CVE-2019-9740 Python urllib CRLF injection vulnerability 浅析

christa / 2019-05-15 09:32:00 / 浏览数 6193 安全技术 WEB安全 顶(1) 踩(0)

0x00前言

2016年曾经爆出过python库中urllib的CRLF HTTP投注入漏洞 CVE-2016-5699，最近又爆出了新的Python urllib CRLF 注入漏洞CVE-2019-9740,有兴趣来分析一下

0x01CRLF

CRLF即为 "回车+换行" (\r\n)的简称,十六进制码为0x0d和0x0a。HTTP中HTTP header和http Body是用两个\n\r来区别的，浏览器根据这两个\r\n来取出HTTP内容并显示出来。因此,当我们能控制HTTP 消息头中的字符，注入一些恶意的换行就能够诸如一些例如会话Cookie或者HTML body的代码。

当我们输入一个`http://127.0.0.1`的时候，其发送的header为

```
GET / HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:67.0) Gecko/20100101 Firefox/67.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Connection: close
Upgrade-Insecure-Requests: 1
```

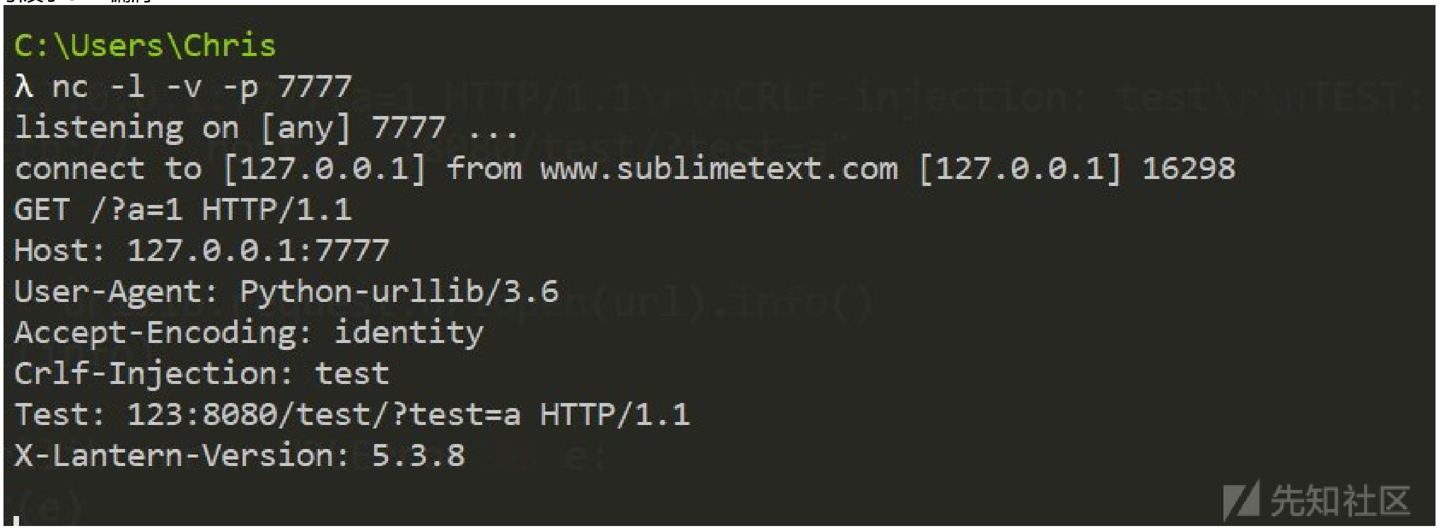
而当我们的url变为`http://127.0.0.1%0d%0a%0d%0aheaders:test`时,其发送的header为

```
GET / HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:67.0) Gecko/20100101 Firefox/67.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Connection: close
headers:test
Upgrade-Insecure-Requests: 1
```

可以看到注入进了header里面
0x02 漏洞研究
官网上的验证代码如下

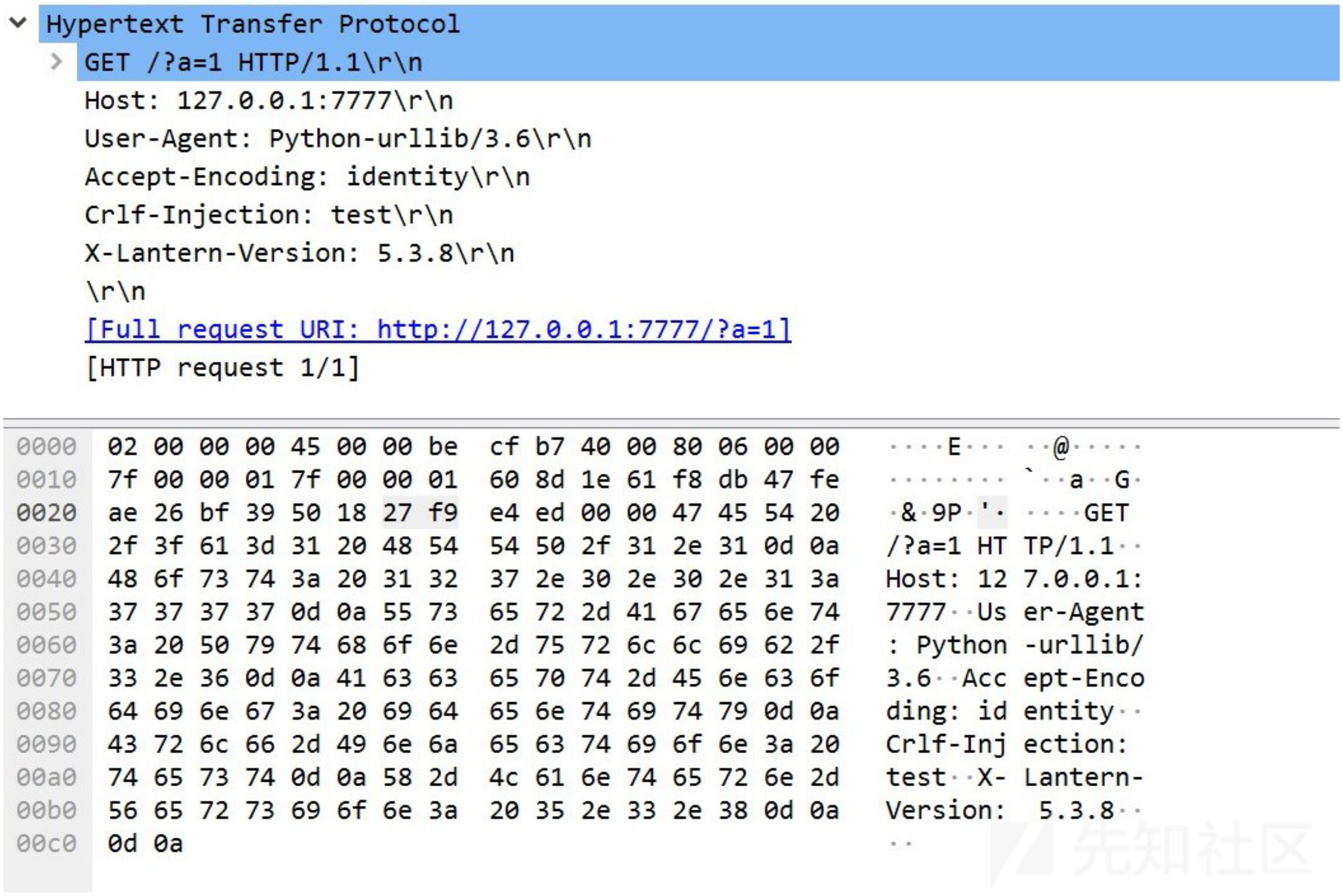
```
import sys
import urllib
import urllib.request
import urllib.error


host = "127.0.0.1:7777?a=1 HTTP/1.1\r\nCRLF-injection: test\r\nTEST: 123"
url = "http://"+ host + ":8080/test/?test=a"

try:
   info = urllib.request.urlopen(url).info()
   print(info)

except urllib.error.URLError as e:
   print(e)
```

引发了CRLF漏洞



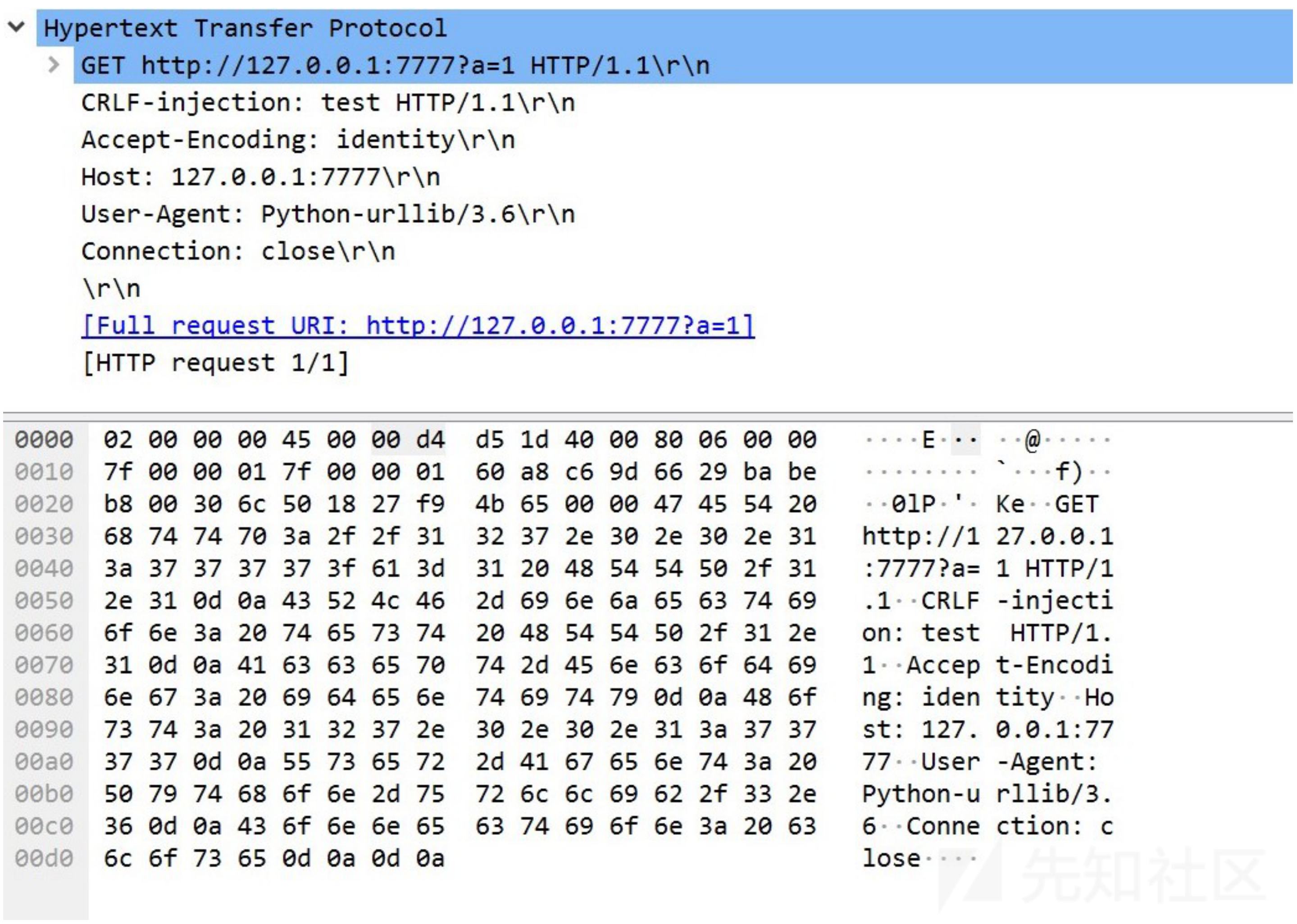
```
C:\Users\Chris
λ nc -l -v -p 7777
listening on [any] 7777 ...
connect to [127.0.0.1] from www.sublimetext.com [127.0.0.1] 16298
GET /?a=1 HTTP/1.1
Host: 127.0.0.1:7777
User-Agent: Python-urllib/3.6
Accept-Encoding: identity
Crlf-Injection: test
Test: 123:8080/test/?test=a HTTP/1.1
X-Lantern-Version: 5.3.8
```

抓包来看



```
Hypertext Transfer Protocol
  GET /?a=1 HTTP/1.1\r\n
    Host: 127.0.0.1:7777\r\n
    User-Agent: Python-urllib/3.6\r\n
    Accept-Encoding: identity\r\n
    Crlf-Injection: test\r\n
    X-Lantern-Version: 5.3.8\r\n
    \r\n
    [Full request URI: http://127.0.0.1:7777/?a=1]
    [HTTP request 1/1]

0000  02 00 00 00 45 00 00 be  cf b7 40 00 80 06 00 00   ····E··· ··@·····
0010  7f 00 00 01 7f 00 00 01  60 8d 1e 61 f8 db 47 fe   ········ `··a··G·
0020  ae 26 bf 39 50 18 27 f9  e4 ed 00 00 47 45 54 20   ·&·9P·'· ····GET 
0030  2f 3f 61 3d 31 20 48 54  54 50 2f 31 2e 31 0d 0a   /?a=1 HT TP/1.1··
0040  48 6f 73 74 3a 20 31 32  37 2e 30 2e 30 2e 31 3a   Host: 12 7.0.0.1:
0050  37 37 37 37 0d 0a 55 73  65 72 2d 41 67 65 6e 74   7777··Us er-Agent
0060  3a 20 50 79 74 68 6f 6e  2d 75 72 6c 6c 69 62 2f   : Python -urllib/
0070  33 2e 36 0d 0a 41 63 63  65 70 74 2d 45 6e 63 6f   3.6··Acc ept-Enco
0080  64 69 6e 67 3a 20 69 64  65 6e 74 69 74 79 0d 0a   ding: id entity··
0090  43 72 6c 66 2d 49 6e 6a  65 63 74 69 6f 6e 3a 20   Crlf-Inj ection: 
00a0  74 65 73 74 0d 0a 58 2d  4c 61 6e 74 65 72 6e 2d   test··X- Lantern-
00b0  56 65 72 73 69 6f 6e 3a  20 35 2e 33 2e 38 0d 0a   Version:  5.3.8··
00c0  0d 0a                                              ··
```

后置的8080的后缀被自动修正，如果不加后缀:8080/test/?test=a的效果为



```
Hypertext Transfer Protocol
  GET http://127.0.0.1:7777?a=1 HTTP/1.1\r\n
  CRLF-injection: test HTTP/1.1\r\n
  Accept-Encoding: identity\r\n
  Host: 127.0.0.1:7777\r\n
  User-Agent: Python-urllib/3.6\r\n
  Connection: close\r\n
  \r\n
  [Full request URI: http://127.0.0.1:7777?a=1]
  [HTTP request 1/1]
```

```
0000  02 00 00 00 45 00 00 d4  d5 1d 40 00 80 06 00 00   ····E··· ··@·····
0010  7f 00 00 01 7f 00 00 01  60 a8 c6 9d 66 29 ba be   ········ `···f)··
0020  b8 00 30 6c 50 18 27 f9  4b 65 00 00 47 45 54 20   ··0lP·'· Ke··GET 
0030  68 74 74 70 3a 2f 2f 31  32 37 2e 30 2e 30 2e 31   http://1 27.0.0.1
0040  3a 37 37 37 37 3f 61 3d  31 20 48 54 54 50 2f 31   :7777?a= 1 HTTP/1
0050  2e 31 0d 0a 43 52 4c 46  2d 69 6e 6a 65 63 74 69   .1··CRLF -injecti
0060  6f 6e 3a 20 74 65 73 74  20 48 54 54 50 2f 31 2e   on: test  HTTP/1.
0070  31 0d 0a 41 63 63 65 70  74 2d 45 6e 63 6f 64 69   1··Accep t-Encodi
0080  6e 67 3a 20 69 64 65 6e  74 69 74 79 0d 0a 48 6f   ng: iden tity··Ho
0090  73 74 3a 20 31 32 37 2e  30 2e 30 2e 31 3a 37 37   st: 127. 0.0.1:77
00a0  37 37 0d 0a 55 73 65 72  2d 41 67 65 6e 74 3a 20   77··User -Agent: 
00b0  50 79 74 68 6f 6e 2d 75  72 6c 6c 69 62 2f 33 2e   Python-u rllib/3.
00c0  36 0d 0a 43 6f 6e 6e 65  63 74 69 6f 6e 3a 20 63   6··Conne ction: c
00d0  6c 6f 73 65 0d 0a 0d 0a                            lose····
```

研究代码，首先看到Lib/urllib文件夹下面的request.py文件，从urlopen这个函数一路跟进

```
def urlopen(url, data=None, timeout=socket._GLOBAL_DEFAULT_TIMEOUT,
            *, cafile=None, capath=None, cadefault=False, context=None):
   global _opener
   if cafile or capath or cadefault:
       import warnings
       warnings.warn("cafile, cpath and cadefault are deprecated, use a "
                     "custom context instead.", DeprecationWarning, 2)
       if context is not None:
           raise ValueError(
               "You can't pass both context and any of cafile, capath, and "
               "cadefault"
           )
       if not _have_ssl:
           raise ValueError('SSL support not available')
       context = ssl.create_default_context(ssl.Purpose.SERVER_AUTH,
                                            cafile=cafile,
                                            capath=capath)
       https_handler = HTTPSHandler(context=context)
       opener = build_opener(https_handler)
   elif context:
       https_handler = HTTPSHandler(context=context)
       opener = build_opener(https_handler)
   elif _opener is None:
       _opener = opener = build_opener()
   else:
       opener = _opener
   return opener.open(url, data, timeout)
```

看到代码中调用了build_opener函数，继续跟进

```
def build_opener(*handlers):
   opener = OpenerDirector()
   default_classes = [ProxyHandler, UnknownHandler, HTTPHandler,
                      HTTPDefaultErrorHandler, HTTPRedirectHandler,
                      FTPHandler, FileHandler, HTTPErrorProcessor,
                      DataHandler]
   if hasattr(http.client, "HTTPSConnection"):
       default_classes.append(HTTPSHandler)
   skip = set()
   for klass in default_classes:
       for check in handlers:
           if isinstance(check, type):
               if issubclass(check, klass):
                   skip.add(klass)
           elif isinstance(check, klass):
               skip.add(klass)
   for klass in skip:
       default_classes.remove(klass)

   for klass in default_classes:
       opener.add_handler(klass())

   for h in handlers:
       if isinstance(h, type):
           h = h()
       opener.add_handler(h)
   return opener
```

在build_opener函数里面根据我们的url来看使用了`HTTPHandler`这个类，继续跟进

```
class HTTPHandler(AbstractHTTPHandler):

   def http_open(self, req):
       return self.do_open(http.client.HTTPConnection, req)

   http_request = AbstractHTTPHandler.do_request_
```

在这个函数带有恶意payload的url通过Request方法进行请求,从self.open方法中也能够看到

```
def do_open(self, http_class, req, **http_conn_args):
       host = req.host
       if not host:
           raise URLError('no host given')

       # will parse host:port
       h = http_class(host, timeout=req.timeout, **http_conn_args)
       h.set_debuglevel(self._debuglevel)

       headers = dict(req.unredirected_hdrs)
       headers.update(dict((k, v) for k, v in req.headers.items()
                           if k not in headers))

       headers["Connection"] = "close"
       headers = dict((name.title(), val) for name, val in headers.items())

       if req._tunnel_host:
           tunnel_headers = {}
           proxy_auth_hdr = "Proxy-Authorization"
           if proxy_auth_hdr in headers:
               tunnel_headers[proxy_auth_hdr] = headers[proxy_auth_hdr]
               # Proxy-Authorization should not be sent to origin
               # server.
               del headers[proxy_auth_hdr]
           h.set_tunnel(req._tunnel_host, headers=tunnel_headers)

       try:
           try:
               h.request(req.get_method(), req.selector, req.data, headers,
                         encode_chunked=req.has_header('Transfer-encoding'))
           except OSError as err: # timeout error
               raise URLError(err)
           r = h.getresponse()
       except:
           h.close()
           raise
       if h.sock:
           h.sock.close()
           h.sock = None

       r.url = req.get_full_url()
       r.msg = r.reason
       return r
```

重新看Lib/http/client.py这个文件中的putheader方法，在之前的CVE-2016-5699漏洞中它的代码如下

```
def putheader(self, header, *values):
        values = list(values)
        for i, one_value in enumerate(values):
            if hasattr(one_value, 'encode'):
                values[i] = one_value.encode('latin-1')
            elif isinstance(one_value, int):
                values[i] = str(one_value).encode('ascii')
        value = b'\r\n\t'.join(values)
        header = header + b': ' + value
        self._output(header)
```

修复后的代码如下

```
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()

        if hasattr(header, 'encode'):
            header = header.encode('ascii')

        if not _is_legal_header_name(header):
            raise ValueError('Invalid header name %r' % (header,))

        values = list(values)
        for i, one_value in enumerate(values):
            if hasattr(one_value, 'encode'):
                values[i] = one_value.encode('latin-1')
            elif isinstance(one_value, int):
                values[i] = str(one_value).encode('ascii')

            if _is_illegal_header_value(values[i]):
                raise ValueError('Invalid header value %r' % (values[i],))

        value = b'\r\n\t'.join(values)
        header = header + b': ' + value
        self._output(header)
```

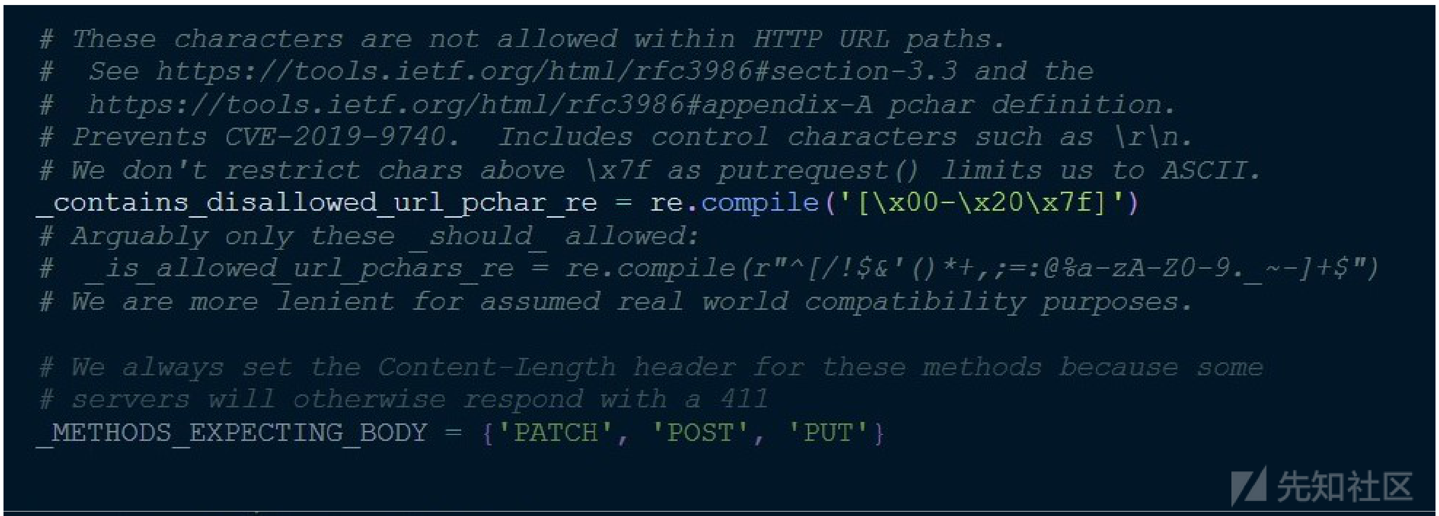
加入了_is_legal_header_name这个方法，方法为

```
_is_legal_header_name = re.compile(rb'[^:\s][^:\r\n]*').fullmatch
```

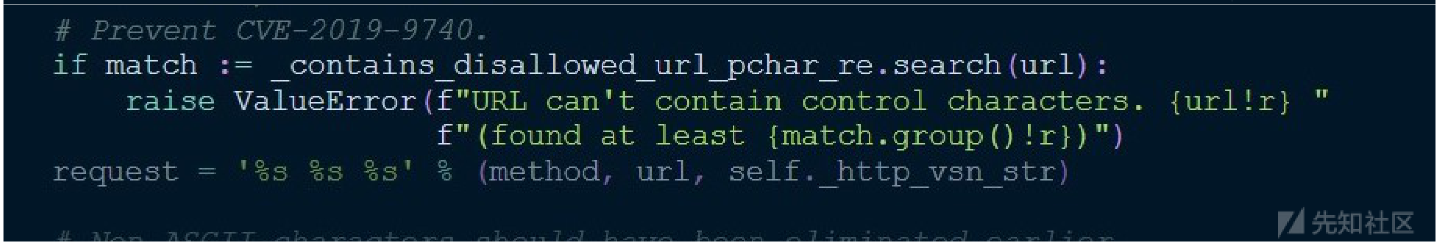
可以看到对：后面的内容进行匹配，匹配了所有\r\n的内容，如果匹配到\r\n,则返回报错`Invalid header name <header>`,但是通过调试发现该检测方法仅为检测返回时候的header头而没有检测到发送除去的url，因此发送出去的payload并没有经过正则的匹配。

0x03官方的修复方法

https://github.githistory.xyz/python/cpython/blob/96aeaec64738b730c719562125070a52ed570210/Lib/http/client.py
在putrequest方法上对url进行检查，匹配所有的ascii码在00到32的所有字符，并且同时匹配\x7f字符



```
# These characters are not allowed within HTTP URL paths.
#  See https://tools.ietf.org/html/rfc3986#section-3.3 and the
#  https://tools.ietf.org/html/rfc3986#appendix-A pchar definition.
# Prevents CVE-2019-9740.  Includes control characters such as \r\n.
# We don't restrict chars above \x7f as putrequest() limits us to ASCII.
_contains_disallowed_url_pchar_re = re.compile('[\x00-\x20\x7f]')
# Arguably only these _should_ allowed:
#  _is_allowed_url_pchars_re = re.compile(r"^[/!$&'()*+,;=:@%a-zA-Z0-9._~-]+$")
# We are more lenient for assumed real world compatibility purposes.

# We always set the Content-Length header for these methods because some
# servers will otherwise respond with a 411
_METHODS_EXPECTING_BODY = {'PATCH', 'POST', 'PUT'}
```



```
 # Prevent CVE-2019-9740.
 if match := _contains_disallowed_url_pchar_re.search(url):
     raise ValueError(f"URL can't contain control characters. {url!r} "
                      f"(found at least {match.group()!r})")
 request = '%s %s %s' % (method, url, self._http_vsn_str)

 # Non-ASCII characters should have been eliminated earlier
```

有错误之处还请师傅们斧正。

Reference

- https://bugs.python.org/issue36276
- https://hg.python.org/cpython/rev/bf3e1c9b80e9
- https://bugs.python.org/issue30458#msg295067
- https://bugs.python.org/issue30458#msg295067
- https://bugs.python.org/issue30458#msg295067

点击收藏 | 1 关注 | 1

上一篇：WinAFL 源码分析 下一篇：CVE-2019-5018：Sql...

1. 1 条回复

lorewalker**** 2019-05-15 21:10:06

好文，收藏了

0 回复Ta

登录 后跟帖

先知社区

现在登录