HTTPolice Documentation

Release 0.2.0

Vasiliy Faronov

July 24, 2016

Contents

HTTPolice is a lint for HTTP requests and responses. It checks them for conformance to standards and best practices.

This manual explains all features of HTTPolice in detail. If you want a brief introduction, see the Quickstart.

There is also a list of all notices that HTTPolice can output.

CHAPTER 1

Quickstart

1.1 Installation

HTTPolice is a Python package that can be installed with pip (on Python 2.7 or 3.4+):

```
$ pip install HTTPolice
```

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

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—let's try jshint.com. All HTTP exchanges made by the browser appear in the Network pane. Right-click inside that pane and select "Save All As HAR".

Now that you have the HAR file, you can feed it into HTTPolice:

```
$ httpolice -i har /path/to/file.har
------------ request 6 : GET /ga.js
------------ response 6 : 200 OK
C 1035 Deprecated media type text/javascript
D 1168 Response from cache
------------ request 7 : GET /r/__utm.gif?utmwv=5.6.7&utm...
------------ response 7 : 200 OK
E 1108 Wrong day of week in Expires
C 1162 Pragma: no-cache in a response
```

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.

1.4 Using mitmproxy

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

```
$ curl -ksiX POST https://eve-demo.herokuapp.com/people \
>   -H 'Content-Type: application/json' \
>   -d '{"firstname":"John", "lastname":"Smith"}'
HTTP/1.1 201 CREATED
Connection: keep-alive
Content-Type: application/json
Content-Length: 279
Server: Eve/0.6.1 Werkzeug/0.10.4 Python/2.7.4
Date: Mon, 25 Apr 2016 09:21:32 GMT
Via: 1.1 vegur

{"_links": {"self": {"href": "people/571de19c4fd7bd0003356826", "title": "person"}}, "_etag": "3b1f9
```

How do you get this into HTTPolice?

One way is to use mitmproxy, an advanced tool for intercepting HTTP traffic. You can install it manually, or from your distribution's packages if they are recent enough (0.15 should work).

And you'll need the integration package:

```
$ pip install mitmproxy-HTTPolice
```

Now, we're going to use mitmproxy's command-line tool—mitmdump. The following command will start mitmdump as an HTTP proxy on port 8080 with HTTPolice integration:

```
$ mitmdump -s "`python -m mitmproxy_httpolice` -o html report.html"
```

With mitmdump running, tell curl to use it as a proxy:

```
$ curl -x localhost:8080 \
>   -ksiX POST https://eve-demo.herokuapp.com/people \
>   -H 'Content-Type: application/json' \
>   -d '{"firstname":"John", "lastname":"Williams"}'
```

In the output of mitmdump, you will see that it has intercepted the exchange. Now, when you stop mitmdump (Ctrl+C), HTTPolice will write an HTML report to `report.html`.

1.5 Django integration

Suppose you're building a Web application with Django (1.8+). 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.

```
$ pip install Django-HTTPolice
```

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

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

Add a couple settings:

```
HTTPOLICE_ENABLE = True
HTTPOLICE_RAISE = True
```

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

```
$ python manage.py test
.E.
======================================================================
ERROR: test_get_plain (example_app.test.ExampleTestCase)
----------------------------------------------------------------------
Traceback (most recent call last):
  [...]
  File "[...]/django_httpolice/middleware.py", line 81, in process_response
    raise ProtocolError(exchange)
django_httpolice.common.ProtocolError: HTTPolice found errors in this response:
------------ request 1 : GET /api/v1/?name=Martha&format=plain
C 1070 No User-Agent header
------------ response 1 : 200 OK
E 1038 Bad JSON body


----------------------------------------------------------------------
Ran 3 tests in 0.351s

FAILED (errors=1)
```

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

1.6 More options

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

CHAPTER 2

Installation

HTTPolice is a Python package that requires Python 2.7 or 3.4+. It can be installed like all other Python packages: with pip from PyPI.

If you're not familiar with pip, you may need to install it manually or from your OS distribution. You may also need development files and tools for lxml dependencies.

2.1 On Debian/Ubuntu

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

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

```
$ pip install --user HTTPolice
```

Or, to install it system-wide:

```
$ sudo pip install HTTPolice
```

Check that the installation was successful:

```
$ httpolice --version
HTTPolice 0.1.0
```

2.2 On Windows

After installing Python 2.7, something like this should do the trick:

```
C:\>Python27\Scripts\pip install HTTPolice

C:\>Python27\Scripts\httpolice --version
HTTPolice 0.1.0
```

CHAPTER 3

General concepts

3.1 Exchanges

HTTPolice takes HTTP *exchanges* (also known as *transactions*) as input. Every exchange can consist of 1 request and 1+ responses. Usually there is just 1 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**.

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.

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 notice 1000, so if you silence it, you **lose a big chunk** of HTTPolice's functionality.

3.3.1 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 ...
```

Every integration method has a similar mechanism. For example, mitmproxy integration understands the same `-s` option.

3.3.2 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
```

CHAPTER 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. HTTP/2 is not yet supported.

You may be familiar with tcpdump, but it won't work: HTTPolice needs the raw TCP streams—just the data sent or received. There are two Unix tools to dump TCP streams: tcpick and tcpflow. Unfortunately, both **sometimes produce incorrect files**, so this may not be 100% reliable.

4.1 tcpick

I have had more success with tcpick. Here's how it can be used:

```
$ mkdir dump

$ cd dump/

$ sudo tcpick -wR 'port 80'
Starting tcpick 0.2.1 at 2016-04-13 05:11 MSK
Timeout for connections is 600
tcpick: listening on wlp4s0
setting filter: "port 80"
```

tcpick 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 tcpick with Ctrl+C. (It is important that connections are closed before tcpick shuts down, otherwise they may be incomplete.)

Now you have one or more pairs of files in this directory:

```
$ ls
tcpick_172.16.0.102_185.72.247.137_http.clnt.dat
tcpick_172.16.0.102_185.72.247.137_http.serv.dat
```

Then you tell HTTPolice to use the `tcpick` input format:

```
$ httpolice -i tcpick .
```

4.2 tcpflow

Very similar to tcpick:

```
$ mkdir dump

$ cd dump/

$ sudo tcpflow -T'%t-%#-%A-%B' port 80
tcpflow: listening on wlp4s0
^Ctcpflow: terminating

$ ls
1460513796-0-172.016.000.102-185.072.247.137  alerts.txt
1460513796-0-185.072.247.137-172.016.000.102  report.xml

$ httpolice -i tcpflow .
```

The cryptic `-T` option is necessary to get the right filenames.

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.

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.

CHAPTER 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 major Web browsers is **erratic**. HTTPolice tries to do a reasonable job on files exported from Chrome, Firefox, and Edge, but some information is simply lost.

If HTTPolice fails 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.

CHAPTER 6

mitmproxy integration

mitmproxy is an advanced HTTP debugging tool. It can intercept TLS-encrypted connections by generating certificates on the fly. It supports HTTP/2, it can work as a reverse proxy... Cool stuff.

HTTPolice comes with an inline script for mitmproxy that will check intercepted exchanges and produce a normal HTTPolice report. It also works with mitmproxy's command-line tool mitmdump.

See mitmproxy docs for instructions on how to install it. Ubuntu 16.04 "Xenial Xerus" has a package for mitmproxy 0.15 that should be recent enough for HTTPolice.

You will also need to install the integration package (see Installation):

```
$ pip install mitmproxy-HTTPolice
```

6.1 Usage

To run HTTPolice together with mitmproxy, use a command like this:

```
$ mitmdump -s "`python -m mitmproxy_httpolice` -o html report.html"
```

Note the backticks. Also, you can replace `mitmdump` with `mitmproxy` if you wish.

`-s` is mitmproxy's option that specifies an inline script to run, along with arguments to that script.

`python -m mitmproxy_httpolice` is a sub-command that prints the path to the script file:

```
$ python -m mitmproxy_httpolice
/home/vasiliy/.local/lib/python2.7/site-packages/mitmproxy_httpolice.py
```

`-o html` tells HTTPolice to produce HTML reports (omit it if you want a plain text report). Finally, `report.html` is the name of the output file.

Now, mitmproxy/mitmdump starts up as usual. Every exchange that it intercepts is checked by HTTPolice. When you stop mitmdump (Ctrl+C) or exit mitmproxy, HTTPolice writes an HTML report to `report.html`.

You can use the `-s` option to *silence* unwanted notices, just as with the `httpolice` command-line tool:

```
$ mitmdump -s "`python -m mitmproxy_httpolice` -s 1089 -s 1194 report.txt"
```

CHAPTER 7

Viewing reports

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

```
------------ request 1 : PUT /articles/109226/
E 1000 Malformed If-Match header
C 1093 User-Agent contains no actual product
------------ response 1 : 100 Continue
------------ response 2 : 204 No Content
C 1110 204 response with no Date header
E 1221 Strict-Transport-Security without TLS
------------ request 2 : POST /articles/109226/comments/
...
```

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

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.)

What if you want full details like in HTML reports, but still in plain text? Just use a text-mode Web browser like w3m:

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

CHAPTER 8

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.

8.1 Example

```
import io
import httpolice

exchanges = [
    httpolice.Exchange(
        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, 1194])      # Errors we don't care about
    httpolice.check_exchange(exch)
    if any(notice.severity == httpolice.ERROR
           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))
```

8.2 API reference

class `httpolice.`**`Request`**(*scheme*, *method*, *target*, *version*, *header_entries*, *body*, *trailer_entries=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`.

`notices`

A list of *`Notice`* objects 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*)

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`.

notices
A list of *Notice* objects 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 Notice

id
The notice's ID (an integer).

severity
The notice's severity. This is an opaque value that should only be compared to the constants `httpolice.ERROR`, `httpolice.COMMENT`, and `httpolice.DEBUG`.

`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).

8.3 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.

CHAPTER 9

Django integration

HTTPolice has a package for integrating with Django 1.8+:

```
$ pip install Django-HTTPolice
```

This package provides `django_httpolice.HTTPoliceMiddleware`. Add it to your `MIDDLEWARE_CLASSES`, as close to the top as possible:

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

This middleware does **nothing** until you also set the `HTTPOLICE_ENABLE` setting to `True`.

When enabled, the middleware checks all exchanges passing through it. Then, there are two different ways to see the results of these checks.

9.1 Viewing the backlog

All exchanges checked by the middleware are stored in a global variable called the *backlog*. By default, it holds up to 20 latest exchanges, but you can override by setting `HTTPOLICE_BACKLOG` to a different number.

The package also provides the `django_httpolice.report_view()` function. Add it to your URLconf like this:

```
import django_httpolice

urlpatterns = [
    # ...
    url(r'^httpolice/$', django_httpolice.report_view),
    # ...
]
```

When you start the server and open `/httpolice/` (or whatever URL you chose), you will see an HTML report on all the exchanges currently in the backlog. The **latest** exchanges are shown at the **top** of the report.

If `HTTPOLICE_ENABLE` is not `True`, the view responds with 404 (Not Found).

You can also access the backlog from your own code: it's in the `django_httpolice.backlog` variable, as a sequence of *`httpolice.Exchange`* objects.

9.2 Raising on errors

If you set the `HTTPOLICE_RAISE` setting to `True`, then the middleware will raise a `django_httpolice.ProtocolError` whenever a **response** is found to have any errors (that are not *silenced*).

This can be used to fail tests on errors:

```
$ python manage.py test
.E.
======================================================================
ERROR: test_get_plain (example_app.test.ExampleTestCase)
----------------------------------------------------------------------
Traceback (most recent call last):
  [...]
  File "[...]/django_httpolice/middleware.py", line 81, in process_response
    raise ProtocolError(exchange)
django_httpolice.common.ProtocolError: HTTPolice found errors in this response:
------------ request 1 : GET /api/v1/?name=Martha&format=plain
C 1070 No User-Agent header
------------ response 1 : 200 OK
E 1038 Bad JSON body


----------------------------------------------------------------------
Ran 3 tests in 0.351s

FAILED (errors=1)
```

The exchange is still added to the backlog.

9.3 Silencing unwanted notices

To *silence* notices you don't care about, you can use the `HTTPOLICE_SILENCE` setting:

```
HTTPOLICE_SILENCE = [1070, 1110, 1194]
```

They will disappear from reports and will not cause `ProtocolError`.

By default, `HTTPOLICE_SILENCE` includes some notices that are irrelevant because of Django specifics, such as 1110.

Of course, the `HTTPolice-Silence` header works, too:

```
def test_unauthorized(self):
    response = self.client.get('/api/v1/products/',
                               HTTP_HTTPOLICE_SILENCE='1194 resp')
    self.assertEqual(response.status_code, 401)
```

h

C

E

H

I

N

P

R

S

T