Java ist reverse_GKCTF 2020 Reverse Writeup

原创

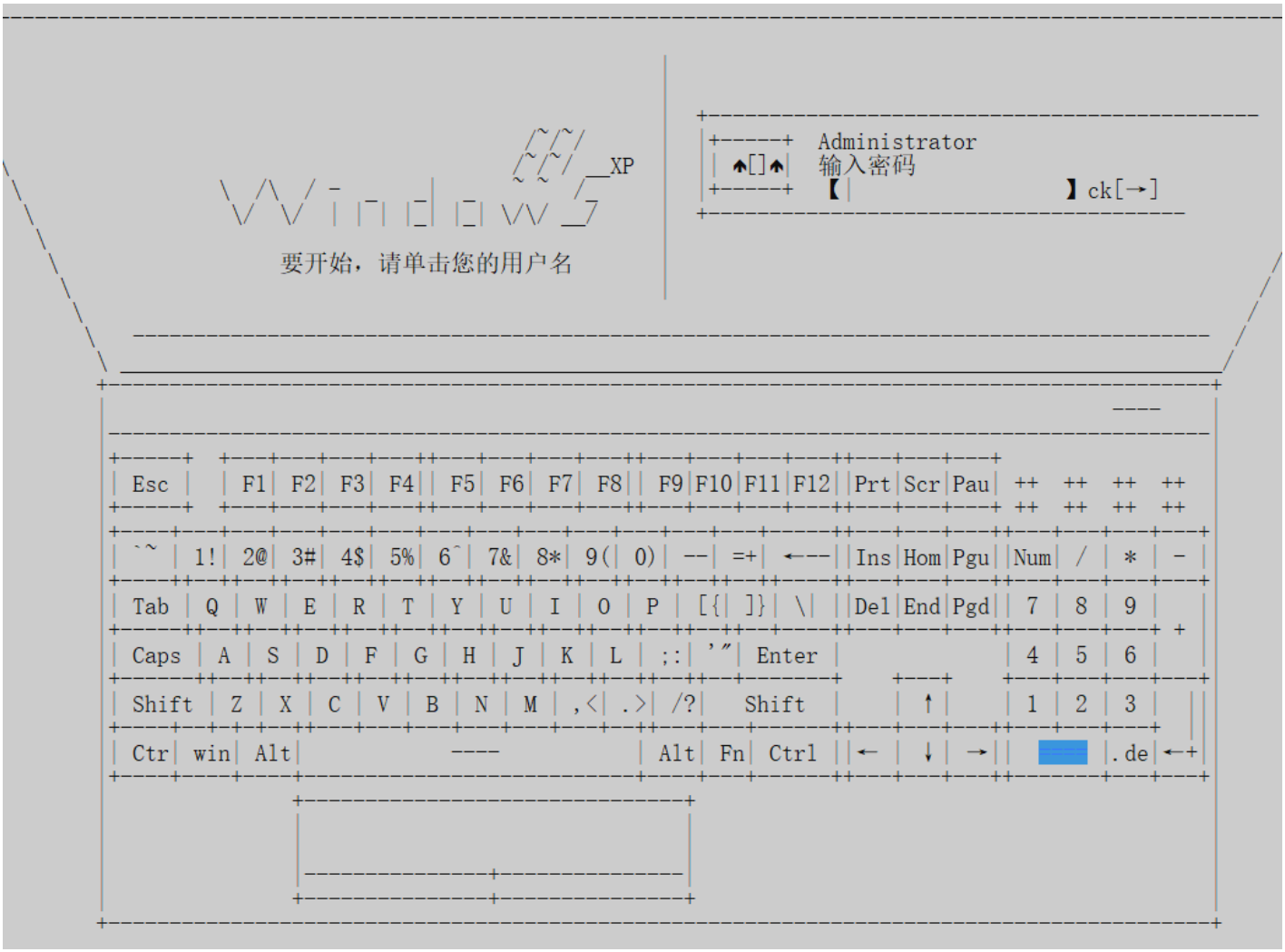
应仁学术 于 2021-02-24 02:38:40 发布 61 收藏
文章标签： Java ist reverse

本文链接：https://blog.csdn.net/weixin_42365492/article/details/114525547

RE

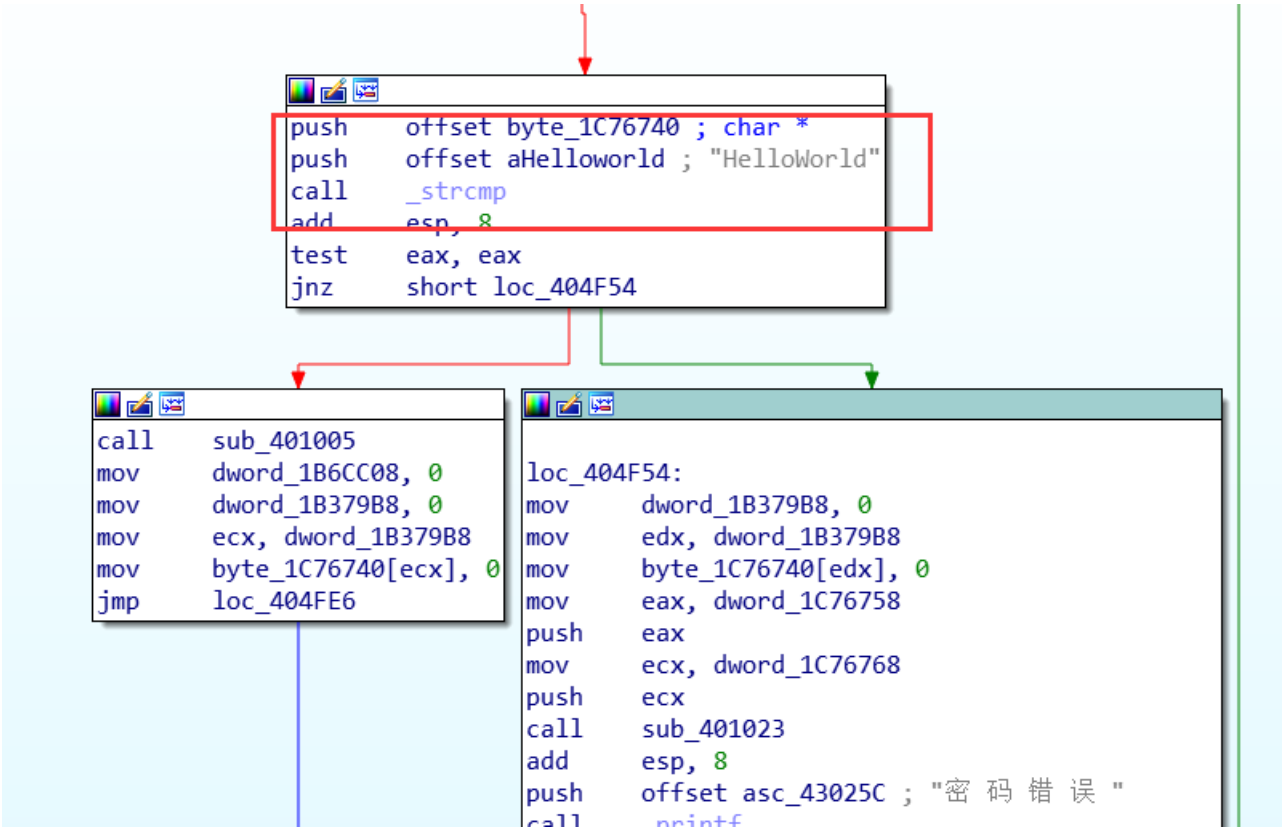
0x1 Check_1n

在B站上看到一个大佬的C语言项目，他在gihub开源了代码，感觉挺有意思的。偷偷拿来魔改了一下当签到题，拿到题目后，如果是按照常规思路走，那就是先找到开机密码：

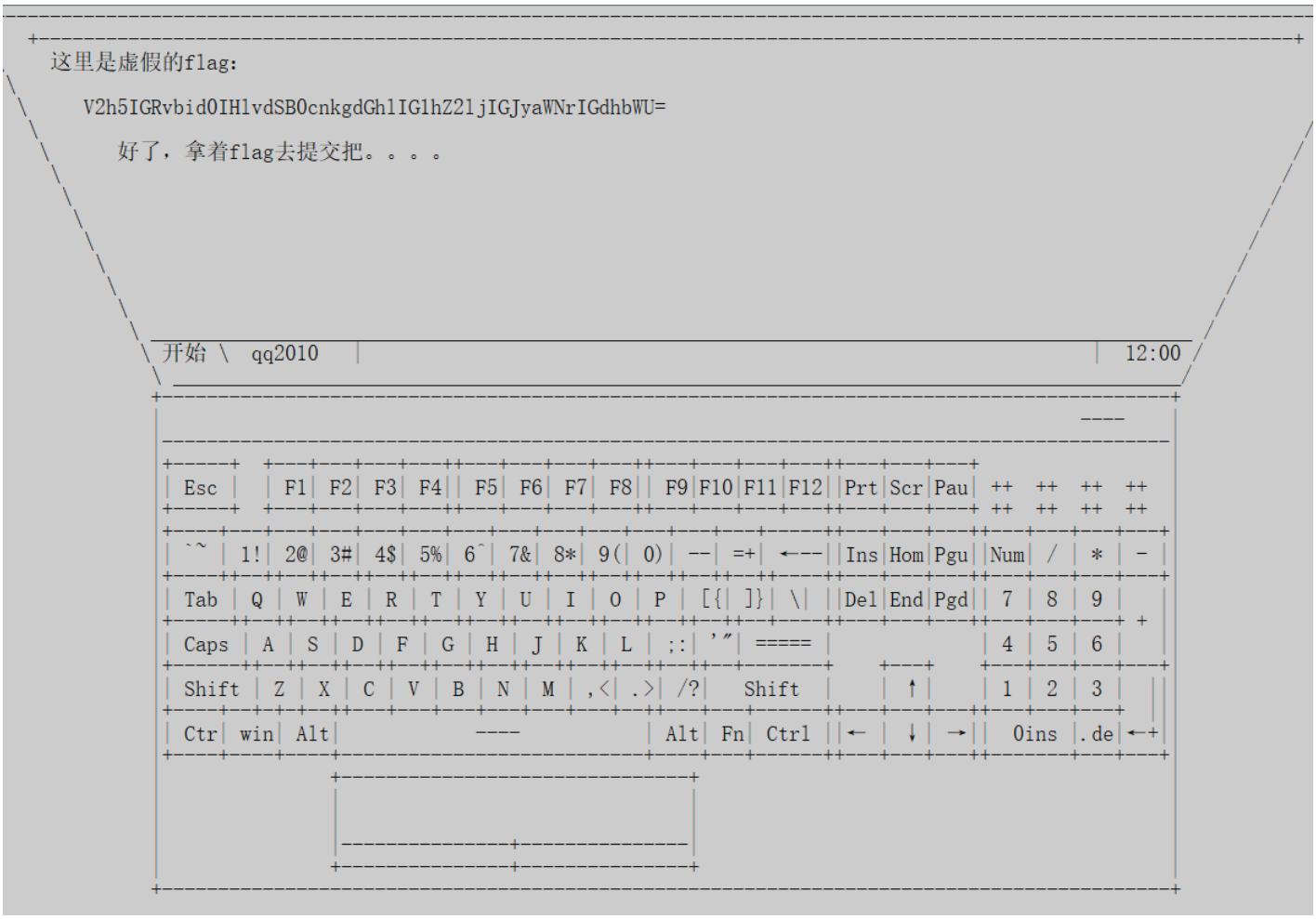


IDA打开搜索关键字符串就可以拿到开机密码：HelloWorld

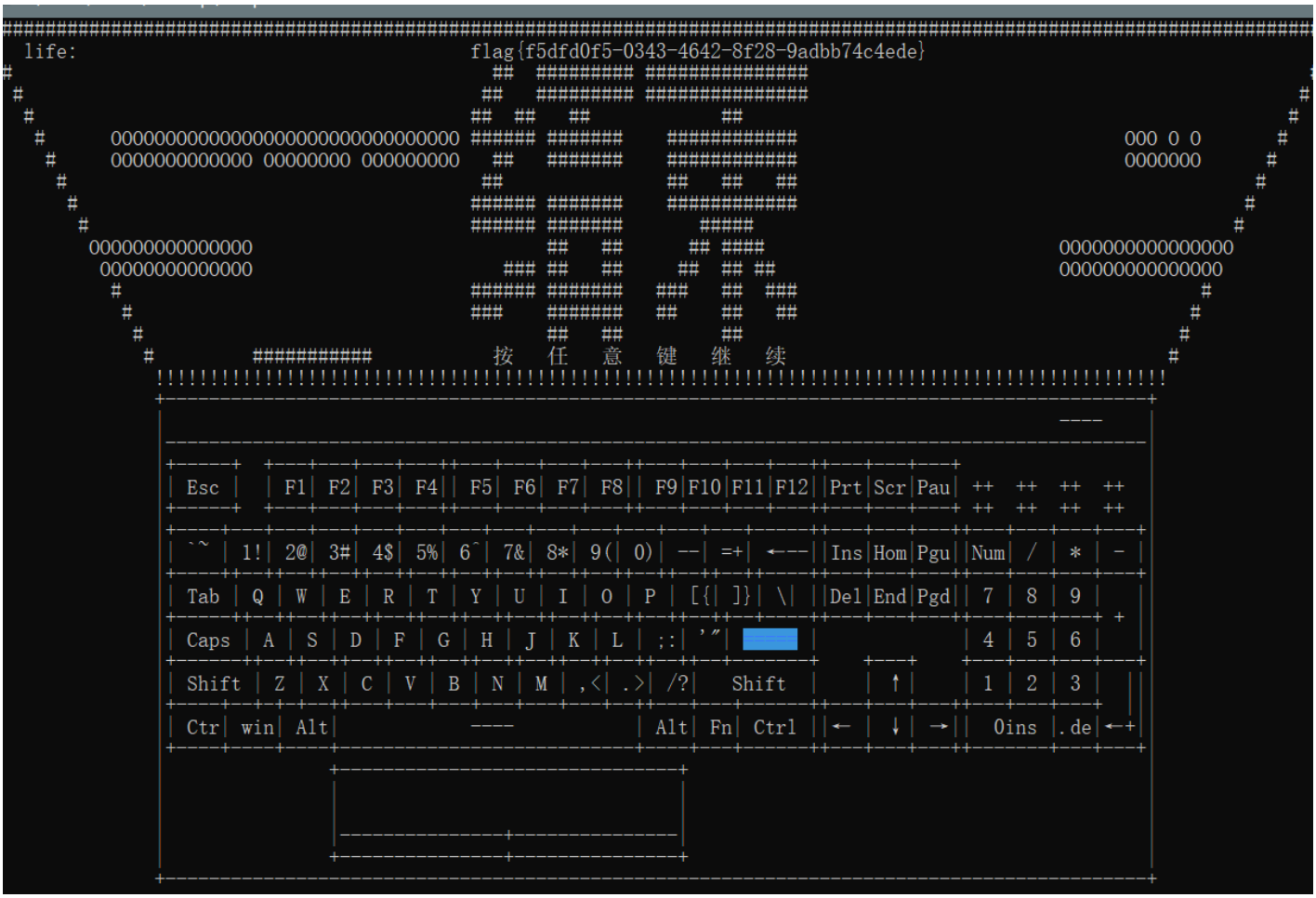
```
.rdata:0…  00000006  C  life:
.rdata:0…  0000003B  C  2i9Q8AtFJTfL3ahU2XGuemEqZJ2ensozjg1EjPJwCHy4RY1Nyvn1ZE1bZe
.rdata:0…  00000030  C
.rdata:0…  00000013  C
.rdata:0…  0000000C  C  密 码 错 误
.rdata:0…  00000009  C  color 07
.rdata:0…  0000000E  C  i386\\chkesp.c
.rdata:0…  000000DC  C  The value of ESP was not properly saved across a function cal…
.rdata:0…  00000009  C  printf.c
.rdata:0…  0000000F  C  format != NULL
```



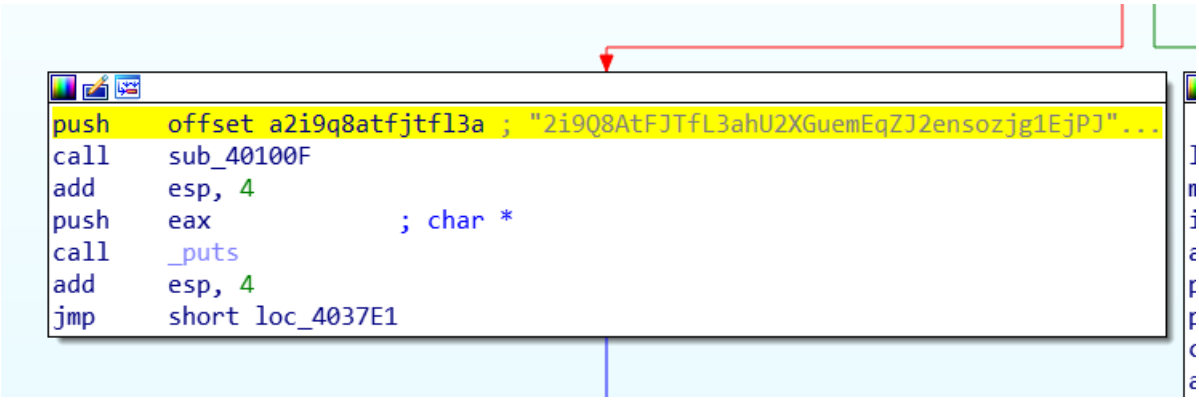
然后，里面有个虚假的flag，base64接出来是Why don't you try the magic brick game，明示让玩打砖块游戏。



进入游戏，没过一会就当场去世以后，得到flag，如下图所示：



当然有些师傅直接看到了base58编码的flag，直接拿去解密速度就更快了。

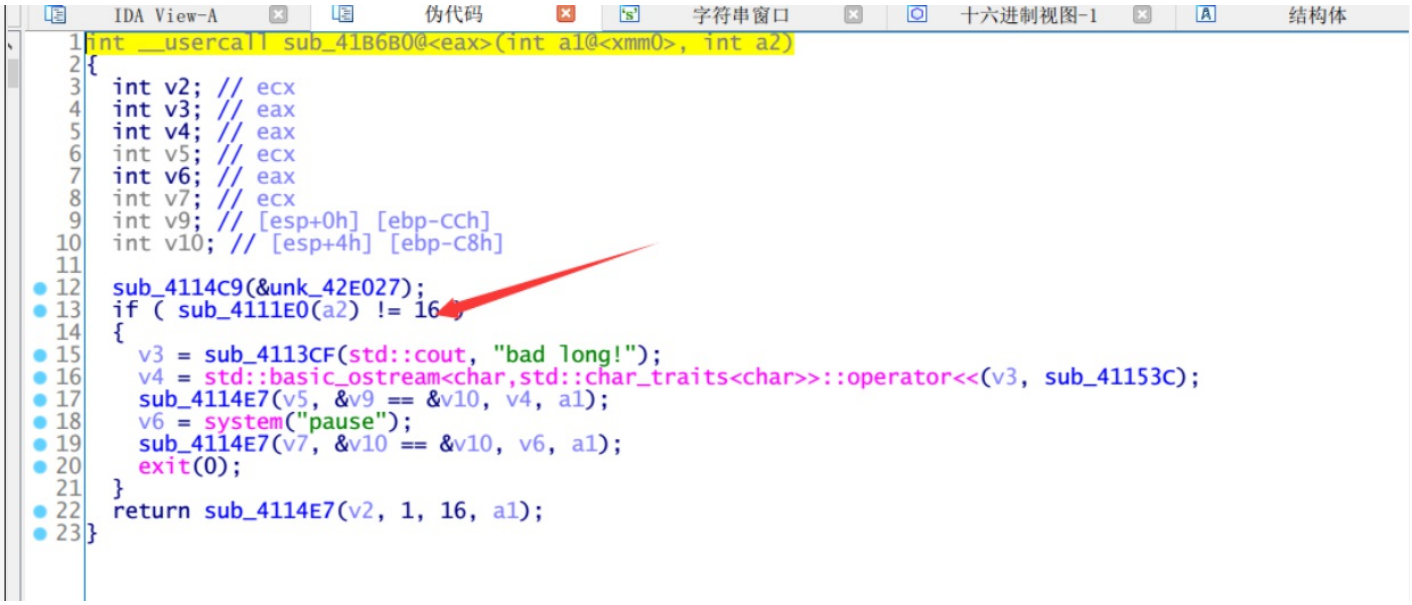


0x2 Chelly's Identity

输入 input 拷贝到 instr，并依次进行将 instr 传入三个函数

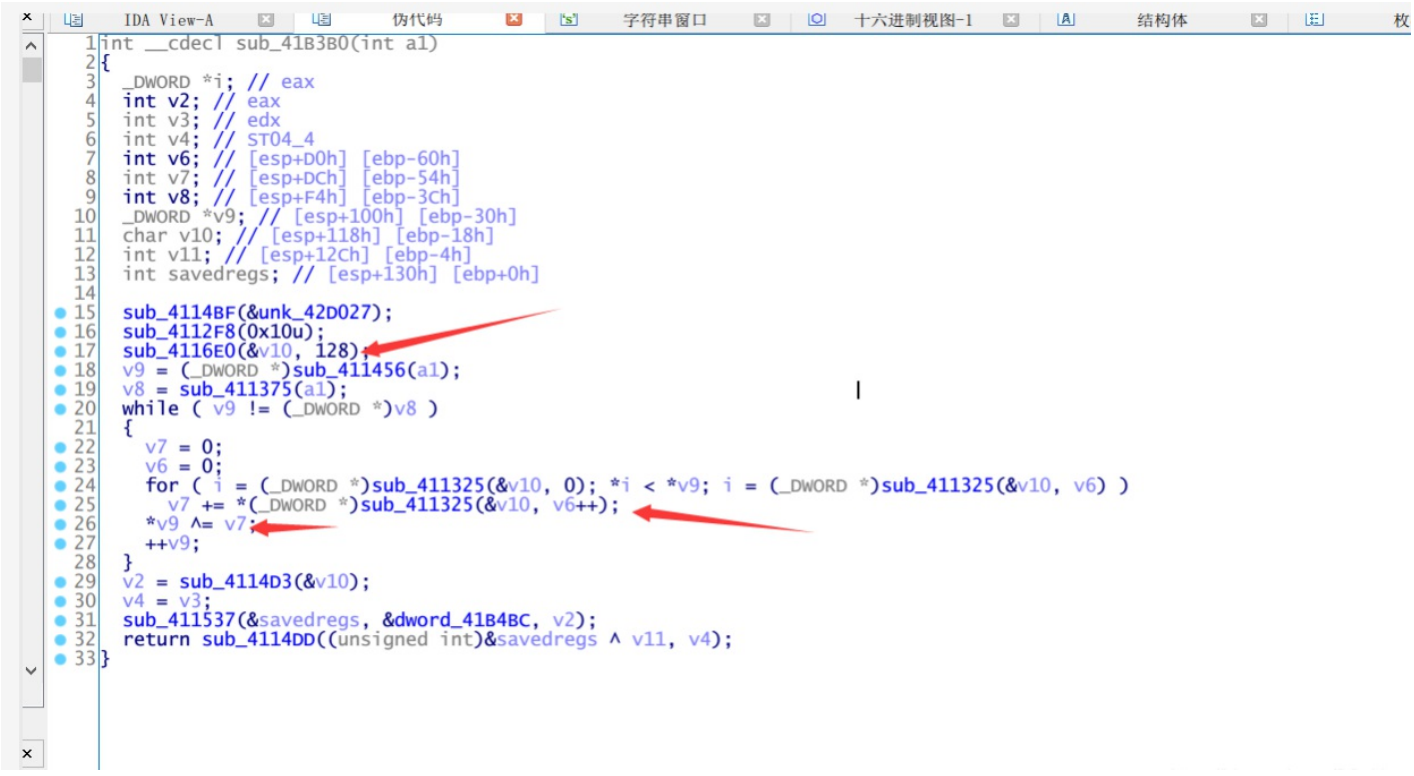
```
v2 = std::basic_ostream<char,std::char_traits<char>>::operator<<(v1, sub_41153C);
sub_4114E7(v3, &v30 == &v30, v2, a1);
v4 = sub_4113CF(std::cout, "Can you speak chelly's identity?");
v5 = std::basic_ostream<char,std::char_traits<char>>::operator<<(v4, sub_41153C);
sub_4114E7(v6, &v30 == &v30, v5, a1);
v7 = sub_4113CF(std::cout, "if you can, I will give you flag.");
v8 = std::basic_ostream<char,std::char_traits<char>>::operator<<(v7, sub_41153C);
sub_4114E7(v9, &v30 == &v30, v8, a1);
v10 = sub_4113CF(std::cout, "Give your answer:");
v11 = std::basic_ostream<char,std::char_traits<char>>::operator<<(v10, sub_41153C);
sub_4114E7(v12, &v30 == &v30, v11, a1);
sub_4110FF(std::cin, &input);
sub_411302(0x10u);
sub_411334(&instr);
LOBYTE(v37) = 1;
for ( i = 0; ; ++i )
{
  v13 = sub_41177B((int)&input, a1);
  if ( v13 == sub_4111E0(&instr) )
    break;
  v32 = *(char *)sub_411532((int)&input, a1, i);
  sub_4115DC(a1, (int)&v32);
}
sub_4111C7(&instr);
sub_41172B(&instr);
if ( (unsigned __int8)sub_41185C(&instr) )
{
  v29 = (char *)sub_41153C;
  v14 = sub_4113CF(std::cout, "You do know Chelly!!!! ");
  v15 = std::basic_ostream<char,std::char_traits<char>>::operator<<(v14, sub_41153C);
  v17 = sub_4114E7(v16, &v28 == (int *)&v29, v15, a1);
  v18 = sub_4113CF(v17, v29);
  v19 = std::basic_ostream<char,std::char_traits<char>>::operator<<(v18, v30);
  sub_4114E7(v20, &v30 == &v30, v19, a1);
}
else
{
  v21 = sub_4113CF(std::cout, "it's not chelly's identity.");
  v22 = std::basic_ostream<char,std::char_traits<char>>::operator<<(v21, sub_41153C);
  sub_4114E7(v23, &v30 == &v30, v22, a1);
}
v24 = system("pause");
```


第一个函数用于限制输入长度为16字符。



第二个函数是加密函数。

首先生成了向量v10，紧接着用一个循环遍历输入，向量中元素小于instr中元素时累加向量的值，之后instr元素与这个值异或



加密函数中向量生成方式如下：

从2至len(此处为128)，若满足if判断则加入向量中。其中if判断是一个判断是否为质数的函数，也就是说，向量是生成len之内的质数

```
int __usercall sub_420530@<eax>(int xmm0_4_0@<xmm0>, int table, int len)
{
  int i; // [esp+E0h] [ebp-30h]
  char v5; // [esp+ECh] [ebp-24h]
  int v6; // [esp+100h] [ebp-10h]
  int v7; // [esp+10Ch] [ebp-4h]
  int savedregs; // [esp+110h] [ebp+0h]

  sub_4115F0((int)&unk_434029);
  sub_4113CF(&v5, 0x10u);
  sub_411406((int)&v5, xmm0_4_0);
  v7 = 0;
  for ( i = 2; i < len; ++i )
  {
    if ( (unsigned __int8)sub_4117FD(xmm0_4_0, i) )
      sub_41173F(xmm0_4_0, (int)&i);
  }
  sub_4119A6(&v5);
  v7 = -1;
  sub_411609((int)&v5, xmm0_4_0);
  sub_41167C((int)&savedregs, (int)&dword_420640);
  return sub_411613((unsigned int)&savedregs ^ v6, 1, table, xmm0_4_0);
}
```

简言之，加密函数将输入的每个字符对一个值进行异或，这个值是向量v10中低于该字符的元素之和。

第三个函数是一个check输入经过加密后是否正确的函数。

与一个数组进行判等。

```
  sub_4112F8(0x10u);
  v9 = 438;
  v10 = 1176;
  v11 = 1089;
  v12 = 377;
  v13 = 377;
  v14 = 1600;
  v15 = 924;
  v16 = 377;
  v17 = 1610;
  v18 = 924;
  v19 = 637;
  v20 = 639;
  v21 = 376;
  v22 = 566;
  v23 = 836;
  v24 = 830;
  v1 = sub_4113A2(&v28);
  v2 = (_DWORD *)sub_411519(&v9, &v25);
  sub_41155F(*v2, v2[1], v1);
  for ( i = 0; ; ++i )
  {
    v3 = sub_4111D6(a1);
    if ( i >= v3 )
      break;
    v4 = (_DWORD *)sub_411325(a1, i);
    if ( *v4 != *(_DWORD *)sub_411325(&v30, i) )
    {
      v27 = 0;
      v6 = sub_4114D3(&v30);
      LOBYTE(v6) = v27;
      goto LABEL_7;
    }
  }
  v26 = 1;
  v6 = sub_4114D3(&v30);
  LOBYTE(v6) = v26;
LABEL_7:
  v7 = v5;
  sub_411537(&savedregs, &dword_41B06C, v6);
  return sub_4114DD((unsigned int)&savedregs ^ v31, v7);
```

因此，只需要生成一个128内的质数数组，用判等数组进行爆破即可。#生成质数数组

```
def is_prime_num(num):
#判断num是否为质数
for i in range(2, num):
if num % i == 0:
return False
return True
def create_table(n):
#生成n之内的质数列表
table = []
for num in range(2, n):
if is_prime_num(num):
table.append(num)
return table
def de_anwser(_key):
#根据key，爆破
table = create_table(128)
flag = ''
for k in _key:
for ch in range(128):
count = 0
i = 0
while table[i] < ch:
count += table[i]
i += 1
tmp = ch ^ count
if tmp == k:
flag += chr(ch)
return flag
key = [438,1176,1089,377,377,1600,924,377,1610,924,637,639,376,566,836,830 ]
flag = de_anwser(key)
```

print(flag) #flag{Che11y_1s_EG0IST}

0x3 BabyDriver

题目思路来源：参考2016 HCTF Reverse题目seven

一看源码，你就知道这是一个打着驱动幌子的简单迷宫，师傅们可能卡在了驱动对应的键盘码这块，我将上下左右设置为为IKJL，键盘过滤驱动捕获点击以后达成条件会输出成功，但不知道为啥卸载驱动以后老是蓝屏。。。

进到驱动程序的入口点DriverEntry里面，可以看到Driverunload，和分发函数，主题人想用KdDisableDebugger：

IDA View-A | Pseudocode-A | Hex View-1 | Structures

```
__int64 __fastcall DriverEntry(PDRIVER_OBJECT DriverObject)
{
  PDRIVER_OBJECT v1; // rsi

  DriverObject->DriverUnload = (PDRIVER_UNLOAD)DriverUnload;
  v1 = DriverObject;
  KdDisableDebugger();
  memset64(v1->MajorFunction, (unsigned __int64)GeneralDispatch, 0x1Bui64);
  v1->MajorFunction[3] = (PDRIVER_DISPATCH)&ReadDispatch;
  v1->MajorFunction[22] = (PDRIVER_DISPATCH)&PowerDispatch;
  v1->MajorFunction[27] = (PDRIVER_DISPATCH)PnPDispatch;
  AttachDevice(v1);
  return 0i64;
}
```

因为是键盘过滤驱动，主要看下IRP读操作的回调函数，也就是CompletionRoutine的位置：

```
  struct _IO_STACK_LOCATION *v7; // rax

  v2 = a2;
  v3 = a1;
  if ( v2->CurrentLocation == 1 )
  {
    v2->IoStatus.Information = 0i64;
    v2->IoStatus.Status = -1073741808;
    IofCompleteRequest(v2, 0);
    result = -1073741808;
  }
  else
  {
    v5 = *(struct _DEVICE_OBJECT **)(a1[8] + 24i64);
    v6 = v2->Tail.Overlay.CurrentStackLocation;
    *(_OWORD *)&v6[-1].MajorFunction = *(_OWORD *)&v6->MajorFunction;
    *(_OWORD *)&v6[-1].Parameters.MountVolume.DeviceObject = *(_OWORD *)&v6->Parameters.MountVolume.DeviceObject;
    *(_OWORD *)&v6[-1].Parameters.WMI.Buffer = *(_OWORD *)&v6->Parameters.WMI.Buffer;
    v6[-1].FileObject = v6->FileObject;
    v6[-1].Control = 0;
    v7 = v2->Tail.Overlay.CurrentStackLocation;
    v7[-1].Context = v3;
    v7[-1].CompletionRoutine = (PIO_COMPLETION_ROUTINE)ReadComp;
    v7[-1].Control = -32;
    result = IofCallDriver(v5, v2);
  }
  return result;
}
```

很快在函数内找到迷宫的地图和行走逻辑，与普通迷宫不同的是，是由键盘过滤驱动获取键盘扫描码来控制上下左右：

```
000140003000 ; Segment permissions: Read/write
000140003000 _data           segment para public 'DATA' use64
000140003000                 assume cs:_data
000140003000                 ;org 140003000h
000140003000 g_Maze          db '****************o.*..*......*..**.**...**.*.*.***.****.**.*.*.***'
000140003000                                         ; DATA XREF: ReadComp+48↑o
000140003000                 db '...**....*.*.*****..***.**.*..**.**.***.**.**.**.**.******.**.**.'
000140003000                 db '**....***.**.**.*****.***....**...***.**********..***......#****.'
000140003000                 db '*****************************',0
0001400030E1                 align 4
0001400030E4 dword_1400030E4 dd 10h                  ; DATA XREF: ReadComp+3C↑r
0001400030E4                                         ; ReadComp+7D↑w ...
```

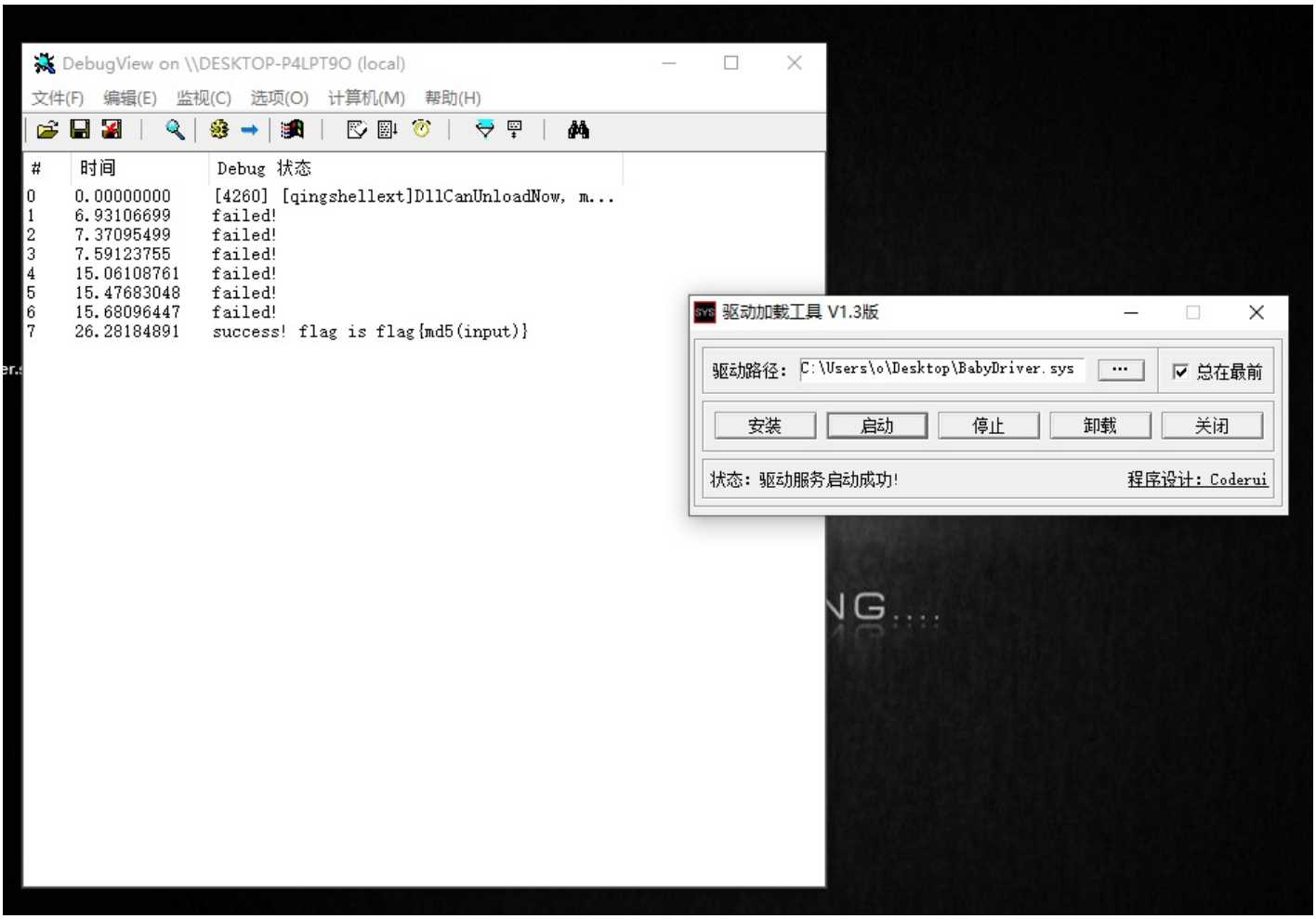
根据键盘过滤驱动获得的key值为，我们可以知道IKJL为上下左右：

键盘扫描码

Esc1		F159	F260	F361	F462		F563	F664	F765	F866		F967	F1068	F1187	F1288
` 41	1 2	2 3	3 4	4 5	5 6	6 7	7 8	8 9	910	011	- 12	= 13	43	← 14	
Tab15	Q 16	W17	E18	R19	T20	Y21	U22	I23	O24	P25	[26	] 27			
Caps58	A 30	S31	D32	F33	G34	H35	J36	K37	L38	;39	' 40	Enter 28			
Shift42	Z 44	X45	C46	V47	B48	N49	M50	,51	.52	/53	Shift 54				
Ctrl29	Win219	Alt184	Space 57								Alt184	Win220	Menu221	Ctrl 157	

```
if ( *v6 == 23 )                                    // I
{
  if ( v5 & 0xFFFFFFF0 )
  {
    v5 -= 16;
    goto LABEL_21;
  }
  v5 += 208;
  dword_1400030E4 = v5;
}
if ( v8 == 37 )                                     // K
{
  if ( (v5 & 0xFFFFFFF0) != 208 )
  {
    v5 += 16;
    goto LABEL_21;
  }
  v5 -= 208;
  dword_1400030E4 = v5;
}
if ( v8 == 36 )                                     // J
{
  if ( v5 & 0xF )
  {
    --v5;
    goto LABEL_21;
  }
  v5 += 15;
  dword_1400030E4 = v5;
}
if ( v8 != 38 )                                     // L
  goto LABEL_22;
if ( (v5 & 0xF) == 15 )
  v5 -= 15;
```

然后在虚拟机里加载下驱动，键盘依次敲击LKKKLLKLKKKLLLKKKLLLLLL，在有节奏感的敲击后，就可以获得success！



然后我虚拟机就蓝了。。。。

0x4 WannaReverse

题目思路来源：WannaCry的加密原理

这道题目被apeng师傅非预期了，膜就完事，首先讲下我认为的常规做法，我们拿到四个文件，主要需要逆向WannaReverse.exe这个文件，libcrypto-1_1.dll是openssl库用来提供rsa的相关加密函数，clickme.exe是仿照wannacry病毒的界面程序(自带换壁纸功能)，flag.txt.Encry是被WannaReverse.exe加密的flag。

题目加密流程和WannaCry的加密原理差不多，也是RSA2048 + AES随机密钥，但是为了大大降低题目难度，也不在加密算法这里留坑了，直接把私钥给了，在clickme.exe里点Decrypt按钮就有私钥(模仿WannaRen直接给密钥)

main函数功能如下图所示：

```
IDA View-A | Hex View-1 | Structures | Enums
1  int __cdecl main(int argc, const char **argv, const char **envp)
2  {
3    int v3; // eax
4    signed int v4; // esi
5    int v5; // ecx
6    char v6; // al
7    const char *v7; // ecx
8    __int128 v9; // [esp+8h] [ebp-34h]
9    __int128 v10; // [esp+18h] [ebp-24h]
10   char v11; // [esp+28h] [ebp-14h]
11   __int64 v12; // [esp+2Ch] [ebp-10h]
12
13   v9 = 0i64;
14   v10 = 0i64;
15   WinExec("clickme.exe", 1u);                    // 运行clickme程序
16   strcpy((char *)&v12, "0123456789");
17   v3 = time64(0);
18   srand(v3);                                     // 用时间做seed生成伪随机数来生成32字节的AES随机密钥
19   v4 = 0;
20   do
21     *((_BYTE *)&v9 + v4++) = *((_BYTE *)&v12 + rand() % 10);
22   while ( v4 < 32 );
23   v11 = 0;
24   v6 = encfile(v5, (int)&v9);                    // 文件加密
25   v7 = "encrypto success!";
26   if ( !v6 )
27     v7 = "encrypto false!";
28   printf(v7);
29   return 0;
30 }
```

然后在encfile函数里进行了关键的操作，最后将文件头+加密的AES密钥+AES加密的文件写入flag.txt.Encry文件：

```
39      v49 = v18;
40      *v18 = 0;
41      v18[1] = 0;
42      *v49 = &v49;
43      v51 = 0;
44      v52 = 15;
45      LOBYTE(v50) = 0;
46      LOBYTE(v54) = 3;
47      std::basic_string_char_std::char_traits_char__std::allocator_char___::basic_string_char_std::char_trai
48      LOBYTE(v54) = 4;
49      v20 = (int *)rsa_pub_encrypt(v19);        // rsa加密随机AES密钥
50      if ( &v49 != (_DWORD **)v20 )
51      {
52        std::basic_string_char_std::char_traits_char__std::allocator_char___::_Tidy_deallocate(&v49);
53        LOBYTE(v38) = 0;
54        std::basic_string_char_std::char_traits_char__std::allocator_char___::_Assign_rv_contents_with_alloc
55          v20,
56          v38);
57      }
58      std::basic_string_char_std::char_traits_char__std::allocator_char___::_Tidy_deallocate(&v42);
59      std::_String_alloc_std::_String_base_types_char_std::allocator_char_____::_Free_proxy(&v42);
60      LOBYTE(v54) = 3;
61      std::basic_string_char_std::char_traits_char__std::allocator_char___::_Tidy_deallocate(&v43);
62      std::_String_alloc_std::_String_base_types_char_std::allocator_char_____::_Free_proxy(&v43);
63      v21 = (int *)base64_encode(v51);          // 将加密的AES结果base64编码
64      if ( &v49 != (_DWORD **)v21 )
65      {
66        std::basic_string_char_std::char_traits_char__std::allocator_char___::_Tidy_deallocate(&v49);
67        LOBYTE(v38) = 0;
68        std::basic_string_char_std::char_traits_char__std::allocator_char___::_Assign_rv_contents_with_alloc
69          v21,
70          v38);
71      }
72      LOBYTE(v54) = 3;
73      std::basic_string_char_std::char_traits_char__std::allocator_char___::_Tidy_deallocate(&v42);
74      std::_String_alloc_std::_String_base_types_char_std::allocator_char_____::_Free_proxy(&v42);
75      v22 = (int *)&v50;
76      if ( v52 >= 0x10 )
```

标准的流程就这些，动调一下应该可以直接梳理清除，

解题脚本：#coding:utf-8

from Crypto.Cipher import AES

from binascii import b2a_hex, a2b_hex

import rsa

import base64

def aes_decrypt(text,key):

key = key.encode('utf-8')

mode = AES.MODE_ECB

cryptor = AES.new(key, mode)

plain_text = cryptor.decrypt(a2b_hex(text))

return plain_text.replace('\x00','')

def decrypt(crypt_text): # 用私钥解密

with open('rsa_private_key.pem', 'r') as privatefile:

p = privatefile.read()

```
privkey = rsa.PrivateKey.load_pkcs1(p)

lase_text = rsa.decrypt(crypt_text, privkey)

return lase_text

if name == 'main':

with open("flag.txt.Encry",'rb') as f:

res = f.read()[0xd:0x165]

print res

res = base64.b64decode(res)

key = decrypt(res)

with open("flag.txt.Encry",'rb') as f:

res = f.read()[0x165:]

print (res.encode('hex'))

print (aes_decrypt(res.encode('hex'),key))
```

```
28      key = decrypt(res)
R6AlR0HASXaugIAawobUR2CafHOfsCvvbAhPmFSODz/audwDYr/c3lQnzjL8eERYk4Tw4roclSen8Nlg4HoPh6F7FFGg+H8MC8JX+zIX
FbStVvvyzgoU3gLZBut3Nz71xEeuuzjPKnz3sf4NfsPW6wB3TXiQXSEaRwp/oIfwp1WFkjYY3Ox9N/25PEPn407RYd/
id9BScQ3h9mh4C/WRU3lxlXnHzuPGrVA7Gb7oEvUCduaPP13zKGwB+4RQMsOoHyID2F06dIp2RFrUiS5nf8T7THo+7HJDwWhxDgqAUK5
zaMaF4Dv3sl38w7nEk3jGSiFmbx83ROVqULkfs+g4fA==
5cbcea89ba2b1827793f130a8a97b49bcd789bd8359205454c22a56937eb6e2b0ebd840f916138f6f1ba991941720791f0266806
61265c2035ddcffc77575481f2f2e4afbfa21d29ae6c083b761b66b8fe72cbd694c3d56ae70c7a28dcbcac80
flag{385fa869-3046-44ee-9d30-c03551273867}
[Finished in 0.9s]
```

但是这里必须贴一下我失误的伪随机用法，apeng师傅的非预期解法，学习到了：

WannaReverse

比赛结束一分钟算出来的...好坑

先随机生成key,然后用rsa加密这串key.以为可以分解N,这里卡了很久.

key作为aes密钥加密flag.

后面直接爆key,从结果中找到包含flag的.初步筛选找不到,试了下是用\x00padding的.然后看到是utf-16编码的...

```
from Crypto.Cipher import AES
from base64 import *

def gen_key(seed):
    k = b""
    for i in range(32):
        seed = seed * 214013 + 2531011
        k+=bytes([0x30+(((seed>>16) & 0x7fff)%10)])
    return k
t = 1590310000
# t = 1589530000

while True:
    key = gen_key(t)
    # print(key)
    aes = AES.new(key, mode=AES.MODE_ECB)
    cipher = b"\\\xbc\xea\x89\xba+\x18\'y?
\x13\n\x8a\x97\xb4\x9b\xcdx\x9b\xd85\x92\x05EL\"\xa5i7\xebn+\x0e\xbd\x84\x0f\x91a8\xf6\xf
1\xba\x99\x19Ar\x07\x91\xf0&h\x06a&\\
5\xdd\xcf\xfcwWT\x81\xf2\xf2\xe4\xaf\xbf\xa2\x1d)\xael\x08;v\x1bf\xb8\xfer\xcb\xd6\x94\xc
3\xd5j\xe7\x0cz(\xdc\xbc\xac\x80"
    if aes.decrypt(cipher).endswith(b"\x00\x00\x00"):
        print(t)
        print(aes.decrypt(cipher).decode("utf-16"))
        exit()
    t-=1
    if t%10000 == 0:
        print(t)
```

师傅的脚本：from Crypto.Cipher import AES

from base64 import *

def gen_key(seed):

k = b""

for i in range(32):

seed = seed * 214013 + 2531011

k+=bytes([0x30+(((seed>>16) & 0x7fff)%10)])

return k

t = 1590310000

```
# t = 1589530000

while True:

key = gen_key(t)

# print(key)

aes = AES.new(key, mode=AES.MODE_ECB)

cipher = b"\\\xbc\xea\x89\xba+\x18\'y?
\x13\n\x8a\x97\xb4\x9b\xcdx\x9b\xd85\x92\x05EL\"\xa5i7\xebn+\x0e\xbd\x84\x0f\x91a8\xf6\xf1\xba\x99\x19Ar\x07
5\xdd\xcf\xfcwWT\x81\xf2\xf2\xe4\xaf\xbf\xa2\x1d)\xael\x08;v\x1bf\xb8\xfer\xcb\xd6\x94\xc3\xd5j\xe7\x0cz(\xdc\xb

if aes.decrypt(cipher).endswith(b"\x00\x00\x00"):

print(t)

print(aes.decrypt(cipher).decode("utf-16"))

exit()

t-=1

if t%10000 == 0:

print(t)
```

我这样的勒索病毒，就是在白给。。。。

0x5 EzMachine

如果没有限制的话，会存在多解的情况，因此后来更新了题目描述，flag的组成仅包括：字母、大括号、大括号回头和下划线。

首先打开IDA，来到main函数，发现有混淆干扰了反汇编，手动patch为nop

```
.text:00401580
.text:00401580 loc_401580:                             ; CODE XREF: sub_401940+19↓p
.text:00401580                 push    ebp
.text:00401581                 mov     ebp, esp
.text:00401583                 sub     esp, 0Ch
.text:00401586                 push    ebx
.text:00401587                 push    esi
.text:00401588                 push    edi
.text:00401589                 call    sub_4014F0
.text:0040158E                 cmp     eax, ecx
.text:00401590                 jnz     short loc_401595
.text:00401592                 jz      short loc_401595
.text:00401594                 nop                                    patch为nop
.text:00401595
.text:00401595 loc_401595:                             ; CODE XREF: .text:00401590↑j
.text:00401595                                         ; .text:00401592↑j ...
.text:00401595                 mov     eax, 1
.text:0040159A                 test    eax, eax
.text:0040159C                 jz      short loc_4015EE
.text:0040159E                 mov     ecx, dword_445BD8
.text:004015A4                 mov     dl, byte_4449A0[ecx]
.text:004015AA                 mov     [ebp-1], dl
.text:004015AD                 mov     dword ptr [ebp-8], 0
.text:004015B4                 jmp     short loc_4015BF
```

然后创建函数并查看伪代码，发现是VM结构，下面开始分析。

off_4448F4处保存各个opcode对应函数的地址，4449A0处为虚拟机代码起始地址，dword_445BD8为ip，off_4427FC处保存有四个寄存器的地址，dword_445BAC为栈的起始地址，dword_445BC8为esp，opcode中有对栈的操作、跳转操作、取输入的操作以及加减乘除异或五种运算，分析各个opcode的作用后，可以写脚本生成伪汇编代码：

```
opcode = {0x00:'nop',0x01:'mov',0x02:'pushi',0x03:'pushr',0x04:'pop',0x05:'print',0x06:'add',0x07:'sub',0x08:'mul',(
```

```
operand = {'nop':0,'mov':2,'pushi':1,'pushr':1,'pop':1,'print':0,'add':2,'sub':2,'mul':2,'div':2,'xor':2,'jmp':1,'cmp':2,'je':1,
```

```
code = [

0x01, 3,3,

0x05, 0x00, 0x00,

0x11, 0x00, 0x00,

0x01, 1,17,

0x0C, 0,1,

0x0D, 10, 0x00,

0x01, 3,1,

0x05, 0x00, 0x00,

0xFF, 0x00, 0x00,

0x01, 2, 0,

0x01, 0, 17,

0x0C, 0, 2,

0x0D, 43, 0x00,

0x14, 0, 2,

0x01, 1, 97,

0x0C, 0, 1,

0x10, 26, 0x00,

0x01, 1, 122,

0x0C, 0, 1,

0x0F, 26, 0x00,

0x01, 1, 71,

0x0A, 0, 1,

0x01, 1, 1,

0x06, 0, 1,
```

0x0B, 36, 0x00,

0x01, 1, 65,

0x0C, 0, 1,

0x10, 36, 0x00,

0x01, 1, 90,

0x0C, 0, 1,

0x0F, 36, 0x00,

0x01, 1, 75,

0x0A, 0, 1,

0x01, 1, 1,

0x07, 0, 1,

0x01, 1, 16,

0x09, 0, 1,

0x03, 1, 0x00,

0x03, 0, 0x00,

0x01, 1, 1,

0x06, 2, 1,

0x0B, 11, 0x00,

0x02, 0x7, 0x00,

0x02, 0xD, 0x00,

0x02, 0x0, 0x00,

0x02, 0x5, 0x00,

0x02, 0x1, 0x00,

0x02, 0xC, 0x00,

0x02, 0x1, 0x00,

0x02, 0x0, 0x00,

0x02, 0x0, 0x00,

0x02, 0xD, 0x00,

0x02, 0x5, 0x00,

0x02, 0xF, 0x00,

0x02, 0x0, 0x00,

0x02, 0x9, 0x00,

0x02, 0x5, 0x00,

0x02, 0xF, 0x00,

0x02, 0x3, 0x00,

0x02, 0x0, 0x00,

0x02, 0x2, 0x00,

0x02, 0x5, 0x00,

0x02, 0x3, 0x00,

0x02, 0x3, 0x00,

0x02, 0x1, 0x00,

0x02, 0x7, 0x00,

0x02, 0x7, 0x00,

0x02, 0xB, 0x00,

0x02, 0x2, 0x00,

0x02, 0x1, 0x00,

0x02, 0x2, 0x00,

0x02, 0x7, 0x00,

0x02, 0x2, 0x00,

0x02, 0xC, 0x00,

0x02, 0x2, 0x00,

0x02, 0x2, 0x00,

0x01, 2, 1,

0x13, 1, 2,

0x04, 0, 0x00,

0x0C, 0, 1,

0x0E, 91, 0x00,

0x01, 1, 34,

0x0C, 2, 1,

0x0D, 89, 0x00,

0x01, 1, 1,

0x06, 2, 1,

0x0B, 78, 0x00,

0x01, 3, 0,

0x05, 0x00, 0x00,

0xFF, 0x00, 0x00,

0x01, 3, 1,

0x05, 0x00, 0x00,

0xFF, 0x00, 0x00,

]

for i in range(0,len(code),3):

if not code[i] in opcode.keys():

print('unknow')

continue

op = opcode[code[i]]

print(op, end = ' ')

if operand[op] != 0:

print(code[i+1:i+1+operand[op]])

else:

print()

得到伪汇编代码：mov [3, 3]

print

input

mov [1, 17] //flag长度

cmp [0, 1]

je [10]

mov [3, 1]

print

quit

mov [2, 0]

mov [0, 17]

cmp [0, 2]

je [43]

```
LoadString [0, 2]
mov [1, 97]
cmp [0, 1]
jl [26] //if input[i] < 'a': jmp
mov [1, 122]
cmp [0, 1]
jg [26] //if input[i] > 'z': jmp
zmov [1, 71]
xor [0, 1]
mov [1, 1]
add [0, 1]
jmp [36]
mov [1, 65]
cmp [0, 1]
jl [36] //if input[i] < 'A': jmp
mov [1, 90]
cmp [0, 1]
jg [36] //if input[i] > 'Z': jmp
mov [1, 75]
xor [0, 1]
mov [1, 1]
sub [0, 1]
mov [1, 16]
div [0, 1]
pushr [1]
pushr [0]
mov [1, 1]
add [2, 1]
jmp [11]
pushi [7]
pushi [13]
```

pushi [0]

pushi [5]

pushi [1]

pushi [12]

pushi [1]

pushi [0]

pushi [0]

pushi [13]

pushi [5]

pushi [15]

pushi [0]

pushi [9]

pushi [5]

pushi [15]

pushi [3]

pushi [0]

pushi [2]

pushi [5]

pushi [3]

pushi [3]

pushi [1]

pushi [7]

pushi [7]

pushi [11]

pushi [2]

pushi [1]

pushi [2]

pushi [7]

pushi [2]

pushi [12]

pushi [2]

pushi [2]

mov [2, 1]

LoadStack [1, 2]

pop [0]

cmp [0, 1]

jn [91]

mov [1, 34]

cmp [2, 1]

je [89]

mov [1, 1]

add [2, 1]

jmp [78]

mov [3, 0]

print

quit

mov [3, 1]

print

quit

伪代码中，类似于mov、add、sub等指令后面的操作数为寄存器编号，例如 mov [1,2]代表将2号寄存器的内容放入1号寄存器。接下来分析伪汇编代码，发现是对输入的值进行处理，小写字母异或71后+1，大写字母异或75后-1，非字母不处理，处理得到的结果除以16，得到的商和余数分别进栈，并在最后部分比较，因此可以写出exp计算

```
flag:array = [0x7,0xd,0x0,0x5,0x1,0xc,0x1,0x0,0x0,0xd,0x5,0xf,0x0,0x9,0x5,0xf,0x3,0x0,0x0,0x2,0x5,0x3,0x3,0x1,0x7,
```

```
array = array[::-1]

for i in range(0, len(array), 2):

c = array[i] + array[i+1]*16

tmp = (c-1) ^ 71

if tmp >= ord('a') and tmp <= ord('z'):

print(chr(tmp), end = "")

continue

tmp = (c+1) ^ 75

if tmp >= ord('A') and tmp <= ord('Z'):
```

print(chr(tmp), end = "")

continue

print(chr(c), end = "")

flag{Such_A_EZVM}

0x6 DbgIsFun

首先检查tls回调，出现了smc，解密方法为第i位与i异或，解密后创建了运行该函数的线程

```
void __stdcall TlsCallback_0(int a1, int a2, int a3)
{
  signed int v3; // eax
  DWORD flOldProtect; // [esp+0h] [ebp-8h]

  if ( a2 == 1 )
  {
    VirtualProtect(StartAddress, 0x42Eu, 0x40u, &flOldProtect);
    v3 = 0;
    do
    {
      *((_BYTE *)StartAddress + v3) ^= v3;
      ++v3;
    }
    while ( v3 < 1070 );
    CreateThread(0, 0, StartAddress, 0, 0, 0);
  }
}
```

main函数的开头可以看到安装了seh函数

```
loc_4015C0:                             ; CODE XREF: start-8D↓p
                push    ebp
                mov     ebp, esp
                push    ecx
                push    ebx
                push    esi
                push    edi
                push    offset loc_401540
                push    large dword ptr fs:0
                mov     large fs:0, esp
                xor     bl, bl
                nop     dword ptr [eax+00h]
```

安装后进行输入，然后对输入的长度减了一个0x1C，并随后触发int3断点来到seh函数。

```
loc_4015C0:                             ; CODE XREF: start-8D↓p
                push    ebp
                mov     ebp, esp
                push    ecx
                push    ebx
                push    esi
                push    edi
                push    offset loc_401540
                push    large dword ptr fs:0
                mov     large fs:0, esp
                xor     bl, bl
                nop     dword ptr [eax+00h]
```

SEH中的判断如图所示。在main函数中对flag长度做的sub操作会影响EFlags的值，当输入长度等于0x1C时，ZFlag置0，SEH函数由此判断输入的长度是否正确，如果错误会将程序的流程引向一段假的flag解密函数。长度正确则对标志位置1。

```
loc_401572:                             ; CODE XREF: .text:00401564↑j
                add     ebx, 2          ; 引向错误的flag解密函数
                mov     [ecx+0B8h], ebx
                jz      short loc_4015AF
                jnz     short loc_4015AF
; ---------------------------------------------------------------------------
                db 0E9h
; ---------------------------------------------------------------------------

loc_401580:                             ; CODE XREF: .text:0040155D↑j
                cmp     edx, 48BCCCCh
                jnz     short loc_4015A2
                mov     dword_41A8E0, 1 ; 标志位置1
                push    1388h
                call    ds:Sleep
                call    sub_403AFA
```

dword_41A8E0这个标志位将在子线程的函数中被循环检查，静态下该函数还未解密，所以需要进行动态调试等待代码段解密。解密后来到子线程函数：

```
0040110A   833D E0A84100   cmp dword ptr ds:[0x41A8E0],0x0      检查标志位是否为0，为0则继续Sleep
00401111   75 0D           jnz short DbgIsFun.00401120
00401113   68 E8030000     push 0x3E8
00401118   FF15 00304100   call dword ptr ds:[<&KERNEL32.Sleep>]   kernel32.Sleep
0040111E   EB EA           jmp short DbgIsFun.0040110A
00401120   C785 ECFFFEFF   mov dword ptr ss:[ebp-0x10014],0x0
0040112A   81BD ECFFFEFF   cmp dword ptr ss:[ebp-0x10014],0x8C
00401134   7D 2E           jge short DbgIsFun.00401164
00401136   B8 40154000     mov eax,DbgIsFun.00401540
0040113B   0385 ECFFFEFF   add eax,dword ptr ss:[ebp-0x10014]
00401141   0FB608          movzx ecx,byte ptr ds:[eax]
00401144   0FB615 DEA8410  movzx edx,byte ptr ds:[0x41A8DE]   取41A8DE起始的代码段字节码之和
0040114B   03D1            add edx,ecx                        DbgIsFun.004010F0
0040114D   8815 DEA84100   mov byte ptr ds:[0x41A8DE],dl
00401153   8B85 ECFFFEFF   mov eax,dword ptr ss:[ebp-0x10014]
00401159   83C0 01         add eax,0x1
0040115C   8985 ECFFFEFF   mov dword ptr ss:[ebp-0x10014],eax
00401162   EB C6           jmp short DbgIsFun.0040112A
```

当dword_41A8E0处的标志位被置1后，函数跳出Sleep死循环开始运行，首先会取41A8DE起始的前0x8C个字节计算字节码之和，如果该段代码被patch或者存在断点，字节码之和就会改变，从而影响后续的计算结果。

```
0040116E   8B8D ECFFFEFF   mov ecx,dword ptr ss:[ebp-0x10014]
00401174   0FBE91 C0A8410  movsx edx,byte ptr ds:[ecx+0x41A8C0]
0040117B   85D2            test edx,edx                       DbgIsFun.004010F0
0040117D   74 33           je short DbgIsFun.004011B2
0040117F   0FB605 DEA8410  movzx eax,byte ptr ds:[0x41A8DE]
00401186   8B8D ECFFFEFF   mov ecx,dword ptr ss:[ebp-0x10014]
0040118C   0FBE91 C0A8410  movsx edx,byte ptr ds:[ecx+0x41A8C0]
00401193   33D0            xor edx,eax                        将前面得到的字节码之后与input的各位异或
00401195   8B85 ECFFFEFF   mov eax,dword ptr ss:[ebp-0x10014]
0040119B   8890 C0A84100   mov byte ptr ds:[eax+0x41A8C0],dl
004011A1   8B8D ECFFFEFF   mov ecx,dword ptr ss:[ebp-0x10014]
004011A7   83C1 01         add ecx,0x1
004011AA   898D ECFFFEFF   mov dword ptr ss:[ebp-0x10014],ecx  DbgIsFun.004010F0
004011B0   EB BC           jmp short DbgIsFun.0040116E
```

计算完字节码之和后与所输入字符串进行异或，再往后就是将异或后的结果进行RC4加密并校验，密钥为GKCTF

```
004011B2   C785 C8FFFEFF  mov dword ptr ss:[ebp-0x10038],0x47     G
004011BC   C785 CCFFFEFF  mov dword ptr ss:[ebp-0x10034],0x4B     K
004011C6   C785 D0FFFEFF  mov dword ptr ss:[ebp-0x10030],0x43     C
004011D0   C785 D4FFFEFF  mov dword ptr ss:[ebp-0x1002C],0x54     T     RC4密钥
004011DA   C785 D8FFFEFF  mov dword ptr ss:[ebp-0x10028],0x46     F
004011E4   C785 F4FFFEFF  mov dword ptr ss:[ebp-0x1000C],0x0
004011EE   EB 0F          jmp short DbgIsFun.004011FF
```

```
004014A5   74 1F          je short DbgIsFun.004014C6
004014A7   75 1D          jnz short DbgIsFun.004014C6
004014A9   E9             db E9
004014AA   2DD40FD0       dd D00FD42D
004014AE   54EE75D0       dd D075EE54
004014B2   E03096E1       dd E19630E0
004014B6   798AE0FE       dd FEE08A79
004014BA   183A27E7       dd E7273A18
004014BE   2F86C9FE       dd FEC9862F
004014C2   6643A775       dd 75A74366
004014C6   33DB           xor ebx,ebx
004014C8   8B0D E4A8410(  mov ecx,dword ptr ds:[0x41A8E4]      Default case of switch 004014E5
004014CE   3BD9           cmp ebx,ecx
004014D0   7D 38          jge short DbgIsFun.0040150A
004014D2   8B85 F8FFFEFF  mov eax,dword ptr ss:[ebp-0x10008]
004014D8   8A0418         mov al,byte ptr ds:[eax+ebx]
004014DB   8A93 AA14400(  mov dl,byte ptr ds:[ebx+0x4014AA]
004014E1   3AC2           cmp al,dl                            比较
004014E3   75 14          jnz short DbgIsFun.004014F9
004014E5   43             inc ebx                              Switch (cases FFFFFFFF..FFFFFFFF)
004014E6   74 E0          je short DbgIsFun.004014C8
004014E8   75 DE          jnz short DbgIsFun.004014C8
004014EA   E9             db E9                                Case FFFFFFFF of switch 004014E5
004014EB   77 72 6F 6E    ascii "wrong■",0
004014F2   72 69 67 68    ascii "right■",0
004014F9   68 EB144000    push DbgIsFun.004014EB               ASCII "wrong\n"
004014FE   E8 3D010000    call DbgIsFun.00401640
00401503   6A 00          push 0x0
00401505   E8 793C0000    call DbgIsFun.00405183
```

如图所示，校验时的数据存放在代码段，因此调试时的反汇编结果可能出现错误，需要手动矫正。从4014AA开始即为校验数据，将这段数据进行RC4解密，密钥GKCTF，再与前面计算得到的字节码之和进行异或即可得到flagfrom Crypto.Cipher import ARC4

key = b"GKCTF"

hexstr = "2DD40FD054EE75D0E03096E1798AE0FE183A27E72F86C9FE6643A775"

newstr = b""

for i in range(0, len(hexstr), 2):

newstr += int(hexstr[i:i+2],16).to_bytes(1,'little')

rc4 = ARC4.new(key)

flag = rc4.decrypt(newstr)

for i in flag:

print(chr(i^0xC9), end = "")

flag{5tay4wayFr0m8reakp0int}

温柔正确的人总是难以生存，因为这世界既不温柔，也不正确