HGAME week1

目录

Pwn24 pwn step0

题目描述： nc 121.42.25.113 10000
binary:http://7xn9bv.dl1.z0.glb.clouddn.com/pwn0.zip
step0 是一个很简单的 stack overflow 原理展示，just read the disassembly code :)
Hint:所谓的 pwn 嘛，你得分析包含漏洞的服务端程序，然后构造特殊的请求来获取 shell 或达到其它目的。step0 完全可以手打 √

文件用 32 位 ida 打开，F5 反汇编后主函数中只有一个关键函数。

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  setbuf(stdin, 0);
  setbuf(stdout, 0);
  setbuf(stderr, 0);
  puts("so, can you find flag?");
  foo(0x12345678);
  return 0;
}
```

跟进 foo 函数，没有 canary，gets 栈溢出可直接覆盖。至少输入 40 个 a，覆盖原 a1 值，使其等于 aaaa，便可得到 flag。

```
// a1 = 0x12345678
int __cdecl foo(int a1)
{
  int result; // eax@1
  char s; // [sp+Ch] [bp-1Ch]@1

  gets(&s);
  result = puts(&s);
  if ( a1 == 0x61616161 )                    // aaaa
    result = getFlag();
  return result;
}
```

Windows 下用 xp 虚拟机的 cmd，输入 telnet 121.42.25.113 10000 远程登录。

```
so, can you find flag?
                    aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
hctf{Pwn_1s_1nteRest1ng}
```

Re23 re?

题目描述: 逆向不止是汇编 FILE:http://7xn9bv.dl1.z0.glb.clouddn.com/hgame.jar

有很多 java 反编译工具，比如 jeb, jd 等，反编译后拿到源码，发现是一道 AES/CBC/PKCS5 解密的题。

由于 CBC 是一种对称加密，加密解密的密钥可从源码中得到。key 和 iv 都是图片中相应位置的比特段，位置在源码里有写，后面是调用 java 自带的 aes 加密函数。

（一）Java 解密

不需要什么解密脚本，直接修改 3 个地方调用 java 自带的解密脚本，编译链接后直接运行即可。

```
Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
    SecretKeySpec skey = new SecretKeySpec(key, "AES");
    IvParameterSpec ivSpec = new IvParameterSpec(iv);
    cipher.init(2, skey, ivSpec);   // 1为aes加密，2为解密
    byte[] en = cipher.doFinal(tFlag);
    byte[] tFlag = { 69, -101, 74, -127, -13, 110, 17, -103, 112, -111, -87, 87, 45, -110, 38, -11 };
    if (Arrays.equals(en, tFlag)) {
      System.out.println(new String(en));
    } else {
      System.out.println("try again");
    }
```

（二）Python 脚本解密

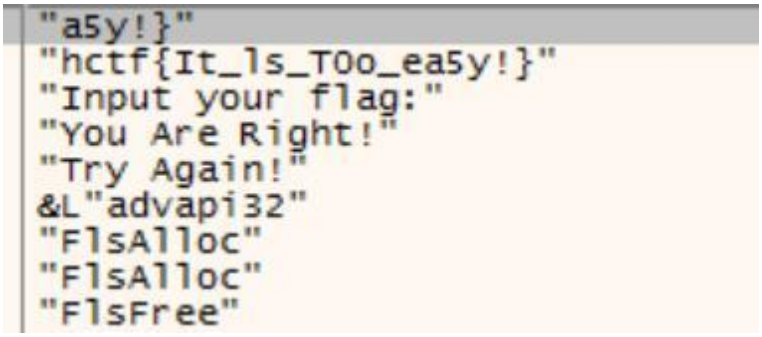
这里遇到了一个坑：在 java 中是没有无符号变量的，所以密文存在负数。用 python 解密时需要给负数加上一个 256。即 java 的 ASCII 范围在 -128-127，而 python 在 0-255。

```
#encoding:utf-8
from Crypto.Cipher import AES
fp = open("ctf.jpg","rb")
data = fp.read(10000)
key = fp.read(16)
fp.read(10000)
iv = fp.read(16)
print key
print iv
fp.close()
mode = AES.MODE_CBC
encryptor = AES.new(bytes(key), mode, bytes(iv))
text = [69,-101,74,-127,-13,110,17,-103,112,-111,-87,87,45,-110,38,-11]
rtext = []
for a in text:
    if a<0:
        a+=256
    rtext.append(a)
data = ''.join([chr(a) for a in rtext])
print data
ciphertext = encryptor.decrypt(data)
print ciphertext
```

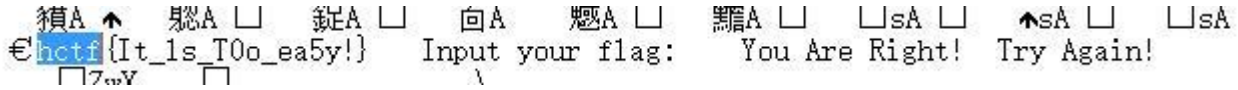
Re28 你看看，逆向多简单！

题目描述： 链接：http://pan.baidu.com/s/1kVrzylx 密码：bxmb

打开文件要求 Input your flag:
（一）od 打开直接查找字符串；



（二）文件后缀改为 txt，查找字符串。



Re29 蛤，这是啥？

题目描述： 链接: http://pan.baidu.com/s/1qXKLkEK 密码: hcwn
Hint: 某算法的魔改版

pyc 是 py 文件经过编译后生成的跨平台字节码文件，是用来保存 python 虚拟机编译生成的 byte code 的。其加载的速度比 py 文件有所提高，而且还可以实现源码隐藏，以及一定程度上的反编译。pyc 的内容，跟 python 的版本有关，不同版本编译后的 pyc 文件是不同的。
pyc 文件解密链接：http://tool.lu/pyc/

Base32 包含字符：A-Z、2-7
加密规则：二进制数据 5 位(bit)一组切分编码成 1 个可见字符,每组的二进制串不足 5 个用 0 补充。计算每组二进制串所对应的十进制，参考 Base32 编码表，找出所对应的编码字符，组合成密文。最后一个分组位数不足 4 个的时候，则用字符“=”编码。

pyc 文件解密后代码中编码表的字符为 A-Z 及 2-7，并且 code.txt 文件内容为 nBRxIZT3mJQxgZK7gmZCC7I= 很容易联想到 Base32 加密。

```
    str_len_mod5 = len(a) % 5
    bin_of_str = ''
    for c in a:
        bin_of_chr = bin(ord(c))[2:]    #字符转二进制，删0b
        length = len(bin_of_chr)
        bin_of_str += '0' * (8 - length) + bin_of_chr    #用0补足8位

    #最后一组位数不足，补位
    if str_len_mod5 == 1:
        extra_zero = '00'
        extra_equal = '======'
    elif str_len_mod5 == 2:
        extra_zero = '0000'
        extra_equal = '===='
    elif str_len_mod5 == 3:
        extra_zero = '0'
        extra_equal = '==='
    elif str_len_mod5 == 4:
        extra_zero = '000'
        extra_equal = '='
    bin_of_str += extra_zero
    continue

    #5位一组
    five_slice = [bin_of_str[i:i + 5] for i in range(0, len(bin_of_str), 5)
    output = ''
    for outchar in five_slice:
        alplabet_ord = int(outchar, 2)  #编号
        output += haihiahia[alplabet_ord]
```

比对后发现为改编的 Base32 加密，仅仅修改了编码表字符中的大小写，可将密文直接改为大写 NBRXIZT3MJQXGZK7GMZCC7I=，进行 Base32 解密。

```
>>> import base64
>>> s = 'NBRXIZT3MJQXGZK7GMZCC7I='
>>> a = base64.b32decode(s)
>>> print a
hctf{base_32!}
```

Re35 奇怪的代码

题目描述： fuckasm
http://ojwp3ihl4.bkt.clouddn.com/re.exe
Hint: 部分思路来自：https://github.com/xoreaxeaxeax/movfuscator，但比这个要简单的多

ida 打开直接 F5 反编译找到加密函数。加密是对输入字符串 Buf 分为 4 组分别加密的。

```
fgets(Buf, 33, v3);
sub_401000();
v4 = 0;
v8 = 0;
do
{
  memset(byte_413D88, 0, 0x100u);
  memset(byte_413C88, 0, 0x100u);
  memset(&unk_413E88, 0, 0x100u);
  memset(&unk_413B88, 0, 0x100u);
  v5 = encode(                                  // 输入字符串分4组加密
         v8,                                    // 循环变量
         *&Buf[4 * v8],
         *&Buf[4 * v8 + 1],
         *&Buf[4 * v8 + 2],
         *&Buf[4 * v8 + 3]);
  if ( v5 + (v6 << 16) != 0x1010101 )           // 0x1010101代表1、2、3、4位正确
  {
    printf(off_403824);                         // fail
    exit(0);
  }
  ++v4;
  v8 = v4;
}
while ( v4 < 8 );
printf(off_403820);                             // sucessful
```

跟进加密函数，重点大概在 c0、c1、c2、c3 这部分。第一个 t 数组大小为 35536(0xFF*0xFF)，值和下标的映射关系为 t[i] = i / 256 ^ i % 256，将 c0、c1、c2、c3 简化如图。

```
int __cdecl sub_401050(int a1, unsigned __int8 a2, unsigned __int8 a3, unsigned __int8 a4, unsigned __int8 a5)
{
  int v5; // eax@1
  int v6; // edx@1
  int v7; // edx@1
  int result; // eax@1

                                                // t[i] = i / 256 ^ i % 256
                                                // tx数组值为下标乘x
                                                // eg: t0xFF[i] = 256 * i
  c0 = t[*(&t1[t0xFFmul4[*(off_403018 + a1)]] + t4[a2])];// c0 = 0x11 * (offset + 1) ^ c0;
  c1 = t[*(&t1[t0xFFmul4[c0]] + t4[a3])];       // c1 = b0 ^ c1;
  c2 = t[*(&t1[t0xFFmul4[c1]] + t4[a4])];       // c2 = b1 ^ c2;
  c3 = t[*(&t1[t0xFFmul4[c2]] + t4[a5])];       // c3 = b2 ^ c3;
  byte_413D88[c0] = 1;
  v5 = t0xFFmul4[byte_413D88[*(off_40301C + t4[a1])]];
  byte_413C88[c1] = 1;
  v6 = t1[t4[a1] + 1];
  LOBYTE(v5) = byte_413C88[*(off_40301C + v6)];
  dword_413B80 = v5;
  byte_413D88[c2] = 1;
  v7 = t1[v6 + 1];
  result = t0xFFmul4[byte_413D88[*(off_40301C + v7)]];
  byte_413D88[c3] = 1;
  LOBYTE(result) = byte_413D88[*(off_40301C + t1[v7 + 1])];
  return result;
}
```

为什么不加后面的呐……因为后面的已经无关紧要了。稍微调试一下可以看出，如果前几个输入的是 hgame{，hgam 的加密可通过第一次循环，结果又恰好与 unk_4020F4 中的值相同。其后的过程

都是用来校验的，只不过是被稍微混淆过而已，所以只需要反向解密 unk_4020F4 的值即可。

```
unk_4020F4      db  79h ; y
                db  1Eh
                db  7Fh ; ■
                db  12h
                db  47h ; G
                db  3Ch ; <
                db  55h ; U
                db  26h ; &
                db  6Ch ; l
                db    5
                db  71h ; q
                db  2Eh ; .
                db  2Dh ; -
                db  43h ; C
                db  37h ; 7
                db  52h ; R
                db  27h ; '
                db  54h ; T
                db  20h
                db  49h ; I
                db    8
                db  6Fh ; o
                db  30h ; 0
                db  44h ; D
                db  18h
                db  47h ; G
                db  2Ah ; *
                db  45h ; E
                db 0E7h ;
                db  91h ;
                db 0AEh ;
                db 0D3h ;
```

解密脚本：

```python
unk_4020F4 = (
    0x79, 0x1e, 0x7f, 0x12, 0x47, 0x3c, 0x55, 0x26,
    0x6c, 0x05, 0x71, 0x2e, 0x2d, 0x43, 0x37, 0x52,
    0x27, 0x54, 0x20, 0x49, 0x08, 0x6f, 0x30, 0x44,
    0x18, 0x47, 0x2a, 0x45, 0xe7, 0x91, 0xae, 0xd3,
)

def decrypt(offset):
    c0, c1, c2, c3 = unk_4020F4[offset << 2 : (offset << 2) + 4]
    m0 = chr(c0 ^ 17 * (offset + 1))
    m1 = chr(c1 ^ c0)
    m2 = chr(c2 ^ c1)
    m3 = chr(c3 ^ c2)
    print(m0, m1, m2, m3)

for i in range(8): decrypt(i)
```

Re36 奇怪的 linux 逆向

题目描述： 逆向不只是 windows ？
http://ojwp3ihl4.bkt.clouddn.com/easy_linux Hint:
1.你可能会用到的工具：readelf，objdump，gdb，ida
2.听说 ida 能手动加载？

大家好我是出题人（
这次出了一个很简单的 ELF，汇编纯手写+复制粘帖：
源码：

```
int main(void){
    int size;
    char code[]=
        "jmp caller\n\
    callee:\n\
        pop ebx\n\
        xor ecx, ecx\n\
        mov cl, [ebx]\n\
        inc ebx\n\
        mov [ebx+ecx], ch\n\
    a:\n\
        xor edi, edi\n\
        xor ecx, ecx\n\
        lea eax, [edi+5]\n\
        int 0x80\n\
        xchg ebx, eax\n\
        xchg ecx, eax\n\
        lea edx, [edi+1]\n\
        lea eax, [edi+0x59]\n\
    loop:\n\
        pushad\n\
        int 0x80 \n\
        xchg esi, ecx\n\
        test eax, eax\n\
        je exit\n\
        mov dx, [esi+8]\n\
        lea ecx, [esi+10]\n\
        mov byte ptr [ecx+edx], 0x0\n\
        inc edx\n\
        mov edx,[ecx]\n\
        cmp edx,0x766572\n\
        jz entry\n\
        popad\n\
        jmp loop\n\
    exit:\n\
```

```
    mov eax,1\n\
    int 0x80\n\
entry:\n\
    mov eax,5\n\
    mov ebx,ecx\n\
    mov ecx,0\n\
    mov edx,0\n\
    int 0x80\n\
    mov ebx,eax\n\
    mov eax,0x3\n\mov ecx,0xbeef080\n\
    mov edx,0x20\n\
    int 0x80\n\
    mov ecx,0x19\n\
loop2:\n\
    mov edi,0xbeef152\n\
    mov ebx,0xbeef080\n\
    add edi,ecx\n\
    add ebx,ecx\n\
    mov dl,BYTE PTR[edi]\n\
    mov dh,BYTE PTR[ebx]\n\
    cmp dl,dh\n\
    jnz exit\n\
    dec ecx\n\
    test ecx,ecx\n\
    jz get_flag\n\
    jmp loop2\n\
get_flag:\n\
    mov eax,0xbeef152\n\
    mov ebx,[eax]\n\
    xor ebx,0xfdce80e0\n\
    mov [eax],ebx\n\
    mov ebx,[eax+4]\n\
    xor ebx,0x96bda8c5\n\
    mov [eax+4],ebx\n\mov ebx,[eax+8]\n\
    xor ebx,0xdf95d3bc\n\
    mov [eax+8],ebx\n\
    mov ebx,[eax+12]\n\
    xor ebx,0x96b4efd8\n\
    mov [eax+12],ebx\n\
    mov ebx,[eax+16]\n\
    xor ebx,0xd7faefec\n\
    mov [eax+16],ebx\n\
    mov ecx,0xbeef152\n\
    mov eax,0x4\n\
    mov ebx,0x1\n\
    mov edx,0x20\n\
    int 0x80\n\
```

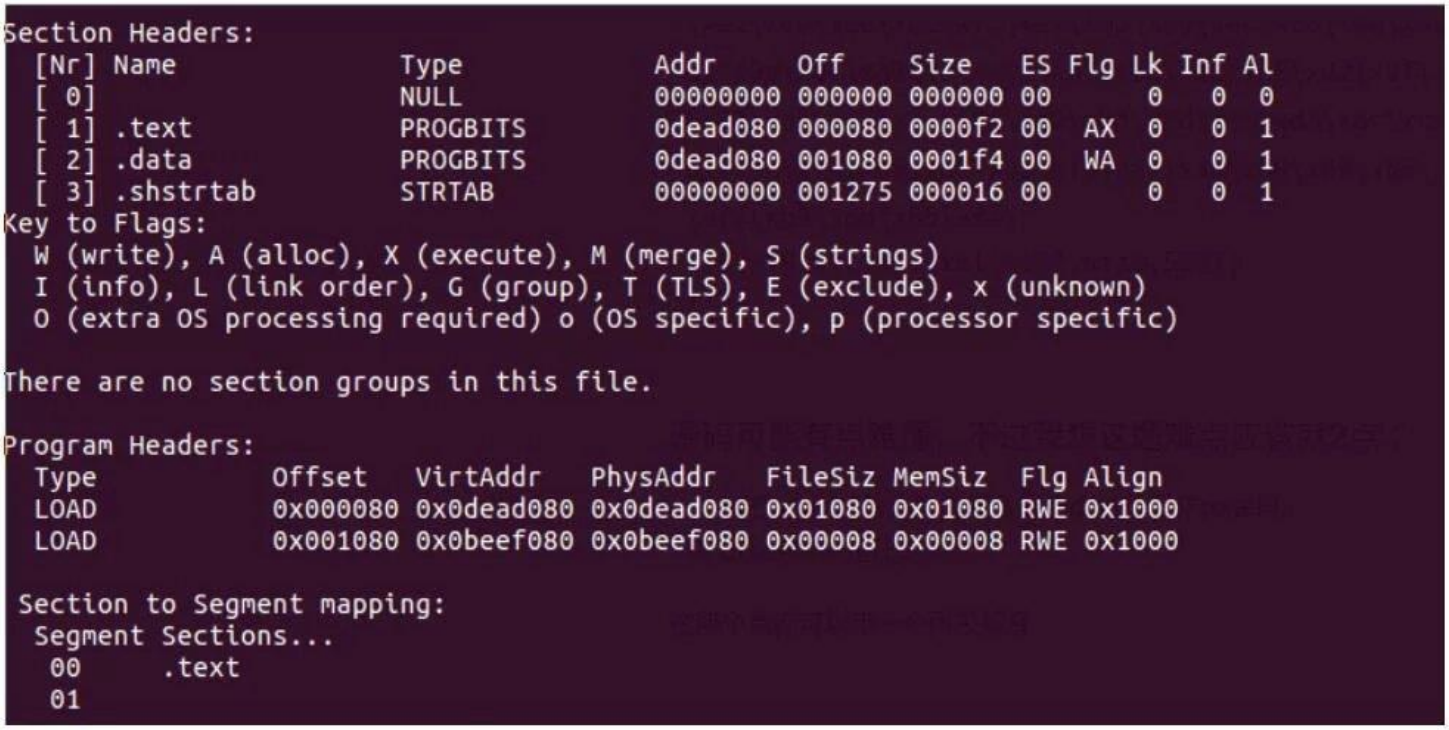
```
        jmp exit\n\
    caller:\n\
        call callee\n\
    arg:\n\
            .byte 2\n\
            .ascii \"./\"\n\
            ";
    char* opcode = make_opcode(code,&size);


create_elf(opcode,"\x91\x9c\xa6\xbe\xd2\xba\xcc\xa9\xfc\xd9\xb7\xf6\xa7\xd5\xe3\xdf\xfa
\xde\xf4\xaf\xd9\xe4\xec\x8c\x97\xa1\xd5\x9c\xeb\xdd\x95\xbb\xcc\xca\x9c\xfc\x90\xb6\x8
7\xa8\xc1\xa2\xef\xd8\xc0\xf7\xe0\x8d\x95\xe2\x98\xf0\x93\xe1\xf8\x9e\xc1\x89\xc1\xae\x
e3\xdb\xc6\xe3\xab\x99\xb9\xb0\xac\xbc\x9e\x8d\x9a\xb5\x92\xbb\xbf\x81\xf6\xd6\xf4\xd0\
xd2\xcc\xb4\xe7\xb2\xd1\x9a\x95\xd9\xec\x85\x80\x8b\xc4\x82\xda\xd4\x90\xf1\x9e\xbe\xb1
\xda\xc8\x92\xba\xb8\x88\xf8\xa6\xce\x80\x9f\xcf\xd1\xd5\xcb\xd5\xfe\x8a\xaf\xfb\xbd\x9
2\x9b\xe7\xc4\xaa\xb3\xab\x94\xf5\xf1\xa3\xba\xa9\xb1\xc6\x8c\xf5\xae\xde\xb4\xa7\xa1\x
ab\xc6\xdb\x81\xf8\xac\x93\xd6\xf9\xc4\xa8\x8b\xac\xda\xdf\x87\xaa\xcd\xdb\xd4\xa5\xd6\
xed\xaa\x9f\xb0\xc9\xaa\xc8\xcf\xb3\xf4\xc9\x8f\x96\xc7\xce\x8e\x8a\xf9\x91\xf8\xf9\xfb
\xc9\x82\xda\xd3\xa8\xd0\x83\xf4\xf5\xca\x92\xc7\xad\xc5\xd3\x8e\xfb\xf5\xb6\x88\xe7\xa
f\x90\xa0\xd3\x8e\xf7\xcf\xaa\xca\xee\xb1\x81\xc1\xee\x91\xef\xfa\xd7\xdd\x84\xe9\xd0\x
a7\x91\x9f\xe5\xe8\xd4\x97\xfd\x85\xde\xc7\xe4\xcd\xa7\xbf\xb2\xb8\x97\x91\xbd\xae\xac\
xa9\xa2\xf2\xd1\x9b\x9f\xcc\xd0\xc8\x83\x82\xd7\xe7\xe6\xb8\xcc\xbe\xe2\xa9\x93\xcc\xeb
\xe7\xc0\x91\x8d\xf2\xfc\xc4\xc1\xbc\xc5\xf0\xbb\xdf\xbd\x86\xbc\x8a\xb5\xf3\xe2\x81\xe
1\xb7\xc0\xd7\xb3\xba\xc9\xb9\xef\xd6\xb0\x8d\xf5\xc0\xd6\x8f\xe9\xeb\xe3\xe5\x99\x91\x
f6\xc6\xb6\x97\xe0\xb6\xa3\x88\x95\xb8\xe4\xb4\xd1\xb5\xbf\xd6\xe3\xce\xfb\xd2\xc9\xfd\
xfe\xb1\xec\x9d\xac\x93\x98\xd6\xcd\xe8\xa8\xdd\x84\xb1\xa8\xde\xa3\x8a\xb8\x90\xfc\xf0
\x97\xaf\xd0\x81\xcc\xf0\xd8\x9d\xe1\xcf\x8c\xe1\x9f\xd4\xe1\xd6\xcd\x89\xfb\xf0\x85\xb
3\xbd\x81\xcc\x86\xdf\x84\x90\xa5\x82\xdb\xde\xcb\x92\xc1\x89\xce\xc3\xe1\xbd\x85\xb2\x
8f\xb1\xde\x92\xe2\x98\xa7\x8a\xc5\x9e\x92\xd5\x9a\x93\xa2\xa2\x86\xac\xd5\xfb\xa7\xb5\
xe9\x8c\xc8\x9b\xbe\xef\xbb\xe9\xee\xd1\x9e\x87\x83\x84\xf4\xb7\xbe\xc1\xe5\xd2\xa7\x8f
\x9a\xf5\xe2\x8d\x93\xa3\x95\xec\x94\xa0\xbd\x8d\xf6\xd3\xac\xc3\x8d\xea\x81\xa9\x82\xa
c\xfa\xb5\x8e\x85\xf3\xa7\xa8\x89\xe4\xb0\xbe\x97\xd7\xf8\xc6\xf9\xb9\x8d\x85\x84\x88\x
f9\xfc\xf7\xe1\xe3",size,500);
}
```

源码可能有点难懂，不过我想这题难点应该就 2 点：

1. elf 在 ida 默认载入的时候会显示 data 段而非 code 段。

2. 0xbeef152 是什么地方

这两个点都可以用一个图来解释：



```
Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .text             PROGBITS        0dead080 000080 0000f2 00  AX  0   0  1
  [ 2] .data             PROGBITS        0dead080 001080 0001f4 00  WA  0   0  1
  [ 3] .shstrtab         STRTAB          00000000 001275 000016 00      0   0  1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings)
  I (info), L (link order), G (group), T (TLS), E (exclude), x (unknown)
  O (extra OS processing required) o (OS specific), p (processor specific)

There are no section groups in this file.

Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  LOAD           0x000080 0x0dead080 0x0dead080 0x01080 0x01080 RWE 0x1000
  LOAD           0x001080 0x0beef080 0x0beef080 0x00008 0x00008 RWE 0x1000

 Section to Segment mapping:
  Segment Sections...
   00     .text
   01
```

Progrom headers 内是两个段的真实加载地址，而 Section header 内是伪造的段地址。
那么这下所有问题都解决了。

程序在干什么呢？
1.读取当前目录查找名为 rev 的文件;
2.从文件中读取 0x20 个字节 3. 和 data 段内的一块地址进行对比，正确的话输出 flag.

最简单的做法是什么呢？
看到源码那个大大的 get_flag 没有？gdb 加载后，set $eip 即可