# HTTPolice Documentation

*Release 0.9.0*

**Vasiliy Faronov**

**Jun 27, 2019**

# Contents

HTTPolice is a validator or "linter" for HTTP requests and responses. It can spot bad header syntax, inappropriate status codes, and other potential problems in your HTTP server or client.

This manual explains all features of HTTPolice in detail. For a hands-on introduction, jump to the *Quickstart*.

CHAPTER 1

# Contents

## 1.1 Quickstart

### 1.1.1 Installation

HTTPolice is a Python package that can be installed with pip (on Python 3.4+):

```
$ pip3 install HTTPolice
```

If you're not familiar with pip, check the manual's *Installation* section.

### 1.1.2 Using HAR files

Let's start with something easy.

If you're running Google Chrome, Firefox, or Microsoft Edge, you can use their developer tools to export HTTP requests and responses as a HAR file, which can then be analyzed by HTTPolice.

For example, in Firefox, press F12 to open the toolbox, and switch to its Network pane. Then, open a simple Web site—I'm going to use h2o.example.net here. All HTTP exchanges made by the browser appear in the Network pane. Right-click inside that pane and select "Save All As HAR".

Then feed this HAR file to HTTPolice:

```
$ httpolice -i har /path/to/file.har
------------ request: GET /
------------ response: 200 OK
E 1000 Syntax error in Server header
E 1013 Multiple Date headers are forbidden
------------ request: GET /search/oktavia-english-search.js
------------ response: 200 OK
E 1000 Syntax error in Server header
E 1013 Multiple Date headers are forbidden
```

(continues on next page)

(continued from previous page)

```
C 1277 Obsolete 'X-' prefix in headers
------------ request: GET /assets/8mbps100msec-nginx195-h2o150.png
C 1276 Accept: */* is as good as image/webp
[...and so on...]
```

### 1.1.3 Better reports

By default, HTTPolice prints a simple text report which may be hard to understand. Use the `-o html` option to make a detailed HTML report instead. You will also need to redirect it to a file:

```
$ httpolice -i har -o html /path/to/file.har >report.html
```

Open `report.html` in your Web browser and enjoy.

### 1.1.4 Using mitmproxy

What if you have an HTTP API that is accessed by special clients? Let's say curl is special enough:

```
$ curl -ksi \
>   -X PATCH https://eve-demo.herokuapp.com/people/5ab12b10662d240004d08b31 \
>   -H 'Content-Type: application/json' \
>   -H 'If-Match: 9f5cdb43a57b04abe342f6a425260f5e6adae359' \
>   -d '{"firstname": "John"}'
HTTP/1.1 200 OK
Connection: keep-alive
Content-Type: application/json
Content-Length: 279
Etag: "9a056970404b9dd20c94c274fbdd6db2239800ac"
Server: Eve/0.7.6 Werkzeug/0.10.4 Python/2.7.4
Date: Sat, 31 Mar 2018 13:23:08 GMT
Via: 1.1 vegur

{"_updated": "Sat, 31 Mar 2018 13:23:08 GMT", "_id": "5ab12b10662d240004d08b31", "_
↪links": {"self": {"title": "person", "href": "people/5ab12b10662d240004d08b31"}}, "_
↪status": "OK", "_etag": "9a056970404b9dd20c94c274fbdd6db2239800ac", "_created":
↪"Tue, 20 Mar 2018 15:38:56 GMT"}
```

How do you get this into HTTPolice?

One way is to use mitmproxy, an advanced tool for inspecting HTTP traffic. Install it in a Python 3.6+ environment with HTTPolice integration:

```
$ pip3 install mitmproxy-HTTPolice
```

See also the instructions for installing mitmproxy via pip3.

---

**Doesn't work on Windows**

Try the Windows Subsystem for Linux, or use Fiddler instead (Fiddler's HAR 1.2 export can get your data into HTTPolice).

---

The following command will start mitmproxy as a reverse proxy in front of your API on port 8080, with HTTPolice integration:

```
$ mitmproxy --mode reverse:https://eve-demo.herokuapp.com \
>   -s "`python3 -m mitmproxy_httpolice`"
```

Now tell your client to talk to port 8080 instead of directly to the API:

```
$ curl -ksi \
>   -X PATCH https://localhost:8080/people/5ab12b10662d240004d08b31 \
>   -H 'Content-Type: application/json' \
>   -H 'If-Match: 9a056970404b9dd20c94c274fbdd6db2239800ac' \
>   -d '{"firstname": "Sam"}'
```

In mitmproxy, you will see that it has intercepted the exchange. Open its details (Enter → Tab → Tab) to see the HTTPolice report on it:

```
Flow Details
2018-03-31 16:26:22 PATCH https://eve-demo.herokuapp.com/people/5ab12b10662d2400
                          04d08b31
                          ← 200 OK application/json 279b 1.10s
          Request              Response              Detail
Metadata:
    HTTPolice: request   E 1000 Syntax error in If-Match header
                         C 1213 Content-Type: application/json is not suitable
                         for PATCH

    HTTPolice: response  C 1214 application/json patch should be rejected

Server Connection:
    Address             eve-demo.herokuapp.com:443
    Resolved Address    174.129.22.75:443
    HTTP Version        HTTP/1.1
Server Certificate:
    Type            RSA, 2048 bits
    SHA1 digest     08:3B:71:72:02:43:6E:CA:ED:42:86:93:BA:7E:DF:81:C4:BC:62:30
    Valid to        2020-06-22 12:00:00
    Valid from      2017-04-19 00:00:00
    Serial          350729763475859927166435998761565448 6
⇩  [1/1]   [reverse:https://eve-demo.her                          [127.0.0.1:8080]
okuapp.com][scripts:1]
```

### 1.1.5 Django integration

Suppose you're building a Web application with Django (1.11+). You probably have a test suite that makes requests to your app and checks responses. You can easily instrument this test suite with HTTPolice and get instant feedback when you break the protocol.

```
$ pip3 install Django-HTTPolice
```

Add the HTTPolice middleware to the top of your middleware list:

```
MIDDLEWARE = [
    'django_httpolice.HTTPoliceMiddleware',
    'django.middleware.common.CommonMiddleware',
    # ...
]
```

Add a couple settings:

```
HTTPOLICE_ENABLE = True
HTTPOLICE_RAISE = 'error'
```

Now let's run the tests and see what's broken:

```
$ python manage.py test
...E
======================================================================
ERROR: test_query_plain (example_app.test.ExampleTestCase)
----------------------------------------------------------------------
Traceback (most recent call last):
  [...]
  File "[...]/django_httpolice/middleware.py", line 92, in process_response
    raise ProtocolError(exchange)
django_httpolice.common.ProtocolError: HTTPolice found problems in this response:
------------ request: GET /api/v1/words/?query=er
------------ response: 200 OK
E 1038 Bad JSON body


----------------------------------------------------------------------
Ran 4 tests in 0.380s

FAILED (errors=1)
```

In this example, the app sent a wrong `Content-Type` header and HTTPolice caught it.

### 1.1.6 More options

There are other ways to get your data into HTTPolice. Check the *full manual*.

## 1.2 Installation

HTTPolice is a Python package that requires Python 3.4+. PyPy is also supported.

Like other Python packages, HTTPolice is installed with pip from PyPI. If you're not familiar with pip, you may need to install it manually or from your OS distribution.

### 1.2.1 On Debian/Ubuntu

Depending on your setup, you may or may not need to install these packages:

```
$ sudo apt-get install python3-pip python3-dev libxml2-dev libxslt1-dev zlib1g-dev
```

Then, to install the HTTPolice command-line tool into `~/.local/bin`:

```
$ pip3 install --user HTTPolice
```

Or, to install it system-wide:

```
$ sudo pip3 install HTTPolice
```

Check that the installation was successful:

```
$ httpolice --version
HTTPolice 0.7.0
```

### 1.2.2 On Fedora

Same as above, but the dependency packages are:

```
$ sudo dnf install python3-pip gcc gcc-c++ redhat-rpm-config python3-devel libxml2-
↪devel libxslt-devel
```

### 1.2.3 On Windows

After installing a recent Python, typically all you need to do is:

```
C:\Users\Vasiliy\...\Python36>Scripts\pip install HTTPolice
```

Check that the installation was successful:

```
C:\Users\Vasiliy\...\Python36>Scripts\httpolice --version
HTTPolice 0.7.0
```

But if `pip install` starts trying (and failing) to compile some libraries, you may need to give it a hand: check the PyPI pages for those libraries (such as lxml or Brotli) to find versions that have suitable pre-built binares (`*-win32.whl`), and install those specific versions first. For example:

```
C:\Users\Vasiliy\...\Python36>Scripts\pip install lxml==3.8.0
C:\Users\Vasiliy\...\Python36>Scripts\pip install Brotli==1.0.4
C:\Users\Vasiliy\...\Python36>Scripts\pip install HTTPolice
```

## 1.3 General concepts

### 1.3.1 Exchanges

HTTPolice takes HTTP *exchanges* (also known as *transactions*) as input. An exchange can consist of one request and one or more responses. Usually there is just one response, but sometimes there are interim (1xx) responses before the main one.

If you only want to check the request, you can omit responses from the exchange.

On the other hand, if you only want to check the *responses*, you should still provide the request (if possible), because responses cannot be properly analyzed without it. If you really have no access to the request, you can omit it, but **many checks will be disabled**.

### 1.3.2 Reports

The output of HTTPolice is a *report* containing *notices*.

Every notice has an ID (such as “1061”) that can be used to *silence* it, and one of three *severities*:

***error*** Something is clearly wrong. For example, a "MUST" requirement of a standard is clearly violated.

Please note that **not all errors may be actual problems**. Sometimes there is a good reason to violate a standard. Sometimes you just don't care. You decide which errors to fix and which to ignore. If you don't want to see an error, you can *silence* it.

***comment*** Something is *possibly* wrong or sub-optimal, but HTTPolice isn't sure. For example, a "SHOULD" requirement of a standard is clearly violated.

***debug*** This just explains why HTTPolice did (or did not do) something. For example, when HTTPolice thinks that a response was served from cache, it will report a debug notice to explain why it thinks so. This may help you understand further cache-related notices for that response.

### 1.3.3 Silencing unwanted notices

You can *silence* notices that you don't want to see. They will disappear from reports and from the *Python API*.

Please note that some notice IDs can stand for a range of problems. For example, most errors in header syntax are reported as notices 1000 or 1158, so if you silence them, you may lose a big chunk of HTTPolice's functionality.

#### Silencing globally

When using the `httpolice` command-line tool, you can use the `-s` option to specify notice IDs to silence:

```
$ httpolice -s 1089 -s 1194 ...
```

Integration methods have similar mechanisms. For example, mitmproxy integration has the `httpolice_silence` option.

#### Silencing locally

You can also silence notices on individual messages by adding the special `HTTPolice-Silence` header to them. Its value is a comma-separated list of notice IDs. For example:

```
HTTP/1.1 405 Method Not Allowed
Content-Length: 0
HTTPolice-Silence: 1089, 1110
```

Requests can also silence notices on responses (but not vice-versa) by adding a `resp` keyword after an ID:

```
GET /index.html HTTP/1.1
User-Agent: Mozilla/5.0
HTTPolice-Silence: 1033 resp, 1031
```

## 1.4 Analyzing raw TCP streams

An obvious way to capture HTTP requests and responses is to dump them with a network sniffer. This only works for cleartext connections (without TLS encryption), but on the other hand, you don't need to change your clients or servers.

HTTPolice can parse HTTP/1.x streams from the ground up. Parsing HTTP/2 is not yet supported.

### 1.4.1 Using tcpflow

You may be familiar with tcpdump, but it won't work: HTTPolice needs the reassembled TCP streams, not individual packets. You can get these streams with a tool called tcpflow:

```
$ mkdir dump

$ cd dump/

$ sudo tcpflow -T'%t-%A-%a-%B-%b-%#' port 80
reportfilename: ./report.xml
reportfilename: ./report.xml
tcpflow: listening on wlp4s0
```

(Note the `-T` option—it is necessary to get the right output.)

tcpflow starts capturing all connections to or from TCP port 80. For example, you can launch a Web browser and go to an 'http:' site. Once you are done, exit the browser, then stop tcpflow with Ctrl+C. (It is important that connections are closed before tcpflow shuts down, otherwise they may be incomplete.)

Now you have one or more pairs of stream files:

```
$ ls
1469847441-054.175.219.008-00080-172.016.000.100-38656-0  report.xml
1469847441-172.016.000.100-38656-054.175.219.008-00080-0
```

Tell HTTPolice to read this directory with the `tcpflow` input format:

```
$ httpolice -i tcpflow .
```

HTTPolice will combine the files into pairs based on their filenames. Due to a limitation in tcpflow, this only works if every combination of source+destination address+port is unique. If there are duplicates, you will get an error.

It's OK if you capture some streams that are not HTTP/1.x. HTTPolice will just complain with notices such as 1279. This means you can run tcpflow without a filter, capturing *all* TCP traffic on a given network interface, and then let HTTPolice sort it out while *silencing* those notices:

```
$ sudo tcpflow -T'%t-%A-%a-%B-%b-%#'

$ httpolice -i tcpflow -o html -s 1279 . >../report.html
```

### 1.4.2 Using tcpick

tcpick is another tool for reassembling TCP streams. It doesn't have the "unique port" limitation of tcpflow, but it has a different problem: sometimes it produces files that are clearly invalid HTTP streams (HTTPolice will fail to parse them with notices like 1009).

Anyway, using it is very similar to using tcpflow:

```
$ mkdir dump

$ cd dump/

$ sudo tcpick -wR -F2 'port 80'
Starting tcpick 0.2.1 at 2016-07-30 06:14 MSK
Timeout for connections is 600
tcpick: listening on wlp4s0
```

(continues on next page)

(continued from previous page)

```
setting filter: "port 80"
[...]
^C
3837 packets captured
30 tcp sessions detected

$ httpolice -i tcpick .
```

(Note the `-wR -F2` options.)

### 1.4.3 Other sniffers

If you use some other tool to capture the TCP streams, use the `streams` input format to pass pairs of files:

```
$ httpolice -i streams requests1.dat responses1.dat requests2.dat ...
```

Or `req-stream` if you only have request streams:

```
$ httpolice -i req-stream requests1.dat requests2.dat ...
```

Or `resp-stream` if you only have response streams (*not recommended*):

```
$ httpolice -i resp-stream responses1.dat responses2.dat ...
```

Note that `resp-stream` may not work at all if any of the requests are HEAD, because responses to HEAD are parsed differently.

### 1.4.4 Combined format

Sometimes you want to compose an HTTP exchange by hand, to test something. To make this easier, there's a special input format that combines the request and response streams into one file:

```
The lines at the beginning are ignored.
You can use them for comments.

======== BEGIN INBOUND STREAM ========
GET / HTTP/1.1
Host: example.com
User-Agent: demo

======== BEGIN OUTBOUND STREAM ========
HTTP/1.1 200 OK
Date: Thu, 31 Dec 2015 18:26:56 GMT
Content-Type: text/plain
Connection: close

Hello world!
```

It must be saved with **CRLF (Windows)** line endings.

Also, for this format, the filename suffix (extension) is important. If it is `.https`, the request URI is assumed to have an `https:` scheme. If it is `.noscheme`, the scheme is unknown. Otherwise, the `http:` scheme is assumed.

Now, tell HTTPolice to use the `combined` format:

```
$ httpolice -i combined exchange1.txt
```

More examples can be found in HTTPolice's test suite.

## 1.5 Analyzing HAR files

HAR is a quasi-standardized JSON format for saving HTTP traffic. It is supported by many HTTP-related tools, including developer consoles of some Web browsers.

HTTPolice can analyze HAR files with the `-i har` option:

```
$ httpolice -i har myfile.har
```

However, please note that HAR support in exporters is **erratic**. HTTPolice tries to do a reasonable job on files exported from major Web browsers and some other HTTP tools, but some information is lost and some checks are skipped to avoid false positives.

If HTTPolice gives unexpected results on your HAR files, feel free to submit an issue (don't forget to attach the files), and I'll see what can be done about it.

## 1.6 Viewing reports

### 1.6.1 Text reports

By default, HTTPolice produces simple plain text reports like this:

```
------------ request: PUT /articles/109226/
E 1000 Malformed If-Match header
C 1093 User-Agent contains no actual product
------------ response: 204 No Content
C 1110 204 response with no Date header
E 1221 Strict-Transport-Security without TLS
------------ request: POST /articles/109226/comments/
...
```

They are intended to be suitable for grep and other Unix-like tools.

### 1.6.2 HTML reports

Use the `-o html` option to enable much more detailed HTML reports. These include explanations for every notice, cross-referenced with the standards, as well as previews of the actual requests and responses.

Please note that these previews **do not represent exactly** what was sent on the wire. For example, in an HTTP/1.x request, a header may have been split into two physical lines, but will be rendered as one line in the report.

#### Options

In the top right hand corner of an HTML report, there's an *options* menu.

The first three options allow you to filter the report on the fly. This is independent from *silencing*: you cannot undo silencing with these options.

***Hide boring exchanges*** Check this to hide all exchanges where no problems were found (only debug notices or none at all).

***Boring notices*** Additional notice IDs or severities that should not be considered problems. For example: `1089 1135 C` (C for "comment").

***Hide boring notices*** Check this if you don't want to see those boring notices at all. They will be hidden even from exchanges that have other problems.

Other options:

***Show remarks*** Check this to show remarks before requests and responses, if any. The nature of these remarks depends on where the data comes from. If you use the `httpolice` command-line tool, remarks will contain the names of the input files and (for *streams input* only) the byte offsets within those files. This can help with debugging.

### 1.6.3 HTML in text

What if you want full details like in HTML reports, but on a textual display? Perhaps you're running HTTPolice on a remote machine via ssh.

You can simply use a text-mode Web browser like w3m:

```
$ httpolice -o html ... | w3m -M -T text/html
```

### 1.6.4 Exit status

When using the `httpolice` command-line tool, there's another channel of information besides the report itself: the command's exit status. If you pass the `--fail-on` option, the exit status will be non-zero if any notices with the given severity (or higher) have been reported. For example:

```
$ httpolice -i combined --fail-on comment test/combined_data/1125_1
------------ request: GET /
------------ response: 304 Not Modified
E 1125 Probably wrong use of status code 304

$ echo $?
1
```

This can be used to take automated action (like failing tests) without parsing the report itself.

## 1.7 Python API

HTTPolice can be used as a Python library: for example, to analyze requests or responses as part of a test suite. It is **not intended** to be used inside live production processes.

### 1.7.1 Example

```
import io
import httpolice

exchanges = [
    httpolice.Exchange(
```

(continues on next page)

(continued from previous page)

```
        httpolice.Request(u'https',
                          u'GET', u'/index.html', u'HTTP/1.1',
                          [(u'Host', b'example.com')],
                          b''),
        [
            httpolice.Response(u'HTTP/1.1', 401, u'Unauthorized',
                               [(u'Content-Type', b'text/plain')],
                               b'No way!'),
        ]
    )
]

bad_exchanges = []

for exch in exchanges:
    exch.silence([1089, 1227])      # Errors we don't care about
    httpolice.check_exchange(exch)
    if any(notice.severity > httpolice.Severity.comment
           for resp in exch.responses       # We only care about responses
           for notice in resp.notices):
        bad_exchanges.append(exch)

if bad_exchanges:
    with io.open('report.html', 'wb') as f:
        httpolice.html_report(bad_exchanges, f)
    print('%d exchanges had problems; report written to file' %
          len(bad_exchanges))
```

### 1.7.2 API reference

**class** httpolice.**Request**(*scheme*, *method*, *target*, *version*, *header_entries*, *body*, *trailer_entries=None*, *remark=None*)

**Parameters**

- **scheme** – The scheme of the request URI, as a Unicode string (usually `u'http'` or `u'https'`), or `None` if unknown (this disables some checks).
- **method** – The request method, as a Unicode string.
- **target** – The request target, as a Unicode string. It must be in one of the four forms defined by RFC 7230. (For HTTP/2, it can be reconstructed from pseudo-headers.)
- **version** – The request's protocol version, as a Unicode string, or `None` if unknown (this disables some checks).

  For requests sent over HTTP/1.x connections, this should be the HTTP version sent in the request line, such as `u'HTTP/1.0'` or `u'HTTP/1.1'`.

  For requests sent over HTTP/2 connections, this should be `u'HTTP/2'`.
- **header_entries** – A list of the request's headers (may be empty). It must **not** include HTTP/2 pseudo-headers.

  Every item of the list must be a `(name, value)` pair.

  `name` must be a Unicode string.

`value` may be a byte string or a Unicode string. If it is Unicode, HTTPolice will assume that it has been decoded from ISO-8859-1 (the historic encoding of HTTP), and will encode it back into ISO-8859-1 before any processing.

- **`body`** – The request's payload body, as a **byte string**, or `None` if unknown (this disables some checks).

  If the request has no payload (like a GET request), this should be the empty string `b''`.

  This must be the payload body as defined by RFC 7230: **after** removing any `Transfer-Encoding` (like `chunked`), but **before** removing any `Content-Encoding` (like `gzip`).

- **`trailer_entries`** – A list of headers from the request's trailer part (as found in chunked coding or HTTP/2), or `None` if there is no trailer part.

  The format is the same as for `header_entries`.

- **`remark`** – If not `None`, this Unicode string will be shown above the request in HTML reports (when the appropriate option is enabled). For example, it can be used to identify the source of the data: `u'from somefile.dat, offset 1337'`.

**`notices`**
A list of *`Complaint`* instances reported on this object.

**`silence`**(*notice_ids*)
Silence unwanted notices on this object.

**Parameters** **`notice_ids`** – An iterable of notice IDs that will be silenced on this object, so they don't appear in *`notices`* or in reports.

**`class`** `httpolice.`**`Response`**(*version*, *status*, *reason*, *header_entries*, *body*, *trailer_entries=None*, *remark=None*)

**Parameters**

- **`version`** – The response's protocol version, as a Unicode string, or `None` if unknown (this disables some checks).

  For responses sent over HTTP/1.x connections, this should be the HTTP version sent in the status line, such as `u'HTTP/1.0'` or `u'HTTP/1.1'`.

  For responses sent over HTTP/2 connections, this should be `u'HTTP/2'`.

- **`status`** – The response's status code, as an integer.

- **`reason`** – The response's reason phrase (such as "OK" or "Not Found"), as a Unicode string, or `None` if unknown (as in HTTP/2).

- **`header_entries`** – A list of the response's headers (may be empty). It must **not** include HTTP/2 pseudo-headers.

  Every item of the list must be a `(name, value)` pair.

  `name` must be a Unicode string.

  `value` may be a byte string or a Unicode string. If it is Unicode, HTTPolice will assume that it has been decoded from ISO-8859-1 (the historic encoding of HTTP), and will encode it back into ISO-8859-1 before any processing.

- **body** – The response's payload body, as a **byte string**, or `None` if unknown (this disables some checks).

  If the response has no payload (like 204 or 304 responses), this should be the empty string `b''`.

  This must be the payload body as defined by RFC 7230: **after** removing any `Transfer-Encoding` (like `chunked`), but **before** removing any `Content-Encoding` (like `gzip`).

- **trailer_entries** – A list of headers from the response's trailer part (as found in chunked coding or HTTP/2), or `None` if there is no trailer part.

  The format is the same as for `header_entries`.

- **remark** – If not `None`, this Unicode string will be shown above this response in HTML reports (when the appropriate option is enabled). For example, it can be used to identify the source of the data: `u'from somefile.dat, offset 1337'`.

**notices**
A list of *Complaint* instances reported on this object.

**silence**(*notice_ids*)
Silence unwanted notices on this object.

**Parameters** **notice_ids** – An iterable of notice IDs that will be silenced on this object, so they don't appear in *notices* or in reports.

**class** httpolice.**Exchange**(*req*, *resps*)

**Parameters**

- **req** – The request, as a *Request* object. If it is not available, you can pass `None`, and the responses will be checked on their own. However, this **disables many checks** which rely on context information from the request.
- **resps** – The responses to `req`, as a list of *Response* objects. Usually this will be a list of 1 element. If you only want to check the request, pass an empty list `[]`.

**request**
The *Request* object passed to the constructor.

**responses**
The list of *Response* objects passed to the constructor.

**silence**(*notice_ids*)
Silence unwanted notices on this object.

**Parameters** **notice_ids** – An iterable of notice IDs that will be silenced on this object, so they don't appear in `notices` or in reports.

httpolice.**check_exchange**(*exch*)
Run all checks on the exchange `exch`, modifying it in place.

**class** httpolice.**Complaint**
A notice as reported in a particular place.

**id**
The notice's ID (an integer).

**severity**
The notice's severity, as a member of the *Severity* enumeration.

**class** httpolice.**Severity**
A notice's severity.

This is a standard Python enumeration with the additional feature that its members are ordered:

```
>>> Severity.comment < Severity.error
True
```

The underlying values of this enumeration are **not** part of the API.

**comment = 1**

**debug = 0**

**error = 2**

httpolice.**text_report**(*exchanges*, *buf*)
Generate a plain-text report with check results.

**Parameters**

- **exchanges** – An iterable of *Exchange* objects. They must be already processed by *check_exchange()*.
- **buf** – The file (or file-like object) to which the report will be written. It must be opened in binary mode (not text).

httpolice.**html_report**(*exchanges*, *buf*)
Generate an HTML report with check results.

**Parameters**

- **exchanges** – An iterable of *Exchange* objects. They must be already processed by *check_exchange()*.

- **buf** – The file (or file-like object) to which the report will be written. It must be opened in binary mode (not text).

### 1.7.3 Integration helpers

Functions that may be useful for integrating with HTTPolice.

`httpolice.helpers.`**`headers_from_cgi`**(*cgi_dict*)
Convert CGI variables into header entries.

**Parameters** **cgi_dict** – A mapping of CGI-like meta-variables, as found in (for example) WSGI's `environ` or `django.http.HttpRequest.META`.

**Returns** A list of header entries, suitable for passing into *`httpolice.Request`*.

`httpolice.helpers.`**`pop_pseudo_headers`**(*entries*)
Remove and return HTTP/2 pseudo-headers from a list of headers.

**Parameters** **entries** – A list of header name-value pairs, as would be passed to *`httpolice.Request`* or *`httpolice.Response`*. It will be modified in-place by removing all names that start with a colon (:).

**Returns** A dictionary of the removed pseudo-headers.

## 1.8 History of changes

### 1.8.1 0.9.0 - 2019-06-27

#### Added

- Basic checks for most of the headers defined by WHATWG Fetch, such as `Access-Control-Allow-Origin`.
- Updated workarounds for HAR files exported from Chrome and Firefox. More checks are now skipped on such files, which means fewer false positives due to missing or mangled data.
- Notice 1282 is now reported on `application/text`.

#### Fixed

- Notice 1276 is now a comment, not an error.
- Notice 1277 is no longer reported on `X-Real-IP`.
- Notice 1029 (`TE` requires `Connection: TE`) is now only reported on HTTP/1.1 requests.

### 1.8.2 0.8.0 - 2019-03-03

- Dropped Python 2 support. If you need it, use the older versions.
- HTTPolice no longer requires six nor singledispatch.
- HTTPolice now pulls in Google's Brotli instead of brotlipy, but this is merely a packaging change; it can work with either.
- Notices 1299 and 1300 are no longer reported on `Alt-Svc`.

### 1.8.3 0.7.0 - 2018-03-31

#### Changed

- Reflecting changes in RFC 8187 and RFC 8259, notices 1253 (bad charset) and 1281 (bad encoding for JSON) are now reported for all encodings other than UTF-8, and notice 1255 (ISO-8859-1 in Content-Disposition) has been removed.

#### Added

- Checks for quoted commas and semicolons that might confuse a naive parser (notices 1299, 1300).
- New checks for Link headers according to RFC 8288 (notices 1307, 1308, 1309).
- Checks for immutable responses (notices 1301, 1302, 1303).
- Early hints are now recognized (due to their idiosyncratic semantics, they avoid many checks that are applied to all other responses).
- Checks for the Accept-Post header (notice 1310).
- Check for no Transfer-Encoding in response to HTTP/1.0 (notice 1306).
- Check for 100 (Continue) before switching protocols (notice 1305).
- Check that the sequence of responses to a request makes sense (notice 1304).
- HAR files exported from Chrome and Insomnia are handled slightly better.

#### Fixed

- Headers like Allow and Accept are now parsed more correctly (RFC Errata 5257).
- `gzip`-encoded payloads are now decompressed more reliably.
- When analyzing TCP streams, HTTPolice now uses a stricter heuristic for detecting HTTP/1.x streams, producing fewer spurious 1006/1009 notices.
- Notice 1291 (Preference-Applied needs Vary) is no longer reported on responses to POST.

### 1.8.4 0.6.0 - 2017-08-02

#### Changed

- Notice 1277 (obsolete 'X-' prefix) is now reported only once per message.
- When parsing TCP streams, HTTPolice no longer attempts to process very long header lines (currently 16K; they will fail with notice 1006/1009) and message bodies (currently 1G; notice 1298).
- Notice 1259 (malformed parameter in Alt-Svc) has been removed: the same problem is now reported as notice 1158.
- The syntax of chunk extensions is no longer checked.

#### Added

- Checks for the Forwarded header (notices 1296, 1297).

#### Fixed

- Fixed a few bugs and design problems that caused HTTPolice to use more time and memory than necessary in various cases (sometimes much more).
- Fixed some Unicode errors under Python 2.
- Notice 1013 is no longer wrongly reported for some headers such as Vary.
- Fixed a crash on some pathological values of 'charset' in Content-Type.

### 1.8.5 0.5.2 - 2017-03-24

- Fixed a few rare crashing bugs found with american fuzzy lop.
- Fixed a couple cosmetic bugs in HTML reports.
- When parsing a message with an unknown transfer coding, HTTPolice now correctly skips any checks on its payload body (such as notice 1038).

### 1.8.6 0.5.1 - 2017-03-15

- Fixed compatibility with httpolice-devtool (when you point it to a local hpoliced instance).

### 1.8.7 0.5.0 - 2017-03-12

#### Added

- When analyzing TCP streams, HTTPolice now reorders exchanges based on the Date header. In other words, messages sent at the same time on different connections are now close to each other in the report.
- Checks for the Prefer mechanism (notices 1285 through 1291).
- The syntax of method and header names and reason phrases is now checked for all messages, not only for those parsed from TCP streams (notices 1292, 1293, 1294).
- Check for method names that are not uppercase (notice 1295).
- The XML-related features removed in 0.4.0 have been restored.
- Check for cacheable 421 (Misdirected Request) responses (notice 1283).
- Check for 202 (Accepted) responses with no body (notice 1284).
- HTML reports have been optimized to load slightly faster in browsers.

#### Changed

- Titles of many notices were changed to make more sense when viewed alone (as in text reports). If you depend on their wording (which you shouldn't), you may need to adjust.

#### Fixed

- Notice 1021 is no longer reported on HTTP/2 requests.

#### Meanwhile

- mitmproxy integration has new features for interactive use.

### 1.8.8 0.4.0 - 2017-01-14

#### Added

- Python 3.6 compatibility.
- Decompression of brotli compressed payloads (`Content-Encoding: br`).
- Checks for JSON charsets (notices 1280 and 1281).
- Checks for some wrong media types, currently `plain/text` and `text/json` (notice 1282).

#### Removed

- The deprecated constants `httpolice.ERROR`, `httpolice.COMMENT`, `httpolice.DEBUG` have been removed. Use `httpolice.Severity` instead.
- When checking XML payloads, HTTPolice no longer takes precautions against denial-of-service attacks, because the defusedxml module does not currently work with Python 3.6. DoS attacks against HTTPolice are considered unlikely and non-critical.
- Notice 1275 ("XML with entity declarations") has been removed for the same reason.

#### Other

- There is now a third-party Chrome extension for HTTPolice.

### 1.8.9 0.3.0 - 2016-08-14

#### Added

- HTTPolice now caches more intermediate values in memory, which makes it significantly faster in many cases.
- HTTPolice now works correctly under PyPy (the 2.7 variant), which, too, can make it faster on large inputs. You will probably need a recent version of PyPy (5.3.1 is OK).
- HTML reports now have an "options" menu to filter exchanges and notices on the fly.
- The `httpolice` command-line tool now has a `--fail-on` option to exit with a non-zero status if any notices with a given severity have been reported.
- Work around various problems in HAR files exported by Firefox and Fiddler.
- HTML reports can now display a remark before every request and response (enabled with the *Show remarks* checkbox in the "options" menu). The `httpolice` command-line tool puts the input filename in this remark. With the Python API, you can put anything there using the `remark` argument to `Request` and `Response` constructors.
- Notices about HTTP/1.x framing errors (such as 1006) now include the input filename as well.
- Check for missing scheme name in authorization headers (notice 1274).
- Check for missing quality values in headers like Accept (notice 1276).

- Check for obsolete 'X-' prefix in experimental headers (notice 1277).
- Notice 1093 recognizes a few more product names as client libraries.

#### Changed

- For the tcpick and tcpflow input modes, you now have to use different options to tcpick/tcpflow (consult the manual).
- Text reports no longer show request/response numbers. If you parse these reports, you may need to adjust.
- Styles in HTML reports have been tweaked to make them more readable.

#### Deprecated

- In the Python API, the constants `httpolice.ERROR`, `httpolice.COMMENT`, `httpolice.DEBUG` have been replaced with a single `httpolice.Severity` enumeration, and will be removed in the next release.

#### Fixed

- The tcpick and tcpflow input modes should now be more reliable, although they still suffer from certain problems.
- CONNECT requests in HAR files are now handled correctly.
- Notices 1053 and 1066 are no longer reported on requests with bodies of length 0.

### 1.8.10 0.2.0 - 2016-05-08

#### Added

- Django integration (as a separate distribution).
- Unwanted notices can now be silenced.
- Checks for OAuth bearer tokens.
- Checks for the Content-Disposition header.
- Checks for RFC 5987 encoded values.
- Checks for alternative services.
- Checks for HTTP/1.1 connection control features prohibited in HTTP/2.
- Stale controls are now recognized.
- Checks for status code 451 (Unavailable For Legal Reasons).

#### Changed

- mitmproxy integration has been moved into a separate distribution.

#### Fixed

- Input files from tcpick are sorted correctly.
- Notice 1108 doesn't crash in non-English locales.
- Notices such as 1038 are not reported on responses to HEAD.

### 1.8.11 0.1.0 - 2016-04-25

- Initial release.

CHAPTER 2

# Supplementary documents

- List of all notices that HTTPolice can output
- Example report produced by HTTPolice

CHAPTER 3

# Integration packages

- mitmproxy integration
- Django integration
- Chrome extension (third-party)

# Python Module Index

## h

# Index