170913 逆向-问鼎杯题库（找flag）

原创

奈沙夜影 于 2017-09-14 00:10:22 发布 4106 收藏 4

分类专栏： CTF 单片机

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。

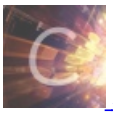
本文链接：https://blog.csdn.net/whklhhhh/article/details/77972746

版权

CTF 同时被 2 个专栏收录

163 篇文章 4 订阅

订阅专栏



单片机

9 篇文章 0 订阅

订阅专栏

1625-5 王子昂 总结《2017年9月13日》【连续第346天总结】

A. 问鼎杯题库-逆向

B.

找flag

运行后是一个单纯的文本输入框和灰色的GetFlag按钮
拖入IDA反编译，没有main函数，只有start函数
start里没看到什么可以继续往下的内容，直接搜索字符串：

Address	Length	Type	String
.rdata:00402120	00000005	C	Flag
.rdata:00402128	00000031	C	flag:{NSCTF_md57e0cad17016b0>?45?f7c>0>4a>1c3a0}
.rdata:00402400	0000000B	C	USER32.dll
.rdata:00402418	0000000D	C	MSVCR120.dll
.rdata:004026AE	0000000D	C	KERNEL32.dll

http://blog.csdn.net/whk1hhhh

一眼看到flag，兴奋地提交
错误OTZ
那看来程序中还有对flag的处理，双击追到内存然后按x查看交叉引用

```
.rdata:00402128 ; char Text[]
.rdata:00402128 Text            db 'flag:{NSCTF_md57e0cad17016b0>?45?f7c>0>4a>1c3a0}',0
.rdata:00402128                                         ; DATA XREF: sub_401000+23↑o
.rdata:00402128                                         ; sub_401070+ED↑o
.rdata:00402159                 align 10h
.rdata:00402160 __load_config_used dd 48h               ; Size
.rdata:00402164                 dd 0                    ; Time stamp
.rdata:00402168                 dw 2 dup(0)             ; Version: 0.0
.rdata:0040216C                 dd 0                    ; GlobalFlagsClear
.rdata:00402170                 dd 0                    ; GlobalFlagsSet
.rdata:00402174                 dd 0                    ; CriticalSectionDefaultTimeout
.rdata:00402178                 dd 0                    ; DeCommitFreeBlockThreshold
```

xrefs to Text

Directio	Ty	Address	Text
Up	o	sub_401000+23	push offset Text ; "flag:{NSCTF_md57e0cad17016b0>?45?f7c>0>"...
Up	o	sub_401070+ED	push offset Text ; "flag:{NSCTF_md57e0cad17016b0>?45?f7c>0>"...

OK Cancel Search Help

http://blog.csdn.net/whk1hhhh

有两处调用，依次查看，发现这个有趣的函数

```
int sub_401000()
{
  char *v0; // eax@1
  CHAR Text; // [sp+0h] [bp-38h]@1
  char Dst; // [sp+1h] [bp-37h]@1
  char v4; // [sp+Fh] [bp-29h]@1

  Text = 0;
  memset(&Dst, 0, 0x30u);
  strncpy_s(&Text, 0x31u, "flag:{NSCTF_md57e0cad17016b0>?45?f7c>0>4a>1c3a0}", 0x30u);//将flag字符串copy到
  v0 = &v4;//v4的地址送入v0中，实际上就是Text[15]
  if ( v4 != 125 )
  {
    do
    {
      *v0 ^= 7u;//逐字符异或循环
      ++v0;
    }
    while ( *v0 != 125 );
  }
  return MessageBoxA(0, &Text, "Flag", 0);
}
```

乍一看循环中的v0和v4都跟字符串和Text没啥关系，但事实上Text和v4都在堆栈中，v4其实就是Text的第15个字符，IDA反编译的时候将其认为是一个新的局部变量（某种意义上也可以算是Text的溢出吧233），对字符串操作的时候IDA经常会有这种误解，需要注意

那么对flag[15:]进行与7逐字符异或后提交，正确~

> NSCTF_md50b7dfc60761e798328a0d9793f96d4f7

再向上溯源的话可以发现

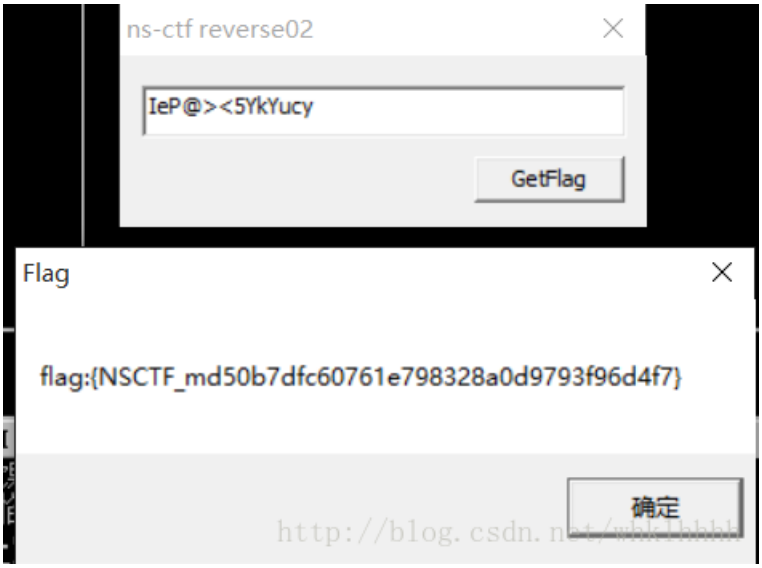
```
int __thiscall sub_401070(int this)
{
  int v1; // edx@8
  int v2; // esi@8
  int result; // eax@18

  if ( ((unsigned __int8)byte_403028 ^ 7) != *(_BYTE *)this
    || ((unsigned __int8)byte_403027 ^ 7) != *(_BYTE *)(this + 1)
    || ((unsigned __int8)byte_403026 ^ 7) != *(_BYTE *)(this + 2)
    || ((unsigned __int8)byte_403025 ^ 7) != *(_BYTE *)(this + 3)
    || ((unsigned __int8)byte_403024 ^ 7) != *(_BYTE *)(this + 4)
    || ((unsigned __int8)byte_403023 ^ 7) != *(_BYTE *)(this + 5)
    || ((unsigned __int8)byte_403022 ^ 7) != *(_BYTE *)(this + 6) )
  {
    v1 = byte_403380;
    v2 = dword_403018;
  }
  else
  {
    v1 = byte_403380 + 2;
    v2 = dword_403018 - 1;
    byte_403380 += 2;
    --dword_403018;
  }
  if ( ((unsigned __int8)byte_403021 ^ 0x33) == *(_BYTE *)(this + 7)
    && ((unsigned __int8)byte_403020 ^ 0x33) == *(_BYTE *)(this + 8)
    && ((unsigned __int8)byte_40301F ^ 0x33) == *(_BYTE *)(this + 9)
    && ((unsigned __int8)byte_40301E ^ 0x33) == *(_BYTE *)(this + 10)
    && ((unsigned __int8)byte_40301D ^ 0x33) == *(_BYTE *)(this + 11)
    && ((unsigned __int8)byte_40301C ^ 0x33) == *(_BYTE *)(this + 12) )
  {
    --v1;
    v2 += 2;
    byte_403380 = v1;
    dword_403018 = v2;
  }
  if ( v2 + v1 == 3 )
    result = sub_401000();//输出正确flag
  else
    result = MessageBoxA(0, "flag:{NSCTF_md57e0cad17016b0>?45?f7c>0>4a>1c3a0}", "Flag", 0);//错误flag
  dword_403018 = 1;
  byte_403380 = 0;
  return result;
}
```

在这个函数对this和内存进行比较，按结果操作v1和v2，最后正确的话call sub_401000，否则直接显示（错误的）flag
很明显this就是输入字符串的指针了，byte_403028-byte_40301c用IDC脚本dump下来，写一个异或的变换即可得到正确输入：

```
a = [78, 98, 87, 71, 57, 59, 50]
b = [106, 88, 106, 70, 80, 74]
for i in a:
    print(chr(i ^ 7), end='')
for i in b:
    print(chr(i ^ 0x33), end='')
```

将这个字符串输入程序，按钮仍然是灰色的
检查了原程序也没有发现EnableMainItem这个可以修改可用性的API，估计是控件属性设置了
以前做CrackMe的时候也遇到过这种，在内存中可以找到包含控件ID的属性设置，将Enable属性修改即可
查看了大佬的WriteUp发现可以直接用Spy++之类的工具，将按钮改为可用后点击，弹出了正确的flag



虽然这个才是最正统的通过方法，不过MessageBox中的flag作为Lable并不能复制啊……不知道该说什么好OTZ
C. 明日计划
问鼎杯逆向