171203 逆向-JarvisOJ（软件密码破解-3）（2）

原创

奈沙夜影 于 2017-12-03 23:36:24 发布 444 收藏

分类专栏： CTF

本文链接：https://blog.csdn.net/whklhhhh/article/details/78705479

1625-5 王子昂 总结《2017年12月3日》【连续第429天总结】

A. JarvisOJ-Re-软件密码破解-3（2）

B.

看来短时间还搞定不了这题了……

上回说到题目中最后计算了一个区域的值的长度，准备研究它的来源

通过查找交叉引用发现了这个地方：

```
13  char v12; // bl@20
14  unsigned int v13; // esi@20
15  int v14; // [sp+4h] [bp-8h]@12
16
17  v1 = input;
18  if ( strlen(input) != 16 )
19    JUMPOUT(*(_DWORD *)&byte_4017DE);
20  v2 = 0;
21  do
22  {
23    v3 = *v1;
24    if ( *v1 <= 0x29 )
25    {
26      v6 = __OFSUB__(v3, 0x40);
27      v4 = v3 == 0x40;
28      v5 = (char)(v3 - 0x40) < 0;
29    }
30    else
31    {
32      v6 = __OFSUB__(v3, '@');
33      v4 = v3 == '@';
34      v5 = (char)(v3 - '@') < 0;
35      if ( v3 < '@' )
36      {
37        v7 = v3 - '0';
38        goto LABEL_11;
39      }
40    }
41    if ( (unsigned __int8)(v5 ^ v6) | v4 || v3 >= 'G' )
42      return 0;
43    v7 = v3 - '7';
44LABEL_11:
45    v1[byte_571458 - input] = v7;               // 转成值写入
46    ++v2;
47    ++v1;
48  }
49  while ( v2 < 16 );
50  v8 = 0;
51  v9 = 1;
52  v14 = 3 - (_DWORD)byte_571459;
53  do                                            // 合成十六进制数
54  {
55    if ( v8 & 1 )
56      byte_571458[v8 >> 1] += byte_571458[v8];
57    else
```


OD跟踪也可以发现确实是这里在负责写入

里面有一些机制来判断数字范围，例如必须为十六进制大写数字，否则将直接return结束该函数

利用这个机制倒是可以写出长度为8的字符串，例如123456780xxxxxxx
但是此时确定按钮又不可用了，原因出在最后

```
    return strlen(byte_571458) != 8
```

而这个函数的返回值被用在有效性的设置上

```
        v5 = sub_4017C0(MultiByteStr) != 0;
        v6 = CWnd::GetDlgItem(v1, 1);
        CWnd::EnableWindow(v6, v5);
```

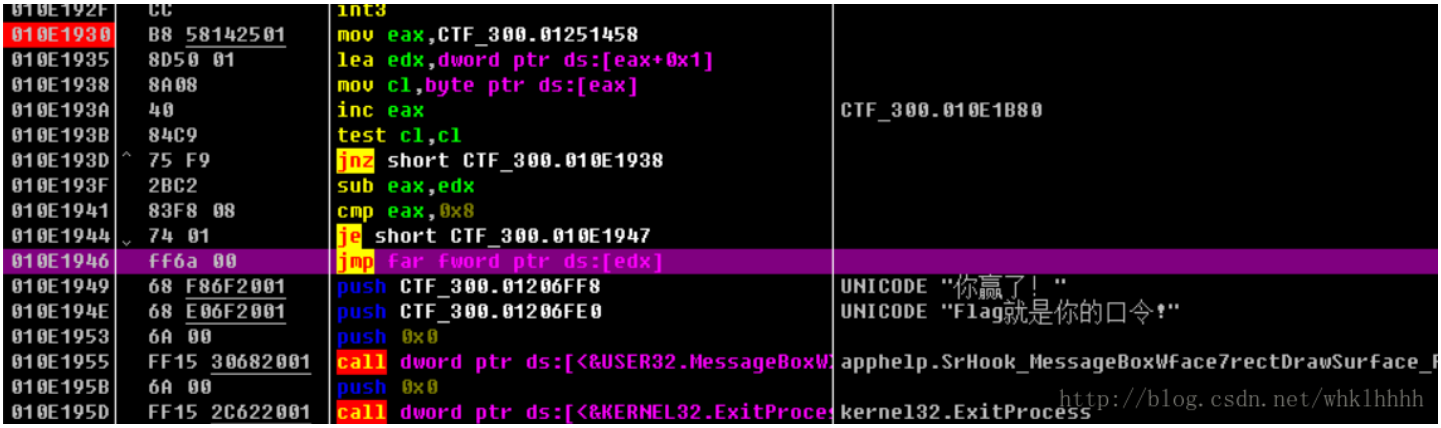
因此这就陷入了一个矛盾的境地–“确定”按钮可用需要长度非8，成功提示则需要长度为8
len和test cl, cl的原理是相同的，因此没有漏洞可以利用

从另一方面来考虑，由于FLAG的唯一性，所以自由组合的flag肯定是不可能的，这里必然不是真正的解题点

找了一圈也没发现，于是去找WriteUp
http://www.mottoin.com/90073.html
这里给出了一些提示，但是太过简略所以还是挺困难的

首先存在很多反调，这个我发现了，但是由于没什么影响所以没太在意
另一方面关键函数是sub_401970，而我虽然在交叉引用中看到这个函数对关键区域有读写，但是却根本没有注意它

学着对其下断，整个运行过程中也没有调用
正当我自暴自弃地继续运行的时候，发现当程序走向错误路径时

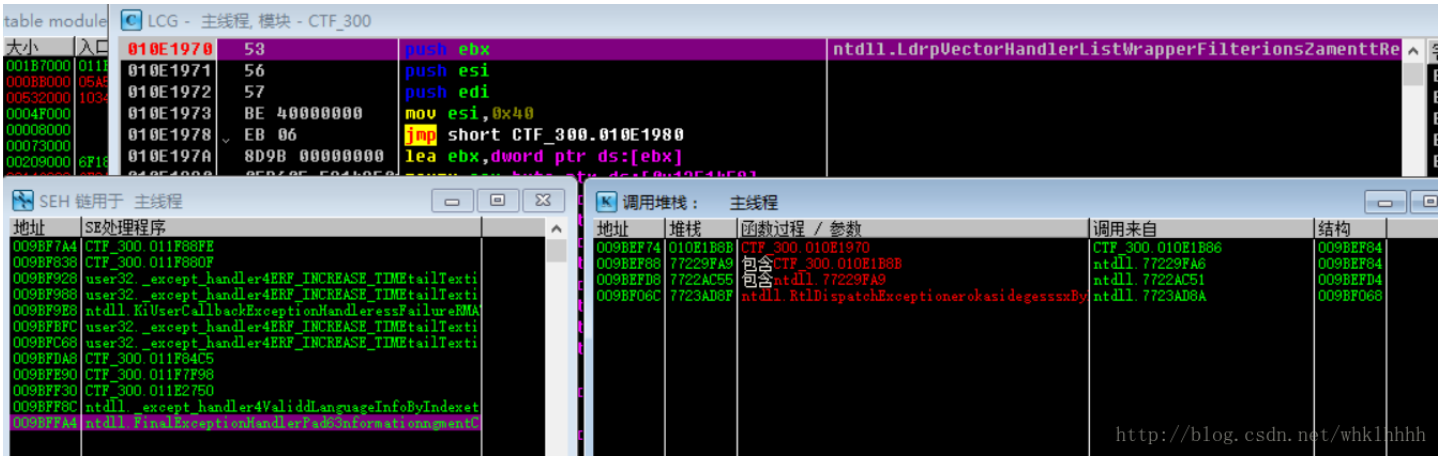


这里[edx]的内容是输入的第2、3个字符，所以jmp必然会引起错误

而之前运行的时候从来没有异常结束的情况

于是运行，发现程序在sub_401970的地方断下了！
说明这是个SEH（异常处理结构）

于是在OD中查看SEH链



果然其中有一些函数插入的程序，这点是我大意没有注意到

但是sub_401970的调用堆栈中也没有看到与SEH链中相重合的内容……

虽然可以确定sub_401970的调用方式肯定是SEH，但是还没搞明白是谁在调用，怎么调用的，下一步研究这一点
另一方面sub_401970对关键区域的读写还需要进一步分析，从而得到flag

C. 明日计划
软件密码破解-3（3）