四叶草交流友谊赛

原创

Gh0st' 于 2020-05-08 10:24:44 发布 363 收藏

分类专栏： re学习笔记

本文链接：https://blog.csdn.net/qq_44115034/article/details/105759651

re学习笔记 专栏收录该内容

4 篇文章 1 订阅

订阅专栏

迷路的菜鸟，CTF从入门到放弃。

---

easyYM

运行程序。

```
Please input flag:123
flag:qwe123987
flag error!!!
请按任意键继续. . .
```

直接输出了flag?

其实并没有，题目描述也说了这不是真正的flag，真正的flag格式是flag{}。

拖进IDA，

```
int __cdecl main_0()
{
  char v1; // [esp+4Ch] [ebp-20h]

  printf("Please input flag:");
  scanf("%s", &v1);
  printf("flag:qwe123987\n");
  if ( !strcmp(&v1, "qwe123987") )
    printf("qwe123987 is not a flag!!!\n");
  else
    printf("flag error!!!\n");
  system("pause");
  return 0;
}
```


好像没什么有用信息。（太菜了，在这里还绕了好久）
看看题目描述，关键函数并不一定执行到了。

看看汇编代码。

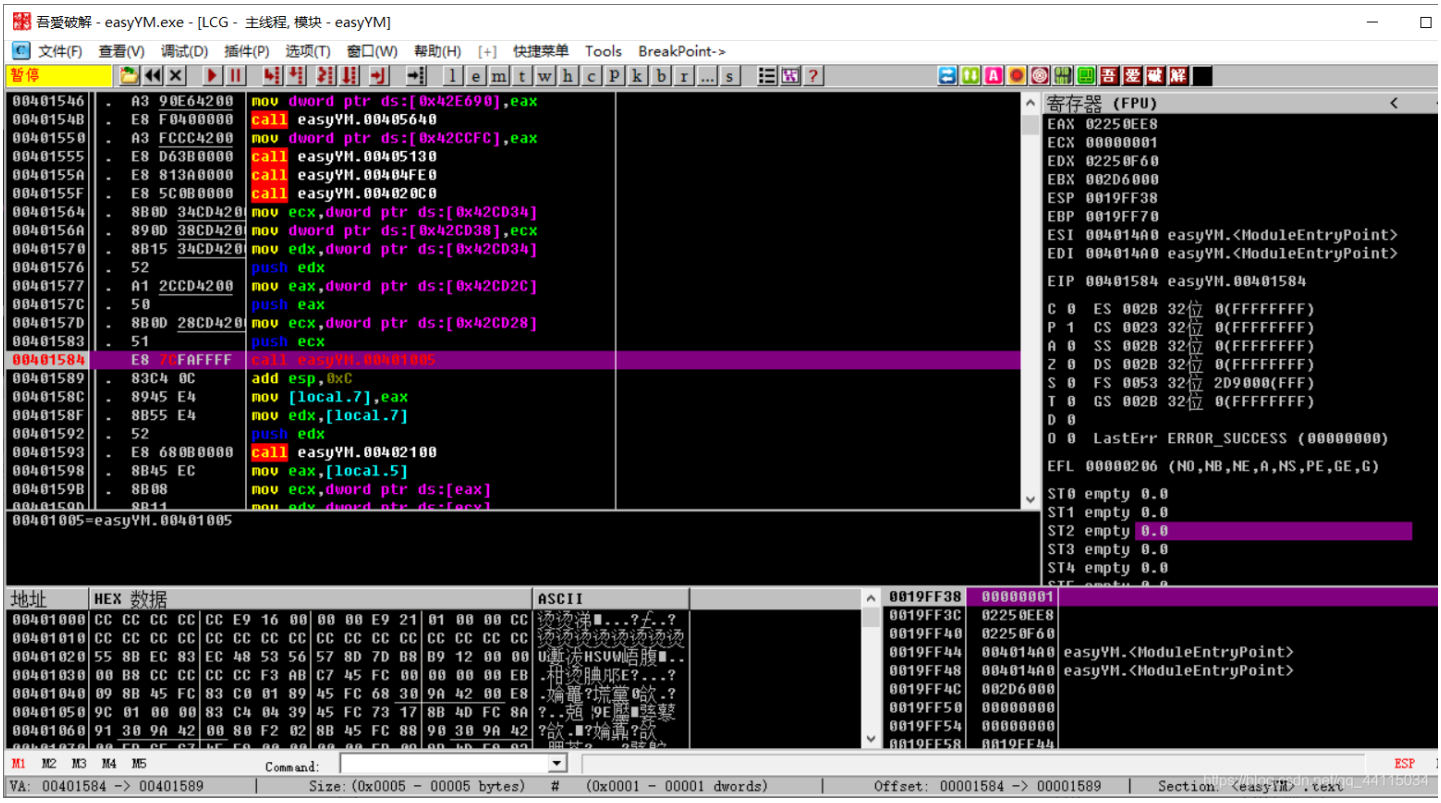
```
.text:00401000                 assume cs:_text
.text:00401000                 ;org 401000h
.text:00401000                 assume es:nothing, ss:nothing, ds:_data, fs:nothing, gs:nothing
.text:00401000                 db 5 dup(0CCh)
.text:00401005 ; ---------------------------------------------------------------------------
.text:00401005                 jmp     loc_401020
.text:0040100A
.text:0040100A ; =============== S U B R O U T I N E =======================================
.text:0040100A
.text:0040100A ; Attributes: thunk
.text:0040100A
.text:0040100A ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:0040100A _main           proc near               ; CODE XREF: start+E4↓p
.text:0040100A                 jmp     _main_0
.text:0040100A _main           endp
.text:0040100A
.text:0040100A ; ---------------------------------------------------------------------------
.text:0040100F                 align 20h
.text:00401020
.text:00401020 loc_401020:                             ; CODE XREF: .text:00401005↑j
.text:00401020                 push    ebp
.text:00401021                 mov     ebp, esp
.text:00401023                 sub     esp, 48h
.text:00401026                 push    ebx
.text:00401027                 push    esi
.text:00401028                 push    edi
.text:00401029                 lea     edi, [ebp-48h]
.text:0040102C                 mov     ecx, 12h
.text:00401031                 mov     eax, 0CCCCCCCCh
.text:00401036                 rep stosd
.text:00401038                 mov     dword ptr [ebp-4], 0
.text:0040103F                 jmp     short loc_40104A
.text:00401041 ; ---------------------------------------------------------------------------
.text:00401041
.text:00401041 loc_401041:                             ; CODE XREF: .text:00401071↓j
.text:00401041                 mov     eax, [ebp-4]
```


好吧，调用主函数之前还执行了下面这段代码，这里应该就是关键代码。在下面其实也发现了可疑字符串，更加明确了这点。

```
.text:00401044                 add     eax, 1
.text:00401047                 mov     [ebp-4], eax
.text:0040104A
.text:0040104A loc_40104A:                             ; CODE XREF: .text:0040103F↑j
.text:0040104A                 push    offset aXozjx1vhz2uzc0 ; "XozjX1vHZ2uzc0TdS{qpZ3@{OU3nal7)dl2?"
.text:0040104F                 call    _strlen
.text:00401054                 add     esp, 4
.text:00401057                 cmp     [ebp-4], eax
.text:0040105A                 jnb     short loc_401073
```

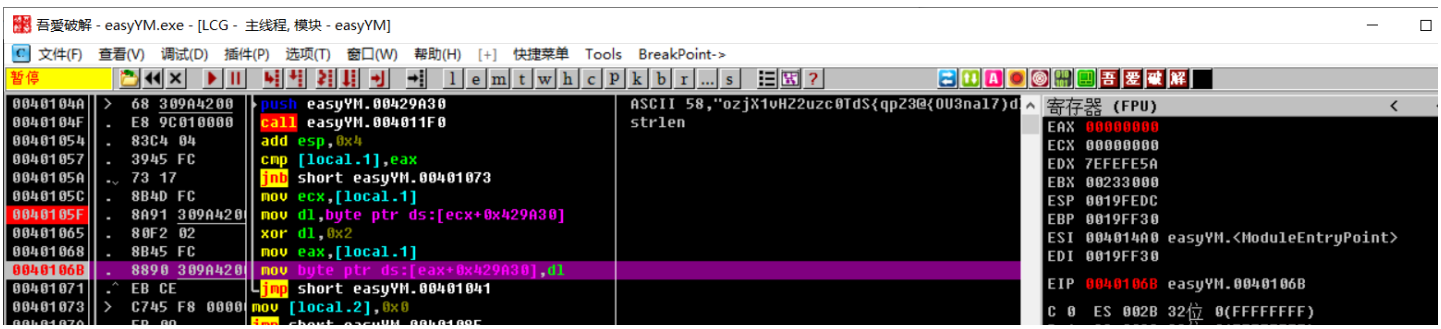
```
.text:0040105C                 mov     ecx, [ebp-4]
.text:0040105F                 mov     dl, byte ptr aXozjx1vhz2uzc0[ecx] ; "XozjX1vHZ2uzc0TdS{qpZ3@{OU3nal7)dl2?"
.text:00401065                 xor     dl, 2
.text:00401068                 mov     eax, [ebp-4]
.text:0040106B                 mov     byte ptr aXozjx1vhz2uzc0[eax], dl ; "XozjX1vHZ2uzc0TdS{qpZ3@{OU3nal7)dl2?"
.text:00401071                 jmp     short loc_401041
.text:00401073 ; ---------------------------------------------------------------------------
.text:00401073
.text:00401073 loc_401073:                             ; CODE XREF: .text:0040105A↑j
.text:00401073                 mov     dword ptr [ebp-8], 0
.text:0040107A                 jmp     short loc_401085
.text:0040107C ; ---------------------------------------------------------------------------
.text:0040107C
.text:0040107C loc_40107C:                             ; CODE XREF: .text:004010AB↓j
.text:0040107C                 mov     ecx, [ebp-8]
.text:0040107F                 add     ecx, 1
.text:00401082                 mov     [ebp-8], ecx
.text:00401085
.text:00401085 loc_401085:                             ; CODE XREF: .text:0040107A↑j
.text:00401085                 push    offset aRngcqgWqgCqg46 ; "Rngcqg]wqg]@cqg46]vm]fgap{rv]vm]egv]vjg"...
.text:0040108A                 call    _strlen
.text:0040108F                 add     esp, 4
.text:00401092                 cmp     [ebp-8], eax
.text:00401095                 jnb     short loc_4010AD
.text:00401097                 mov     edx, [ebp-8]
.text:0040109A                 mov     al, byte ptr aRngcqgWqgCqg46[edx] ; "Rngcqg]wqg]@cqg46]vm]fgap{rv]vm]egv]vjg"...
.text:004010A0                 xor     al, 2
```

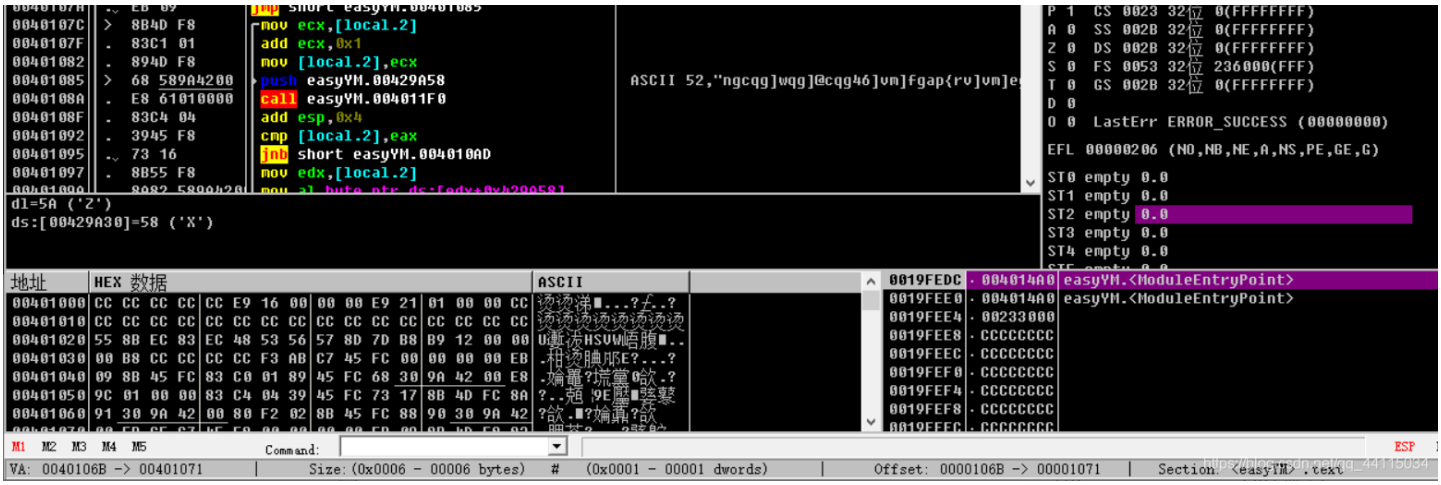

拖进OD调试，



我们在这里调用主函数的地方设置断点，修改一下汇编代码，调用之前的关键代码。

然后跟进，

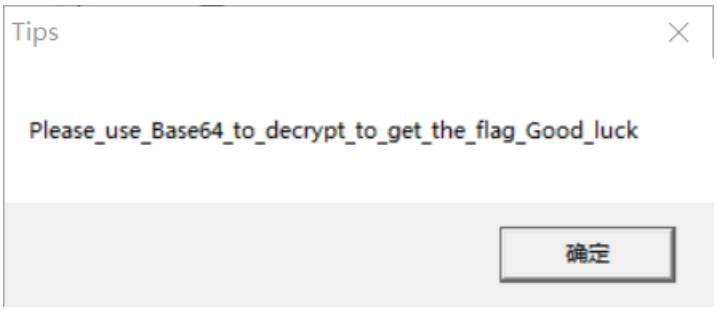




随后发现在此处每次取出字符串中的一个字符与2异或，然后替换原来的字符串。设置断点跟踪查看即可得到替换后的字符串。

```
o - m
z - x
h - j
X - Z
1 - 3
v - t
H - J
Z - H
2 - 0
u - w
z - x
c - a
0 - 2
T - V
d - f
S - Q
{ - y
q - s
p - r
Z - X
3 - 1
@ - B
{ - y
O - M
U - W
3 - 1
n - l
a - c
l - n
7 - 5
) - +
d - f
l - n
2 - 0
? - =
```

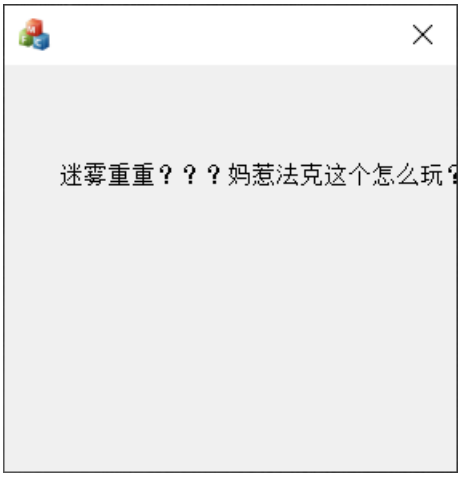
base64? 这里调试完也给出提示了。



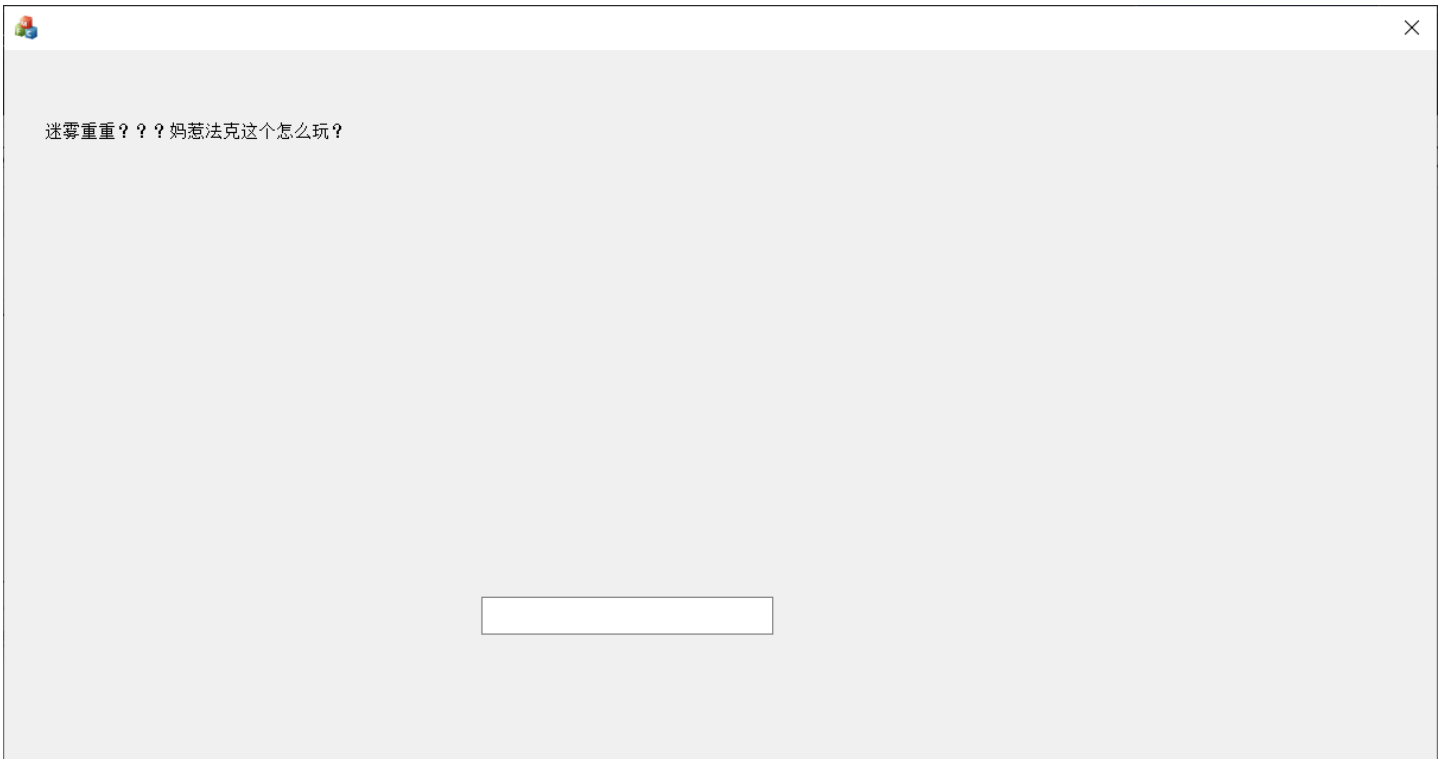
> flag{I_L1ke_C++_Pr1mer~~~}

mwcc

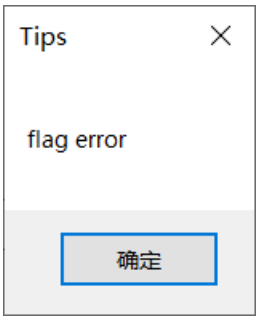
运行程序。



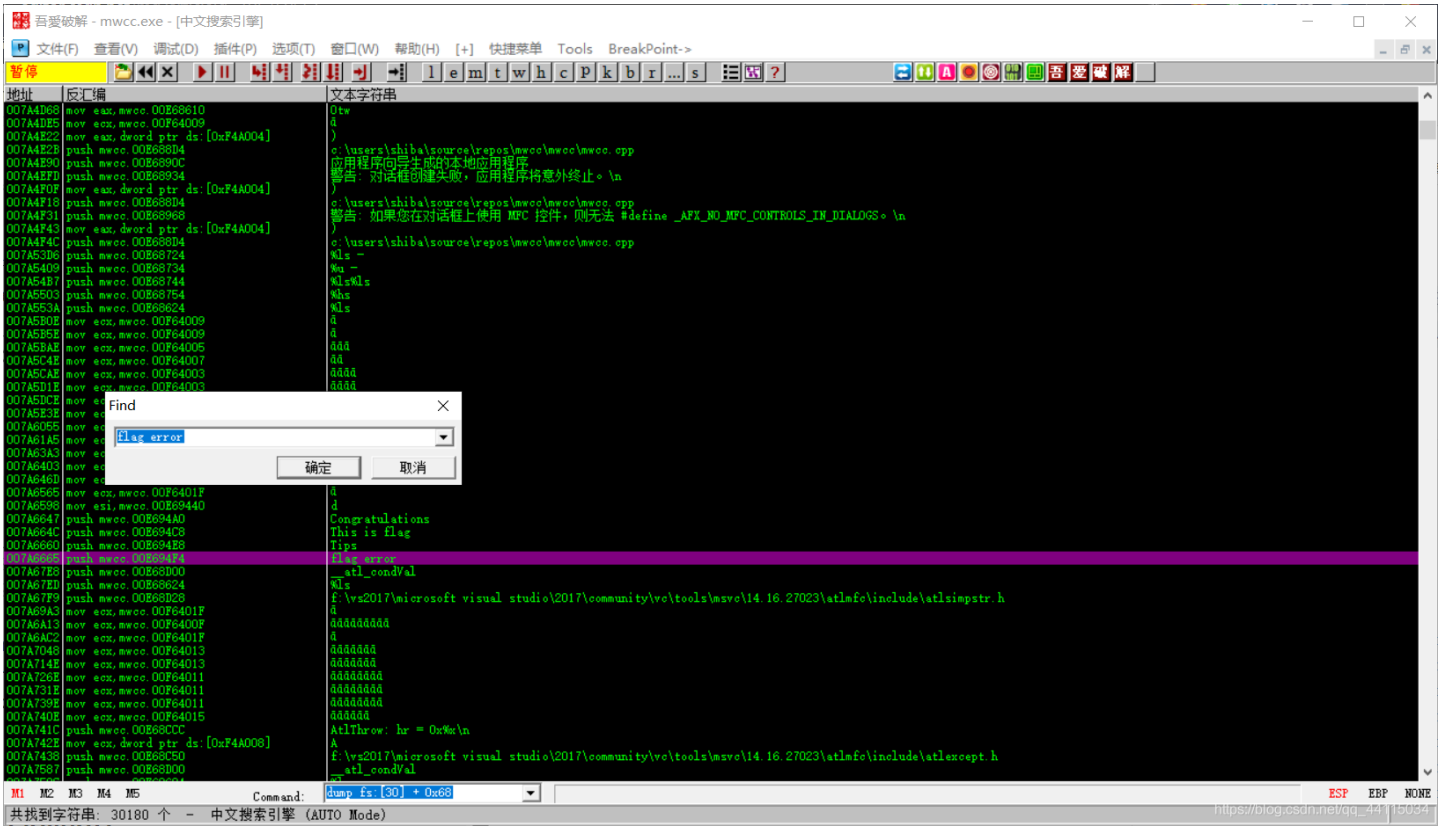
好像没什么有用信息，
但是细心将程序边框扩展开的话。



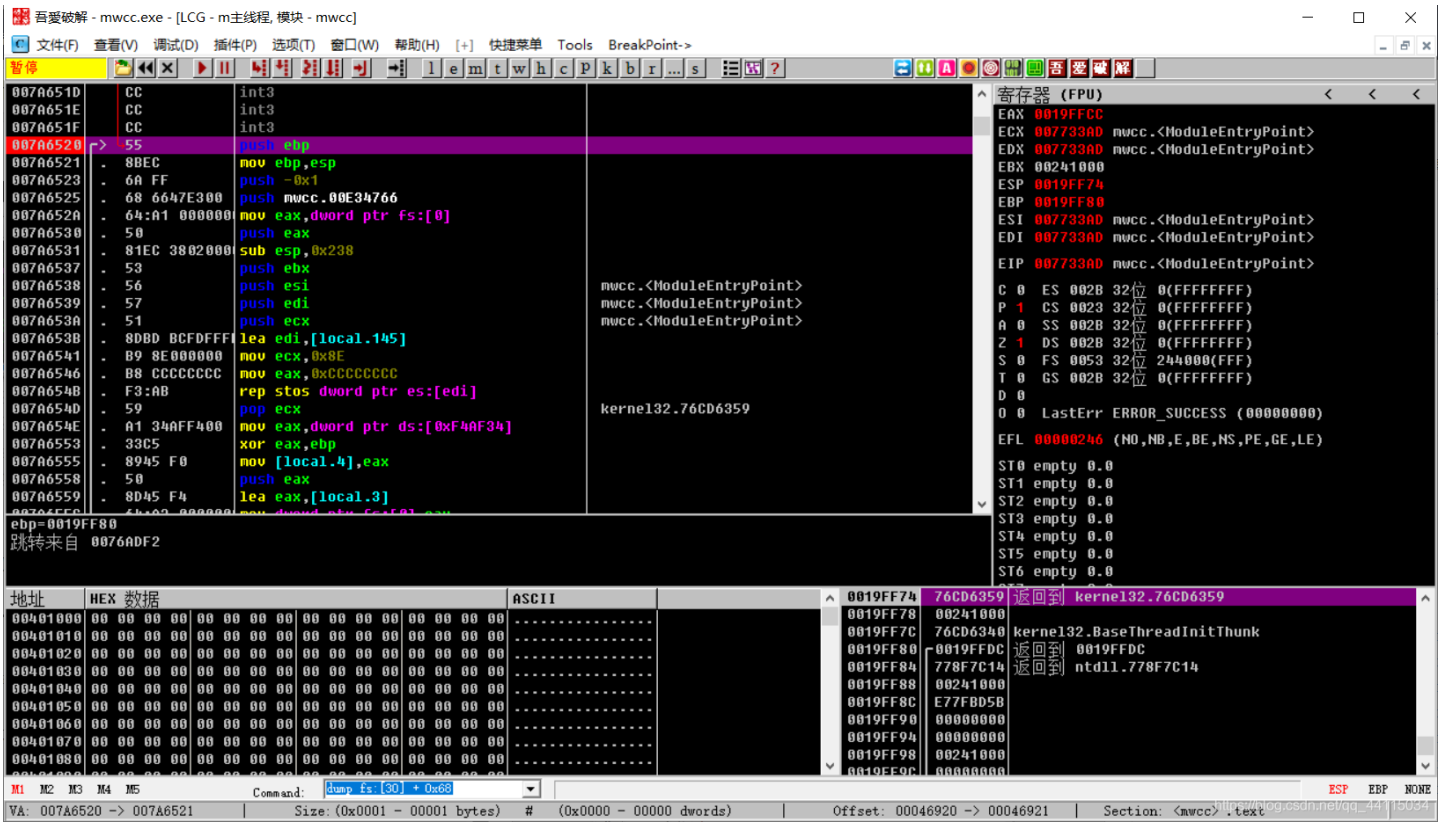
我们就发现可以进行输入了，也有对应的输出。
我们随便输入一个flag，获取相关信息。



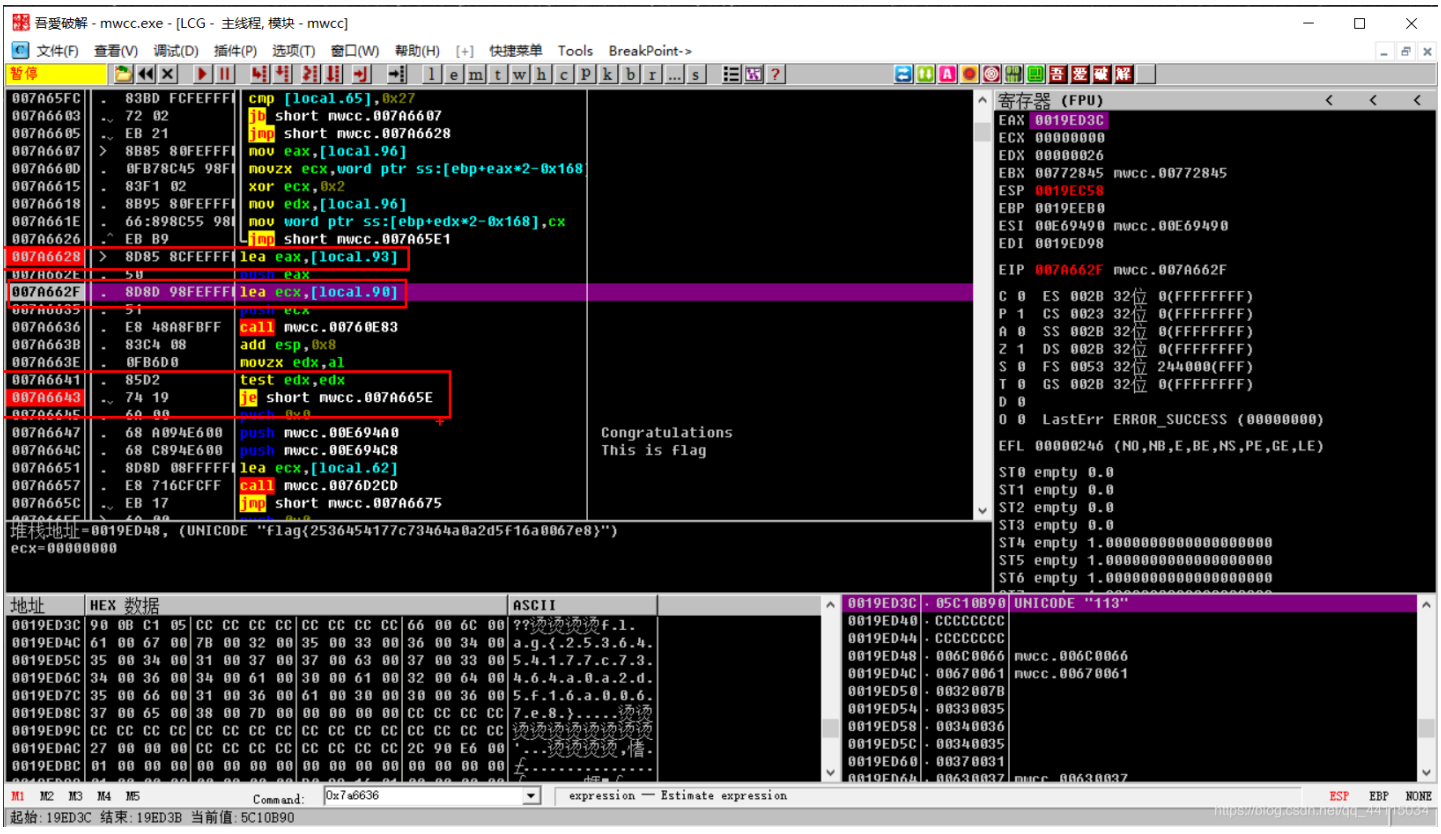
弹窗，Tips,flag error。
接下来静态，动态分析即可。
拖进IDA没有发现什么有用信息，
尝试动态分析。



跟进，并找到该函数的开始。



单步调试，在下图所示发现关键代码。



第一处将我们的输入压入堆栈，

第二处将类似flag的字符串压入堆栈。

第三处进行检验，

若输入正确，上面call的函数会将eax置1，输出this is flag。

若输出错误，上面call的函数会将eax置0，输出flag error。

故

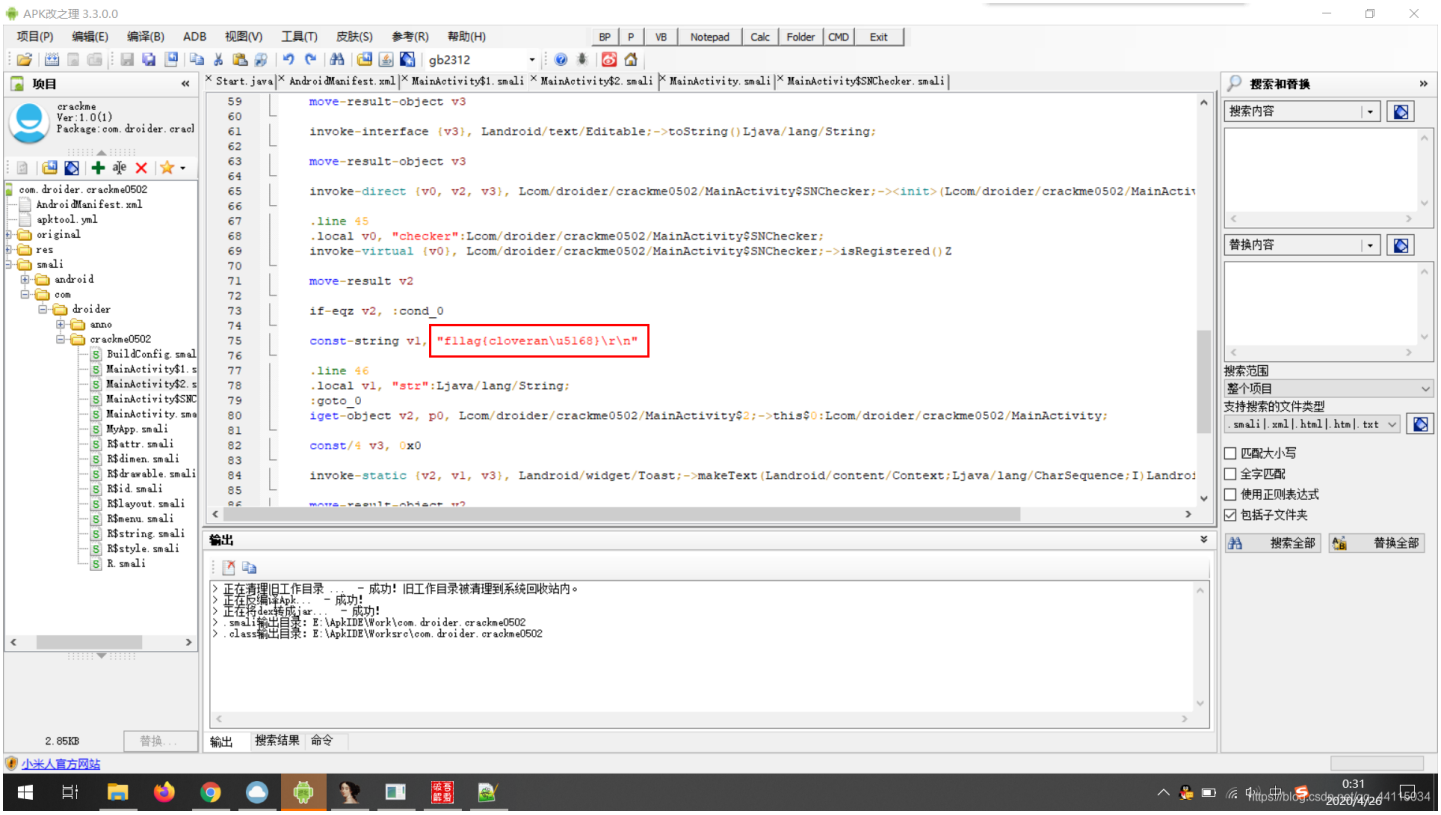
> flag{2536454177c73464a0a2d5f16a0067e8}

CrackMe_clover01

> 这几道是Android逆向入门题，但作者之前没接触Android逆向。(菜醒。。)

大体思路其实都差不多，也是先查壳，
并没有加壳。

使用APKIDE或AndriodKiller，
查看MainActivity，直接找到flag



> flag{cloveran}

CrackMe_clover02

运行程序，
随意输入一个用户名和注册码。

查找关键字

直接搜索运行程序时反馈的字符串
flag就写在下面。

```
<string name="unsuccessed">无效用户名或注册码</string>
<string name="successed">恭喜您！注册成功 F--21（） a--g {CloverSec-android reverse}</string>
```

当然，查看smali代码修改关键的跳转指令也可。

> flag{CloverSec-android reverse}