CTF-攻防世界 Reverse新手练习解析

原创

F1ash000 于 2019-08-20 03:25:04 发布 10457 收藏 48

分类专栏： # CTF 信息安全 文章标签： CTF Reverse 新手 小白

版权声明：本文为博主原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接和本声明。
本文链接：https://blog.csdn.net/weixin_42621117/article/details/99768988
版权

CTF 同时被 2 个专栏收录

2 篇文章 1 订阅

订阅专栏

信息安全

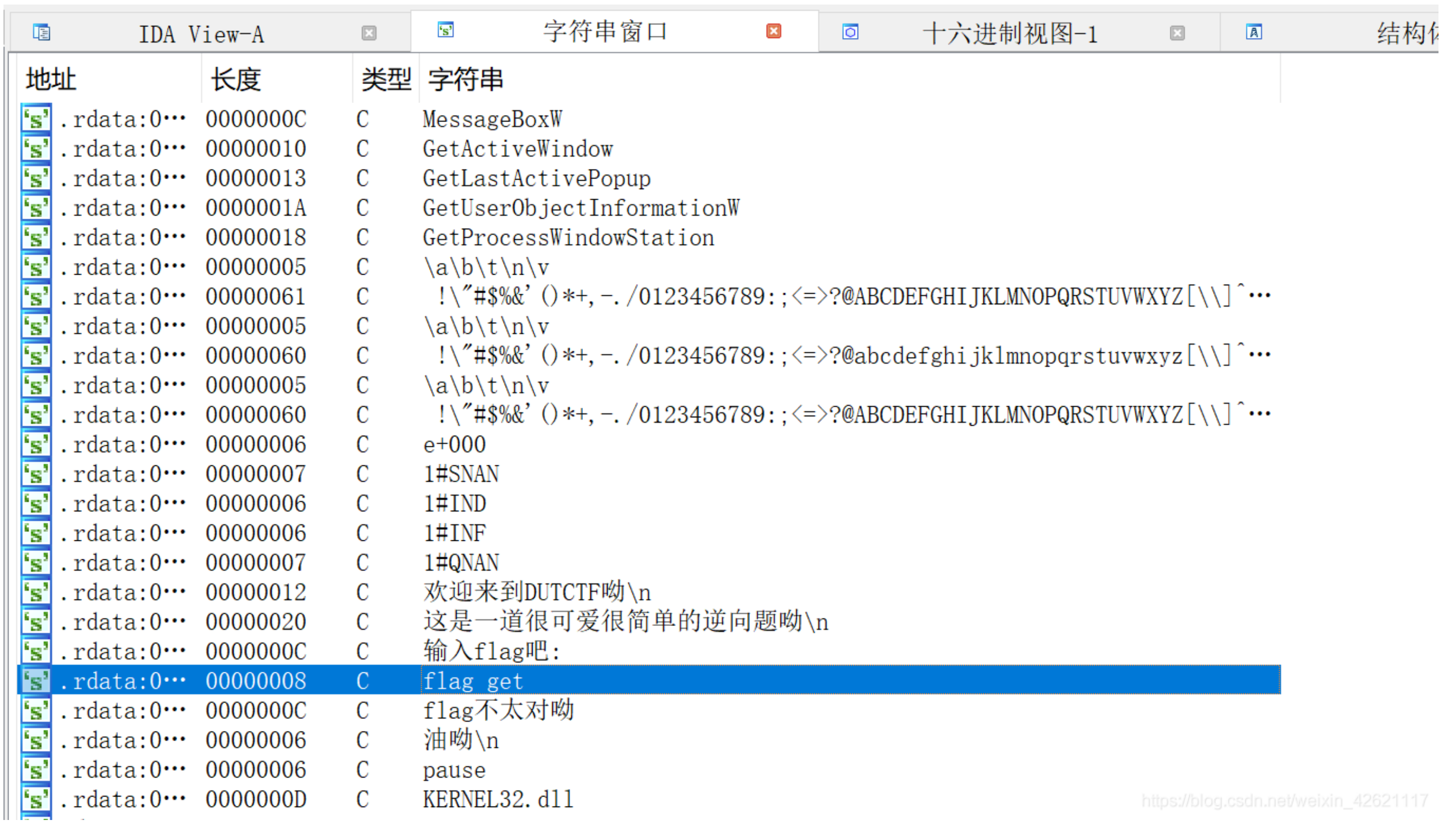
3 篇文章 0 订阅

订阅专栏

博主也是CTF小白，入门ing。。。方向是RE＋PWN。文章可能多有纰漏，但会持续更新更正。希望大家多多指出不足之处。

0x1. re1

解析：
这道题很简单：
打开.exe随便输点东西进去，发现不对，退出。
用IDA打开，点到十六进制视图
点IDA视图–打开子视图–字符串（英文版IDA应该就是view这种常见的单词）。或者直接按shift+F12。然后在一大堆东西中找到这个：flag get



地址	长度	类型	字符串
.rdata:0···	0000000C	C	MessageBoxW
.rdata:0···	00000010	C	GetActiveWindow
.rdata:0···	00000013	C	GetLastActivePopup
.rdata:0···	0000001A	C	GetUserObjectInformationW
.rdata:0···	00000018	C	GetProcessWindowStation
.rdata:0···	00000005	C	\a\b\t\n\v
.rdata:0···	00000061	C	!\"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\\]^···
.rdata:0···	00000005	C	\a\b\t\n\v
.rdata:0···	00000060	C	!\"#$%&'()*+,-./0123456789:;<=>?@abcdefghijklmnopqrstuvwxyz[\\]^···
.rdata:0···	00000005	C	\a\b\t\n\v
.rdata:0···	00000060	C	!\"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\\]^···
.rdata:0···	00000006	C	e+000
.rdata:0···	00000007	C	1#SNAN
.rdata:0···	00000006	C	1#IND
.rdata:0···	00000006	C	1#INF
.rdata:0···	00000007	C	1#QNAN
.rdata:0···	00000012	C	欢迎来到DUTCTF呦\n
.rdata:0···	00000020	C	这是一道很可爱很简单的逆向题呦\n
.rdata:0···	0000000C	C	输入flag吧:
.rdata:0···	00000008	C	flag get
.rdata:0···	0000000C	C	flag不太对呦
.rdata:0···	00000006	C	油呦\n
.rdata:0···	00000006	C	pause
.rdata:0···	0000000D	C	KERNEL32.dll

https://blog.csdn.net/weixin_42621117

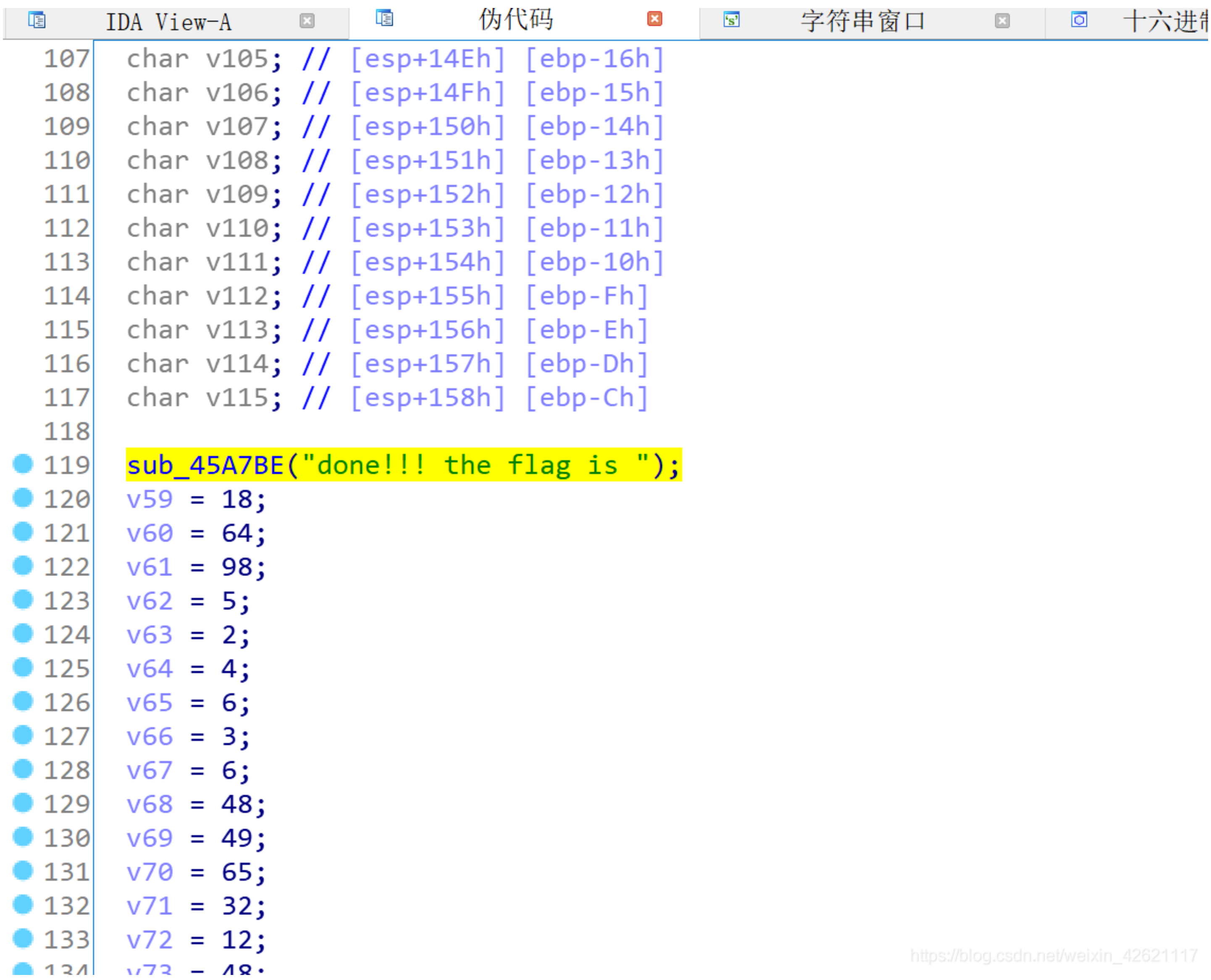
英文应该都能看懂吧……这就是关键
双击，跳到十六进制视图窗口就可轻松获得flag：DUTCTF{We1c0met0DUTCTF}

0x2.game

n是灯的序列号，m是灯的状态
如果第N个灯的m为1，则它打开，如果不是，则关闭
起初所有的灯都关闭了
现在您可以输入n来更改其状态
但是你应该注意一件事，如果改变第N盏灯的状态，第（N-1）和第（N＋1）的状态也会改变
当所有灯都亮起时，将出现标志
现在，输入n
（来自谷歌翻译……QAQ）

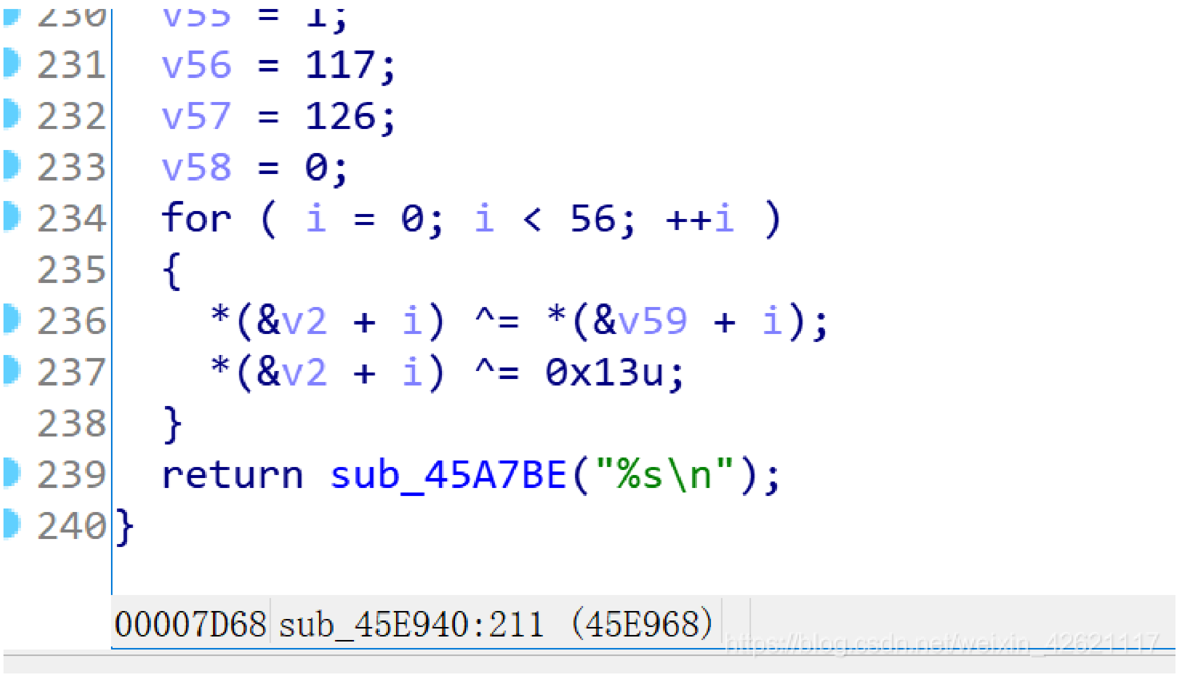
解析：
依旧什么都不用管，直接拖到IDA打开
shift+F12
Alt+T（搜索字符串），搜索： flag
直接跳出来：done!!!the flag is
双击，跳到IDA View-A（这里说一下，字符串窗口双击跳转的窗口是打开字符串窗口时停留的窗口。也就是说，当你页面停在IDA View-A时，你打开了字符串窗口，那在字符串窗口双击，就跳转到IDA View-A）
Ctrl+X（交叉引用）
F5（生成伪代码）
如下图

如下图



```
char v105; // [esp+14Eh] [ebp-16h]
char v106; // [esp+14Fh] [ebp-15h]
char v107; // [esp+150h] [ebp-14h]
char v108; // [esp+151h] [ebp-13h]
char v109; // [esp+152h] [ebp-12h]
char v110; // [esp+153h] [ebp-11h]
char v111; // [esp+154h] [ebp-10h]
char v112; // [esp+155h] [ebp-Fh]
char v113; // [esp+156h] [ebp-Eh]
char v114; // [esp+157h] [ebp-Dh]
char v115; // [esp+158h] [ebp-Ch]

sub_45A7BE("done!!! the flag is ");
v59 = 18;
v60 = 64;
v61 = 98;
v62 = 5;
v63 = 2;
v64 = 4;
v65 = 6;
v66 = 3;
v67 = 6;
v68 = 48;
v69 = 49;
v70 = 65;
v71 = 32;
v72 = 12;
v73 = 48;
```

https://blog.csdn.net/weixin_42621117



```
v55 = 1;
v56 = 117;
v57 = 126;
v58 = 0;
for ( i = 0; i < 56; ++i )
{
  *(&v2 + i) ^= *(&v59 + i);
  *(&v2 + i) ^= 0x13u;
}
return sub_45A7BE("%s\n");
}
```

```
00007D68 sub_45E940:211 (45E968)
```

https://blog.csdn.net/weixin_42621117

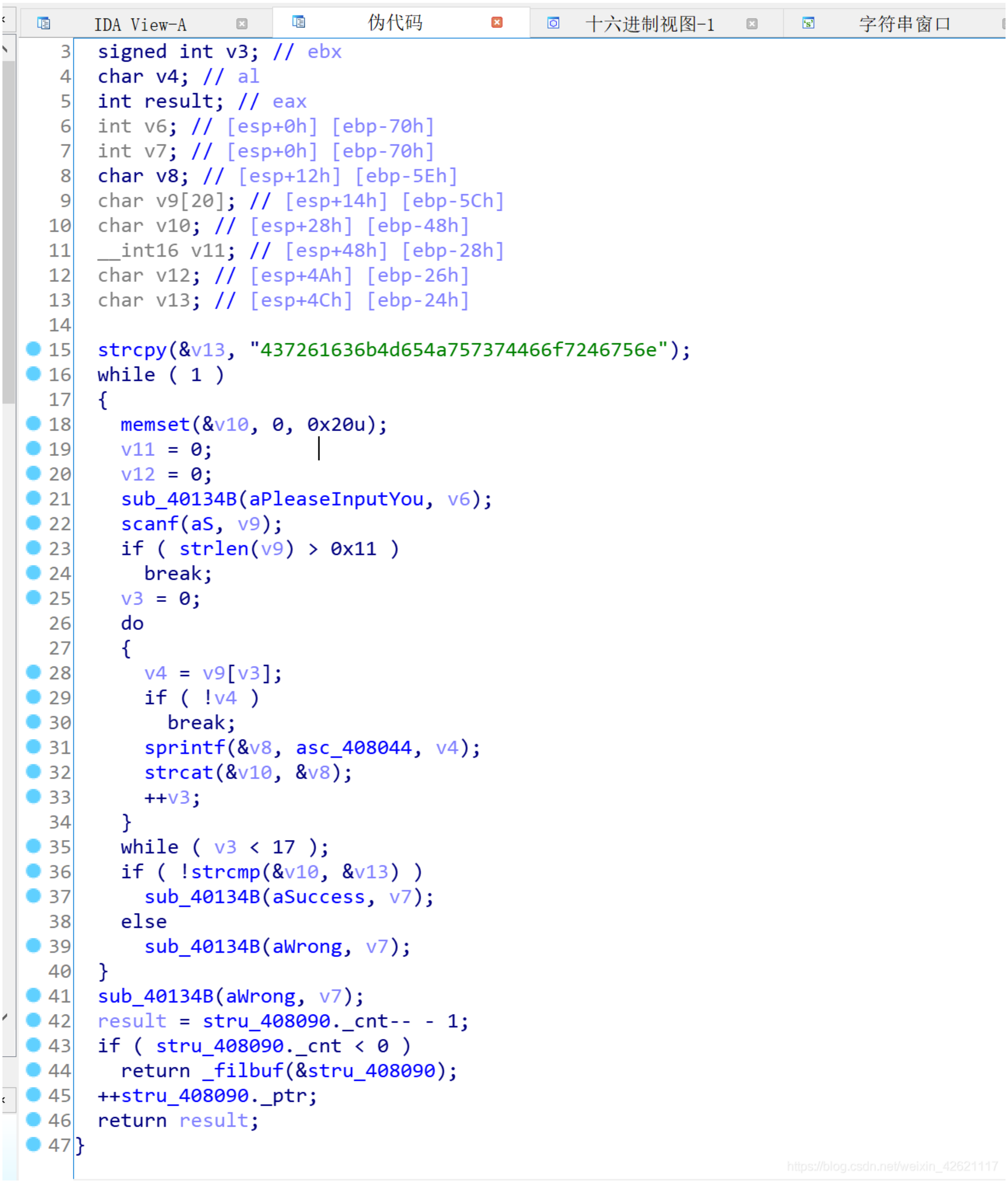
这里我们就初步接触到了逆向的加解密，加解密其实也就是算法的使用。这里加密比较简单，甚至都不能称为加密。*(&v2 + i)的值练起来就是flag的值
所以得到解密代码：（博主使用python，其他语言均可）

```
#v2：原代码v2-v58的值
v2 = [123,32,18,98,119,108,65,41,124,80,125,38,124,111,74,49,83,108,94,108,84,6,96,83,44,121,104,110,32,95,117,101,99,123,127,119,96,48,107,71,92,29,81,107,90,85,64,12,43,76,86,13,114,1,117,126,0]

#v59：原代码v59-v115的值
v59 = [18,64,98,5,2,4,6,3,6,48,49,65,32,12,48,65,31,78,62,32,49,32,1,57,96,3,21,9,4,62,3,5,4,1,2,3,44,65,78,32,16,97,54,16,44,52,32,64,89,45,32,65,15,34,18,16,0]

s = ""

for i in range(57):
    v2[i] = v2[i] ^ v59[i]
    v2[i] = v2[i] ^ 19
    s += chr(v2[i])

print(s)
```

得到flag：zsctf{T9is_tOpic_1s_v5ry_int7resting_b6t_others_are_n0t}

0x3.Hello,CTF

解析：
老办法，遇到.exe直接打开看看是啥玩意儿。随便输，发现会弹出来wrong，输出巨长字符串后会直接退出
拖到IDA打开，shift+F12
发现和我们的程序中有一个东西是匹配的：“wrong!\n”，关键点get到!
双击进IDA View-A，Ctrl+X，F5



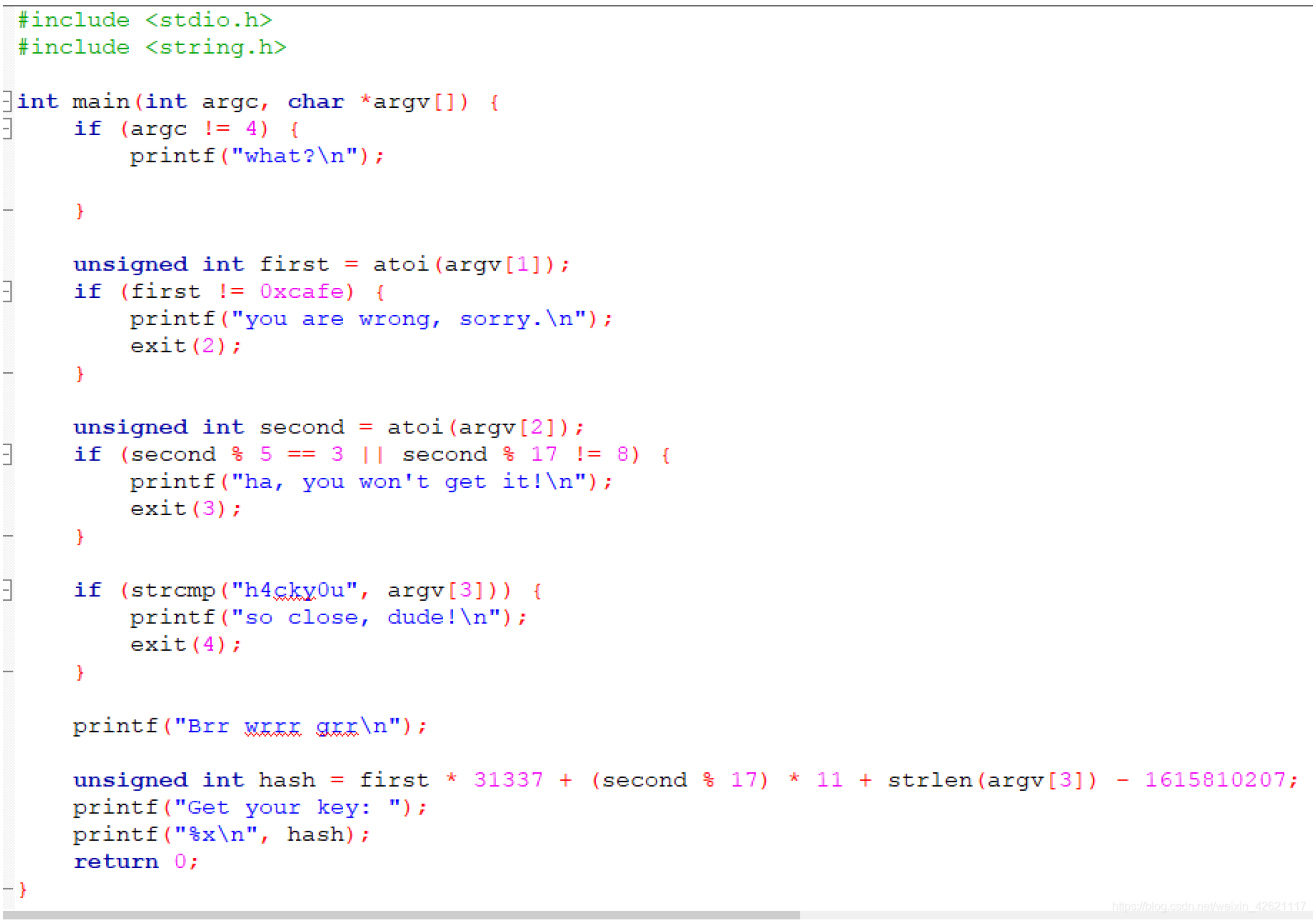
```
  signed int v3; // ebx
  char v4; // al
  int result; // eax
  int v6; // [esp+0h] [ebp-70h]
  int v7; // [esp+0h] [ebp-70h]
  char v8; // [esp+12h] [ebp-5Eh]
  char v9[20]; // [esp+14h] [ebp-5Ch]
  char v10; // [esp+28h] [ebp-48h]
  __int16 v11; // [esp+48h] [ebp-28h]
  char v12; // [esp+4Ah] [ebp-26h]
  char v13; // [esp+4Ch] [ebp-24h]

  strcpy(&v13, "437261636b4d654a757374466f7246756e");
  while ( 1 )
  {
    memset(&v10, 0, 0x20u);
    v11 = 0;
    v12 = 0;
    sub_40134B(aPleaseInputYou, v6);
    scanf(aS, v9);
    if ( strlen(v9) > 0x11 )
      break;
    v3 = 0;
    do
    {
      v4 = v9[v3];
      if ( !v4 )
        break;
      sprintf(&v8, asc_408044, v4);
      strcat(&v10, &v8);
      ++v3;
    }
    while ( v3 < 17 );
    if ( !strcmp(&v10, &v13) )
      sub_40134B(aSuccess, v7);
    else
      sub_40134B(aWrong, v7);
  }
  sub_40134B(aWrong, v7);
  result = stru_408090._cnt-- - 1;
  if ( stru_408090._cnt < 0 )
    return _filbuf(&stru_408090);
  ++stru_408090._ptr;
  return result;
}
```

https://blog.csdn.net/weixin_42621117

简单的逻辑推理：
v9为我们的输入，长度≤0x11（10进制的17）
v10储存的就是v9，和v13进行比较。相同就success
到这里我们就知道输入必须就是v13这个字符串相同。但是发现引号中字符数＞17，所以判断这是个16进制数表示的字符串（ASCII码），用网上16进制转字符串得到flag：CrackMeJustForFun

0x4.open-source

拿到源码了嘤嘤嘤，就直接IDE打开不解释！
源码如下图：



```c
#include <stdio.h>
#include <string.h>

int main(int argc, char *argv[]) {
    if (argc != 4) {
        printf("what?\n");

    }

    unsigned int first = atoi(argv[1]);
    if (first != 0xcafe) {
        printf("you are wrong, sorry.\n");
        exit(2);
    }

    unsigned int second = atoi(argv[2]);
    if (second % 5 == 3 || second % 17 != 8) {
        printf("ha, you won't get it!\n");
        exit(3);
    }

    if (strcmp("h4cky0u", argv[3])) {
        printf("so close, dude!\n");
        exit(4);
    }

    printf("Brr wrrr grr\n");

    unsigned int hash = first * 31337 + (second % 17) * 11 + strlen(argv[3]) - 1615810207;
    printf("Get your key: ");
    printf("%x\n", hash);
    return 0;
}
```

https://blog.csdn.net/weixin_42621117

解析：
这个题不用逆向也能做，纯源码分析就能得到答案。
逆向做法：
随便输入几个参数编译链接执行发现wrong
拖IDA，shift+F12，发现“Get your key：”，双击，Ctrl+X+确定，F5
发现v3就是key，写出代码求得v3

```python
v3 = 11 * (25 % 17) + 1628458542 + len("h4cky0u") - 1615810207
print(v3)
```

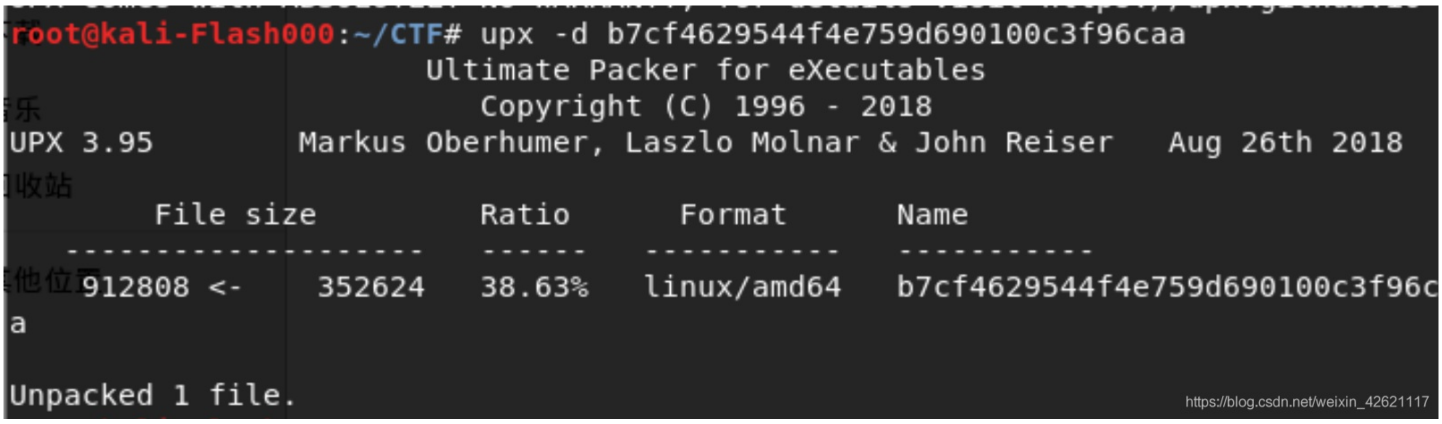
得到12648430，转16进制得到flag：c0ffee

0x5.simple-unpack

解析：
从题目就知道需要脱壳，但是让我们假装不知道QAQ！依旧还是拖到IDA里面看看，果然！
什么都看不懂……那还是老步骤：shift+F12，发现了一个关键字：upx，说明他是upx压缩的文件，所以就需要upx解压

这里博主还是推荐大家装一个kali，双系统或者虚拟机都可以。如果原本就用的Ubuntu等Linux可以忽略这句话QWQ

upx -d filename脱壳

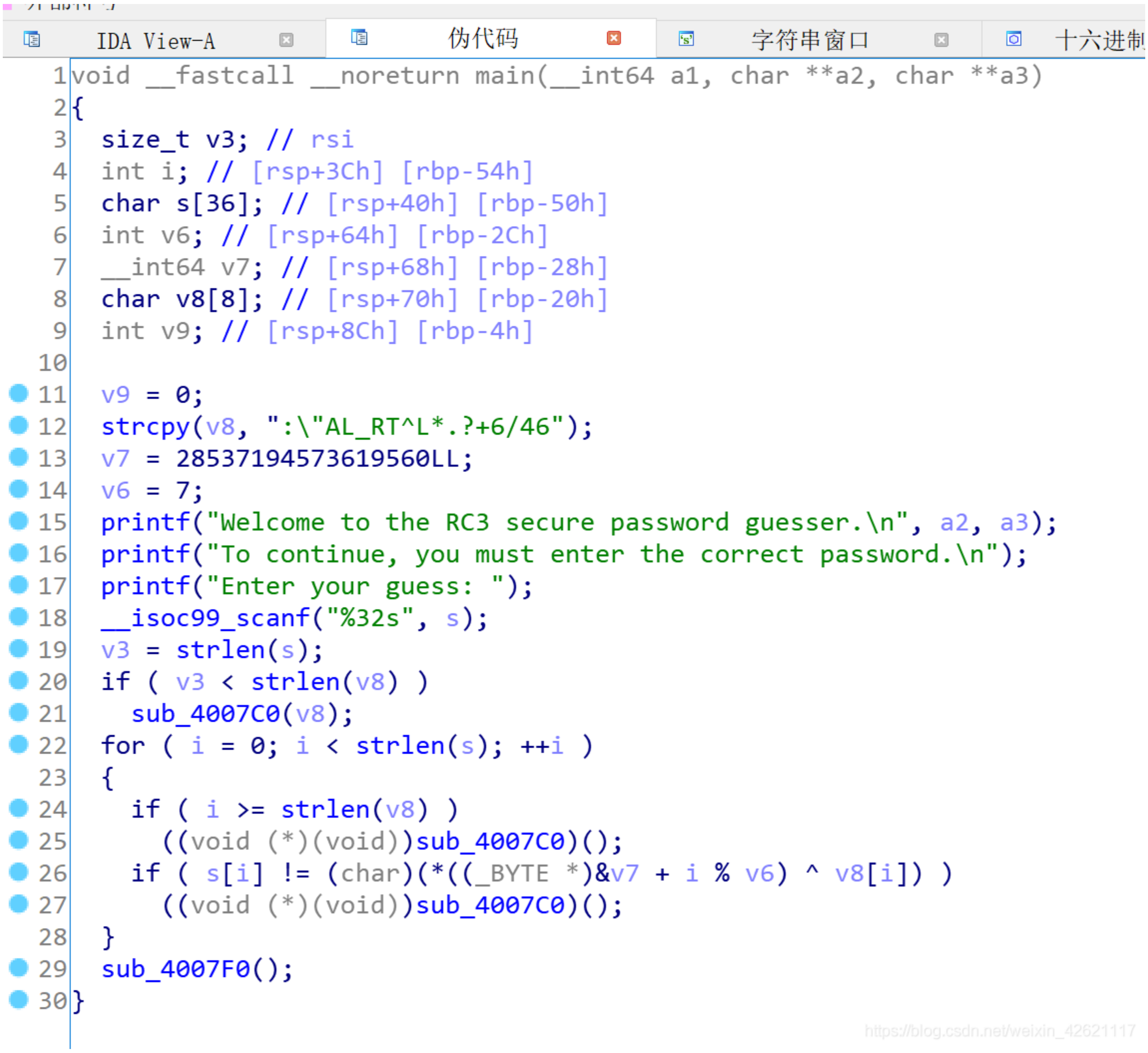


```
root@kali-Flash000:~/CTF# upx -d b7cf4629544f4e759d690100c3f96caa
                       Ultimate Packer for eXecutables
                          Copyright (C) 1996 - 2018
UPX 3.95        Markus Oberhumer, Laszlo Molnar & John Reiser   Aug 26th 2018

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
    912808 <-    352624   38.63%   linux/amd64   b7cf4629544f4e759d690100c3f96c
a

Unpacked 1 file.
```
https://blog.csdn.net/weixin_42621117

拖到IDA，shift+F12直接得到flag：flag{Upx_1s_n0t_a_d3liv3r_c0mp4ny}

0x6.logmein

解析：
日常拖IDA，shift+F12
第一次经验性进You entered the correct password!\nGreat job!\n，发现反编译出来的函数没啥用，所以第二次选择进输入点Enter your guess（类似于找OEP时先找PUSHAD和POPAD）



```
void __fastcall __noreturn main(__int64 a1, char **a2, char **a3)
{
  size_t v3; // rsi
  int i; // [rsp+3Ch] [rbp-54h]
  char s[36]; // [rsp+40h] [rbp-50h]
  int v6; // [rsp+64h] [rbp-2Ch]
  __int64 v7; // [rsp+68h] [rbp-28h]
  char v8[8]; // [rsp+70h] [rbp-20h]
  int v9; // [rsp+8Ch] [rbp-4h]

  v9 = 0;
  strcpy(v8, ":\"AL_RT^L*.?+6/46");
  v7 = 28537194573619560LL;
  v6 = 7;
  printf("Welcome to the RC3 secure password guesser.\n", a2, a3);
  printf("To continue, you must enter the correct password.\n");
  printf("Enter your guess: ");
  __isoc99_scanf("%32s", s);
  v3 = strlen(s);
  if ( v3 < strlen(v8) )
    sub_4007C0(v8);
  for ( i = 0; i < strlen(s); ++i )
  {
    if ( i >= strlen(v8) )
      ((void (*)(void))sub_4007C0)();
    if ( s[i] != (char)(*((_BYTE *)&v7 + i % v6) ^ v8[i]) )
      ((void (*)(void))sub_4007C0)();
  }
  sub_4007F0();
}
```

https://blog.csdn.net/weixin_42621117

逻辑分析：
v8是给定的字符串,v7是long long的数据类型
s是输入，v3是s的长度，v3必须≥v8的长度17，否则会进入提示输入错误的函数sub_4007C0()
重点是：(_BYTE *)&v7的意思是，把longlong型的v7强制转化为byte型的地址，简单的说，就是把它看成字符串（C语言字符串本质都是指针首地址+偏移）。
所以我们用先用v7的值10进制转16进制，然后16进制转文本得到：ebmarah
重点来了！为什么直接套这个字符串不对，根本原因是因为在机器虚拟化内存后，规定地址排列规则时使用了小端法（最低有效字节在前面）。因此我们真正的解码文本应该是把上面的答案倒过来写：harambe

```
v8 = ":\"AL_RT^L*.?+6/46"
v7 = 'harambe'

for i in range(len(v8)):
    char = ord(v7[i % 7]) ^ ord(v8[i])
    print(chr(char),end='')
```

得到flag：RC3-2016-XORISGUD

当然个人感觉最简单的办法还是C++重现一遍。。。就不用考虑这么多

```
#include <iostream>
using namespace std;

int main(){
 long long v7 = 28537194573619560;
 char *p = (char*)&v7;
 char v8[] = ":\"AL_RT^L*.?+6/46";
 for(int i = 0;v8[i]!=0;i++){
  v8[i] = v8[i]^p[i%7];
 }
 cout<<v8<<endl;
 return 0;
}
```

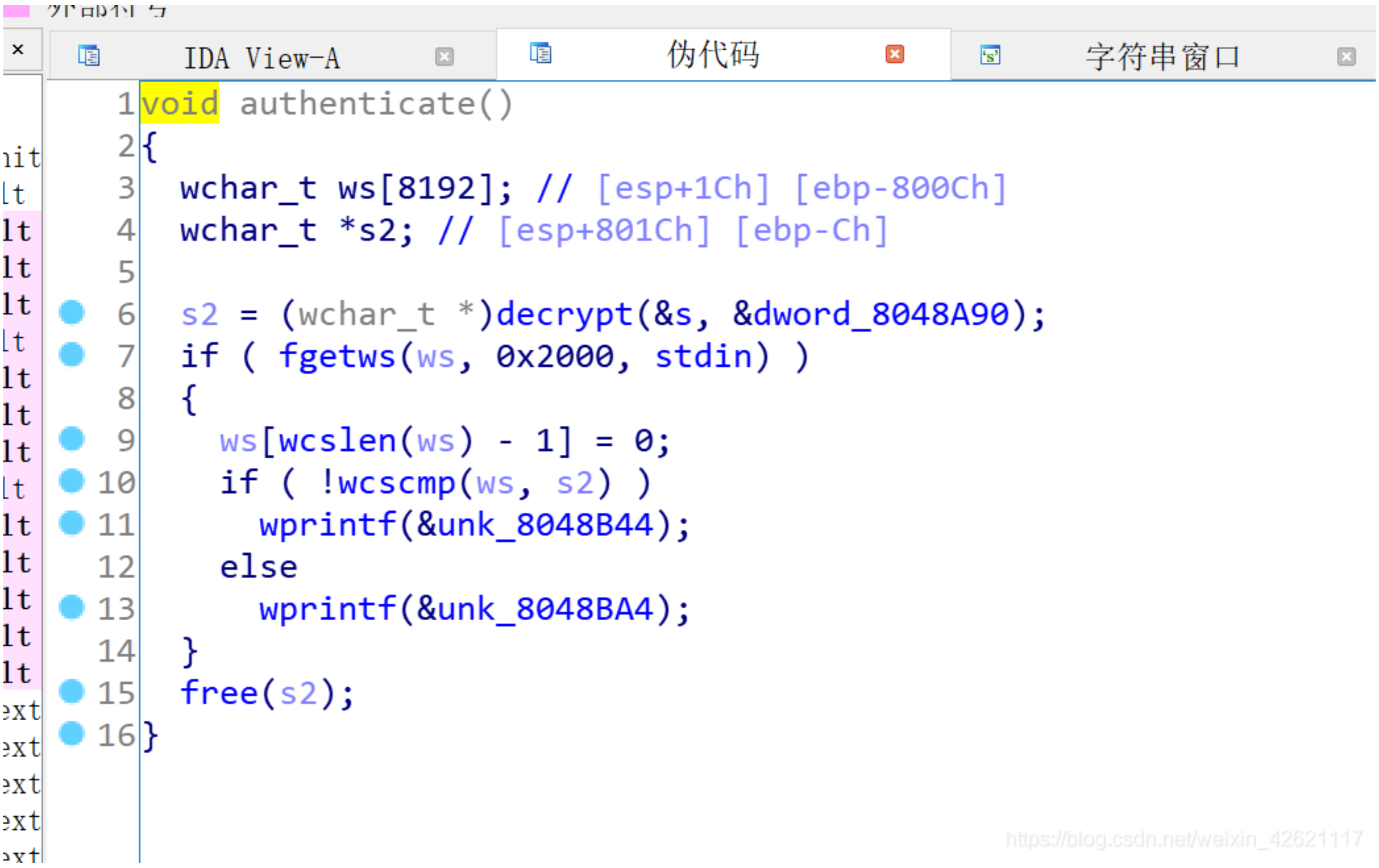
0x7. insanity

解析：
这个真的不知道咋解析……至于为啥放这里，也许就和题目所言一样吧，希望大家身心愉悦继续肝吧……
拖IDA，shift+F12直接拿到flag：9447{This_is_a_flag}

0x8.no-strings-attached

这个题是真的有难度QAQ

解析：
正常步骤拖到IDA静态分析，shfit+F12，发现第一行赫然出现：/lib/ld-linux.so.2。看见这个大家心里应该都有数了，和linux有关没跑了。同时也说明这是个ELF文件
字符串没有关键字，就从IDA左边函数列表找到main函数双击进去，F5反汇编，再进到authenticate函数看看（有的东西做多了就知道了），如下：



```
void authenticate()
{
  wchar_t ws[8192]; // [esp+1Ch] [ebp-800Ch]
  wchar_t *s2; // [esp+801Ch] [ebp-Ch]

  s2 = (wchar_t *)decrypt(&s, &dword_8048A90);
  if ( fgetws(ws, 0x2000, stdin) )
  {
    ws[wcslen(ws) - 1] = 0;
    if ( !wcscmp(ws, s2) )
      wprintf(&unk_8048B44);
    else
      wprintf(&unk_8048BA4);
  }
  free(s2);
}
```

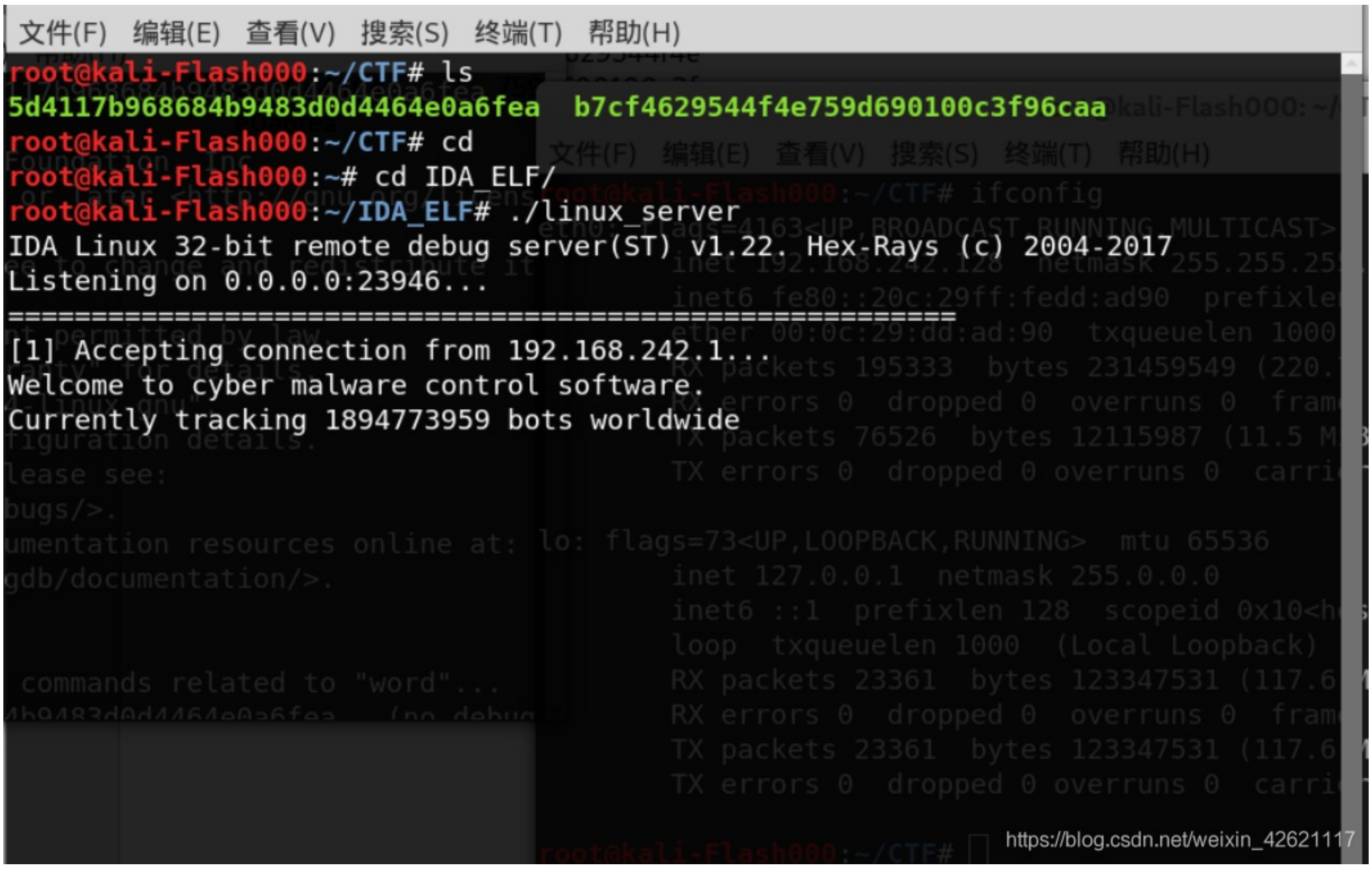
https://blog.csdn.net/weixin_42621117

此时就真的看英语了……计算s2的函数decrypt正是非常专业的术语：解密。
粗略的看一下下面的伪码，得出：ws是输入，ws==s2时就是正确的flag
此时我们需要转变一下思维：之前我们都是各种找、各种逻辑推断正确输入。但是我们忽略了一件事，那个与输入的比较的正确答案，一定是加载到内存里面之后，才与输入比较。要是我们能跟踪到这个正确答案储存在内存的位置然后把他拿出来，这不也行嘛！！！（Reverse！）

思路有了，还需要实际的操作。这里就不能用静态分析了。这里插一句，我们逆向分析分为静态分析和动态分析，直接拖到IDA反汇编看伪代码，逻辑推断等等都属于静态分析。换言之，在没有执行程序或程序是静态时的分析。
所以要用IDA动态调试ELF—IDA remote linux debugger

环境配置参考IDA动态调试ELF写的非常清楚

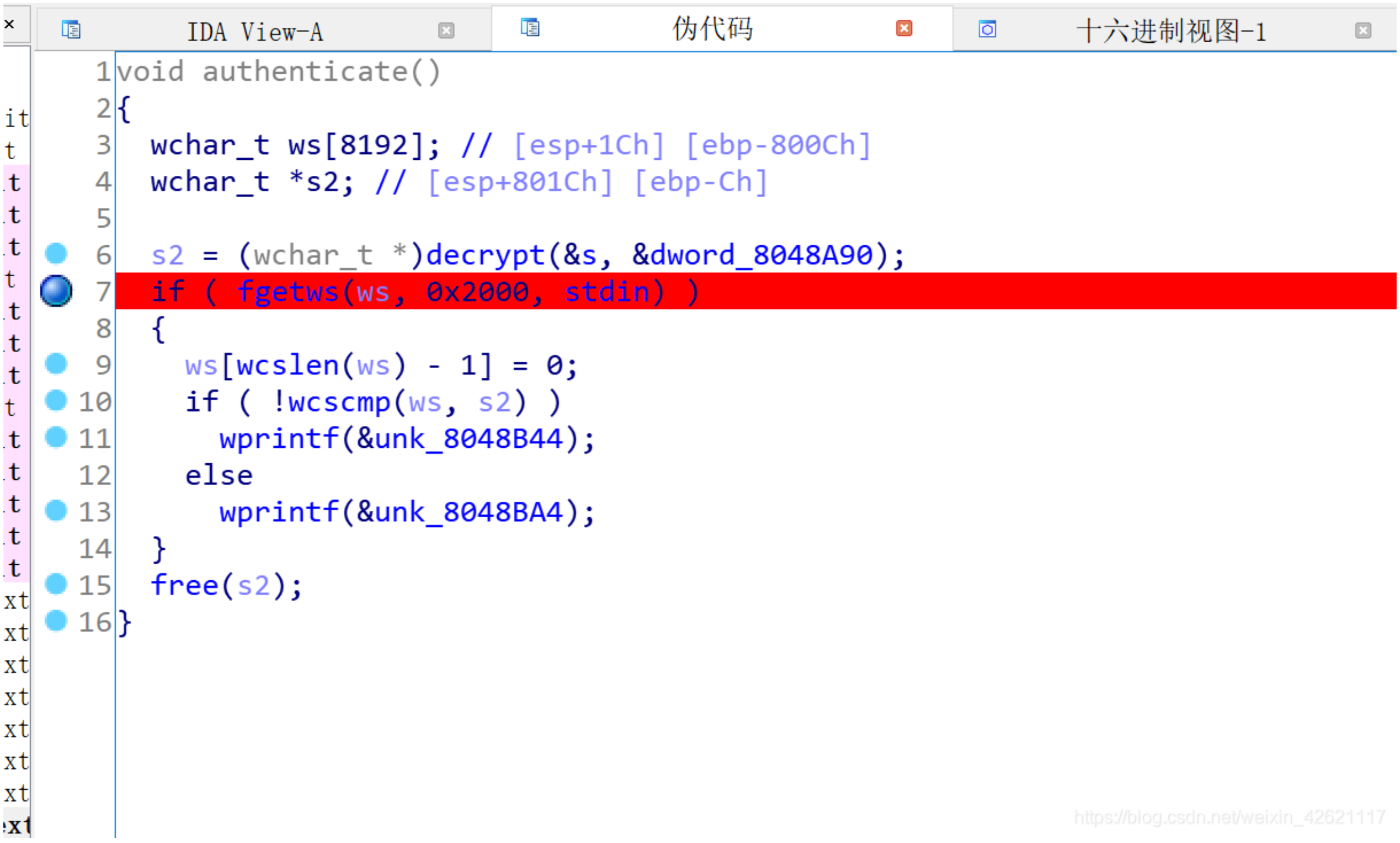
为了检验连通性，可以看看kali的命令行，如下图



```
root@kali-Flash000:~/CTF# ls
5d4117b968684b9483d0d4464e0a6fea  b7cf4629544f4e759d690100c3f96caa
root@kali-Flash000:~/CTF# cd
root@kali-Flash000:~# cd IDA_ELF/
root@kali-Flash000:~/IDA_ELF# ./linux_server
IDA Linux 32-bit remote debug server(ST) v1.22. Hex-Rays (c) 2004-2017
Listening on 0.0.0.0:23946...
=========================================================
[1] Accepting connection from 192.168.242.1...
Welcome to cyber malware control software.
Currently tracking 1894773959 bots worldwide
```

https://blog.csdn.net/weixin_42621117

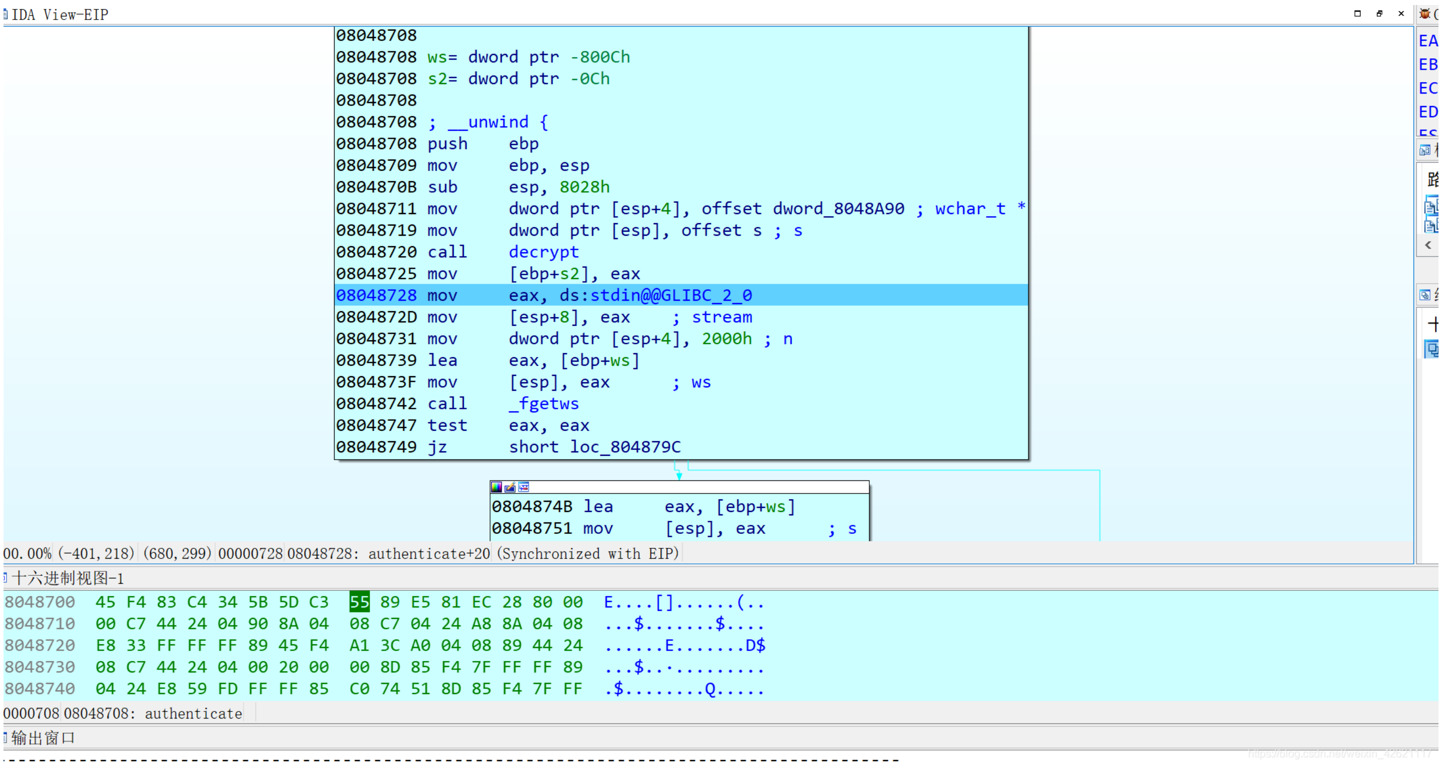
首先我们进入authenticate，F5，点左边设置断点，如下图（在s2刚被赋值完毕后停止，找s2的值）



```
void authenticate()
{
  wchar_t ws[8192]; // [esp+1Ch] [ebp-800Ch]
  wchar_t *s2; // [esp+801Ch] [ebp-Ch]

  s2 = (wchar_t *)decrypt(&s, &dword_8048A90);
  if ( fgetws(ws, 0x2000, stdin) )
  {
    ws[wcslen(ws) - 1] = 0;
    if ( !wcscmp(ws, s2) )
      wprintf(&unk_8048B44);
    else
      wprintf(&unk_8048BA4);
  }
  free(s2);
}
```

https://blog.csdn.net/weixin_42621117

然后按F9，编译链接运行到断点停止，如下图



```
08048708
08048708 ws= dword ptr -800Ch
08048708 s2= dword ptr -0Ch
08048708
08048708 ; __unwind {
08048708 push    ebp
08048709 mov     ebp, esp
0804870B sub     esp, 8028h
08048711 mov     dword ptr [esp+4], offset dword_8048A90 ; wchar_t *
08048719 mov     dword ptr [esp], offset s ; s
08048720 call    decrypt
08048725 mov     [ebp+s2], eax
08048728 mov     eax, ds:stdin@@GLIBC_2_0
0804872D mov     [esp+8], eax    ; stream
08048731 mov     dword ptr [esp+4], 2000h ; n
08048739 lea     eax, [ebp+ws]
0804873F mov     [esp], eax      ; ws
08048742 call    _fgetws
08048747 test    eax, eax
08048749 jz      short loc_804879C

0804874B lea     eax, [ebp+ws]
08048751 mov     [esp], eax      ; s
```

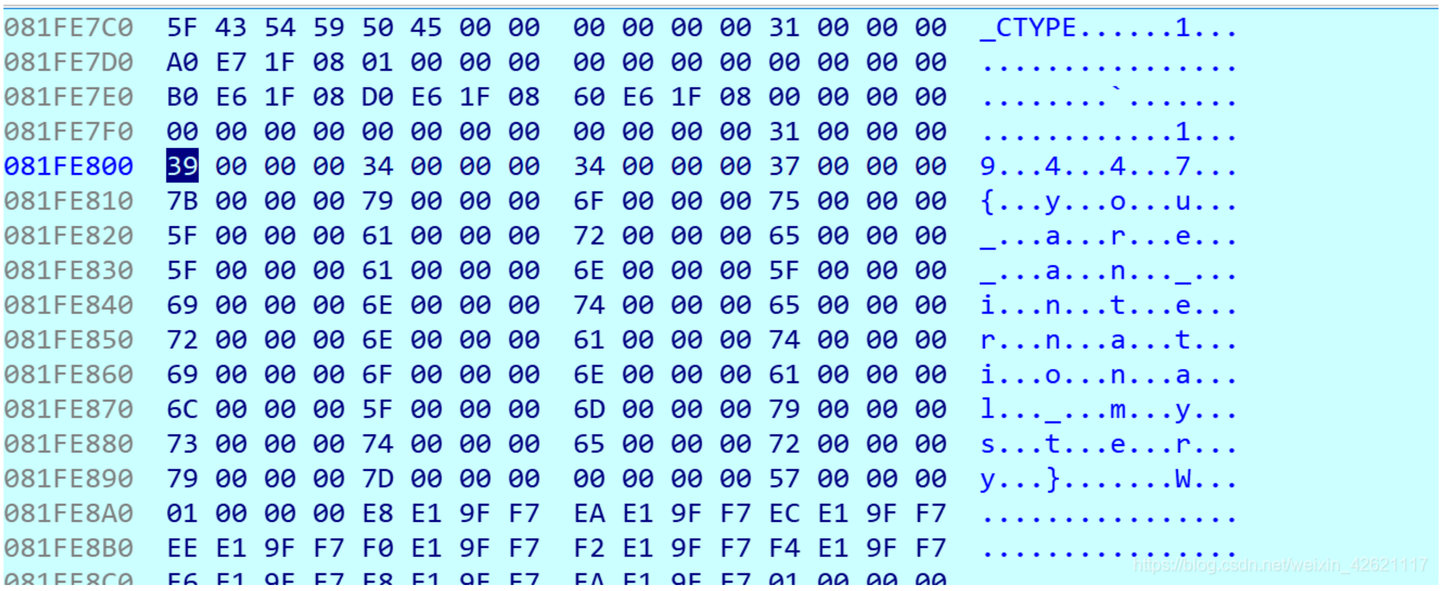
```
00.00% (-401,218) (680,299) 00000728 08048728: authenticate+20 (Synchronized with EIP)
```

十六进制视图-1

```
8048700  45 F4 83 C4 34 5B 5D C3  55 89 E5 81 EC 28 80 00  E....[]......(..
8048710  00 C7 44 24 04 90 8A 04  08 C7 04 24 A8 8A 04 08  ...$.......$....
8048720  E8 33 FF FF FF 89 45 F4  A1 3C A0 04 08 89 44 24  ......E.......D$
8048730  08 C7 44 24 04 00 20 00  00 8D 85 F4 7F FF FF 89  ...$............
8048740  04 24 E8 59 FD FF FF 85  C0 74 51 8D 85 F4 7F FF  .$.......Q.....
0000708 08048708: authenticate
```

输出窗口

这个时候，我们看到了s2就储存在寄存器eax中，所以我们在下面的Hex View窗口中右键，synchronized with，选eax，就能看到值啦，这就是flag，如下图

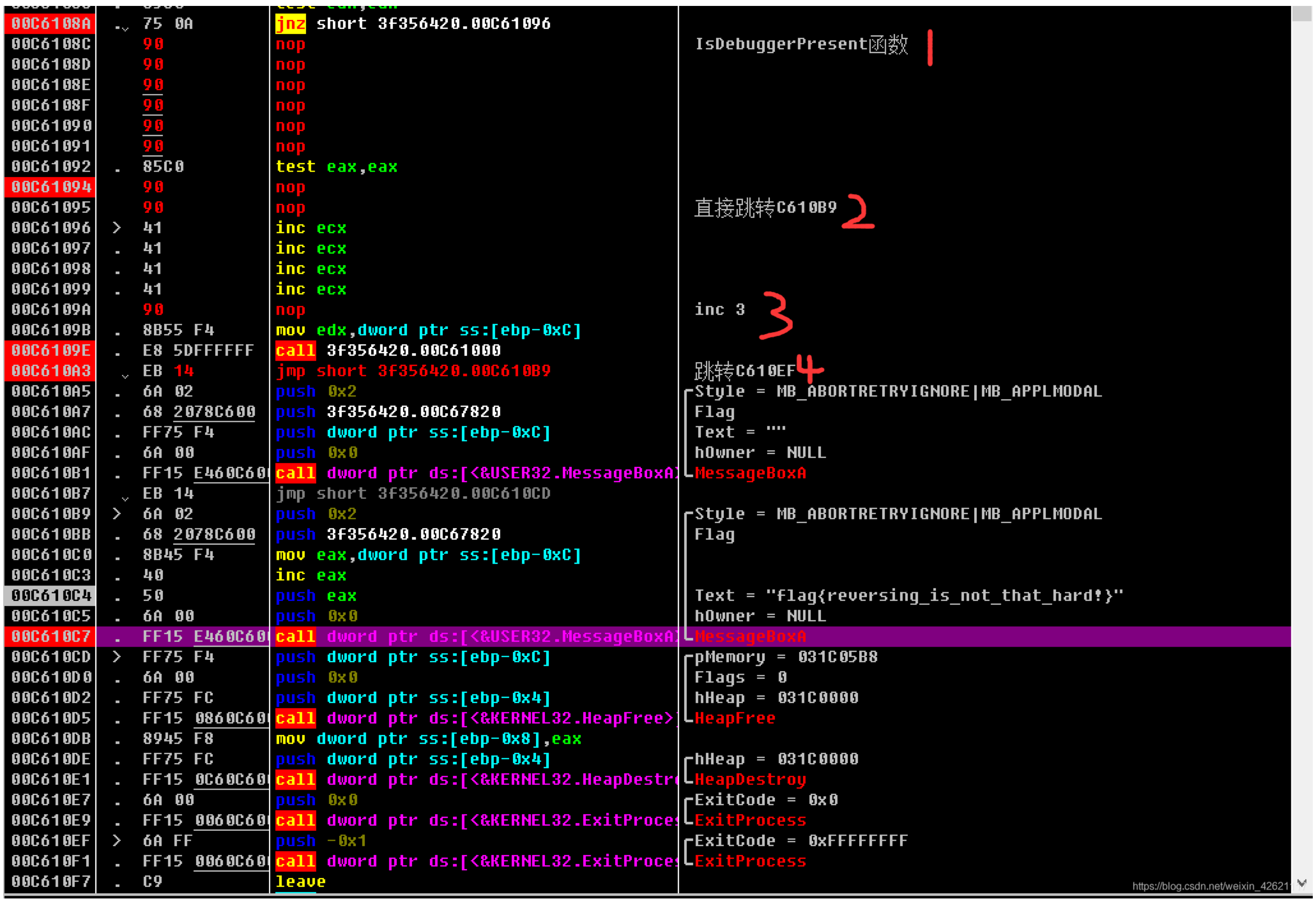


```
081FE7C0  5F 43 54 59 50 45 00 00  00 00 00 00 31 00 00 00  _CTYPE......1...
081FE7D0  A0 E7 1F 08 01 00 00 00  00 00 00 00 00 00 00 00  ................
081FE7E0  B0 E6 1F 08 D0 E6 1F 08  60 E6 1F 08 00 00 00 00  ........`.......
081FE7F0  00 00 00 00 00 00 00 00  00 00 00 00 31 00 00 00  ............1...
081FE800  39 00 00 00 34 00 00 00  34 00 00 00 37 00 00 00  9...4...4...7...
081FE810  7B 00 00 00 79 00 00 00  6F 00 00 00 75 00 00 00  {...y...o...u...
081FE820  5F 00 00 00 61 00 00 00  72 00 00 00 65 00 00 00  _...a...r...e...
081FE830  5F 00 00 00 61 00 00 00  6E 00 00 00 5F 00 00 00  _...a...n..._...
081FE840  69 00 00 00 6E 00 00 00  74 00 00 00 65 00 00 00  i...n...t...e...
081FE850  72 00 00 00 6E 00 00 00  61 00 00 00 74 00 00 00  r...n...a...t...
081FE860  69 00 00 00 6F 00 00 00  6E 00 00 00 61 00 00 00  i...o...n...a...
081FE870  6C 00 00 00 5F 00 00 00  6D 00 00 00 79 00 00 00  l..._...m...y...
081FE880  73 00 00 00 74 00 00 00  65 00 00 00 72 00 00 00  s...t...e...r...
081FE890  79 00 00 00 7D 00 00 00  00 00 00 00 57 00 00 00  y...}.......W...
081FE8A0  01 00 00 00 E8 E1 9F F7  EA E1 9F F7 EC E1 9F F7  ................
081FE8B0  EE E1 9F F7 F0 E1 9F F7  F2 E1 9F F7 F4 E1 9F F7  ................
```

至此拿到flag：9447{you_are_an_international_mystery}

0x9.csaw2013reversing2

解析：拖到IDA中分析发现有重要的函数IsDebuggerPresent()，这个函数目的就是反调试（检测是否处于调试环境中）。既然如此千方百计阻止我们调试，那就直接OD动态走起。
我们拖到OD中，ctrl+n找到IsDebuggerPresent()，确定他的位置之后下断点开始调试程序，发现底下有两个对话框的代码（能看见注释那里有Flag，Text字样就ok），手动F8看一次，发现00C61000那里的函数没有执行。本着现在是“你不让干的事我偏要搞一次”的思想，我们修改程序跳转代码，发现flag赫然出现！
由于这样的方法强行改汇编跳转也存在“试”的成分，所以直接给修改完成的代码（修改了4处），如下图：



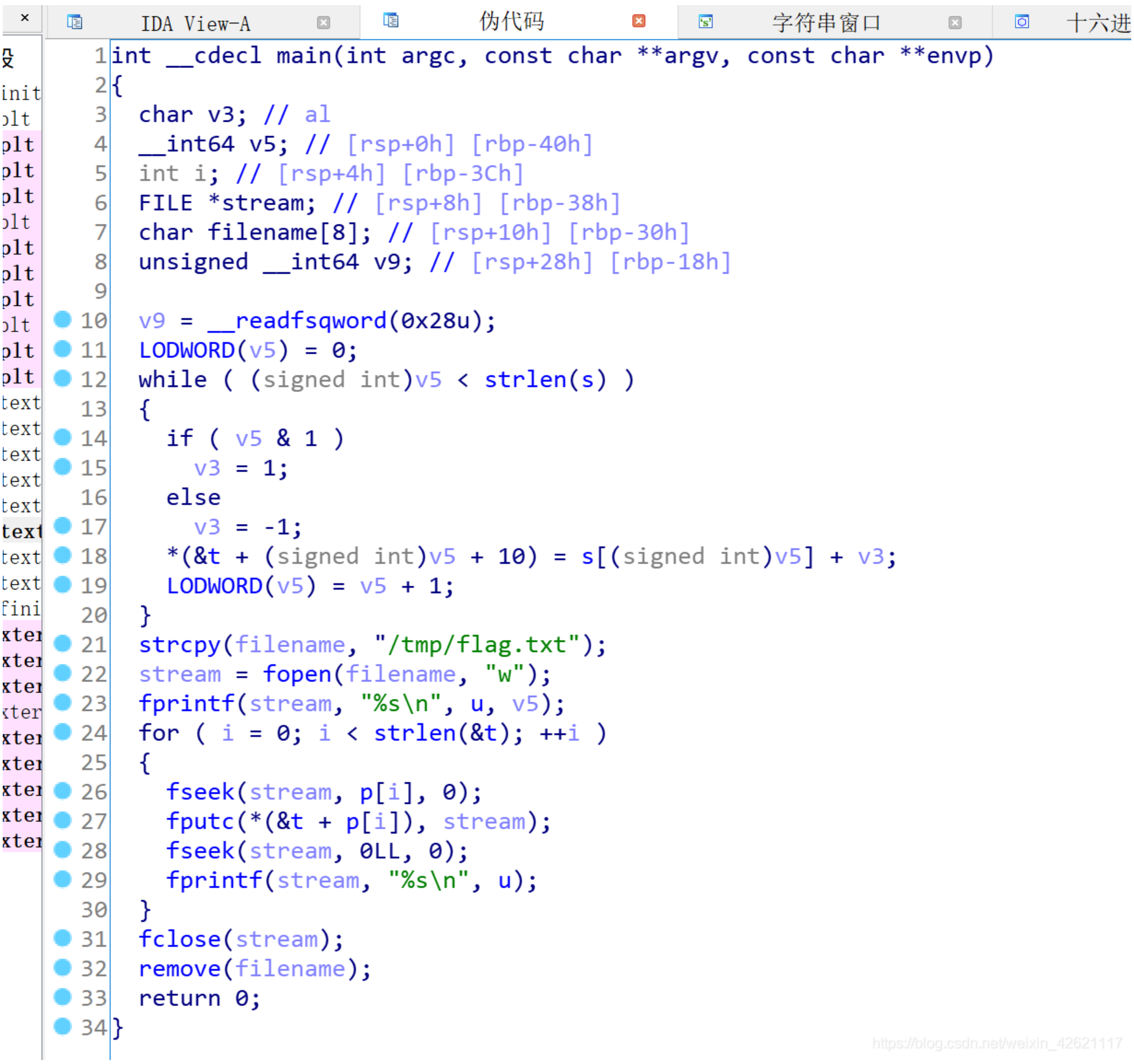
```
00C6108A  .v  75 0A          jnz short 3f356420.00C61096
00C6108C      90             nop                                         IsDebuggerPresent函数 1
00C6108D      90             nop
00C6108E      90             nop
00C6108F      90             nop
00C61090      90             nop
00C61091      90             nop
00C61092  .   85C0           test eax,eax
00C61094      90             nop
00C61095      90             nop                                         直接跳转C610B9 2
00C61096  >   41             inc ecx
00C61097  .   41             inc ecx
00C61098  .   41             inc ecx
00C61099  .   41             inc ecx
00C6109A      90             nop                                         inc 3 3
00C6109B  .   8B55 F4        mov edx,dword ptr ss:[ebp-0xC]
00C6109E  .   E8 5DFFFFFF    call 3f356420.00C61000
00C610A3   v  EB 14          jmp short 3f356420.00C610B9                 跳转C610EF 4
00C610A5  .   6A 02          push 0x2                                   ┌Style = MB_ABORTRETRYIGNORE|MB_APPLMODAL
00C610A7  .   68 2078C600    push 3f356420.00C67820                     │Flag
00C610AC  .   FF75 F4        push dword ptr ss:[ebp-0xC]                │Text = ""
00C610AF  .   6A 00          push 0x0                                   │hOwner = NULL
00C610B1  .   FF15 E460C60   call dword ptr ds:[<&USER32.MessageBoxA]   └MessageBoxA
00C610B7   v  EB 14          jmp short 3f356420.00C610CD
00C610B9  >   6A 02          push 0x2                                   ┌Style = MB_ABORTRETRYIGNORE|MB_APPLMODAL
00C610BB  .   68 2078C600    push 3f356420.00C67820                     │Flag
00C610C0  .   8B45 F4        mov eax,dword ptr ss:[ebp-0xC]             │
00C610C3  .   40             inc eax                                    │
00C610C4  .   50             push eax                                   │Text = "flag{reversing_is_not_that_hard!}"
00C610C5  .   6A 00          push 0x0                                   │hOwner = NULL
00C610C7  .   FF15 E460C60   call dword ptr ds:[<&USER32.MessageBoxA]   └MessageBoxA
00C610CD  >   FF75 F4        push dword ptr ss:[ebp-0xC]                ┌pMemory = 031C05B8
00C610D0  .   6A 00          push 0x0                                   │Flags = 0
00C610D2  .   FF75 FC        push dword ptr ss:[ebp-0x4]                │hHeap = 031C0000
00C610D5  .   FF15 0860C60   call dword ptr ds:[<&KERNEL32.HeapFree>]   └HeapFree
00C610DB  .   8945 F8        mov dword ptr ss:[ebp-0x8],eax
00C610DE  .   FF75 FC        push dword ptr ss:[ebp-0x4]                ┌hHeap = 031C0000
00C610E1  .   FF15 0C60C60   call dword ptr ds:[<&KERNEL32.HeapDestro   └HeapDestroy
00C610E7  .   6A 00          push 0x0                                   ┌ExitCode = 0x0
00C610E9  .   FF15 0060C60   call dword ptr ds:[<&KERNEL32.ExitProces   └ExitProcess
00C610EF  >   6A FF          push -0x1                                  ┌ExitCode = 0xFFFFFFFF
00C610F1  .   FF15 0060C60   call dword ptr ds:[<&KERNEL32.ExitProces   └ExitProcess
00C610F7  .   C9             leave
```

https://blog.csdn.net/weixin_42621

所以直接能拿flag啦：flag{reversing_is_not_that_hard!}

0xa.getit

解析：依旧老套路，拖IDA，shift+F12看字符串发现linux和一个很像flag形式的字符串"SharifCTF{???}"，双击点进去，然后在左边的框找到主函数，反汇编成伪代码。如下图：



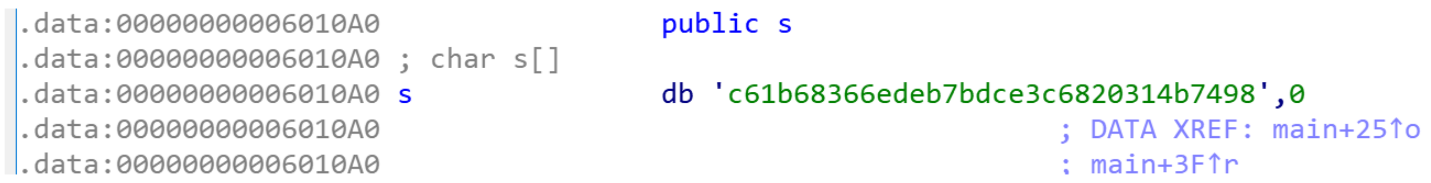
```c
 1 int __cdecl main(int argc, const char **argv, const char **envp)
 2 {
 3   char v3; // al
 4   __int64 v5; // [rsp+0h] [rbp-40h]
 5   int i; // [rsp+4h] [rbp-3Ch]
 6   FILE *stream; // [rsp+8h] [rbp-38h]
 7   char filename[8]; // [rsp+10h] [rbp-30h]
 8   unsigned __int64 v9; // [rsp+28h] [rbp-18h]
 9
10   v9 = __readfsqword(0x28u);
11   LODWORD(v5) = 0;
12   while ( (signed int)v5 < strlen(s) )
13   {
14     if ( v5 & 1 )
15       v3 = 1;
16     else
17       v3 = -1;
18     *(&t + (signed int)v5 + 10) = s[(signed int)v5] + v3;
19     LODWORD(v5) = v5 + 1;
20   }
21   strcpy(filename, "/tmp/flag.txt");
22   stream = fopen(filename, "w");
23   fprintf(stream, "%s\n", u, v5);
24   for ( i = 0; i < strlen(&t); ++i )
25   {
26     fseek(stream, p[i], 0);
27     fputc(*(&t + p[i]), stream);
28     fseek(stream, 0LL, 0);
29     fprintf(stream, "%s\n", u);
30   }
31   fclose(stream);
32   remove(filename);
33   return 0;
34 }
```

https://blog.csdn.net/weixin_42621117

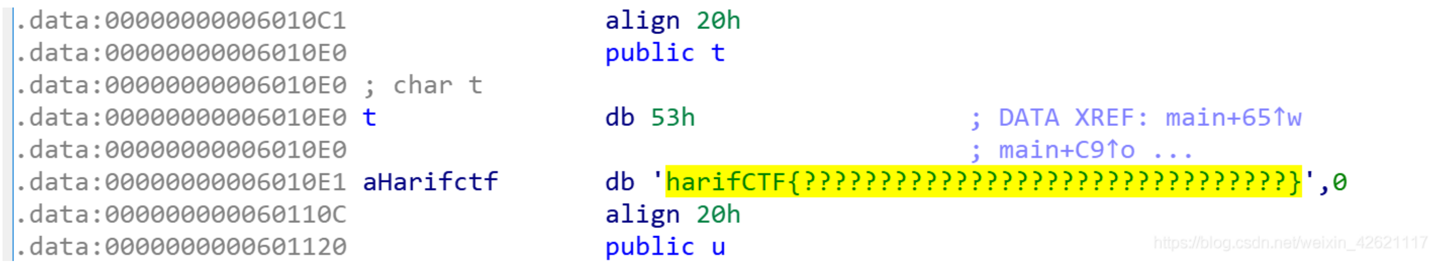
简单分析代码：（重点是11~20行）s长度限定，v5条件选择，v3偏移量，用参数操作s，t为最终存放数组，最后用流写入tmp文件夹下的flag.txt中。但是/tmp是linux主目录下一个存放临时文件的文件夹，程序return后写入的临时文件也一并丢弃。

这里额外说一下，这道题可以用在linux环境下运行，然后设置断点去/tmp文件夹下找，或者直接更改流写入的目标文件夹都是可以的。这里我们使用windows纯代码分析的方法。

通过分析我们发现v3,v5已知，需要知道s和t。我们在IDA的IDA View-A的窗口中找到s的值，如下图
：


```
.data:00000000006010A0                          public s
.data:00000000006010A0 ; char s[]
.data:00000000006010A0 s                        db 'c61b68366edeb7bdce3c6820314b7498',0
.data:00000000006010A0                                                   ; DATA XREF: main+25↑o
.data:00000000006010A0                                                   ; main+3F↑r
```

找t的值，代码如下：（注意这里是题目有bug！！！t的值是SharifCTF{???},可在16进制视图窗口查看）


```
.data:00000000006010C1                  align 20h
.data:00000000006010E0                  public t
.data:00000000006010E0 ; char t
.data:00000000006010E0 t                db 53h                   ; DATA XREF: main+65↑w
.data:00000000006010E0                                           ; main+C9↑o ...
.data:00000000006010E1 aHarifctf        db 'harifCTF{????????????????????????????????}',0
.data:000000000060110C                  align 20h
.data:0000000000601120                  public u
```
https://blog.csdn.net/weixin_42621117

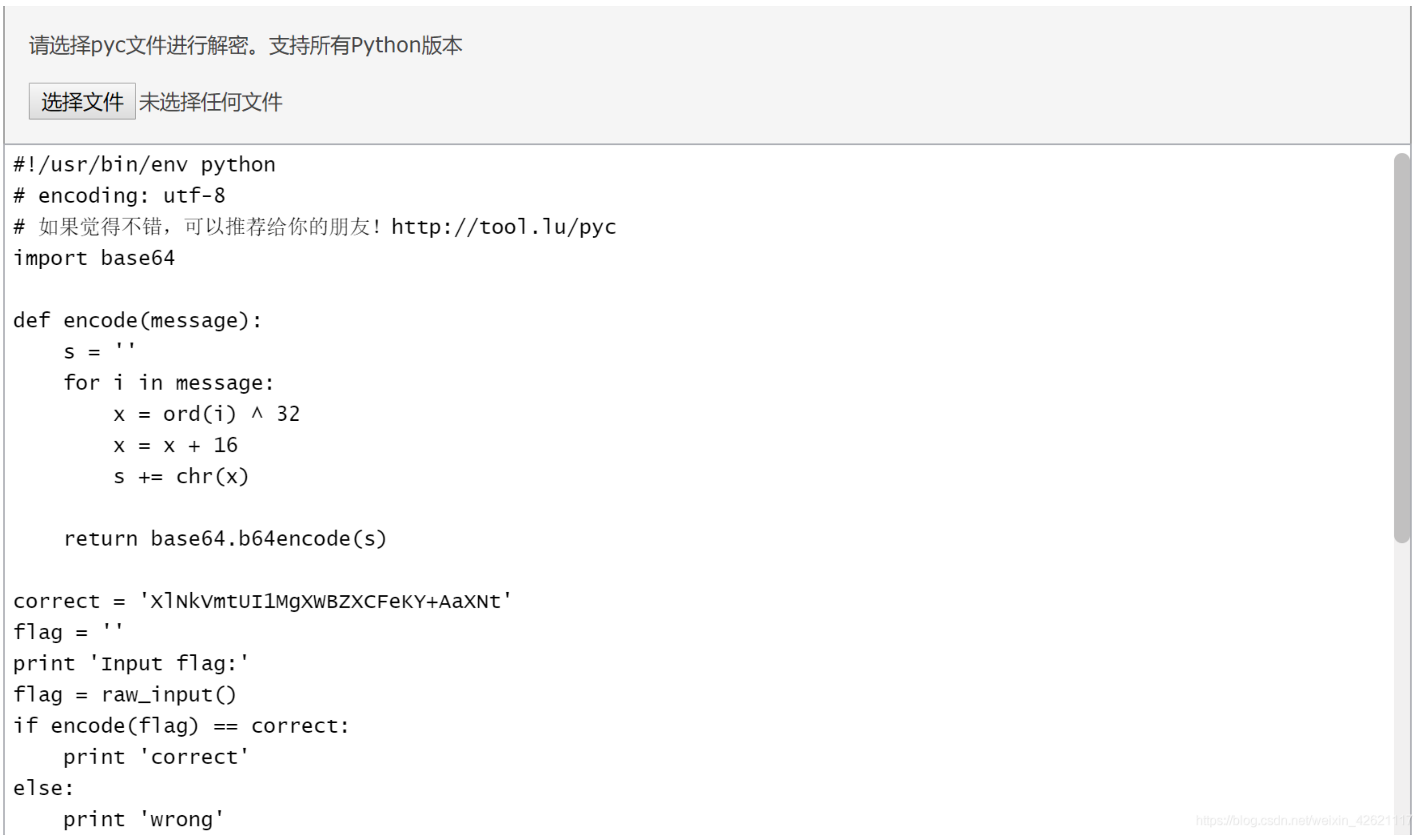
最后写出代码

```
v5 = 0
s = 'c61b68366edeb7bdce3c6820314b7498'
t = ['S','h','a','r','i','f','C','T','F','{','?','?','?','?','?','?','?','?','?','?','?','?','?','?','?','?','?'
,'?','?','?','?','?','?','?','?','?','?','?','?','?','?','?','}']
v3 = 0
l = len(s)
while(v5 < l):
    if( v5 & 1 ):
        v3 = 1
    else:
        v3 = -1
    t[10+v5] = chr(ord(s[v5])+v3)
    v5 += 1
flag = ''
for x in t:
    flag+=x
print(flag)
```

得到flag：SharifCTF{b70c59275fcfa8aebf2d5911223c6589}

0xB.python-trade

解析：
下载完文件发现是一个.pyc文件，百度得知.pyc文件其实是PyCodeObject的一种持久化保存方式（感兴趣可自行搜索学习）。所以思路就比较清晰了：用python反编译在线工具反编译这个.pyc文件得到源码，如下图



```
#!/usr/bin/env python
# encoding: utf-8
# 如果觉得不错，可以推荐给你的朋友！http://tool.lu/pyc
import base64

def encode(message):
    s = ''
    for i in message:
        x = ord(i) ^ 32
        x = x + 16
        s += chr(x)

    return base64.b64encode(s)

correct = 'XlNkVmtUI1MgXWBZXCFeKY+AaXNt'
flag = ''
print 'Input flag:'
flag = raw_input()
if encode(flag) == correct:
    print 'correct'
else:
    print 'wrong'
```

https://blog.csdn.net/weixin_4262117

关键点：encode(flag) == correct
所以就很容易写出逆向解码的代码：

```
# encoding: utf-8
import base64
s = "XlNkVmtUI1MgXWBZXCFeKY+AaXNt"
flag = ""

#base64
b = base64.b64decode(s)# print(b)

#encode
for i in b:
    i -= 16
    i ^= 32
    flag += chr(i)
print(flag)
```

拿到flag：nctf{d3c0mpil1n9_PyC}

0xC.maze

解析：ELF文件，日常拖到IDA，查找字符串，交叉引用，F5大法好。
分析代码，s1储存输入对象，比较前5位是不是"nctf{"，第25位最后一位是不是"}"。之后发现asc_601060中储存的是一个8*8的迷宫,迷宫如下：

```
  ******
*   *  *
*** * **
**  * **
**  * **
*  *#  *
** *** *
**     *
********
```

通过分析，发现v4是玩家输入的方向：‘O’–左，‘o’–右，’.’–上，‘0’–下，由迷宫得到轨迹：右下右右下下左下下下右右右右上上左左
所以flag就是：nctf{o0oo00O000oooo…OO}

到这里，整个攻防世界Reverse的Exercise area就解答完毕了，希望大家能多多交【pi】流【ping】！
RE真好玩~强颜欢笑.jpg