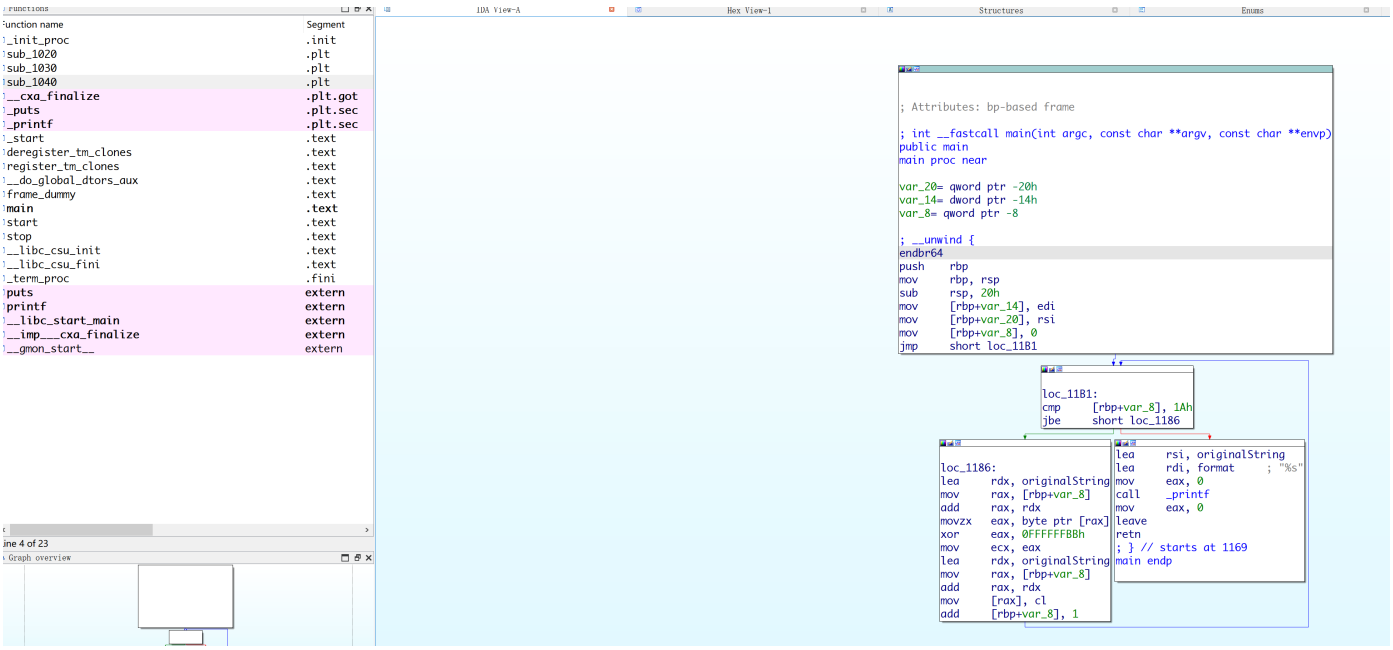
0x01 (dis)con

考察点：构造与析构函数；

查看程序架构

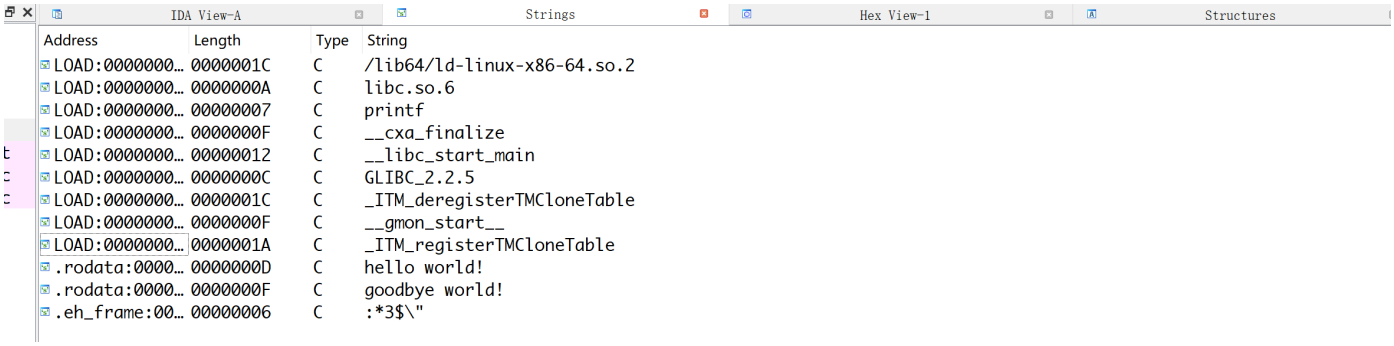
IDA分析

函数不多



字符串查看

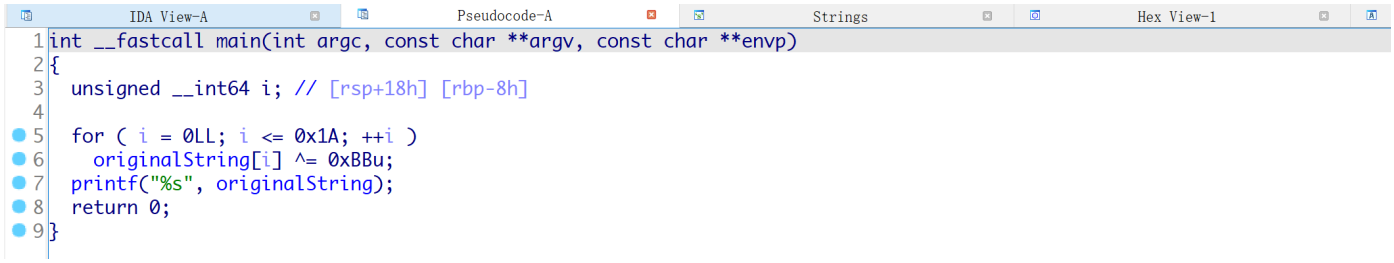
字符串里并没有flag字样



Address	Length	Type	String
LOAD:0000000…	0000001C	C	/lib64/ld-linux-x86-64.so.2
LOAD:0000000…	0000000A	C	libc.so.6
LOAD:0000000…	00000007	C	printf
LOAD:0000000…	0000000F	C	__cxa_finalize
LOAD:0000000…	00000012	C	__libc_start_main
LOAD:0000000…	0000000C	C	GLIBC_2.2.5
LOAD:0000000…	0000001C	C	_ITM_deregisterTMCloneTable
LOAD:0000000…	0000000F	C	__gmon_start__
LOAD:0000000…	0000001A	C	_ITM_registerTMCloneTable
.rodata:0000…	0000000D	C	hello world!
.rodata:0000…	0000000F	C	goodbye world!
.eh_frame:00…	00000006	C	:*3$\"

main函数

查看main函数，发现有对originalString的异或操作；



```
int __fastcall main(int argc, const char **argv, const char **envp)
{
  unsigned __int64 i; // [rsp+18h] [rbp-8h]

  for ( i = 0LL; i <= 0x1A; ++i )
    originalString[i] ^= 0xBBu;
  printf("%s", originalString);
  return 0;
}
```

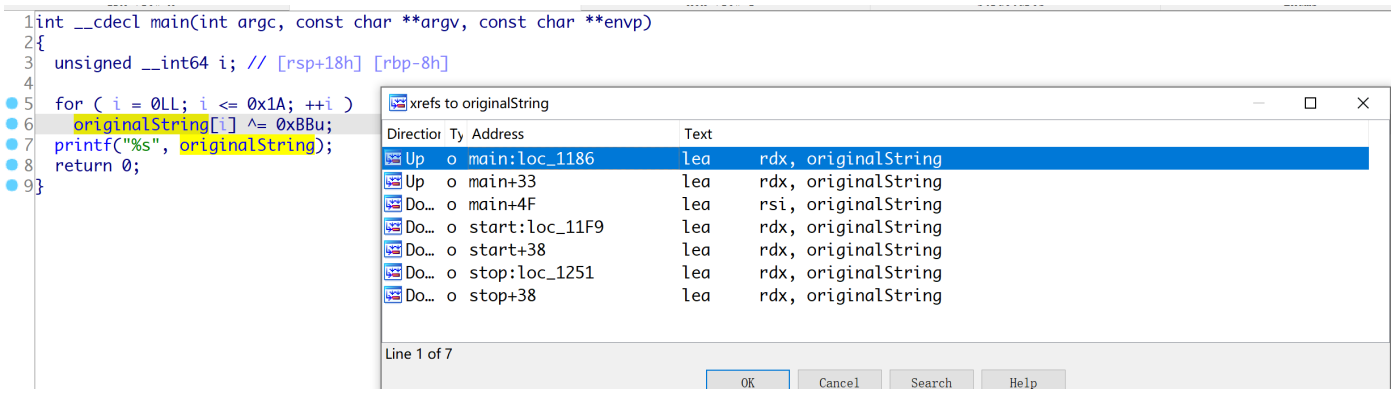
双击看定义,取出字符序列

```
flag_0 = [0x90, 0x92, 0x9e, 0x8e, 0x9e, 0x89, 0x9b, 0xa6, 0x9a, 0xed,
0xed, 0xb9, 0x82, 0xa9, 0x95, 0xb4, 0xae, 0x82, 0xec, 0xae, 0x82, 0xa9,
0xb5, 0xee, 0x82, 0xbb, 0xb1, 0x9c, 0xba, 0xa0]
```

这里直接异或算出结果并不是flag，而是乱码；

变量交叉引用

交叉引用看originalString；可以看到在start，main，stop三个函数都使用了变量；



```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  unsigned __int64 i; // [rsp+18h] [rbp-8h]

  for ( i = 0LL; i <= 0x1A; ++i )
    originalString[i] ^= 0xBBu;
  printf("%s", originalString);
  return 0;
}
```

xrefs to originalString

Direction	Ty	Address	Text
Up	o	main:loc_1186	lea rdx, originalString
Up	o	main+33	lea rdx, originalString
Do…	o	main+4F	lea rsi, originalString
Do…	o	start:loc_11F9	lea rdx, originalString
Do…	o	start+38	lea rdx, originalString
Do…	o	stop:loc_1251	lea rdx, originalString
Do…	o	stop+38	lea rdx, originalString

Line 1 of 7

OK Cancel Search Help

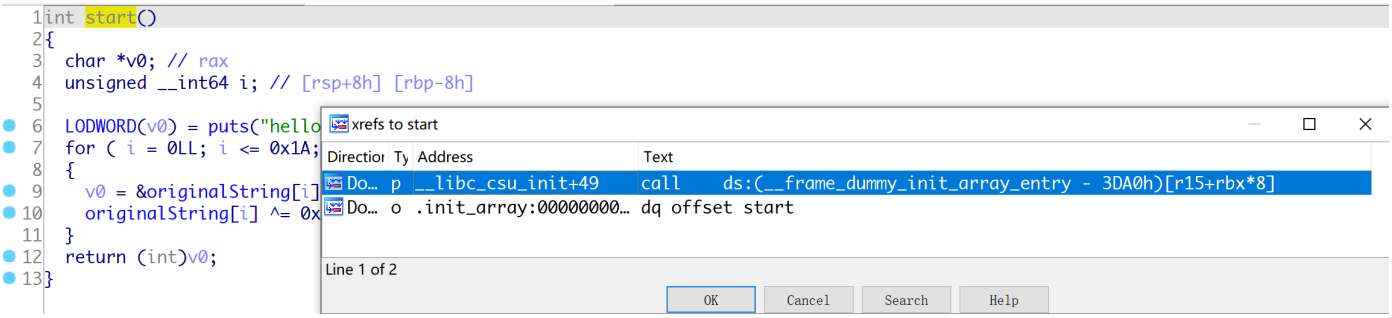
start函数

函数处理；

```
int start()
{
  char *v0; // rax
  unsigned __int64 i; // [rsp+8h] [rbp-8h]

  LODWORD(v0) = puts("hello world!");
  for ( i = 0LL; i <= 0x1A; ++i )
  {
    v0 = &originalString[i];
    originalString[i] ^= 0xAAu;
  }
  return (int)v0;
}
```

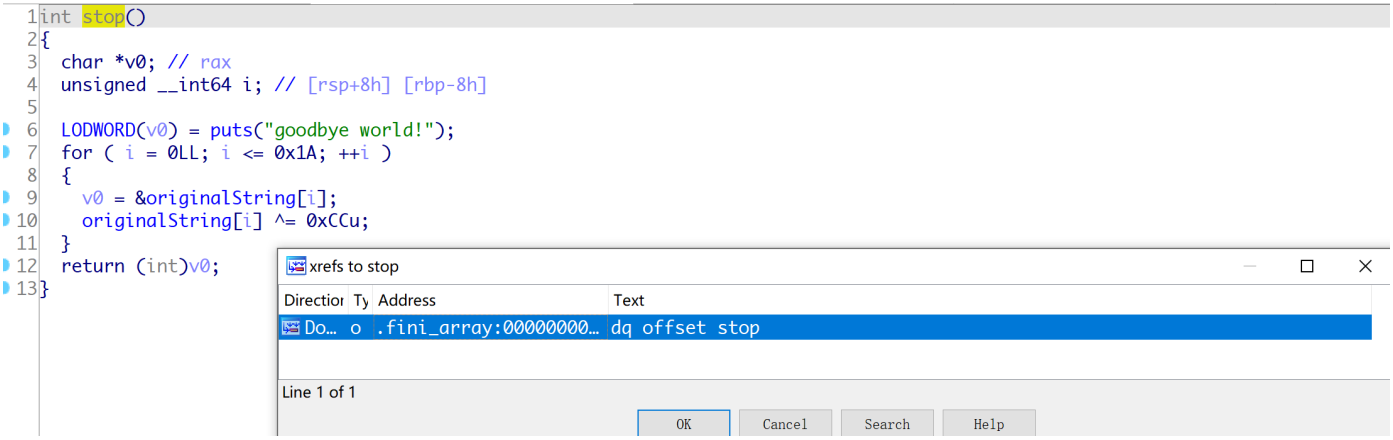
怎么知道调用了函数呢；看节；参考链接；

```
https://b0ldfrev.gitbook.io/note/pwn/zhong-xie-.finiarray-han-shu-zhi-zhen
```



stop函数

```
int stop()
{
  char *v0; // rax
  unsigned __int64 i; // [rsp+8h] [rbp-8h]

  LODWORD(v0) = puts("goodbye world!");
  for ( i = 0LL; i <= 0x1A; ++i )
  {
    v0 = &originalString[i];
    originalString[i] ^= 0xCCu;
  }
  return (int)v0;
}
```



函数处理流程

start --> main -> stop ， 这个题目没有增加难度，函数里面只进行了xor操作，如果进行多种操作，会涉及到顺序；这里不涉及顺序，直接xor三个hex数据即可；

Get flag

```
# Write Python 3 code in this online editor and run it.
flag_0 = [0x90, 0x92, 0x9e, 0x8e, 0x9e, 0x89, 0x9b, 0xa6, 0x9a, 0xed, 0xed, 0xb9, 0x82, 0xa9, 0x95, 0xb4, 0xae, 0x82, 0xec, 0xae, 0x82, 0xa9, 0xb5, 0xee, 0x82, 0xbb, 0xb1, 0x9c, 0xba, 0xa0]
#flag_0="MOCSCTF{G00d_tHis_1s_th3_flAg}"
flag=""
for i in flag_0:
    flag += chr(i^0xaa^0xbb^0xcc)
print(flag)

```

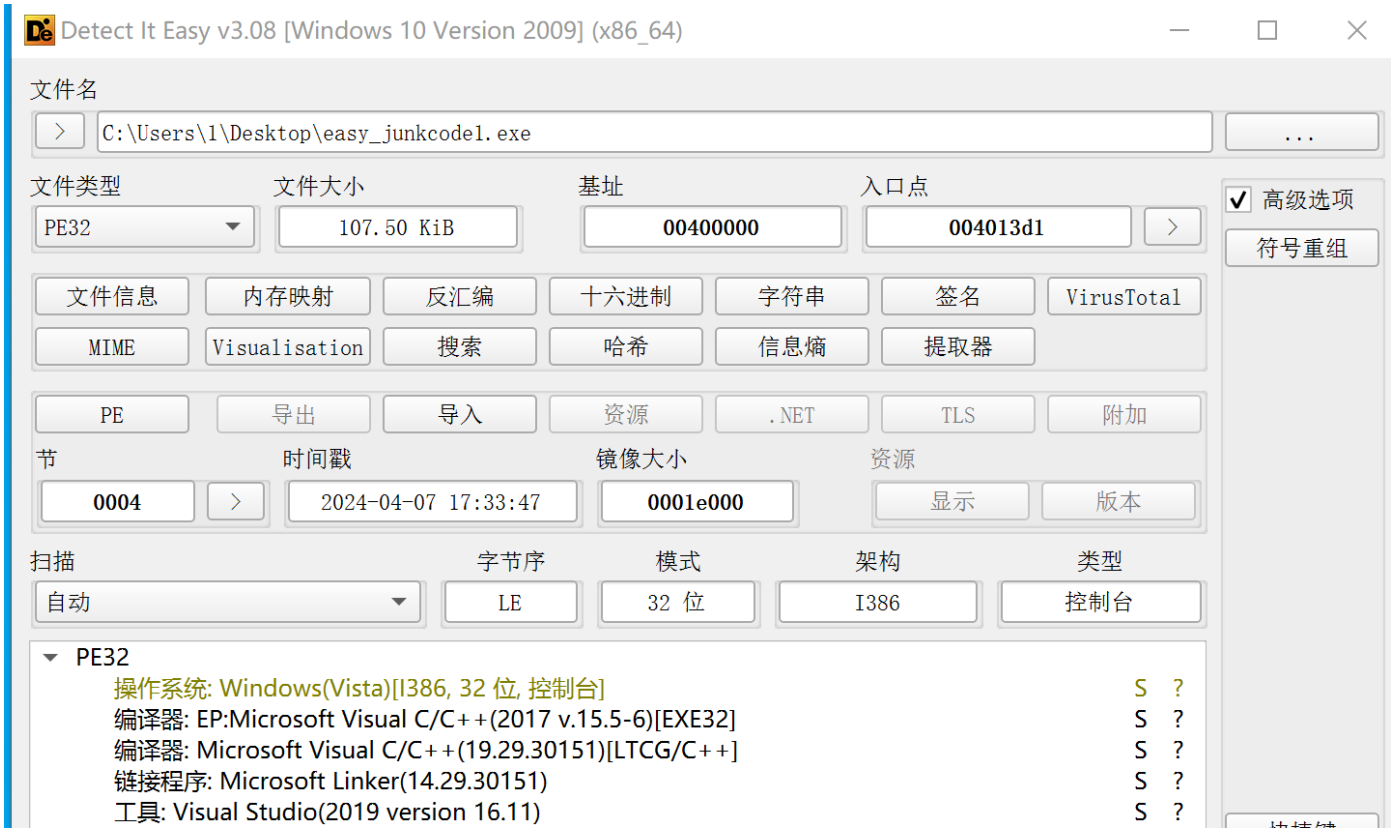
```
# Write Python 3 code in this online editor and run it.
flag_0 = [0x90, 0x92, 0x9e, 0x8e, 0x9e, 0x89, 0x9b, 0xa6, 0x9a, 0xed, 0xed, 0xb9, 0x82, 0xa9, 0x95, 0xb4, 0xae, 0x82, 0xec, 0xae, 0x82, 0xa9, 0xb5, 0xee, 0x82, 0xbb, 0xb1, 0x9c, 0xba, 0xa0]
#flag_0="MOCSCTF{G00d_tHis_1s_th3_flAg}"
flag=""
for i in flag_0:
    flag += chr(i^0xaa^0xbb^0xcc)
print(flag)
```

```
MOCSCTF{G00d_tHis_1s_th3_flAg}

=== Code Execution Successful ===
```

0x02 easy_junkr1.exe

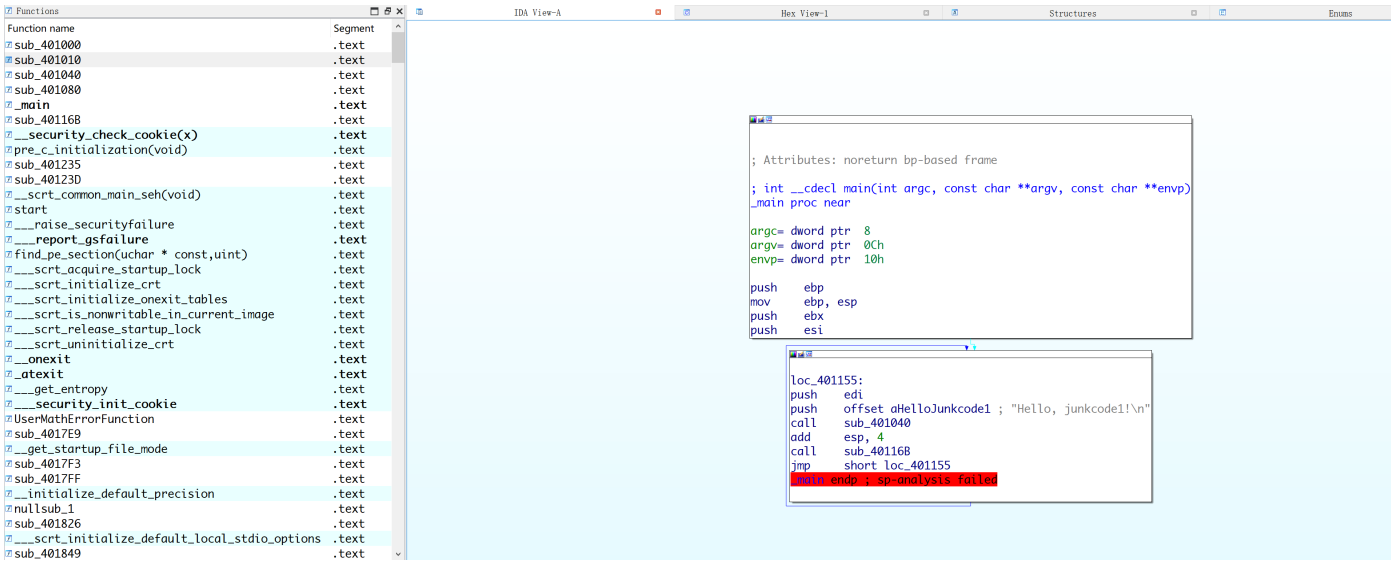
考察点：主程序花指令去除；

程序架构



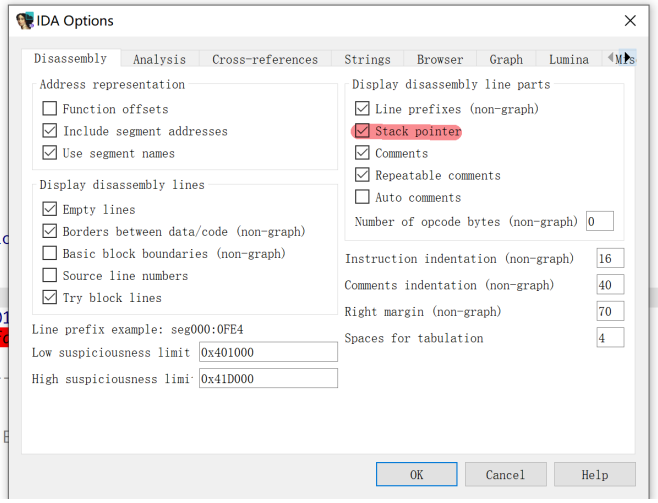
IDA分析

存在sp指针错误，题目提示了花指令，肯定是加了料了；



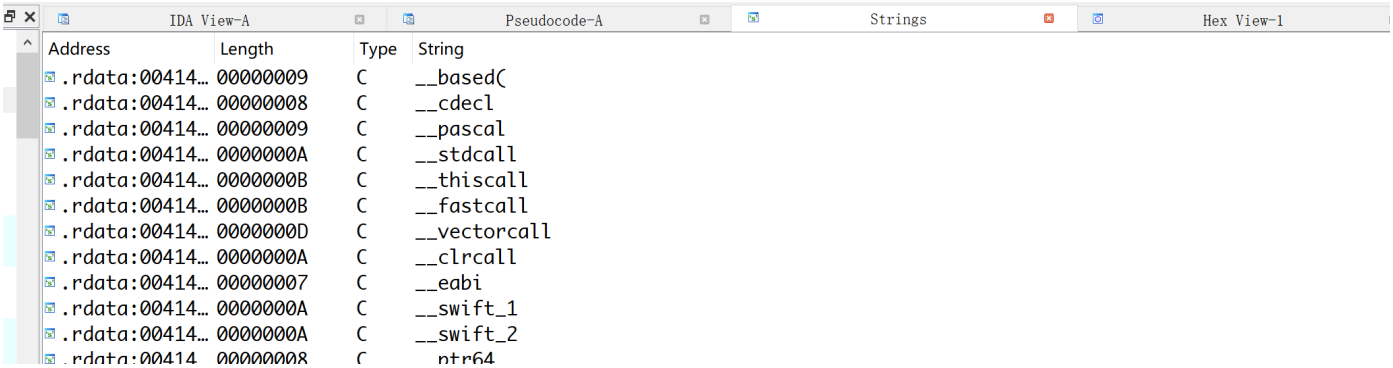
设置下栈指针，便于清除花指令；

```
.text:00401150 ; =============== S U B R O U T I N E =======================================
.text:00401150
.text:00401150 ; Attributes: noreturn bp-based frame
.text:00401150
.text:00401150 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00401150 _main           proc near               ; CODE XREF: __scrt_common_main_seh(void)+F5↓p
.text:00401150
.text:00401150 argc            = dword ptr  8
.text:00401150 argv            = dword ptr  0Ch
.text:00401150 envp            = dword ptr  10h
.text:00401150
.text:00401150                 push    ebp
.text:00401151                 mov     ebp, esp
.text:00401153                 push    ebx
.text:00401154                 push    esi
.text:00401155
.text:00401155 loc_401155:
.text:00401155                 push    edi
.text:00401156                 push    offset aHello
.text:0040115B                 call    sub_401040
.text:00401160                 add     esp, 4
.text:00401163                 call    sub_40116B
.text:00401168                 jmp     short loc_401
.text:00401168 _main           endp ; sp-analysis fa
.text:00401168
.text:00401168 ; ---------------------------------------
.text:0040116A                 db 5
.text:0040116B
.text:0040116B ; =============== S U B R O U T I N E
.text:0040116B
.text:0040116B
.text:0040116B sub_40116B      proc near
```



字符串查看

无flag相关字符串提示；



先不清除花指令看伪C

看不到flag相关函数；

```
1  int __cdecl __noreturn main(int argc, const char **argv, const char **envp)
2  {
3    char v3; // [esp+0h] [ebp-Ch]
4
5    while ( 1 )
6    {
7      sub_401040("Hello, junkcode1!\n", v3);
8      sub_40116B();
9    }
10 }
```

```
1  void sub_40116B()
2  {
3    char *retaddr; // [esp+0h] [ebp+0h]
4
5    ++retaddr;
6  }
```

分析红框内代码，影响了伪C代码的生成；

```
.text:00401155     loc_401155:                             ; CODE XREF: _main+18↓j
.text:00401155 00C                 push    edi
.text:00401156 010                 push    offset aHelloJunkcode1 ; "Hello, junkcode1!\n"
.text:0040115B 014                 call    sub_401040
.text:00401160 014                 add     esp, 4
.text:00401163 010                 call    sub_40116B
.text:00401168 010                 jmp     short loc_401155
.text:00401168     _main           endp ; sp-analysis failed
.text:00401168
.text:00401168     ; ---------------------------------------------------------------------------
.text:0040116A                     db 5
.text:0040116B
.text:0040116B     ; =============== S U B R O U T I N E =======================================
.text:0040116B
.text:0040116B
.text:0040116B     ; void sub_40116B()
.text:0040116B     sub_40116B      proc near               ; CODE XREF: _main+13↑p
.text:0040116B 000                 add     dword ptr [esp+0], 1
.text:0040116F 000                 retn
.text:0040116F     sub_40116B      endp
.text:0040116F
```

直接call了sub_40116B， 中间的jump和db定义很多余；将灰色代码nop掉；

```
.text:00401160                     add     esp, 4
.text:00401163                     call    sub_40116B
.text:00401168                     jmp     short loc_401155
.text:00401168 _main               endp ; sp-analysis failed
.text:00401168
.text:00401168 ; ---------------------------------------------------------------------------
.text:0040116A                     db 5
.text:0040116B
.text:0040116B ; =============== S U B R O U T I N E =======================================
.text:0040116B
.text:0040116B
.text:0040116B sub_40116B      proc near               ; CODE XREF: _main+13↑p
.text:0040116B                 add     dword ptr [esp+0], 1
.text:0040116F                 retn
.text:0040116F sub_40116B      endp
.text:0040116F
```

undefine下

```
text:00401150 ; int __cdecl main(int argc, const char **argv, const char **envp)
text:00401150 _main           proc near                                   ; __main_seh(void)+F5↓p
text:00401150
text:00401150 argc            = dword ptr  8
text:00401150 argv            = dword ptr  0Ch
text:00401150 envp            = dword ptr  10h
text:00401150
text:00401150                 push    ebp
text:00401151                 mov     ebp, es
text:00401153                 push    ebx
text:00401154                 push    esi
text:00401155                 push    edi
text:00401156                 push    offset                                ; "!\n"
text:0040115B                 call    sub_401
text:00401160                 add     esp, 4
text:00401163                 nop
text:00401164                 nop
text:00401165                 nop
text:00401166                 nop
text:00401167                 nop
text:00401168                 nop
text:00401169                 nop
text:00401169 _main           endp ; sp-analysis failed
text:00401169
text:0040116A                 nop
text:0040116B
text:0040116B ; =============== S U B R O U T I N E =======================================
text:0040116B
text:0040116B
text:0040116B sub_40116B      proc near
text:0040116B                 nop
text:0040116C                 nop
text:0040116D                 nop
text:0040116E                 nop
text:0040116F                 nop
text:0040116F sub_40116B      endp
text:0040116F
text:00401170                 call    sub_401080
text:00401175                 xor     eax, eax
text:00401177                 pop     edi
text:00401178                 pop     esi
text:00401179                 pop     ebx
text:0040117A                 pop     ebp
text:0040117B                 retn
```

```
.text:00401145                 align 10h
.text:00401150 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00401150 _main           db  55h ; U             ; CODE XREF: __scrt_common_main_seh(void)+F5↓p
.text:00401151                 db  8Bh
.text:00401152                 db 0ECh
.text:00401153                 db  53h ; S
.text:00401154                 db  56h ; V
.text:00401155                 db  57h ; W
.text:00401156                 db  68h ; h
.text:00401157                 db  10h ; OFF32 SEGDEF [_rdata,419810]
.text:00401158                 db  98h
.text:00401159                 db  41h ; A
.text:0040115A                 db    0
.text:0040115B                 db 0E8h
.text:0040115C                 db 0E0h
.text:0040115D                 db 0FEh
.text:0040115E                 db 0FFh
.text:0040115F                 db 0FFh
.text:00401160                 db  83h
.text:00401161                 db 0C4h
.text:00401162                 db    4
.text:00401163                 db  90h
.text:00401164                 db  90h
.text:00401165                 db  90h
.text:00401166                 db  90h
.text:00401167                 db  90h
.text:00401168                 db  90h
.text:00401169                 db  90h
.text:0040116A                 db  90h
.text:0040116B                 db  90h
.text:0040116C                 db  90h
.text:0040116D                 db  90h
.text:0040116E                 db  90h
.text:0040116F                 db  90h
.text:00401170                 db 0E8h
.text:00401171                 db  0Bh
.text:00401172                 db 0FFh
.text:00401173                 db 0FFh
.text:00401174                 db 0FFh
.text:00401175                 db  33h ; 3
.text:00401176                 db 0C0h
.text:00401177                 db  5Fh ; _
.text:00401178                 db  5Eh ; ^
.text:00401179                 db  5Bh ; [
.text:0040117A                 db  5Dh ; ]
.text:0040117B                 db 0C3h
```

转代码后创建函数

```
.text:00401150
.text:00401150 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00401150 _main:                                  scrt_common_main_seh(void)+F5↓p
.text:00401150                 push    ebp
.text:00401151                 mov     ebp
.text:00401153                 push    ebx
.text:00401154                 push    esi
.text:00401155                 push    edi
.text:00401156                 push    off                junkcode1!\n"
.text:0040115B                 call    sub
.text:00401160                 add     esp
.text:00401163                 nop
.text:00401164                 nop
.text:00401165                 nop
.text:00401166                 nop
.text:00401167                 nop
.text:00401168                 nop
.text:00401169                 nop
.text:0040116A                 nop
.text:0040116B                 nop
.text:0040116C                 nop
.text:0040116D                 nop
.text:0040116E                 nop
.text:0040116F                 nop
.text:00401170                 call    sub_401080
.text:00401175                 xor     eax, eax
.text:00401177                 pop     edi
.text:00401178                 pop     esi
.text:00401179                 pop     ebx
.text:0040117A                 pop     ebp
.text:0040117B                 retn
```


创建完毕

```
.text:00401150 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00401150 _main           proc near               ; CODE XREF: __scrt_common_main_seh(void)+F5↓p
.text:00401150                 push    ebp
.text:00401151                 mov     ebp, esp
.text:00401153                 push    ebx
.text:00401154                 push    esi
.text:00401155                 push    edi
.text:00401156                 push    offset aHelloJunkcode1 ; "Hello, junkcode1!\n"
.text:0040115B                 call    sub_401040
.text:00401160                 add     esp, 4
.text:00401163                 nop
.text:00401164                 nop
.text:00401165                 nop
.text:00401166                 nop
.text:00401167                 nop
.text:00401168                 nop
.text:00401169                 nop
.text:0040116A                 nop
.text:0040116B                 nop
.text:0040116C                 nop
.text:0040116D                 nop
.text:0040116E                 nop
.text:0040116F                 nop
.text:00401170                 call    sub_401080
.text:00401175                 xor     eax, eax
.text:00401177                 pop     edi
.text:00401178                 pop     esi
.text:00401179                 pop     ebx
.text:0040117A                 pop     ebp
.text:0040117B                 retn
.text:0040117B _main           endp
```

看伪C

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

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  sub_401040("Hello, junkcode1!\n");
  sub_401080();
  return 0;
}
```

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

```
unsigned int sub_401080()
{
  unsigned int result; // eax
  int v1; // [esp+8h] [ebp-24h]
  unsigned int i; // [esp+Ch] [ebp-20h]
  char *v3; // [esp+10h] [ebp-1Ch]
  int v5[4]; // [esp+18h] [ebp-14h]

  v5[0] = 49;
  v5[1] = 65;
  v5[2] = 81;
  v5[3] = 97;
  v1 = 0;
  result = sub_401040(&unk_41980C);
  for ( i = 0; ; ++i )
  {
    v3 = byte_41B8B0;
    do
      LOBYTE(result) = *v3;
    while ( *v3++ );
    if ( i >= v3 - &byte_41B8B0[1] )
      break;
    byte_41B8B0[i] ^= LOBYTE(v5[v1++]);
    if ( v1 == 4 )
      v1 = 0;
    result = i + 1;
  }
  return result;
}
```

查看取出数据进行异或运算；

功能程序分析

byte_41B8B0 和 v5进行的异或操作

Get flag

```
flag_0=[0x7c, 0xe, 0x12, 0x32, 0x72, 0x15, 0x17, 0x1a, 0x72, 0x71,
0x3f, 0x6, 0x43, 0x0, 0x25, 0x14, 0x5d, 0x20, 0x25, 0x8, 0x1, 0x2f,
0x22, 0x3e, 0x68, 0x71, 0x24, 0x3e, 0x5a, 0x2f, 0x61, 0x16, 0x6e, 0x2b,
0x24, 0x2f, 0x5a, 0x1e, 0x32, 0x51, 0x55, 0x24, 0x2c]

flag_2=[0x31, 0x41, 0x51, 0x61]
flag=""
cnt = 0
for i in flag_0:
    flag += chr(i^flag_2[cnt])
    cnt = cnt+1
    if cnt==4:
        cnt =0

print(flag)
```

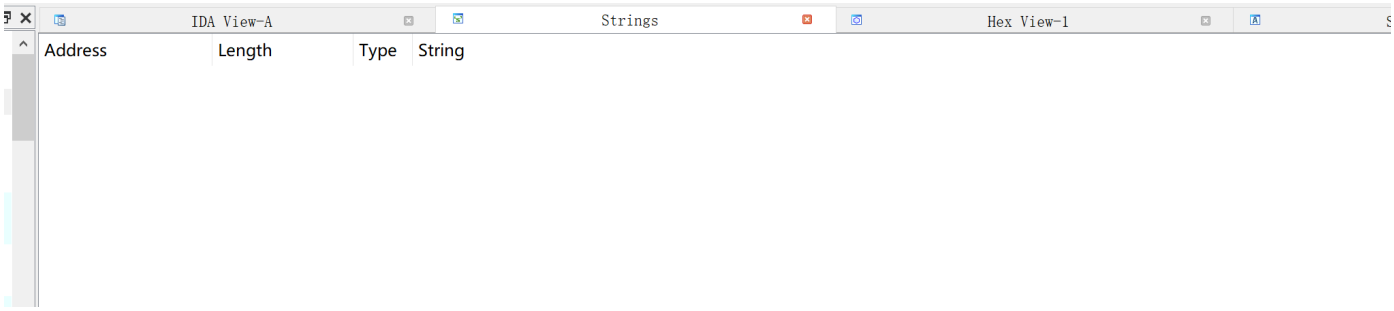
0x03 easy_junkr2.exe

考察点：子程序花指令去除；

程序架构

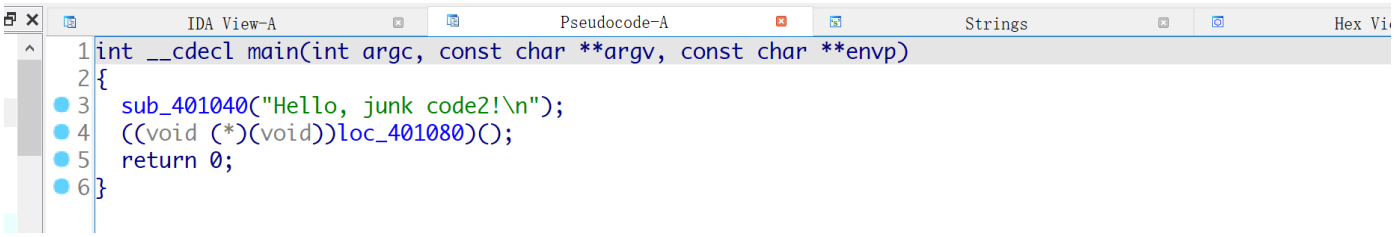
IDA分析

字符串查看



看主程序查看伪C

发现有loc标号；进去看看；



似曾相识的一幕

```
.text:00401080 loc_401080:                             ; CODE XREF: _main+10↓p
.text:00401080                 push    ebp
.text:00401081                 mov     ebp, esp
.text:00401083                 sub     esp, 2Ch
.text:00401086                 mov     eax, ___security_cookie
.text:0040108B                 xor     eax, ebp
.text:0040108D                 mov     [ebp-4], eax
.text:00401090                 push    ebx
.text:00401091                 push    esi
.text:00401092                 push    edi
.text:00401093                 mov     dword ptr [ebp-14h], 31h ; '1'
.text:0040109A                 mov     dword ptr [ebp-10h], 41h ; 'A'
.text:004010A1                 mov     dword ptr [ebp-0Ch], 51h ; 'Q'
.text:004010A8                 mov     dword ptr [ebp-8], 61h ; 'a'
.text:004010AF
.text:004010AF loc_4010AF:                             ; CODE XREF: .text:004010C8↓j
.text:004010AF                 mov     dword ptr [ebp-24h], 0
.text:004010B6                 push    offset unk_41980C
.text:004010BB                 call    sub_401040
.text:004010C0                 add     esp, 4
.text:004010C3                 call    sub_4010CB
.text:004010C8                 jmp     short near ptr loc_4010AF+6
.text:004010C8 ; ---------------------------------------------------------------------------
.text:004010CA                 db 5
.text:004010CB
.text:004010CB ; =============== S U B R O U T I N E =======================================
.text:004010CB
.text:004010CB
.text:004010CB sub_4010CB      proc near               ; CODE XREF: .text:004010C3↑p
.text:004010CB                 add     dword ptr [esp+0], 1
.text:004010CF                 retn
.text:004010CF sub_4010CB      endp
.text:004010CF
.text:004010D0 ; ---------------------------------------------------------------------------
.text:004010D0                 mov     dword ptr [ebp-20h], 0
.text:004010D7                 jmp     short loc_4010E2
.text:004010D9 ; ---------------------------------------------------------------------------
```

去除花指令

nop掉

```
.text:004010C0                 add     esp, 4
.text:004010C3                 call    sub_4010CB
.text:004010C8                 jmp     short near ptr loc_4010AF+6
.text:004010C8 ; ---------------------------------------------------------------------------
.text:004010CA                 db 5
.text:004010CB
.text:004010CB ; =============== S U B R O U T I N E =======================================
.text:004010CB
.text:004010CB
.text:004010CB sub_4010CB      proc near               ; CODE XREF: .text:004010C3↑p
.text:004010CB                 add     dword ptr [esp+0], 1
.text:004010CF                 retn
.text:004010CF sub_4010CB      endp
.text:004010CF
```

```
.text:00401080
.text:00401080                 push    ebp
.text:00401081                 mov     ebp, esp
.text:00401083                 sub     esp, 2Ch
.text:00401086                 mov     eax, ___security_cookie
.text:0040108B                 xor     eax, ebp
.text:0040108D                 mov     [ebp+var_4], eax
.text:00401090                 push    ebx
.text:00401091                 push    esi
.text:00401092                 push    edi
.text:00401093                 mov     [ebp+var_14], 31h ; '1'
.text:0040109A                 mov     [ebp+var_10], 41h ; 'A'
.text:004010A1                 mov     [ebp+var_C], 51h ; 'Q'
.text:004010A8                 mov     [ebp+var_8], 61h ; 'a'
.text:004010AF                 mov     [ebp+var_24], 0
.text:004010B6                 push    offset unk_41980C
.text:004010BB                 call    sub_401040
.text:004010C0                 add     esp, 4
.text:004010C3                 nop
.text:004010C4                 nop
.text:004010C5                 nop
.text:004010C6                 nop
.text:004010C7                 nop
.text:004010C8                 nop
.text:004010C9                 nop
.text:004010CA                 nop
.text:004010CB                 nop
.text:004010CC                 nop
.text:004010CD                 nop
.text:004010CE                 nop
.text:004010CF                 nop
.text:004010D0                 mov     [ebp+var_20], 0
.text:004010D7                 jmp     short loc_4010E2
.text:004010D9 ; ---------------------------------------------------------------------------
.text:004010D9
.text:004010D9 loc_4010D9:                             ; CODE XREF: sub_401080:loc_401145↓j
.text:004010D9                 mov     eax, [ebp+var_20]
.text:004010DC                 add     eax, 1
```

再看伪C

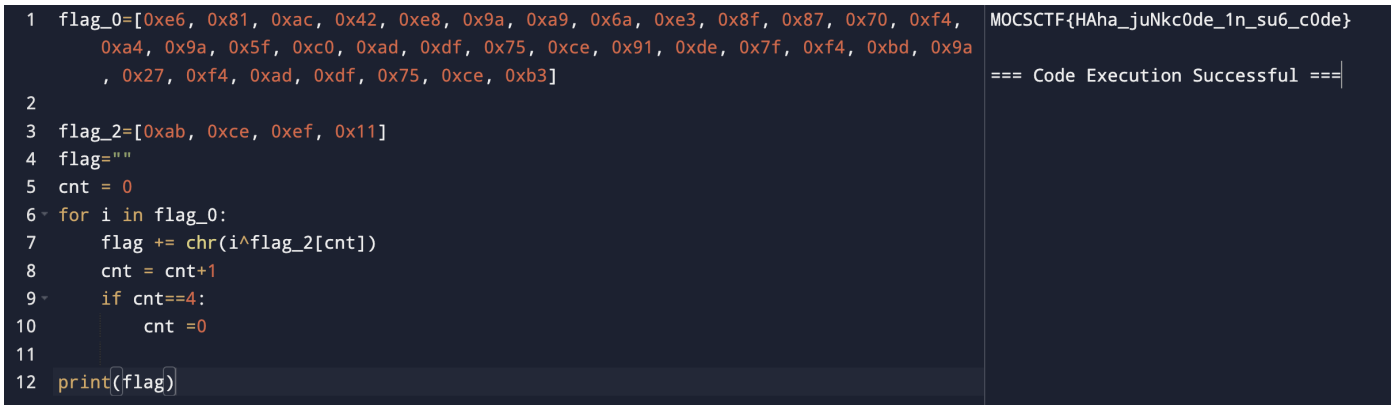
相同的算法和操作；

```
1  unsigned int sub_401080()
2  {
3    unsigned int result; // eax
4    int v1; // [esp+14h] [ebp-24h]
5    unsigned int i; // [esp+18h] [ebp-20h]
6    char *v3; // [esp+1Ch] [ebp-1Ch]
7    int v5[4]; // [esp+24h] [ebp-14h]
8
9    v5[0] = 49;
10   v5[1] = 65;
11   v5[2] = 81;
12   v5[3] = 97;
13   v1 = 0;
14   result = sub_401040(&unk_41980C);
15   for ( i = 0; ; ++i )
16   {
17     v3 = byte_41B8B0;
18     do
19       LOBYTE(result) = *v3;
20     while ( *v3++ );
21     if ( i >= v3 - &byte_41B8B0[1] )
22       break;
23     byte_41B8B0[i] ^= LOBYTE(v5[v1++]);
24     if ( v1 == 4 )
25       v1 = 0;
26     result = i + 1;
27   }
28   return result;
29 }
```

Get flag

取出数据异或运算；

```
flag_0=[0xe6, 0x81, 0xac, 0x42, 0xe8, 0x9a, 0xa9, 0x6a, 0xe3, 0x8f,
0x87, 0x70, 0xf4, 0xa4, 0x9a, 0x5f, 0xc0, 0xad, 0xdf, 0x75, 0xce, 0x91,
0xde, 0x7f, 0xf4, 0xbd, 0x9a, 0x27, 0xf4, 0xad, 0xdf, 0x75, 0xce, 0xb3]

flag_2=[0xab, 0xce, 0xef, 0x11]
flag=""
cnt = 0
for i in flag_0:
    flag += chr(i^flag_2[cnt])
    cnt = cnt+1
    if cnt==4:
        cnt =0

print(flag)
```



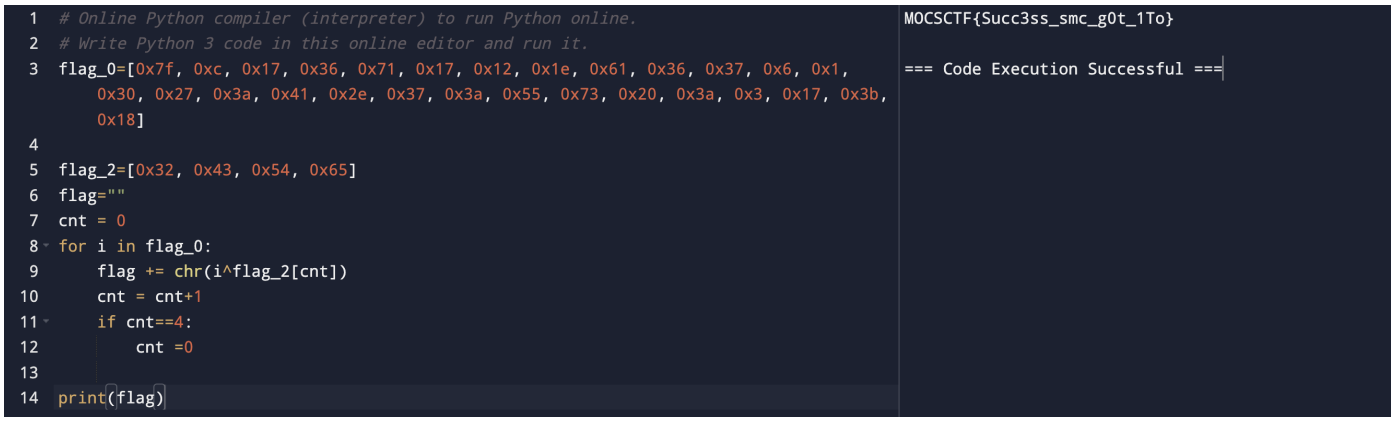
0x04 easy_rev

考察点：自定义函数、花指令去除

与上类似，加了不同花指令；取出后算法相同；

Get flag

```
flag_0=[0x7f, 0xc, 0x17, 0x36, 0x71, 0x17, 0x12, 0x1e, 0x61, 0x36,
0x37, 0x6, 0x1, 0x30, 0x27, 0x3a, 0x41, 0x2e, 0x37, 0x3a, 0x55, 0x73,
0x20, 0x3a, 0x3, 0x17, 0x3b, 0x18]

flag_2=[0x32, 0x43, 0x54, 0x65]
flag=""
cnt = 0
for i in flag_0:
    flag += chr(i^flag_2[cnt])
    cnt = cnt+1
    if cnt==4:
        cnt =0

print(flag)
```

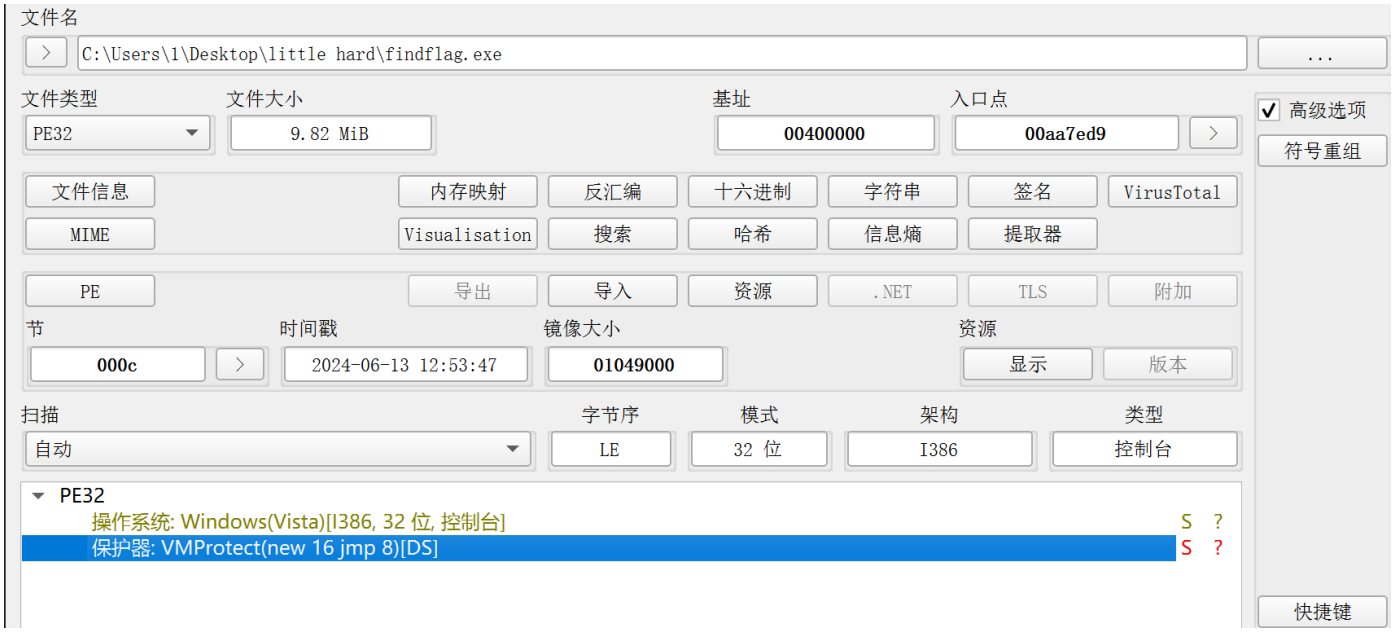


0x05 Little hard

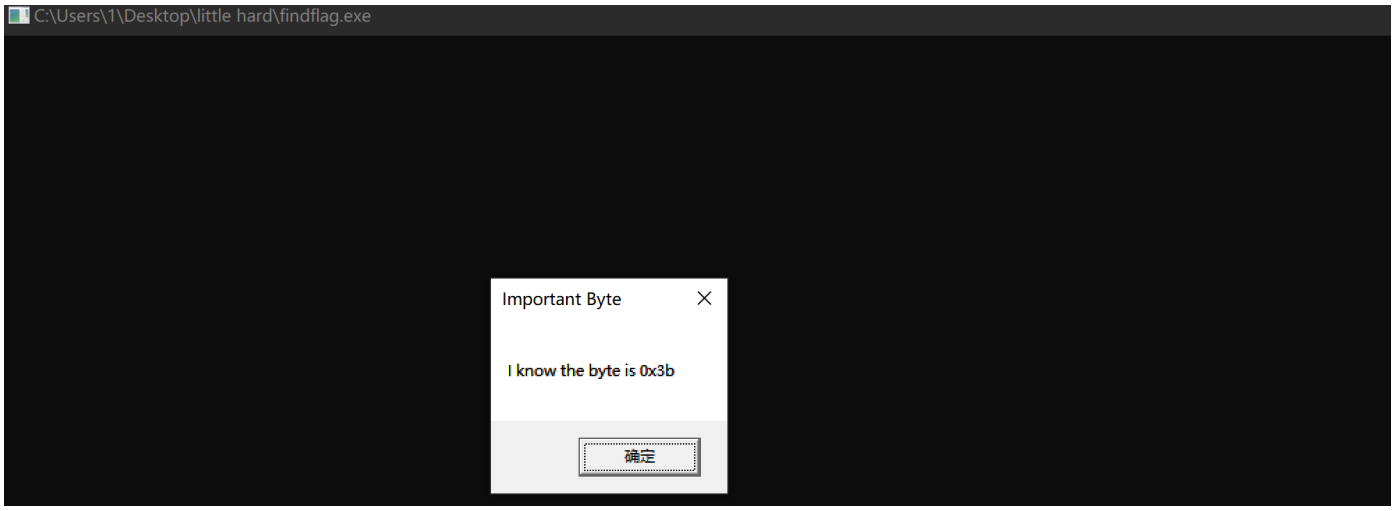
原题思路来自xx木马，实际findflag.exe是加载器，findflag.exe通过内存加载find.inc （exe文件，可以直接改后缀运行它）；而find.inc 通过手工加载dll文件执行其导出函数；findflag.exe做了VMP加壳，其实这个是误导；关键点在dll文件中的resource节；这也是恶意木马常用的隐藏payload的手段；通过运行findflag.exe 拿到加密key：0x3b；通过单字节（除0x00外）解密resource数据，直接命令行运行获取flag；

详细分析过程

查看文件加了壳；

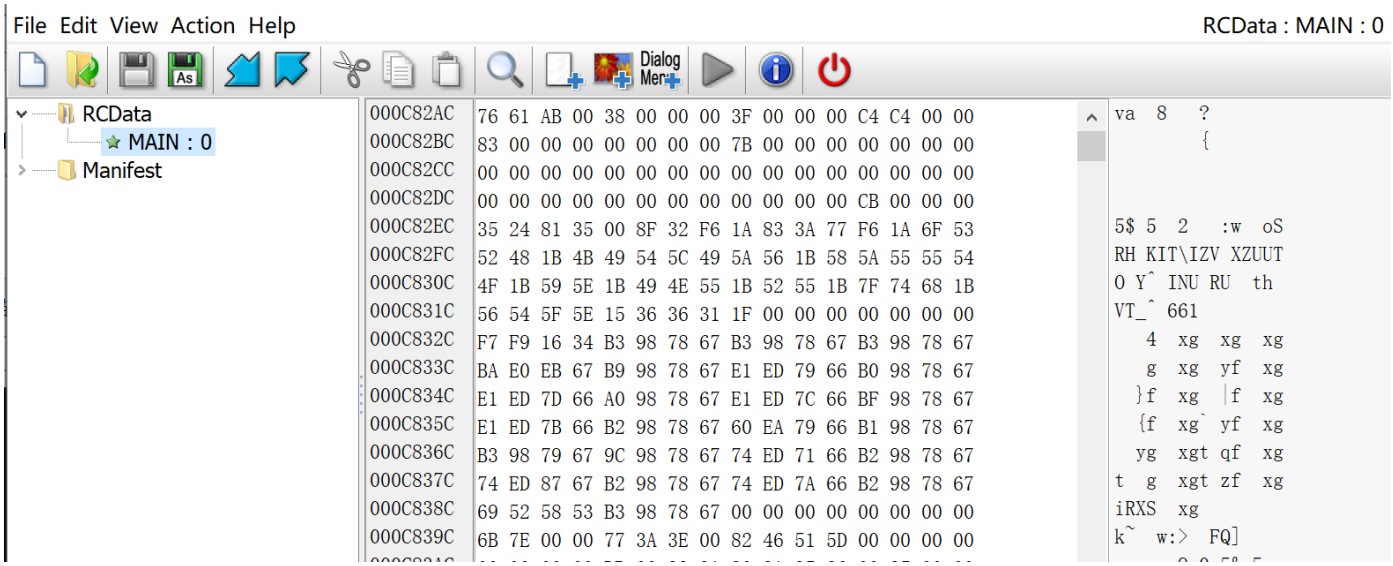


运行提示重要字节是 0x3b；



IDA没发现有用线索；

查看dll文件；resourceheader看下；发现疑似PE文件头被加密；



dump下来；运行脚本解密；

Get flag

```python
def modify_exe(input_file, output_file):
    with open(input_file, 'rb') as f:
        content = bytearray(f.read())

    for i in range(len(content)):
        if content[i] != 0x3b and content[i] != 0x00:
            content[i] ^= 0x3b

    with open(output_file, 'wb') as f:
        f.write(content)

if __name__ == "__main__":
    input_file = "MAIN.exe"
    output_file = "output.exe"
    modify_exe(input_file, output_file)
    print("Modification complete.")

```

前面提示重要字节0x3b；显然是单字节xor，这里直接异或得不到结果，观察异或后数据，发现需要剔除0x00字节；

```
C:\Users\1\Desktop\little hard>python a.py
Modification complete.

C:\Users\1\Desktop\little hard>"C:\Users\1\Desktop\little hard\output.exe"
MOCSCTF{Haha_Y0u_f1nd_th3_flAg}
```

或拖入IDA看下；

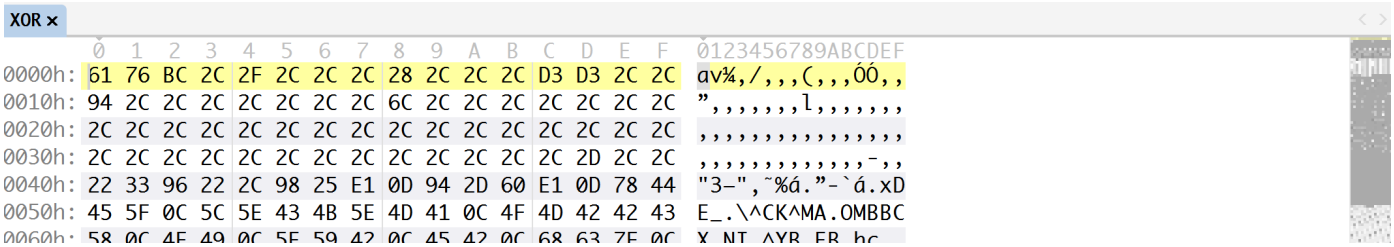
```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char ArgList[31]; // [esp+0h] [ebp-24h] BYREF

  qmemcpy(ArgList, "MOCSCTF{Haha_Y0u_f1nd_th3_flAg}", sizeof(ArgList));
  sub_401010("%s", (char)ArgList);
  return 0;
}
```

```
MOCSCTF{Haha_Y0u_f1nd_th3_flAg}
```

0x06 XOR

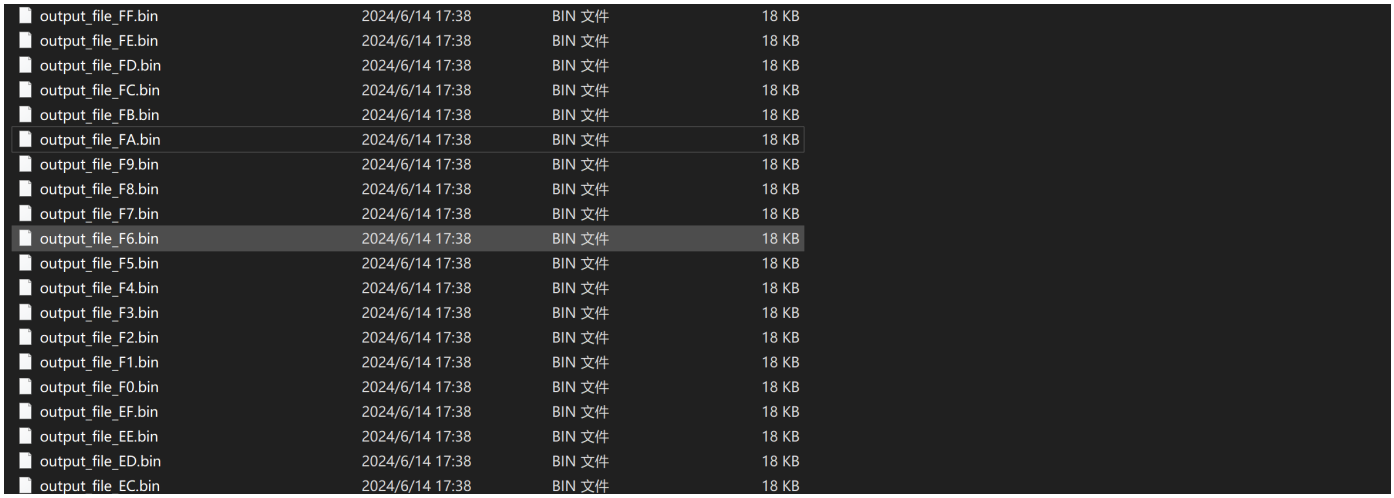
考察PE识别，题目提示了XOR，但是没提示XOR哪个字节；给的是一堆HEX数据，想到XOR后可能是个二进制文件；

将文本文件复制，粘贴进010editor中；




```python
import subprocess

def xor_file_and_extract_strings(filename, output_dir, strings_output_file):
    all_strings = []

    with open(filename, 'rb') as f:
        content = f.read()

    for xor_value in range(1, 256):  # 0x01 to 0xff
        xored_content = bytes([byte ^ xor_value for byte in content])
        output_file = f"output_file_{xor_value:02X}.bin"  # Save with the XOR value in the filename
        with open(output_file, 'wb') as f_out:
            f_out.write(xored_content)


# Example usage:
input_filename = "main.bin"
output_dir = ""
strings_output_file = "all_strings.txt"
xor_file_and_extract_strings(input_filename, output_dir, strings_output_file)

```

执行下生成255个文件



简单方法打包上传威胁情报看下；

找到恶意文件；

或者在linux下直接 file * 命令看下类型；找到目标文件分析；

```
output_file_2B.bin: data
output_file_2C.bin: PE32 executable (console) Intel 80386, for MS Windows
output_file_2D.bin: data
output_file_2E.bin: data
```

此题目实际为PE加载执行shellcode，还原exe后可以直接执行（不可信文件在vm中运行）；也可以还原下shellcode，拖入IDA；

```
int sub_0()
{
  __int64 v0; // rax
  int v1; // eax
  int result; // eax
  int (__stdcall *v4)(_DWORD, char *, _DWORD, _DWORD); // [esp+10h] [ebp-8Ch]
  char *v5; // [esp+1Ch] [ebp-80h]
  char *v6; // [esp+20h] [ebp-ACh]
  int v7; // [esp+24h] [ebp-A8h]
  char v8[40]; // [esp+28h] [ebp-A4h] BYREF
  char v9[16]; // [esp+50h] [ebp-7Ch] BYREF
  char v10[16]; // [esp+60h] [ebp-6Ch] BYREF
  char v11[12]; // [esp+70h] [ebp-5Ch] BYREF
  char v12[12]; // [esp+7Ch] [ebp-50h] BYREF
  __int64 v13; // [esp+88h] [ebp-44h]
  __int64 i; // [esp+90h] [ebp-3Ch]
  int (__stdcall *v15)(int); // [esp+98h] [ebp-34h]
  int (__stdcall *v16)(char *, char *); // [esp+9Ch] [ebp-30h]
  unsigned __int16 *v17; // [esp+A0h] [ebp-2Ch]
  _DWORD *v18; // [esp+A4h] [ebp-28h]
  __int64 v19; // [esp+A8h] [ebp-24h]
  _DWORD *v20; // [esp+B0h] [ebp-1Ch]
  int v21; // [esp+B4h] [ebp-18h]
  __int16 v22; // [esp+B8h] [ebp-14h]
  _DWORD *v23; // [esp+BCh] [ebp-10h]
  __int64 v24; // [esp+C0h] [ebp-Ch]
  _BYTE *v25; // [esp+C8h] [ebp-4h]

  v24 = 0i64;
  for ( i = *(int *)(*(_DWORD *)(__readfsdword(0x30u) + 0xC) + 0x14); i; i = *(unsigned int *)i )
  {
    v13 = *(int *)(i + 0x28);
    v22 = *(_WORD *)(i + 0x24);
    v19 = 0i64;
    do
    {
      v19 = (unsigned int)__ROR4__(v19, 0xD);
      if ( *(unsigned __int8 *)v13 < 0x61u )
        v0 = *(unsigned __int8 *)v13;
      else
  }
  if ( !v16 || !v15 )
    __debugbreak();
  strcpy(v12, "USER32.dll");
  strcpy(v11, "MessageBoxA");
  strcpy(v8, "MOCSCTF{Brut3Forc3_th3_flAg_Us3_X0r}");
  v1 = v16(v12, v11);
  v4 = (int (__stdcall *)(_DWORD, char *, _DWORD, _DWORD))v15(v1);
  if ( !v4 )
    __debugbreak();
  result = v4(0, v8, 0, 0);
  __debugbreak();
  return result;
}
```

```
MOCSCTF{Brut3Forc3_th3_flAg_Us3_X0r}
```

更为简洁的方法，看到文件存在大量2C，直接异或2C即可；这就是为什么一般的恶意软件使用异或算法时会将自身和0x00避过的原因；

XOR ×

```
          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
0000h: 61 76 BC 2C 2F 2C 2C 2C 28 2C 2C 2C D3 D3 2C 2C av¼,/,,,(,,,ÓÓ,,
0010h: 94 2C 2C 2C 2C 2C 2C 2C 6C 2C 2C 2C 2C 2C 2C 2C ”,,,,,,,l,,,,,,,
0020h: 2C 2C 2C 2C 2C 2C 2C 2C 2C 2C 2C 2C 2C 2C 2C 2C ,,,,,,,,,,,,,,,,
0030h: 2C 2C 2C 2C 2C 2C 2C 2C 2C 2C 2C 2C 2C 2D 2C 2C ,,,,,,,,,,,,,-,,
0040h: 22 33 96 22 2C 98 25 E1 0D 94 2D 60 E1 0D 78 44 "3–",˜%á.”-`á.xD
0050h: 45 5F 0C 5C 5E 43 4B 5E 4D 41 0C 4F 4D 42 42 43 E_.\^CK^MA.OMBBC
0060h: 58 0C 4E 49 0C 5E 59 42 0C 45 42 0C 68 63 7E 0C X.NI.^YB.EB.hc~.
```