

# Week 1 Solution

## (5 pt) What is so called stream?

The network is so bad that I can't even send TCP stream through Internet. Wondering if I can use "UDP streams"...

[capture.pcap](#)

Try to find `flag` in this file, the flag format is: `picoCTF{***}`

*Hint1: Wireshark may be useful.*

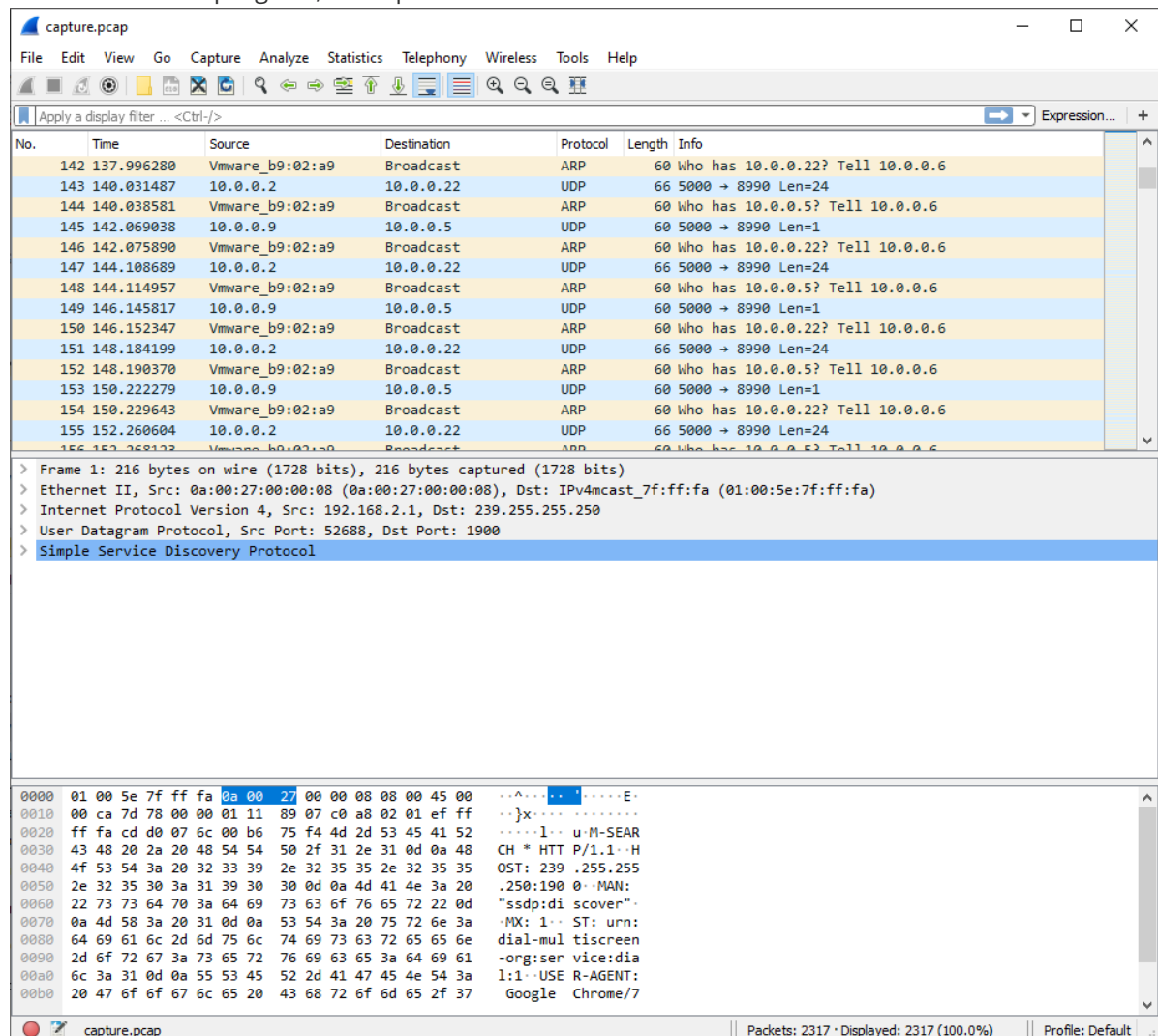
## Writeup

Challenge from picoCTF 2019 `shark on wire 1` by DANNY

Writeup from <https://zomny1.github.io/shark-on-wire-1/>

I download the file, and the file extension is pcap, So it's Wireshark file. For those who don't know what it is, [Wireshark](#) is a sniffing and packet analyzer program.

So download the program, and open the file with it -



Most of the time, the packets we are interested in are UDP and TCP protocols, as I scroll down I could see a lot of UDP packets. I chose randomly one of those packets (just click on it), and

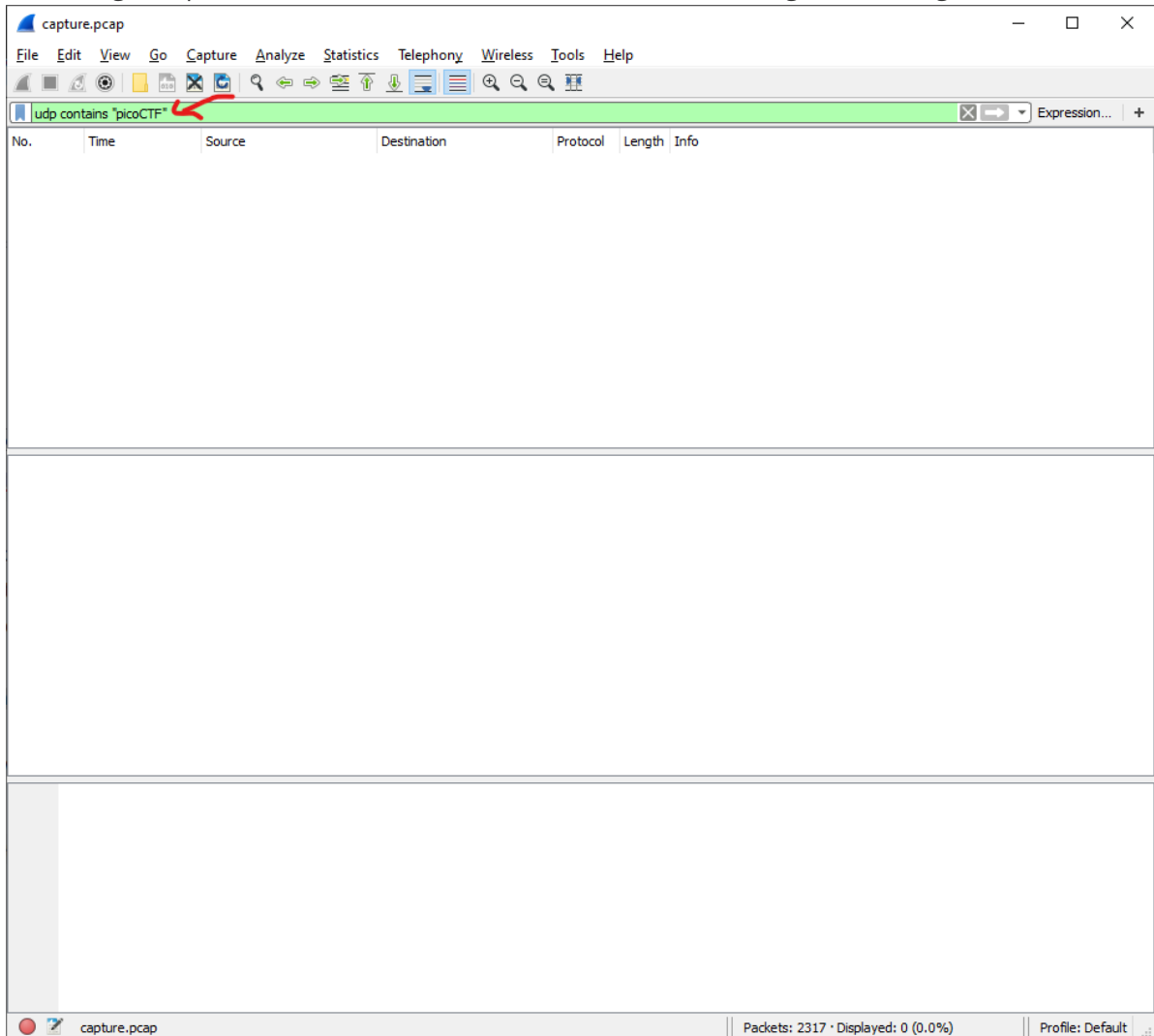
opened the data tab -

The image shows a Wireshark window titled 'capture.pcap'. The main pane displays a list of network packets. Packet 153 is highlighted in blue and has a red arrow pointing to it. The detailed view pane below shows the structure of packet 153, which is a UDP datagram. The data field is expanded to show a single byte containing the character 'b'. The hex dump at the bottom shows the raw bytes of the data field: 0000 ff ff ff ff ff ff 00 0c 29 b9 02 a9 08 00 45 00 ... .. ).....E.  
0010 00 1d 00 01 00 00 40 11 66 c2 0a 00 00 09 0a 00 .....@. f.....  
0020 00 05 13 88 23 1e 00 09 53 28 62 00 00 00 00 .....#... S(b.....  
0030 00 00 00 00 00 00 00 00 00 00 00 00 ..... ..

We can see that it's just a 'b', tried this on other packets but nothing came out. I cant check each packet...So Wireshark has it own filtering system, So i wanted to search a udp packets that contains the picoCTF format, after some searching it the web I found this as the filter I wanted -

udp contains "picoCTF"

(Don't forget to press enter to filter) tried it in Wireshark but nothing came out again..



So maybe the data is fragmented between many packets? In order to check this there is something called [udp streams](#) So in order to check those streams, I deleted the filter (and pressed

Enter), chose random udp packet and press right click on it -> follow -> udp stream:

The screenshot shows the Wireshark interface with a packet capture named 'capture.pcap'. The main pane displays a list of captured packets. Packet 107 is selected, and a context menu is open over it. The 'Follow' option is highlighted, and a sub-menu is visible with 'UDP Stream' selected. The packet details pane shows the structure of the selected packet: Ethernet II, Internet Protocol Version 4, User Datagram Protocol, and Data (24 bytes). The hex dump pane shows the raw data of the packet.

No.	Time	Source	Destination	Protocol	Length	Info
100	99.246656	Vmware_b9:02:a9	Broadcast	ARP	60	Who has 10.0.0.13? Tell 10.0.0.6
101	101.278404	10.0.0.2	10.0.0.13	UDP	60	5000 → 8888 Len=1
102	101.285764	Vmware_b9:02:a9	Broadcast	ARP	60	Who has 10.0.0.22? Tell 10.0.0.6
103	103.320177	10.0.0.2	10.0.0.22	UDP	66	5000 → 8990 Len=24
104	103.328171	Vmware_b9:02:a9	Broadcast	ARP	60	Who has 10.0.0.5? Tell 10.0.0.6
105	105.361299	10.0.0.9	10.0.0.5	UDP	60	5000 → 8990 Len=1
106	105.367747	Vmware_b9:02:a9	Broadcast	ARP	60	Who has 10.0.0.22? Tell 10.0.0.6
107	107.396948	10.0.0.2	10.0.0.22	UDP	66	5000 → 8990 Len=24
108	107.404035	Vmware_b9:02:a9	Broadcast	ARP	60	Who has 10.0.0.5? Tell 10.0.0.6
109	109.441269	10.0.0.9	10.0.0.5	UDP	60	5000 → 8990 Len=1
110	109.447538	Vmware_b9:02:a9	Broadcast	ARP	60	Who has 10.0.0.22? Tell 10.0.0.6
111	111.476713	10.0.0.2	10.0.0.22	UDP	66	5000 → 8990 Len=24
112	111.484618	Vmware_b9:02:a9	Broadcast	ARP	60	Who has 10.0.0.5? Tell 10.0.0.6
113	113.513065	10.0.0.9	10.0.0.5	UDP	60	5000 → 8990 Len=1
114	113.519607	Vmware_b9:02:a9	Broadcast	ARP	60	Who has 10.0.0.22? Tell 10.0.0.6

Frame 107: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface  
> Ethernet II, Src: Vmware\_b9:02:a9 (00:0c:29:b9:02:a9), Dst: Broadcast  
> Internet Protocol Version 4, Src: 10.0.0.2, Dst: 10.0.0.22  
> User Datagram Protocol, Src Port: 5000, Dst Port: 8990  
Data (24 bytes)  
Data: 666a6473616b663b6c616e6b65666c6b73616e6c6b66646e  
[Length: 24]

```
0000  ff ff ff ff ff ff 00 0c 29 b9 02 a9 08 00 45 00  ..... ).....E.  
0010  00 34 00 01 00 00 40 11 66 a1 0a 00 00 02 0a 00  -4...@. f.....  
0020  00 16 13 88 23 1e 00 20 c4 2a 66 6a 64 73 61 6b  ...#... *fjdsak  
0030  66 3b 6c 61 6e 6b 65 66 6c 6b 73 61 6e 6c 6b 66  f;lankef lksanlkf  
0040  64 6e                                             dn
```

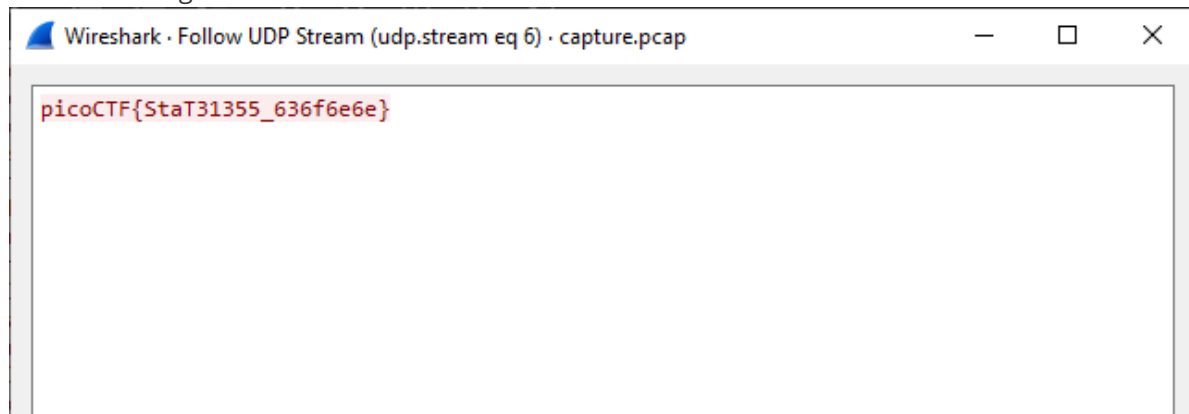
A window open with the whole text of this stream, but it just junk...

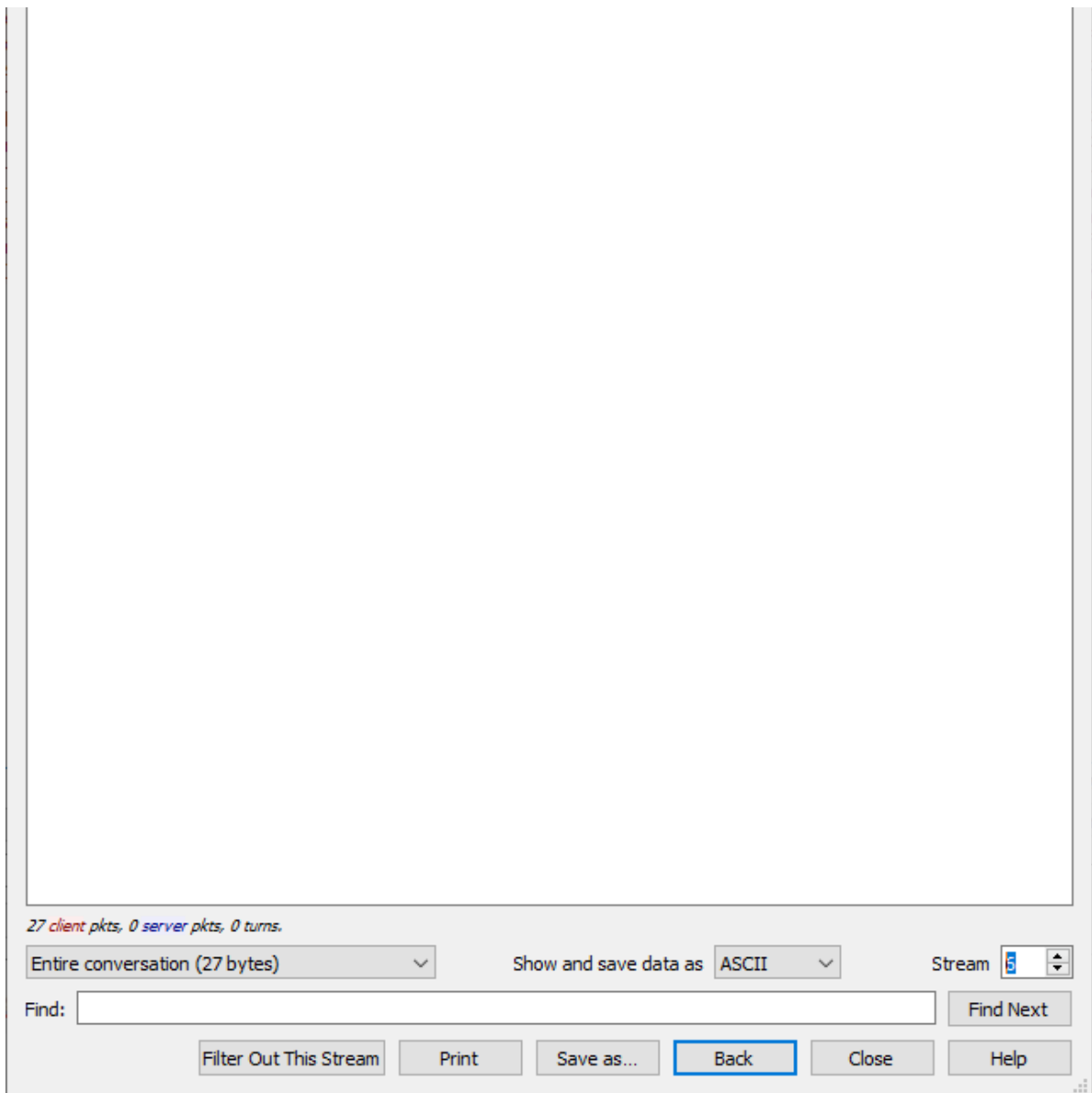


Maybe this is not the correct stream, I tried to look at the others with change the stream id:



I found the flag in the 5th stream!





## (5 pt) HTTPS with secret sauce

Solved the network problem yesterday, but I found some guy was sniffing my network traffic. I need to be careful to protect my flag. Decide to use HTTPS to submit my flag to `web01`.



By the way, upload my **super☆secret☆file** to network disk.

[capture.pcapng](#)

[pre-master secret.txt](#)

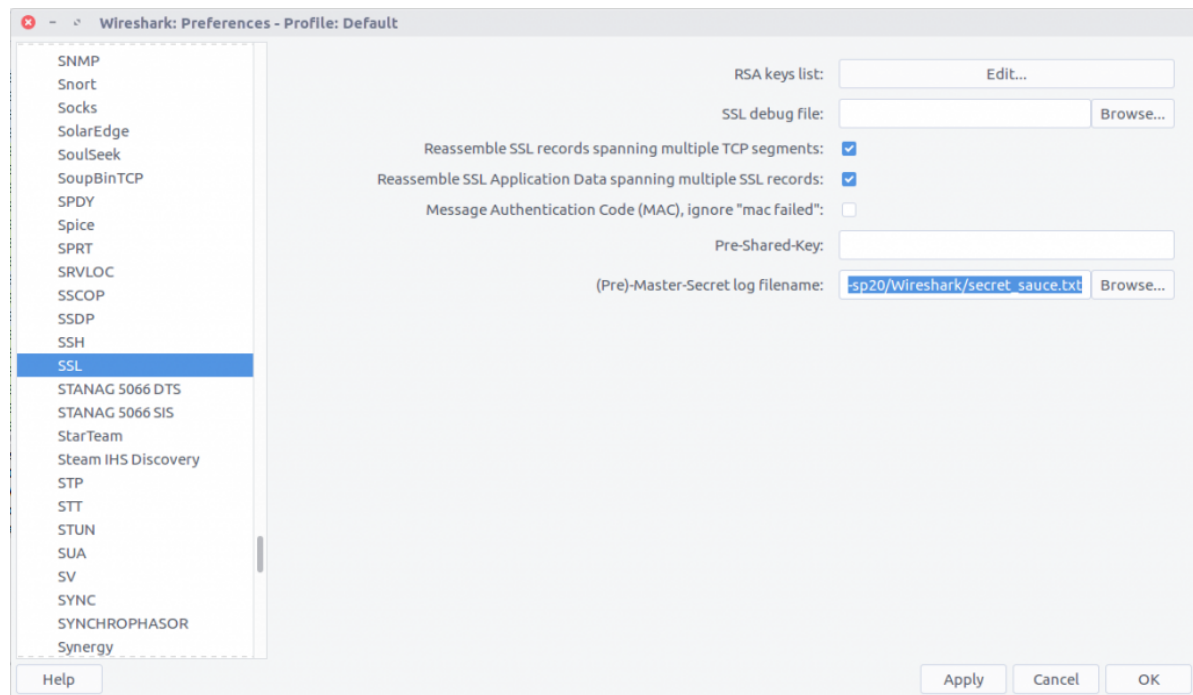
Try to find `f1ag` in this file, the flag format is: `f1ag{y2***}`

# Writeup

Challenge from DFA/CCSC Spring 2020 CTF `Some secret sauce`

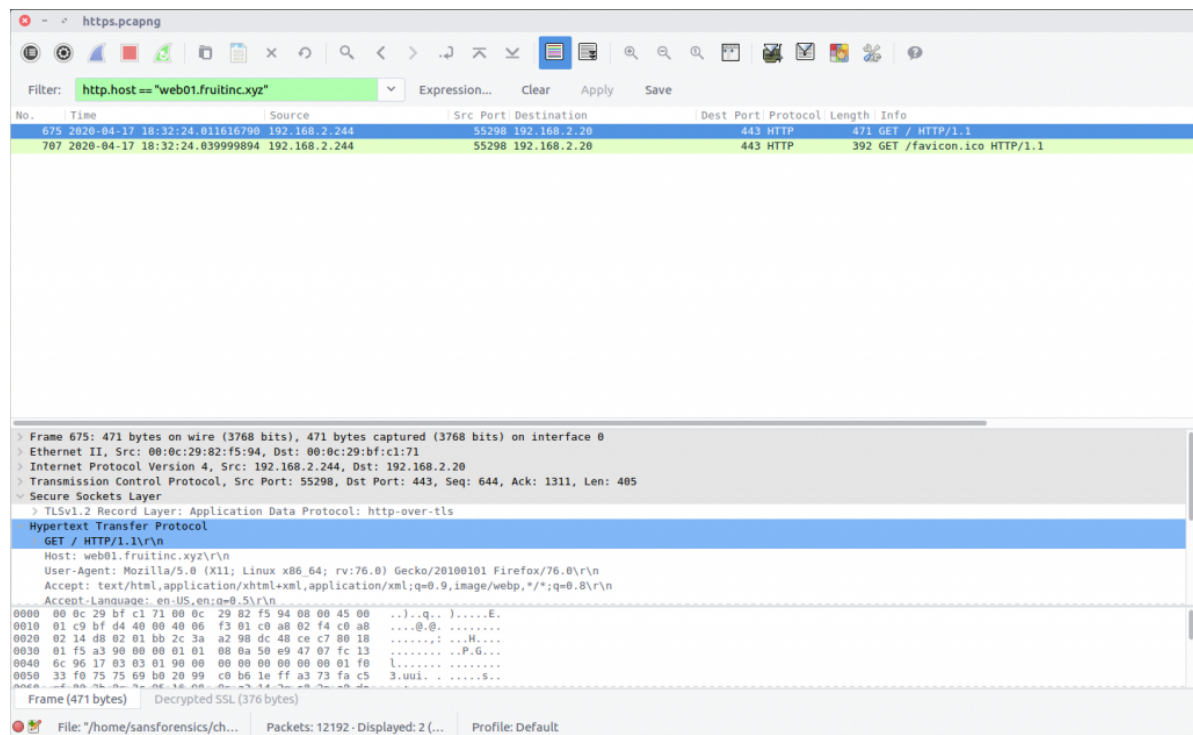
Writeup from <https://www.petermstewart.net/dfa-ccsc-spring-2020-ctf-wireshark-https-pcapng-write-up/>

Before we can start answering questions we need to decrypt the encrypted traffic. *Wireshark* allows us to [decrypt TLS traffic by supplying the Pre-Master Secret](#) helpfully provided in the `secret-sauce.txt` file that was included with the challenge PCAPs.



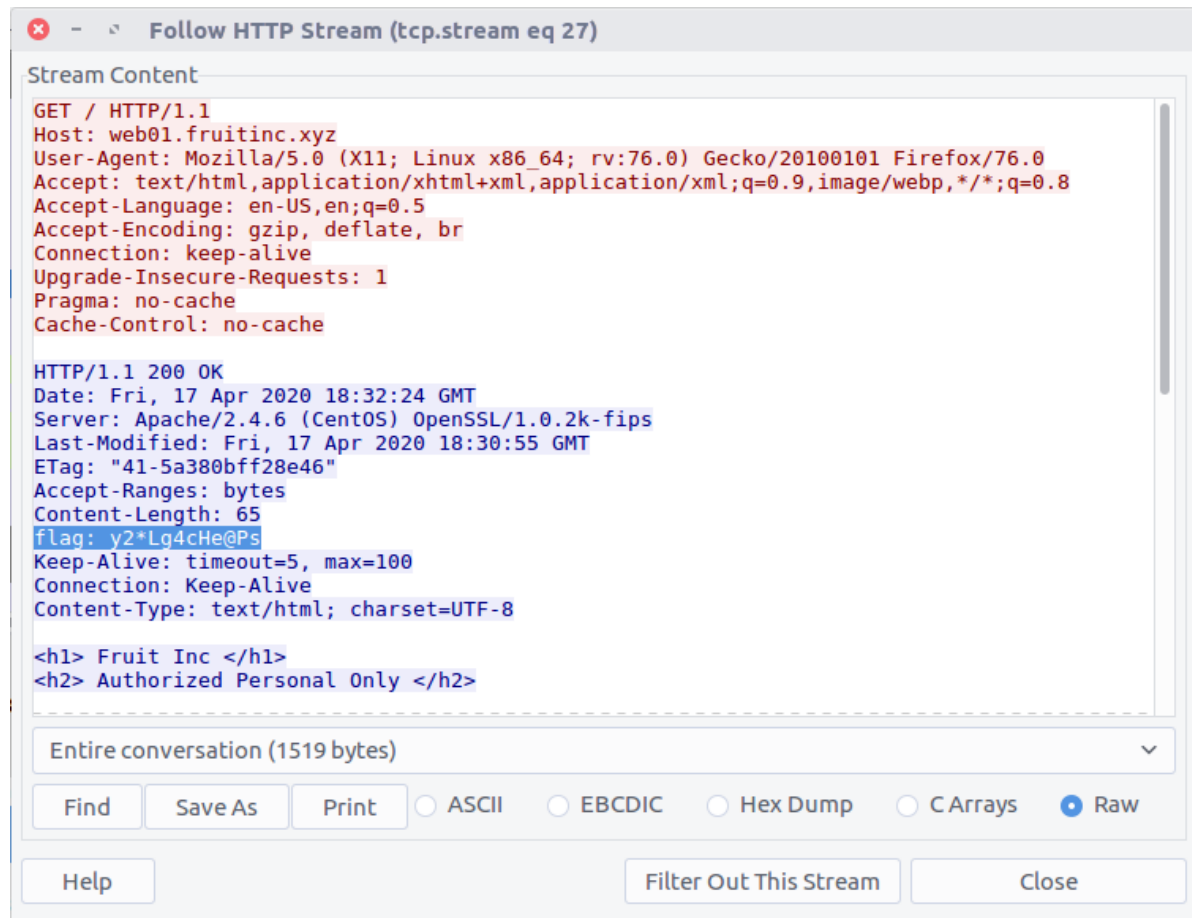
After decrypting the traffic we can filter for traffic to the web server in question:

```
http.host == "web01.fruitinc.xyz"
```





This filter only shows us packets specifically containing the HTTP request headers, but by selecting the *Follow HTTP Stream* option (*Stream #27*) we can more easily read the exchange between the client and server, including the flag inserted into the server response headers.



The screenshot shows the 'Follow HTTP Stream (tcp.stream eq 27)' window in Wireshark. The 'Stream Content' pane displays the following text:

```
GET / HTTP/1.1
Host: web01.fruitinc.xyz
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:76.0) Gecko/20100101 Firefox/76.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Pragma: no-cache
Cache-Control: no-cache

HTTP/1.1 200 OK
Date: Fri, 17 Apr 2020 18:32:24 GMT
Server: Apache/2.4.6 (CentOS) OpenSSL/1.0.2k-fips
Last-Modified: Fri, 17 Apr 2020 18:30:55 GMT
ETag: "41-5a380bff28e46"
Accept-Ranges: bytes
Content-Length: 65
flag: y2*Lg4cHe@Ps
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8

<h1> Fruit Inc </h1>
<h2> Authorized Personal Only </h2>
```

Below the stream content, there are buttons for 'Find', 'Save As', 'Print', and radio buttons for 'ASCII', 'EBCDIC', 'Hex Dump', 'C Arrays', and 'Raw' (selected). There are also buttons for 'Help', 'Filter Out This Stream', and 'Close'.

flag<y2\*Lg4cHe@Ps>

## (BONUS 5 pt) Bytes through network

That hacker still got my flag! Fine, I'm going to send my file byte by byte. Besides, combined with my knowledge of **programming, encryption, and steganography** I'm going to fight the final round. WE ARE IN THE ENDGAME NOW.

[capture.pcapng](#)

Try to find `flag` in this file, the flag format is: `flag{***}`

*This challenge is extremely hard. The winner will get a badge for solving this.*

## Writeup

Challenge from 强网杯 2021 ExtremelySlow

Writeup from [https://miaotony.xyz/2021/06/28/CTF\\_2021qiangwang](https://miaotony.xyz/2021/06/28/CTF_2021qiangwang) by MiaoTony

首先是一个流量包，里面全是 TCP 和 HTTP 流量。而且是 206 分段传输，每个包传 1byte。

于是先导出为 JSON，然后写个脚本提取其中的每个 byte，最后合并得到一个二进制文件。

wireshark 直接导出的 JSON 里 `http.response.line` 包含多个，如果直接用 `json.loads` 只保留最后一个了，所以先要去掉无关的内容。

python

```

import json
import re

with open('http.json', 'r', encoding='utf-8') as fin:
    s = fin.read()

re_num = re.compile(
    r'"http.response.line": "content-range: bytes (\d+)-\d+/1987\\r\\n"'
)
re_nonnum = re.compile(
    r'(\\"http.response.line": (?!\\"content-range: bytes (\d+)-\d+/1987\\r\\n",).*)'
)
s1 = re.sub(re_nonnum, '', s)

with open('http_sub.json', 'w', encoding='utf-8') as fout:
    fout.write(s1)

http = json.loads(s1)
total = [b''] * 1987
# total = [''] * 1987
idx_list = []
for x in http:
    source = x['_source']
    layers = source['layers']
    # get data
    data = layers['data']['data.data']
    data = bytes([int(data, 16)])
    # find index
    n = layers['http']['http.response.line']
    idx = int(re.search(r'(\d+)-\d+/1987', n)[1])
    idx_list.append(idx)
    total[idx] = data

print(total)
t = b''.join(total)
# t = ''.join(total)
# print(len(t)/2)
with open('decode.pyc', 'wb') as f:
    f.write(t)
# with open('decode1.pyc', 'w') as f:
#     f.write(t)

```

或者直接命令行用 tshark 更快，不过当时就没想到这么写喵喵呜呜。

按 index 把这个合并就行，bash 脚本类似这样

bash

```

tshark -r ExtremelySlow.pcapng -T fields -e data -Y "http.response.line == \"content-range: bytes $idx-$idx/1987\\x0d\\x0a\"" 2>/dev/null

```

根据文件内容得知是个 pyc 文件。

但是直接拿在线工具或者 uncompile6 反编译都不成，发现 magic number 有误。

参考

[Python's magic numbers](#)

[Python Uncompyle6 反编译工具使用与 Magic Number 详解](#)

<https://github.com/google/pytype/blob/master/pytype/pyc/magic.py>

[Understanding Python Bytecode](#)

可以发现文件头的这个 magic number 是随版本号递增的，而且比最新的 3.9.5 跨了一大截。

于是考虑拉个 py3.10 的镜像下来。

bash

```
docker run --rm -it python:3.10.0b2
```

根据 magic number 确定就是最新的 Python 3.10.0b2

```
Python 3.10.0b2 (default, Jun 7 2021, 20:33:14) [GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import imp
<stdin>:1: DeprecationWarning: the imp module is deprecated in favour of
tation for alternative uses
>>> imp.get_magic()
b'o\r\r\n'
>>> imp.get_magic().hex()
'6f0d0d0a'
>>>
```

但还是需要反编译这个 pyc

[uncompyle6](#) <https://pypi.org/project/uncompyle6/> 目前只支持 python 2.4-3.8

<https://github.com/rocky/python-decompile3> 不行

dis 可

python

```
>>> import marshal, dis
>>> with open('decode.pyc', 'rb') as f:
...     metadata = f.read(16)
...     code_obj = marshal.load(f)
...
>>> dis.dis(code_obj)
4          0 LOAD_CONST           0 (0)
          2 LOAD_CONST           1 (None)
          4 IMPORT_NAME         0 (sys)
          6 STORE_NAME         0 (sys)

6          8 LOAD_CONST           0 (0)
         10 LOAD_CONST           2 (('sha256',))
         12 IMPORT_NAME         1 (hashlib)
         14 IMPORT_FROM         2 (sha256)
         16 STORE_NAME         2 (sha256)
         18 POP_TOP

16         20 LOAD_CONST           3 (<code object KSA at 0x7f1199dc7890,
file "main.py", line 6>)
         22 LOAD_CONST           4 ('KSA')
         24 MAKE_FUNCTION       0
         26 STORE_NAME         3 (KSA)
```

```

26          28 LOAD_CONST          5 (<code object PRGA at 0x7f1199dc7940,
file "main.py", line 16>)
          30 LOAD_CONST          6 ('PRGA')
          32 MAKE_FUNCTION       0
          34 STORE_NAME         4 (PRGA)

30          36 LOAD_CONST          7 (<code object RC4 at 0x7f1199dc7aa0,
file "main.py", line 26>)
          38 LOAD_CONST          8 ('RC4')
          40 MAKE_FUNCTION       0
          42 STORE_NAME         5 (RC4)

33          44 LOAD_CONST          9 (<code object xor at 0x7f1199dd4500,
file "main.py", line 30>)
          46 LOAD_CONST         10 ('xor')
          48 MAKE_FUNCTION       0
          50 STORE_NAME         6 (xor)

34          52 LOAD_NAME          7 (__name__)
          54 LOAD_CONST         11 ('__main__')
          56 COMPARE_OP           2 (==)
          58 POP_JUMP_IF_FALSE    139 (to 278)

35          60 LOAD_CONST         12
(b'\xf6\xef\x10H\xa9\x0f\x9f\xb5\x80\xc1d\xae\xd3\x03\xb2\x84\xc2\xb4\x0e\xc8\x
f3<\x151\x19\n\x8f')
          62 STORE_NAME         8 (w)

38          64 LOAD_CONST         13
(b'$\r9\xa3\x18\xddw\xc9\x97\xf3\xa7\xa8R~')
          66 STORE_NAME         9 (e)

39          68 LOAD_CONST         14 (b'geo')
          70 STORE_NAME        10 (b)

41          72 LOAD_CONST         15
(b'}\xce`xbej\xa2\x120\xb5\x8a\x94\x14{\xa3\x86\xc8\xc7\x01\x98\xa3_\x91\xd8\x8
2T*v\xab\xe0\xa1\x141')
          74 STORE_NAME        11 (s)

42          76 LOAD_CONST         16 (b"Q_\xe2\xf8\x8c\x11M}'<@\xceT\xf6?
_m\xa4\xf8\xb4\xea\xca\xc7:\xb9\xe6\x06\x8b\xeb\xfabH\x85xJ3$\xdd\xde\xb6\xdc\xa
0\xb8b\x961\xb7\x13=\x17\x13\xb1")
          78 STORE_NAME        12 (t)

43          80 LOAD_CONST         17 (115)
          82 LOAD_CONST         18 (97)
          84 LOAD_CONST         19 (117)
          86 LOAD_CONST         20 (114)
          88 LOAD_CONST         21 ((2, 8, 11, 10))
          90 BUILD_CONST_KEY_MAP  4
          92 STORE_NAME        13 (m)

44          94 LOAD_CONST         22 (119)
          96 LOAD_CONST         23 (116)
          98 LOAD_CONST         24 (124)
         100 LOAD_CONST         25 (127)
         102 LOAD_CONST         26 ((3, 7, 9, 12))

```

```

104 BUILD_CONST_KEY_MAP      4
106 STORE_NAME               14 (n)

45      108 LOAD_NAME             13 (m)
      110 LOAD_CONST              27 (<code object <dictcomp> at
0x7f1199dd4c90, file "main.py", line 44>)
      112 LOAD_CONST              28 ('<dictcomp>')
      114 MAKE_FUNCTION            0
      116 LOAD_NAME               14 (n)
      118 GET_ITER
      120 CALL_FUNCTION            1
      122 INPLACE_OR
      124 STORE_NAME             13 (m)

47      126 LOAD_NAME             13 (m)
      128 LOAD_CONST              29 (<code object <genexpr> at
0x7f1199dd5b00, file "main.py", line 45>)
      130 LOAD_CONST              30 ('<genexpr>')
      132 MAKE_FUNCTION            0
      134 LOAD_NAME               10 (b)
      136 GET_ITER
      138 CALL_FUNCTION            1
      140 INPLACE_OR
      142 STORE_NAME             13 (m)

48      144 LOAD_NAME             5 (RC4)
      146 LOAD_NAME             15 (list)
      148 LOAD_NAME             16 (map)
      150 LOAD_CONST              31 (<code object <lambda> at
0x7f1199a42d90, file "main.py", line 47>)
      152 LOAD_CONST              32 ('<lambda>')
      154 MAKE_FUNCTION            0
      156 LOAD_NAME             17 (sorted)
      158 LOAD_NAME             13 (m)
      160 LOAD_METHOD             18 (items)
      162 CALL_METHOD              0
      164 CALL_FUNCTION            1
      166 CALL_FUNCTION            2
      168 CALL_FUNCTION            1
      170 CALL_FUNCTION            1
      172 STORE_NAME             19 (stream)

49      174 LOAD_NAME             20 (print)
      176 LOAD_NAME             6 (xor)
      178 LOAD_NAME             8 (w)
      180 LOAD_NAME             19 (stream)
      182 CALL_FUNCTION            2
      184 LOAD_METHOD             21 (decode)
      186 CALL_METHOD              0
      188 CALL_FUNCTION            1
      190 POP_TOP

50      192 LOAD_NAME             0 (sys)
      194 LOAD_ATTR               22 (stdin)
      196 LOAD_ATTR               23 (buffer)
      198 LOAD_METHOD             24 (read)
      200 CALL_METHOD              0
      202 STORE_NAME             25 (p)

```

```

52      204 LOAD_NAME          6 (xor)
        206 LOAD_NAME          9 (e)
        208 LOAD_NAME         19 (stream)
        210 CALL_FUNCTION      2
        212 STORE_NAME        9 (e)

53      214 LOAD_NAME          6 (xor)
        216 LOAD_NAME         25 (p)
        218 LOAD_NAME         19 (stream)
        220 CALL_FUNCTION      2
        222 STORE_NAME        26 (c)

54      224 LOAD_NAME          2 (sha256)
        226 LOAD_NAME         26 (c)
        228 CALL_FUNCTION      1
        230 LOAD_METHOD        27 (digest)
        232 CALL_METHOD        0
        234 LOAD_NAME          11 (s)
        236 COMPARE_OP        2 (==)
        238 POP_JUMP_IF_FALSE 131 (to 262)

56      240 LOAD_NAME          20 (print)
        242 LOAD_NAME          6 (xor)
        244 LOAD_NAME         12 (t)
        246 LOAD_NAME         19 (stream)
        248 CALL_FUNCTION      2
        250 LOAD_METHOD        21 (decode)
        252 CALL_METHOD        0
        254 CALL_FUNCTION      1
        256 POP_TOP
        258 LOAD_CONST          1 (None)
        260 RETURN_VALUE

33      >> 262 LOAD_NAME          20 (print)
        264 LOAD_NAME          9 (e)
        266 LOAD_METHOD        21 (decode)
        268 CALL_METHOD        0
        270 CALL_FUNCTION      1
        272 POP_TOP
        274 LOAD_CONST          1 (None)
        276 RETURN_VALUE
        >> 278 LOAD_CONST          1 (None)
        280 RETURN_VALUE

```

Disassembly of <code object KSA at 0x7f1199dc7890, file "main.py", line 6>:

```

8      0 LOAD_GLOBAL          0 (len)
        2 LOAD_FAST             0 (key)
        4 CALL_FUNCTION      1
        6 STORE_FAST          1 (keylength)

9      8 LOAD_GLOBAL          1 (list)
       10 LOAD_GLOBAL          2 (range)
       12 LOAD_CONST          1 (256)
       14 CALL_FUNCTION      1
       16 CALL_FUNCTION      1
       18 STORE_FAST          2 (S)

```

```

10      20 LOAD_CONST          2 (0)
        22 STORE_FAST          3 (j)

11      24 LOAD_GLOBAL          2 (range)
        26 LOAD_CONST          1 (256)
        28 CALL_FUNCTION        1
        30 GET_ITER
    >>  32 FOR_ITER              29 (to 92)
        34 STORE_FAST          4 (i)

12      36 LOAD_FAST             3 (j)
        38 LOAD_FAST             2 (s)
        40 LOAD_FAST             4 (i)
        42 BINARY_SUBSCR
        44 BINARY_ADD
        46 LOAD_FAST             0 (key)
        48 LOAD_FAST             4 (i)
        50 LOAD_FAST             1 (keylength)
        52 BINARY_MODULO
        54 BINARY_SUBSCR
        56 BINARY_ADD
        58 LOAD_CONST          1 (256)
        60 BINARY_MODULO
        62 STORE_FAST          3 (j)

13      64 LOAD_FAST             2 (s)
        66 LOAD_FAST             3 (j)
        68 BINARY_SUBSCR
        70 LOAD_FAST             2 (s)
        72 LOAD_FAST             4 (i)
        74 BINARY_SUBSCR
        76 ROT_TWO
        78 LOAD_FAST             2 (s)
        80 LOAD_FAST             4 (i)
        82 STORE_SUBSCR
        84 LOAD_FAST             2 (s)
        86 LOAD_FAST             3 (j)
        88 STORE_SUBSCR
    >>  90 JUMP_ABSOLUTE        16 (to 32)
        92 LOAD_FAST             2 (s)
        94 RETURN_VALUE

```

Disassembly of <code object PRGA at 0x7f1199dc7940, file "main.py", line 16>:

```

17      0 GEN_START            0

18      2 LOAD_CONST            1 (0)
        4 STORE_FAST            1 (i)

19      6 LOAD_CONST            1 (0)
        8 STORE_FAST            2 (j)

20      10 NOP

21    >>  12 LOAD_FAST             1 (i)
        14 LOAD_CONST            3 (1)
        16 BINARY_ADD
        18 LOAD_CONST            4 (256)
        20 BINARY_MODULO

```

	22	STORE_FAST	1 (i)
22	24	LOAD_FAST	2 (j)
	26	LOAD_FAST	0 (S)
	28	LOAD_FAST	1 (i)
	30	BINARY_SUBSCR	
	32	BINARY_ADD	
	34	LOAD_CONST	4 (256)
	36	BINARY_MODULO	
	38	STORE_FAST	2 (j)
23	40	LOAD_FAST	0 (S)
	42	LOAD_FAST	2 (j)
	44	BINARY_SUBSCR	
	46	LOAD_FAST	0 (S)
	48	LOAD_FAST	1 (i)
	50	BINARY_SUBSCR	
	52	ROT_TWO	
	54	LOAD_FAST	0 (S)
	56	LOAD_FAST	1 (i)
	58	STORE_SUBSCR	
	60	LOAD_FAST	0 (S)
	62	LOAD_FAST	2 (j)
	64	STORE_SUBSCR	
24	66	LOAD_FAST	0 (S)
	68	LOAD_FAST	0 (S)
	70	LOAD_FAST	1 (i)
	72	BINARY_SUBSCR	
	74	LOAD_FAST	0 (S)
	76	LOAD_FAST	2 (j)
	78	BINARY_SUBSCR	
	80	BINARY_ADD	
	82	LOAD_CONST	4 (256)
	84	BINARY_MODULO	
	86	BINARY_SUBSCR	
	88	STORE_FAST	3 (K)
19	90	LOAD_FAST	3 (K)
	92	YIELD_VALUE	
	94	POP_TOP	
	96	JUMP_ABSOLUTE	6 (to 12)

Disassembly of <code object RC4 at 0x7f1199dc7aa0, file "main.py", line 26>:

28	0	LOAD_GLOBAL	0 (KSA)
	2	LOAD_FAST	0 (key)
	4	CALL_FUNCTION	1
	6	STORE_FAST	1 (S)
	8	LOAD_GLOBAL	1 (PRGA)
	10	LOAD_FAST	1 (S)
	12	CALL_FUNCTION	1
	14	RETURN_VALUE	

Disassembly of <code object xor at 0x7f1199dd4500, file "main.py", line 30>:

31	0	LOAD_GLOBAL	0 (bytes)
	2	LOAD_GLOBAL	1 (map)
	4	LOAD_CLOSURE	0 (stream)
	6	BUILD_TUPLE	1



```

      8 LOAD_CONST          1 (<code object <lambda> at
Disassembly of <code object <lambda> at 0x7f1199dd5dc0, file "main.py", line 31>)
      8 LOAD_CONST          2 ('xor.<locals>.<lambda>')
     10 LOAD_CONST          8 (closure)
     12 MAKE_FUNCTION
     14 LOAD_FAST           0 (p)
     16 CALL_FUNCTION       2
     18 CALL_FUNCTION       1
     20 RETURN_VALUE

```

Disassembly of <code object <lambda> at 0x7f1199dd5dc0, file "main.py", line 31>:

```

      0 LOAD_FAST           0 (x)
      2 LOAD_DEREF          0 (stream)
      4 LOAD_METHOD         0 (__next__)
      6 CALL_METHOD         0
      8 BINARY_XOR
     10 RETURN_VALUE

```

Disassembly of <code object <dictcomp> at 0x7f1199dd4c90, file "main.py", line 44>:

```

      0 BUILD_MAP           0
      2 LOAD_FAST           0 (.0)
     >> 4 FOR_ITER           9 (to 24)
      6 STORE_FAST         1 (x)
      8 LOAD_FAST           1 (x)
     10 LOAD_FAST           1 (x)
     12 LOAD_GLOBAL         0 (n)
     14 LOAD_FAST           1 (x)
     16 BINARY_SUBSCR
     18 BINARY_XOR
     20 MAP_ADD             2
     22 JUMP_ABSOLUTE     2 (to 4)
     >> 24 RETURN_VALUE

```

Disassembly of <code object <genexpr> at 0x7f1199dd5b00, file "main.py", line 45>:

```

      0 GEN_START           0
      2 LOAD_FAST           0 (.0)
     >> 4 FOR_ITER           9 (to 24)
      6 STORE_FAST         1 (i)
      8 LOAD_FAST           1 (i)
     10 LOAD_METHOD         0 (bit_count)
     12 CALL_METHOD         0
     14 LOAD_FAST           1 (i)
     16 BUILD_TUPLE        2
     18 YIELD_VALUE
     20 POP_TOP
     22 JUMP_ABSOLUTE     2 (to 4)
     >> 24 LOAD_CONST       0 (None)
     26 RETURN_VALUE

```

Disassembly of <code object <lambda> at 0x7f1199a42d90, file "main.py", line 47>:

```

      0 LOAD_FAST           0 (x)
      2 LOAD_CONST          1 (1)
      4 BINARY_SUBSCR
      6 RETURN_VALUE

```

人工手动逆向得到对应 python 代码大概如下

(有些地方没有完全按照字节码来写

python

```
import sys
from hashlib import sha256

w =
b'\xf6\xef\x10H\xa9\x0f\x9f\xb5\x80\xc1\xd\xae\xd3\x03\xb2\x84\xc2\xb4\x0e\xc8\xf
3<\x151\x19\n\x8f'

e = b'$\r9\xa3\x18\xddw\xc9\x97\xf3\xa7\xa8R~'
b = b'geo'

s =
b'}\xce`xbej\xa2\x120\xb5\x8a\x94\x14{\xa3\x86\xc8\xc7\x01\x98\xa3_\x91\xd8\x82
T*v\xab\xe0\xa1\x141'
t = b"Q_\xe2\xf8\x8c\x11M}'<@\xceT\xf6?
_m\xa4\xf8\xb4\xea\xca\xc7:\xb9\xe6\x06\x8b\xeb\xfabH\x85xJ3$\xdd\xde\xb6\xdc\xa
0\xb8b\x961\xb7\x13=\x17\x13\xb1"
m = {2:115, 8:97, 11:117, 10:114}
n = {3:119, 7:116, 9:124, 12:127}

def KSA(key):
    keylength = len(key)

    s = list(range(256))

    j = 0
    for i in range(256):
        j = (j + s[i] + key[i % keylength]) % 256
        s[i], s[j] = s[j], s[i]

    return s

def PRGA(S):
    i = 0
    j = 0
    while True:
        i = (i + 1) % 256
        j = (j + S[i]) % 256
        S[i], S[j] = S[j], S[i]

        K = S[(S[i] + S[j]) % 256]
        yield K

def RC4(key):
    S = KSA(key)
    return PRGA(S)

def xor(p, stream):
    return bytes(map(lambda x:x ^ stream.__next__(), p))

# n = {2:115, 8:97, 11:117, 10:114}
# x:x^n[x] -> <dictcomp>
m |= {x: x^n[x] for x in n}
```

```

m |= ((i.bit_count(), i) for i in b)
stream = RC4(list(map(lambda m:m[1], sorted(m.items()))))
# print welcome banner...
# print(stream)

print(xor(w, stream).decode())
p = sys.stdin.buffer.readline()
e = xor(e, stream)
# print(e)
c = xor(p, stream)

if sha256(c).digest() != s: # error
    print(e.decode())
    exit()

print(xor(t, stream)) # true?

```

大约可以直到，这个地方通过爆破输入字符的长度，得到  $t$  的真实数据

可以发现，输入长度为 26 的时候，会提示说 `Congratulations! Now you should now what the flag is`，这个就是  $t$  的解密结果。而其他情况都不能正确解码。

于是就去哪里还有这个输入。

然后发现用 pyc 隐写了一部分内容，使用脚本 stegosaurus 导出 pyc 隐写。

[一文让你完全弄懂Stegosaurus](#)

<https://github.com/AngelKitty/stegosaurus>

需要魔改一下 header，python 3.10 长度是 16.

```

119
120
121 def _loadBytecode(carrier, logger):
122     try:
123         f = open(carrier, "rb")
124         header = f.read(16)
125         code = marshal.load(f)
126         logger.debug("Read header and bytecode from carrier")
127     finally:
128         f.close()
129
130     return (header, code)
131

```

另外输出的话不用转 str，直接 bytes 就好了。

```

78
79 def _extractPayload(mutableBytecodeStack, explodeAfter, logger):
80     payloadBytes = bytearray()
81
82     for bytes, byteIndex in _bytesAvailableForPayload(mutableBytecodeStack, explodeAft
83         byte = bytes[byteIndex]
84         if byte == 0:
85             break
86         payloadBytes.append(byte)
87
88     # payload = str(payloadBytes) # , "utf-8"
89     payload = payloadBytes # , "utf-8"
90
91     print("Extracted payload: {}".format(payload))
92

```

```

root@61034719bc70:/ctf# python3 decode.pyc^C
root@61034719bc70:/ctf# cd stegosaurus/
root@61034719bc70:/ctf/stegosaurus# python3 stegosaurus.py -x ../decode.pyc
Extracted payload: bytearray(b'\xe5\n2\xd6"\xf0}I\xb0\xcd\xa2\x11\xf0\xb4U\x166\xc5o\xdb\xc9\xead\x04\x15b')
root@61034719bc70:/ctf/stegosaurus#

```

得到长度为 26 的 bytes

python

```
b'\xe5\n2\xd6"\xf0}I\xb0\xcd\xa2\x11\xf0\xb4U\x166\xc5o\xdb\xc9\xead\x04\x15b'
```

最后将这个作为输入，然后让上述代码的 `c` 打印出来，即为 flag

```
flag{P0w5rFu1_0pEn_50urcE}
```