characters into a single byte.

Type: boolean

Valid values: `true, false`

Default value: `true`

utf8_bare_byte

Specifies whether to normalize UTF-8 characters which include bytes that are not URL or percent encoded. We recommend that you enable the `utf8_bare_byte` parameter.

Type: boolean

Valid values: `true, false`

Default value: `false`

iis_unicode

Specifies whether to normalize the characters in the HTTP message with the Unicode code point.

Note We recommend that you enable the `iis_unicode` parameter. Unicode is commonly seen in attacks and evasion attempts.

Type: boolean

Valid values: `true, false`

Default value: `false`

iis_unicode_code_page

Specifies whether to use the code page from the IIS Unicode map file.

Type: integer

Valid range: `1` to `65535`

Default value: `1252`

iis_double_decode

Specifies whether to normalize characters by performing double decoding of URL encoded characters. Decodes IIS double encoded traffic by making two passes through the request URI. We recommend that you enable the `iis_double_decode` parameter. Double encoding is typically found only in attack scenarios.

Type: boolean

Valid values: `true, false`

Default value: `true`

oversize_dir_length

Specifies the maximum number of bytes allowed for the URL directory.

Type: integer

Valid range: `1` to `65535`

Default value: `300`

backslash_to_slash

Specifies whether to replace the backslash (`\`) with forward slash (`/`) in the URIs.

Type: boolean

Valid values: `true, false`

Default value: `true`

plus_to_space

Specifies whether to replace the plus sign (+) with <sp> in the URIs.

Type: boolean

Valid values: `true, false`

Default value: `true`

simplify_path

Specifies whether to reduce the URI directory path to the simplest form. A URI directory path that includes extra traversals may include: `.`, `..`, and `/`.

Type: boolean

Valid values: `true, false`

Default value: `true`

xff_headers

Specifies the types of `X-Forwarded-For` HTTP header to examine. In the `xff_headers` parameter, list the `X-Forwarded-For` headers from highest to lowest preference.

You can define custom `X-Forwarded-For` type headers. The HTTP header, which carries the original client IP address, can have a vendor-specific header name. In this scenario, the `xff_headers` parameter provides a way to introduce custom headers to the HTTP inspector.

The `xff_headers` default value is `x-forwarded-for true-client-ip`, two commonly known headers. If both default headers are present in the stream, `x-forwarded-for` is preferred over `true-client-ip`. When specifying multiple `X-Forwarded-For` HTTP headers, delimit the header names with a space.

Type: string

Valid values: `x-forwarded-for`, `true-client-ip`

Default value: `x-forwarded-for true-client-ip`

HTTP Inspect Inspector Rules

Enable the `http_inspect` inspector rules to generate events and, in an inline deployment, drop offending packets.

Table 1: HTTP Inspect Inspector Rules

GID:SID	Rule Message
119:1	URI has percent-encoding of an unreserved character
119:2	URI is percent encoded and the result is percent encoded again
119:3	URI has non-standard %u-style Unicode encoding
119:4	URI has Unicode encodings containing bytes that were not percent-encoded
119:6	URI has two-byte or three-byte UTF-8 encoding

GID:SID	Rule Message
119:7	URI has unicode map code point encoding
119:8	URI path contains consecutive slash characters
119:9	backslash character appears in the path portion of a URI
119:10	URI path contains /./ pattern repeating the current directory
119:11	URI path contains /../ pattern moving up a directory
119:12	Tab character in HTTP start line
119:13	HTTP start line or header line terminated by LF without a CR
119:14	Normalized URI includes character from bad_characters list
119:15	URI path contains a segment that is longer than the oversize_dir_length parametery
119:16	chunk length exceeds configured maximum_chunk_length
119:18	URI path includes /../ that goes above the root directory
119:19	HTTP header line exceeds 4096 bytes
119:20	HTTP message has more than 200 header fields
119:21	HTTP message has more than one Content-Length header value
119:24	Host header field appears more than once or has multiple values
119:25	length of HTTP Host header field value exceeds maximum_host_length option
119:28	HTTP POST or PUT request without content-length or chunks
119:31	HTTP request method is not known to Snort
119:32	HTTP request uses primitive HTTP format known as HTTP/0.9
119:33	HTTP request URI has space character that is not percent-encoded
119:34	HTTP connection has more than 100 simultaneous pipelined requests that have not been answered
119:102	invalid status code in HTTP response
119:104	HTTP response has UTF character set that failed to normalize
119:105	HTTP response has UTF-7 character set
119:109	more than one level of JavaScript obfuscation
119:110	consecutive JavaScript whitespaces exceed maximum allowed
119:111	multiple encodings within JavaScript obfuscated data

GID:SID	Rule Message
119:112	SWF file zlib decompression failure
119:113	SWF file LZMA decompression failure
119:114	PDF file deflate decompression failure
119:115	PDF file unsupported compression type
119:116	PDF file with more than one compression applied
119:117	PDF file parse failure
119:201	not HTTP traffic or unrecoverable HTTP protocol error
119:202	chunk length has excessive leading zeros
119:203	white space before or between HTTP messages
119:204	request message without URI
119:205	control character in HTTP response reason phrase
119:206	illegal extra whitespace in start line
119:207	corrupted HTTP version
119:209	format error in HTTP header
119:210	chunk header options present
119:211	URI badly formatted
119:212	unrecognized type of percent encoding in URI
119:213	HTTP chunk misformatted
119:214	white space adjacent to chunk length
119:215	white space within header name
119:216	excessive gzip compression
119:217	gzip decompression failed
119:218	HTTP 0.9 requested followed by another request
119:219	HTTP 0.9 request following a normal request
119:220	message has both Content-Length and Transfer-Encoding
119:221	status code implying no body combined with Transfer-Encoding or nonzero Content-Length
119:222	Transfer-Encoding not ending with chunked

GID:SID	Rule Message
119:223	Transfer-Encoding with encodings before chunked
119:224	misformatted HTTP traffic
119:225	unsupported Content-Encoding used
119:226	unknown Content-Encoding used
119:227	multiple Content-Encodings applied
119:228	server response before client request
119:229	PDF/SWF/ZIP decompression of server response too big
119:230	nonprinting character in HTTP message header name
119:231	bad Content-Length value in HTTP header
119:232	HTTP header line wrapped
119:233	HTTP header line terminated by CR without a LF
119:234	chunk terminated by nonstandard separator
119:235	chunk length terminated by LF without CR
119:236	more than one response with 100 status code
119:237	100 status code not in response to Expect header
119:238	1XX status code other than 100 or 101
119:239	Expect header sent without a message body
119:240	HTTP 1.0 message with Transfer-Encoding header
119:241	Content-Transfer-Encoding used as HTTP header
119:242	illegal field in chunked message trailers
119:243	header field inappropriately appears twice or has two values
119:244	invalid value chunked in Content-Encoding header
119:245	206 response sent to a request without a Range header
119:246	HTTP in version field not all upper case
119:247	white space embedded in critical header value
119:248	gzip compressed data followed by unexpected non-gzip data
119:249	excessive HTTP parameter key repeats
119:253	HTTP CONNECT request with a message body

GID:SID	Rule Message
119:254	HTTP client-to-server traffic after CONNECT request but before CONNECT response
119:255	HTTP CONNECT 2XX response with Content-Length header
119:256	HTTP CONNECT 2XX response with Transfer-Encoding header
119:257	HTTP CONNECT response with 1XX status code
119:258	HTTP CONNECT response before request message completed
119:259	malformed HTTP Content-Disposition filename parameter
119:260	HTTP Content-Length message body was truncated
119:261	HTTP chunked message body was truncated
119:262	HTTP URI scheme longer than 10 characters
119:263	HTTP/1 client requested HTTP/2 upgrade
119:264	HTTP/1 server granted HTTP/2 upgrade
119:265	bad token in JavaScript
119:266	unexpected script opening tag in JavaScript
119:267	unexpected script closing tag in JavaScript
119:268	JavaScript code under the external script tags
119:269	script opening tag in a short form
119:270	max number of unique JavaScript identifiers
119:271	JavaScript bracket nesting is over capacity
119:272	Consecutive commas in HTTP Accept-Encoding header
119:273	missed PDUs during JavaScript normalization
119:274	JavaScript scope nesting is over capacity
119:275	HTTP/1 version other than 1.0 or 1.1e
119:276	HTTP version in start line is 0
119:277	HTTP version in start line is higher than 1

HTTP Inspect Inspector Intrusion Rule Options

http_client_body

Sets the detection cursor to the body of an HTTP request. When an HTTP message does not specify an HTTP header, Snort normalizes `http_client_body` using URI normalization. URI normalization is typically applied to `http_header`.

Syntax: `http_client_body;`

Examples: `http_client_body;`

http_cookie

Sets the detection cursor to the extracted HTTP `Cookie` header field. The `http_cookie` rule option includes the parameters: `http_cookie.request`, `http_cookie.with_header`, `http_cookie.with_body`, and `http_cookie.with_trailer`.

Syntax: http_cookie: <parameter>, <parameter>

Examples: `http_cookie: request;`

http_cookie.request

Matches the HTTP cookie found in the HTTP request message. Use the HTTP request cookie when examining the HTTP response. The `http_cookie.request` parameter is optional.

Syntax: `http_cookie: request;`

Examples: `http_cookie: request;`

http_cookie.with_header

Specifies that the rule can only examine the HTTP message headers. The `http_cookie.with_header` parameter is optional.

Syntax: `http_cookie: with_header;`

Examples: `http_cookie: with_header;`

http_cookie.with_body

Specifies that another part of the rule examines the HTTP message body, not the `http_cookie` rule option. The `http_cookie.with_body` parameter is optional.

Syntax: `http_cookie: with_body;`

Examples: `http_cookie: with_body;`

http_cookie.with_trailer

Specifies that another part of the rule examines the HTTP message trailers, not the `http_cookie` rule option. The `http_cookie.with_trailer` parameter is optional.

Syntax: `http_cookie: with_trailer;`

Examples: `http_cookie: with_trailer;`

http_header

Sets the detection cursor to the normalized HTTP headers. You can specify individual header names using the `field` option.

The `http_header` rule option includes the parameters: `http_header.field`, `http_header.request`, `http_header.with_header`, `http_header.with_body`, and `http_header.with_trailer`.

Syntax: `http_header: field <field_name>,<parameter>, <parameter>`

Examples: `http_header: field Content-Type, with_trailer;`

http_header.field

Matches the specified header name to the normalized HTTP headers. The header name is case insensitive. If you do not specify a header name, the HTTP inspector examines all headers except the HTTP cookie headers (`Cookie` and `Set-Cookie`).

Type: string

Syntax: `http_header: field <field_name>;`

Valid values: An HTTP header name.

Examples: `http_header: field Content-Type;`

http_header.request

Matches the headers found in the HTTP request. Use the HTTP request headers when examining the HTTP response. The `http_header.request` parameter is optional.

Syntax: `http_header: request;`

Examples: `http_header: request;`

http_header.with_header

Specifies that the rule can only examine the HTTP message headers. The `http_header.with_header` parameter is optional.

Syntax: `http_header: with_header;`

Examples: `http_header: with_header;`

http_header.with_body

Specifies that another part of the rule examines the HTTP message body, not the `http_header` rule option. The `http_header.with_body` parameter is optional.

Syntax: `http_header: with_body;`

Examples: `http_header: with_body;`

http_header.with_trailer

Specifies that another part of the rule examines the HTTP message trailers, not the `http_header` rule option. The `http_header.with_trailer` parameter is optional.

Syntax: `http_header: with_trailer;`

Examples: `http_header: with_trailer;`

http_method

Sets the detection cursor to the method of the HTTP request. The common HTTP request method values are `GET`, `POST`, `OPTIONS`, `HEAD`, `DELETE`, `PUT`, `TRACE`, and `CONNECT`.

The `http_method` rule option includes the parameters: `http_method.with_header`, `http_method.with_body`, and `http_method.with_trailer`.

Syntax: `http_method: <parameter>, <parameter>;`

Examples: `http_method; content:"GET";`

http_method.with_header

Specifies that the rule can only examine the HTTP message headers. The `http_method.with_header` parameter is optional.

Syntax: `http_method: with_header;`

Examples: `http_method: with_header;`

http_method.with_body

Specifies that another part of the rule examines the HTTP message body, not the `http_header` rule option. The `http_method.with_body` parameter is optional.

Syntax: `http_method: with_body;`

Examples: `http_method: with_body;`

http_method.with_trailer

Specifies that another part of the rule examines the HTTP message trailers, not the `http_header` rule option. The `http_method.with_trailer` parameter is optional.

Syntax: `http_method: with_trailer;`

Examples: `http_method: with_trailer;`

http_param

Sets the detection cursor to the specified HTTP parameter key. The HTTP parameter key may appear in the query or request body.

The `http_param` rule option includes the parameters: `http_param.param` and `http_method.nocase`.

Syntax: `http_param: <parameter_key>, nocase;`

Examples: `http_param: offset, nocase;`

http_param.param

Matches the specified parameter.

Type: string

Syntax: `http_param: <http_parameter>;`

Valid values: A request query parameter or request body field.

Examples: `http_param: offset;`

http_param.nocase

Match the specified parameter, but do not consider case. The `http_param.nocase` parameter is optional.

Syntax: `http_param: nocase;`

Examples: `http_param: nocase;`

http_raw_body

Sets the detection cursor to the unnormalized request or response message body.

Syntax: `http_raw_body;`

Examples: `http_raw_body;`

http_raw_cookie

Sets the detection cursor to the unnormalized HTTP `Cookie` header. The `http_raw_cookie` rule option includes the parameters: `http_raw_cookie.request`, `http_raw_cookie.with_header`, `http_raw_cookie.with_body`, and `http_raw_cookie.with_trailer`.

Syntax: `http_raw_cookie: <parameter>, <parameter>;`

Examples: `http_raw_cookie: request;`

http_raw_cookie.request

Matches the cookie found in the HTTP request. Use the HTTP request cookie when examining the response message. The `http_raw_cookie.request` parameter is optional.

Syntax: `http_raw_cookie: request;`

Examples: `http_raw_cookie: request;`

http_raw_cookie.with_header

Specifies that the rule can only examine the HTTP message headers. The `http_raw_cookie.with_header` parameter is optional.

Syntax: `http_raw_cookie: with_header;`

Examples: `http_raw_cookie: with_header;`

http_raw_cookie.with_body

Specifies that another part of the rule examines the HTTP message body, not the `http_raw_cookie` rule option. The `http_raw_cookie.with_body` parameter is optional.

Syntax: `http_raw_cookie: with_body;`

Examples: `http_raw_cookie: with_body;`

http_raw_cookie.with_trailer

Specifies that another part of the rule examines the HTTP message trailers, not the `http_raw_cookie` rule option. The `http_raw_cookie.with_trailer` parameter is optional.

Syntax: `http_raw_cookie: with_trailer;`

Examples: `http_raw_cookie: with_trailer;`

http_raw_header

Sets the detection cursor to the unnormalized headers. `http_raw_header` includes all of the unmodified header names and values in the original message.

The `http_raw_header` rule option includes the parameters: `http_raw_header.field`, `http_raw_header.request`, `http_raw_header.with_header`, `http_raw_header.with_body`, and `http_raw_header.with_trailer`.

Syntax: `http_raw_header: field <field_name>, <parameter>, <parameter>;`

Examples: `http_raw_header: field Content-Type, with_trailer;`

http_raw_header.field

Matches the specified header name to the unnormalized HTTP headers. The header name is case insensitive. If you do not specify a header name, the HTTP inspector examines all headers except the HTTP cookie headers (Cookie and Set-Cookie).

Type: string

Syntax: `http_raw_header: field <field_name>`

Valid values: An HTTP header name.

Examples: `http_raw_header: field Content-Type;`

http_raw_header.request

Matches the headers found in the HTTP request message. Use the HTTP request headers when examining the response message. The `http_raw_header.request` parameter is optional.

Syntax: `http_raw_header: request;`

Examples: `http_raw_header: request;`

http_raw_header.with_header

Specifies that the rule can only examine the HTTP message headers. The `http_raw_header.with_header` parameter is optional.

Syntax: `http_raw_header: with_header;`

Examples: `http_raw_header: with_header;`

http_raw_header.with_body

Specifies that another part of the rule examines the HTTP message body, not the `http_raw_header` rule option. The `http_raw_header.with_body` parameter is optional.

Syntax: `http_raw_header: with_body;`

Examples: `http_raw_header: with_body;`

http_raw_header.with_trailer

Specifies that another part of the rule examines the HTTP message trailers, not the `http_raw_header` rule option. The `http_raw_header.with_trailer` parameter is optional.

Syntax: `http_raw_header: with_trailer;`

Examples: `http_raw_header: with_trailer;`

http_raw_request

Sets the detection cursor to the unnormalized request line. To examine a specific part of the first header line, use one of the following rule options: `http_method`, `http_raw_uri`, or `http_version`.

The `http_raw_request` rule option includes the parameters: `http_raw_request.with_header`, `http_raw_request.with_body`, and `http_raw_request.with_trailer`.

Syntax: `http_raw_request: <parameter>, <parameter>;`

Examples: `http_raw_request: with_header;`

http_raw_request.with_header

Specifies that the rule can only examine the HTTP message headers. The `http_raw_request.with_header` parameter is optional.

Syntax: `http_raw_request: with_header;`

Examples: `http_raw_request: with_header;`

http_raw_request.with_body

Specifies that another part of the rule examines the HTTP message body, not the `http_raw_request` rule option. The `http_raw_request.with_body` parameter is optional.

Syntax: `http_raw_request: with_body;`

Examples: `http_raw_request: with_body;`

http_raw_request.with_trailer

Specifies that another part of the rule examines the HTTP message trailers, not the `http_raw_request` rule option. The `http_raw_request.with_trailer` parameter is optional.

Syntax: `http_raw_request: with_trailer;`

Examples: `http_raw_request: with_trailer;`

http_raw_status

Sets the detection cursor to the unnormalized status line. To examine a specific part of the status line, use one of the following rule options: `http_version`, `http_stat_code`, or `http_stat_msg`.

The `http_raw_status` rule option includes the parameters: `http_raw_status.with_body` and `http_raw_status.with_trailer`.

Syntax: `http_raw_status: <parameter>, <parameter>;`

Examples: `http_raw_status: with_body;`

http_raw_status.with_body

Specifies that another part of the rule examines the HTTP message body, not the `http_raw_status` rule option. The `http_raw_status.with_body` parameter is optional.

Syntax: `http_raw_status: with_body;`

Examples: `http_raw_status: with_body;`

http_raw_status.with_trailer

Specifies that another part of the rule examines the HTTP message trailers, not the `http_raw_status` rule option. The `http_raw_status.with_trailer` parameter is optional.

Syntax: `http_raw_status: with_trailer;`

Examples: `http_raw_status: with_trailer;`

http_raw_trailer

Sets the detection cursor to the unnormalized HTTP trailers. Trailers contain information about the message content. The trailers are not available when the client request creates HTTP headers.

`http_raw_trailer` is identical to `http_raw_header`, except that it applies to the end headers. You must create separate rules to inspect the HTTP headers and trailers.

The `http_raw_trailer` rule option includes the parameters: `http_raw_trailer.field`, `http_raw_trailer.request`, `http_raw_trailer.with_header`, `http_raw_trailer.with_body`.

Syntax: `http_raw_trailer: field <field_name>, <parameter>, <parameter>;`

Examples: `http_raw_trailer: field <field_name>, request;`

http_raw_trailer.field

Matches the specified trailer name to the unnormalized HTTP trailers. The trailer name is case insensitive.

Type: string

Syntax: `http_raw_trailer: field <field_name>;`

Valid values: An HTTP trailer name.

Examples: `http_raw_trailer: field trailer-timestamp;`

http_raw_trailer.request

Matches the trailers found in the HTTP request message. Use the HTTP request trailers when examining the response message. The `http_raw_trailer.request` parameter is optional.

Syntax: `http_raw_trailer: request;`

Examples: `http_raw_trailer: request;`

http_raw_trailer.with_header

Specifies that the rule can only examine the HTTP response headers. The `http_raw_trailer.with_header` parameter is optional.

Syntax: `http_raw_trailer: with_header;`

Examples: `http_raw_trailer: with_header;`

http_raw_trailer.with_body

Specifies that another part of the rule examines the HTTP response message body, not the `http_raw_trailer` rule option. The `http_raw_trailer.with_body` parameter is optional.

Syntax: `http_raw_trailer: with_body;`

Examples: `http_raw_trailer: with_body;`

http_raw_uri

Sets the detection cursor to the unnormalized URI.

The `http_raw_uri` rule option includes:

- `http_raw_uri.with_header`
- `http_raw_uri.with_body`
- `http_raw_uri.with_trailer`
- `http_raw_uri.scheme`
- `http_raw_uri.host`
- `http_raw_uri.port`
- `http_raw_uri.path`
- `http_raw_uri.query`
- `http_raw_uri.fragment`

Syntax: `http_raw_uri: <parameter>, <parameter>;`

Examples: `http_raw_uri: with_header, path, query;`

http_raw_uri.with_header

Specifies that the rule can only examine the HTTP message headers. The `http_raw_uri.with_header` parameter is optional.

Syntax: `http_raw_uri: with_header;`

Examples: `http_raw_uri: with_header;`

http_raw_uri.with_body

Specifies that another part of the rule examines the HTTP message body, not the `http_raw_uri` rule option. The `http_raw_uri.with_body` parameter is optional.

Syntax: `http_raw_uri: with_body;`

Examples: `http_raw_uri: with_body;`

http_raw_uri.with_trailer

Specifies that another part of the rule examines the HTTP message trailers, not the `http_raw_uri` rule option. The `http_raw_uri.with_trailer` parameter is optional.

Syntax: `http_raw_uri: with_trailer;`

Examples: `http_raw_uri: with_trailer;`

http_raw_uri.scheme

Matches only against the scheme of the URI. The `http_raw_uri.scheme` parameter is optional.

Syntax: `http_raw_uri: scheme;`

Examples: `http_raw_uri: scheme;`

http_raw_uri.host

Matches only against the host (domain name) of the URI. The `http_raw_uri.host` parameter is optional.

Syntax: `http_raw_uri: host;`

Examples: `http_raw_uri: host;`

http_raw_uri.port

Matches only against the port (TCP port) of the URI. The `http_raw_uri.port` parameter is optional.

Syntax: `http_raw_uri: port;`

Examples: `http_raw_uri: port;`

http_raw_uri.path

Matches only against the path section (directory and file) of the URI. The `http_raw_uri.path` parameter is optional.

Syntax: `http_raw_uri: path;`

Examples: `http_raw_uri: path;`

http_raw_uri.query

Matches only against the query parameters in the URI. The `http_raw_uri.query` parameter is optional.

Syntax: `http_raw_uri: query;`

Examples: `http_raw_uri: query;`

http_raw_uri.fragment

Matches only against the fragment section of the URI. A fragment is part of the file requested, normally found only inside a browser and not transmitted over the network. The `http_raw_uri.fragment` parameter is optional.

Syntax: `http_raw_uri: fragment;`

Examples: `http_raw_uri: fragment;`

http_stat_code

Sets the detection cursor to the HTTP status code. The HTTP status code is a three-digit number ranging between 100 – 599.

The `http_stat_code` rule option includes the parameters: `http_stat_code.with_body` and `http_stat_code.with_trailer`.

Syntax: `http_stat_code: <parameter>, <parameter>;`

Examples: `http_stat_code: with_trailer;`

http_stat_code.with_body

Specifies that another part of the rule examines the HTTP message body, not the `http_stat_code` rule option. The `http_stat_code.with_body` parameter is optional.

Syntax: `http_stat_code: with_body;`

Examples: `http_stat_code: with_body;`

http_stat_code.with_trailer

Specifies that another part of the rule examines the HTTP message trailers, not the `http_stat_code` rule option. The `http_stat_code.with_trailer` parameter is optional.

Syntax: `http_stat_code: with_trailer;`

Examples: `http_stat_code: with_trailer;`

http_stat_msg

Sets the detection cursor to the HTTP status message. The HTTP status message describes the HTTP status code in plain text, for example: `OK`.

The `http_stat_msg` rule option includes the parameters: `http_stat_msg.with_body` and `http_stat_msg.with_trailer`.

Syntax: `http_stat_msg: <parameter>, <parameter>;`

Examples: `http_stat_msg: with_body;`

http_stat_msg.with_body

Specifies that another part of the rule examines the HTTP message body, not the `http_stat_msg` rule option. The `http_stat_msg.with_body` parameter is optional.

Syntax: `http_stat_msg: with_body;`

Examples: `http_stat_msg: with_body;`

http_stat_msg.with_trailer

Specifies that another part of the rule examines the HTTP message trailers, not the `http_stat_msg` rule option. The `http_stat_msg.with_trailer` parameter is optional.

Syntax: `http_stat_msg: with_trailer;`

Examples: `http_stat_msg: with_trailer;`

http_trailer

Sets the detection cursor to the normalized trailers. Trailers contain information about the message content. The trailers are not available when the client request creates HTTP headers.

`http_trailer` is identical to `http_header`, except that it applies to the end headers. You must create separate rules to inspect the HTTP headers and trailers.

The `http_trailer` rule option includes the parameters: `http_trailer.field`, `http_trailer.request`, `http_trailer.with_header`, `http_trailer.with_body`.

Syntax: `http_trailer: field <field_name>, <parameter>, <parameter>;`

Examples: `http_trailer: field trailer-timestamp, with_body;`

http_trailer.field

Matches the specified trailer name to the normalized HTTP trailers. The trailer name is case insensitive.

Type: string

Syntax: `http_trailer: field <field_name>;`

Valid values: An HTTP trailer name.

Examples: `http_trailer: field trailer-timestamp;`

http_trailer.request

Matches the trailers found in the HTTP request message. Use the HTTP request trailers when examining the response message. The `http_trailer.request` parameter is optional.

Syntax: `http_trailer: request;`

Examples: `http_trailer: request;`

http_trailer.with_header

Specifies that another part of the rule examines the HTTP message headers, not the `http_trailer` rule option. The `http_trailer.with_header` parameter is optional.

Syntax: `http_trailer: with_header;`

Examples: `http_trailer: with_header;`

http_trailer.with_body

Specifies that another part of the rule examines the HTTP message body, not the `http_trailer` rule option. The `http_trailer.with_body` parameter is optional.

Syntax: `http_trailer: with_body;`

Examples: `http_trailer: with_body;`

http_true_ip

Sets the detection cursor to the final client IP address. When a client sends a request, the proxy server stores the final client IP address. A client IP address is the last IP address listed in the `X-Forwarded-For`, `True-Client-IP`, or any other custom `X-Forwarded-For` type header. If multiple headers are present, Snort considers the headers defined in `xff_headers`.

The `http_true_ip` rule option includes the parameters: `http_true_ip.with_header`, `http_true_ip.with_body`, and `http_true_ip.with_trailer`.

Syntax: `http_true_ip: <parameter>, <parameter>;`

Examples: `http_true_ip: with_header;`

http_true_ip.with_header

Specifies that the rule can only examine the HTTP message headers. The `http_true_ip.with_header` parameter is optional.

Syntax: `http_true_ip: with_header;`

Examples: `http_true_ip: with_header;`

http_true_ip.with_body

Specifies that another part of the rule examines the HTTP message body, not the `http_true_ip` rule option. The `http_true_ip.with_body` parameter is optional.

Syntax: `http_true_ip: with_body;`

Examples: `http_true_ip: with_body;`

http_true_ip.with_trailer

Specifies that another part of the rule examines the HTTP message trailers, not the `http_true_ip` rule option. The `http_true_ip.with_trailer` parameter is optional.

Syntax: `http_true_ip: with_trailer;`

Examples: `http_true_ip: with_trailer;`

http_uri

Sets the detection cursor to the normalized URI buffer.

- `http_uri.with_header`
- `http_uri.with_body`
- `http_uri.with_trailer`
- `http_uri.scheme`
- `http_uri.host`
- `http_uri.port`
- `http_uri.path`
- `http_uri.query`
- `http_uri.fragment`

Syntax: `http_uri: <parameter>, <parameter>;`

Examples: `http_uri: with_trailer, path, query;`

http_uri.with_header

Specifies that the rule can only examine the HTTP message headers. The `http_uri.with_header` parameter is optional.

Syntax: `http_uri: with_header;`

Examples: `http_uri: with_header;`

http_uri.with_body

Specifies that another part of the rule examines the HTTP message body, not the `http_uri` rule option. The `http_uri.with_body` parameter is optional.

Syntax: `http_uri: with_body;`

Examples: `http_uri: with_body;`

http_uri.with_trailer

Specifies that another part of the rule examines the HTTP message trailers, not the `http_uri` rule option. The `http_uri.with_trailer` parameter is optional.

Syntax: `http_uri: with_trailer;`

Examples: `http_uri: with_trailer;`

http_uri.scheme

Matches only against the scheme of the URI. The `http_uri.scheme` parameter is optional.

Syntax: `http_uri: scheme;`

Examples: `http_uri: scheme;`

http_uri.host

Matches only against the host (domain name) of the URI. The `http_uri.host` parameter is optional.

Syntax: `http_uri: host;`

Examples: `http_uri: host;`

http_uri.port

Matches only against the port (TCP port) of the URI. The `http_uri.port` parameter is optional.

Syntax: `http_uri: port;`

Examples: `http_uri: port;`

http_uri.path

Matches only against the path (directory and file) of the URI. The `http_uri.path` parameter is optional.

Syntax: `http_uri: path;`

Examples: `http_uri: path;`

http_uri.query

Matches only against the query parameters in the URI. The `http_uri.query` parameter is optional.

Syntax: `http_uri: uri;`

Examples: `http_uri: query;`

http_uri.fragment

Matches only against the fragment section of the URI. A fragment is part of the file requested, normally found only inside a browser and not transmitted over the network. The `http_uri.fragment` parameter is optional.

Syntax: `http_uri: fragment;`

Examples: `http_uri: fragment;`

http_version

Sets the detection cursor to the beginning of the HTTP version buffer. `http_version` accepts various HTTP versions. The most commonly found versions are: `HTTP/1.0` and `HTTP/1.1`. The `http_version` rule option includes the parameters: `http_version.request`, `http_version.with_header`, `http_version.with_body`, and `http_version.with_trailer`.

Syntax: `http_version: <parameter>, <parameter>;`

Examples: `http_version; content:"HTTP/1.1";`

http_version.request

Matches the version found in the HTTP request. Use the request version when examining the response message. The `http_version.request` parameter is optional.

Syntax: `http_version: request;`

Examples: `http_version: request;`

http_version.with_header

Specifies that the rule can only examine the HTTP message headers. The `http_version.with_header` parameter is optional.

Syntax: `http_version: with_header;`

Examples: `http_version: with_header;`

http_version.with_body

Specifies that another part of the rule examines the HTTP message body, not the `http_version` rule option. The `http_version.with_body` parameter is optional.

Syntax: `http_version: with_body;`

Examples: `http_version: with_body;`

http_version.with_trailer

Specifies that another part of the rule examines the HTTP message trailers, not the `http_version` rule option. The `http_version.with_trailer` parameter is optional.

Syntax: `http_version: with_trailer;`

Examples: `http_version: with_trailer;`

http_version_match

Specifies a list of HTTP versions to match against the standard HTTP versions. Separate multiple versions with a space character. An HTTP request or status line may contain a version. If the version is present, Snort compares this version with the list specified in `http_version_match`.

If the version doesn't have a format of `[0-9].[0-9]` it is considered malformed. A version in the format of `[0-9].[0-9]` that is not `1.0` or `1.1` is considered `other`.

Type: string

Syntax: `http_version_match: <version_list>`

Valid values: `1.0, 1.1, 2.0, 0.9, other, malformed`

Examples: `http_version_match: "1.0 1.1";`

js_data

Sets the detection cursor to the normalized JavaScript data. This option is specific to the enhanced JavaScript normalizer.

Syntax: `js_data;`

Examples: `js_data;`

vba_data

Sets the detection cursor to the Microsoft Office Visual Basic for Applications macros buffer.

Syntax: `vba_data;`

Examples: `vba_data;`