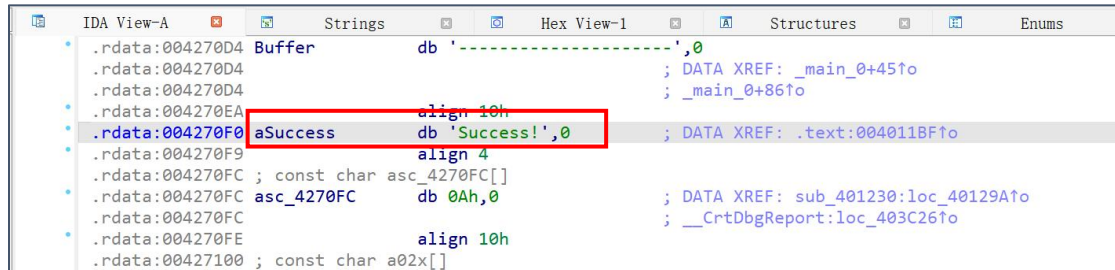


## 测试报告

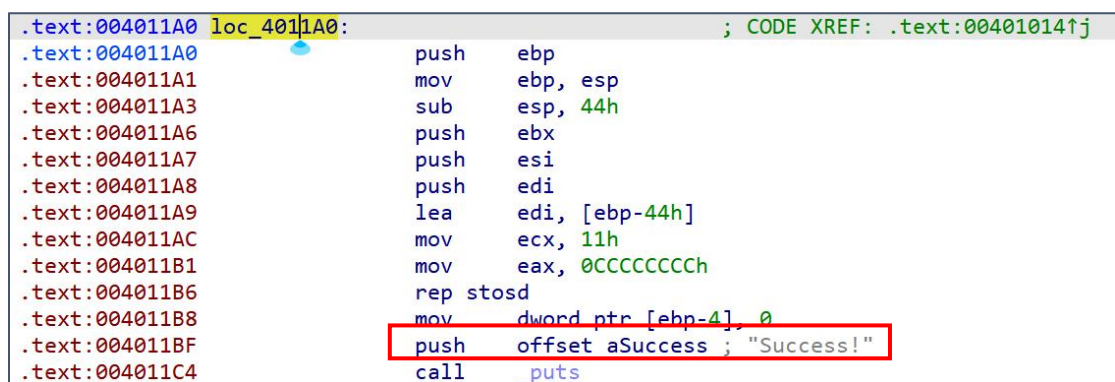
用 IDA 打开可执行文件，先进行静态分析

搜索 Success 字符串



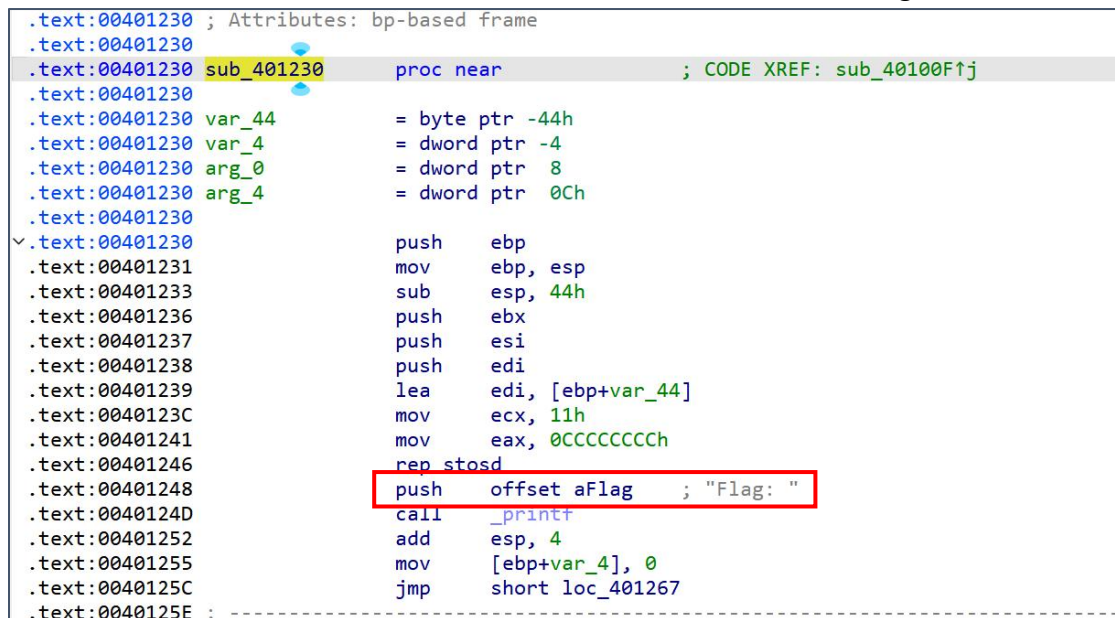
```
.idata:004270D4 Buffer db '-----',0
.rdata:004270D4 ; DATA XREF: _main_0+45fo
.rdata:004270D4 ; _main_0+86fo
.rdata:004270EA align 10h
.rdata:004270F0 aSuccess db 'Success!',0 ; DATA XREF: .text:004011BFfo
.rdata:004270F9 align 4
.rdata:004270FC ; const char asc_4270FC[]
.rdata:004270FC asc_4270FC db 0Ah,0 ; DATA XREF: sub_401230:loc_40129Afo
.rdata:004270FE ; _CrtDbgReport:loc_403C26fo
.rdata:00427100 align 10h
.rdata:00427100 ; const char a02x[]
```

查看引用字符串处函数



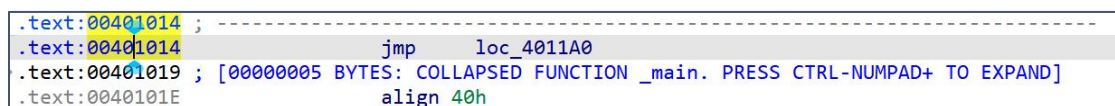
```
.text:004011A0 loc_4011A0: ; CODE XREF: .text:00401014fj
.text:004011A0 push ebp
.text:004011A1 mov ebp, esp
.text:004011A3 sub esp, 44h
.text:004011A6 push ebx
.text:004011A7 push esi
.text:004011A8 push edi
.text:004011A9 lea edi, [ebp-44h]
.text:004011AC mov ecx, 11h
.text:004011B1 mov eax, 0CCCCCCCCh
.text:004011B6 rep stosd
.text:004011B8 mov dword ptr [ebp-4], 0
.text:004011BF push offset aSuccess ; "Success!"
.text:004011C4 call _puts
```

同时发现 success 函数中调用了一个函数，进入该函数发现是 flag 打印函数



```
.text:00401230 ; Attributes: bp-based frame
.text:00401230 sub_401230 proc near ; CODE XREF: sub_40100Ffj
.text:00401230 var_44 = byte ptr -44h
.text:00401230 var_4 = dword ptr -4
.text:00401230 arg_0 = dword ptr 8
.text:00401230 arg_4 = dword ptr 0Ch
.text:00401230 push ebp
.text:00401231 mov ebp, esp
.text:00401233 sub esp, 44h
.text:00401236 push ebx
.text:00401237 push esi
.text:00401238 push edi
.text:00401239 lea edi, [ebp+var_44]
.text:0040123C mov ecx, 11h
.text:00401241 mov eax, 0CCCCCCCCh
.text:00401246 rep stosd
.text:00401248 push offset aFlag ; "Flag: "
.text:0040124D call _printf
.text:00401252 add esp, 4
.text:00401255 mov [ebp+var_4], 0
.text:0040125C jmp short loc_401267
.text:0040125E ;
```

到此已得知目标函数的地址为 004011A0，下一步是分析如何转移到该函数  
但发现程序中跳转到该函数的语句并不出现在主程序中



```
.text:00401014 ; -----
.text:00401014 jmp loc_4011A0
.text:00401019 ; [00000005 BYTES: COLLAPSED FUNCTION _main. PRESS CTRL-NUMPAD+ TO EXPAND]
.text:0040101E align 40h
```

下一步分析主程序，看是否有漏洞可以利用，反汇编主程序代码，结果如下：

```

1 int __cdecl sub_4012E0(int a1)
2 {
3     int v1; // eax
4     _BYTE FileName[552]; // [esp+4Ch] [ebp-228h] BYREF
5
6     FileName[260] = 0;
7     memset(&unk_42D054, 0, 0x10u);
8     memset(&FileName[264], 0, 0x120u);
9     puts("Please input your account:");
10    scanf("%s", &unk_42D054);
11    if ( a1 == 1 )
12    {
13        puts("Please input your passwd:");
14        scanf("%s", &FileName[264]);
15        v1 = strlen(&FileName[264]);
16        FileName[260] = v1;
17    }
18    else
19    {
20        memset(FileName, 0, 0x12Cu);
21        puts("Please input your file_name:");
22        scanf("%s", FileName);
23        *(_DWORD *)&FileName[256] = fopen(FileName, "rb");
24        if ( !*(_DWORD *)&FileName[256] )
25        {
26            puts("file dose'nt exist!");
27            return 0;
28        }
29        FileName[260] = fread(&FileName[264], 1u, 0x10Eu, *(FILE **)&FileName[256]);
30        v1 = fclose(*(FILE **)&FileName[256]);
31    }
32    LOBYTE(v1) = FileName[260];
33    return sub_401005(&FileName[264], v1);
34 }

```

可以看到在文件输入模式下，函数会读入用户名和文件名，并试图打开文件读取字符串。尝试进入字符串读取函数，发现会出现栈溢出的 strcpy 函数

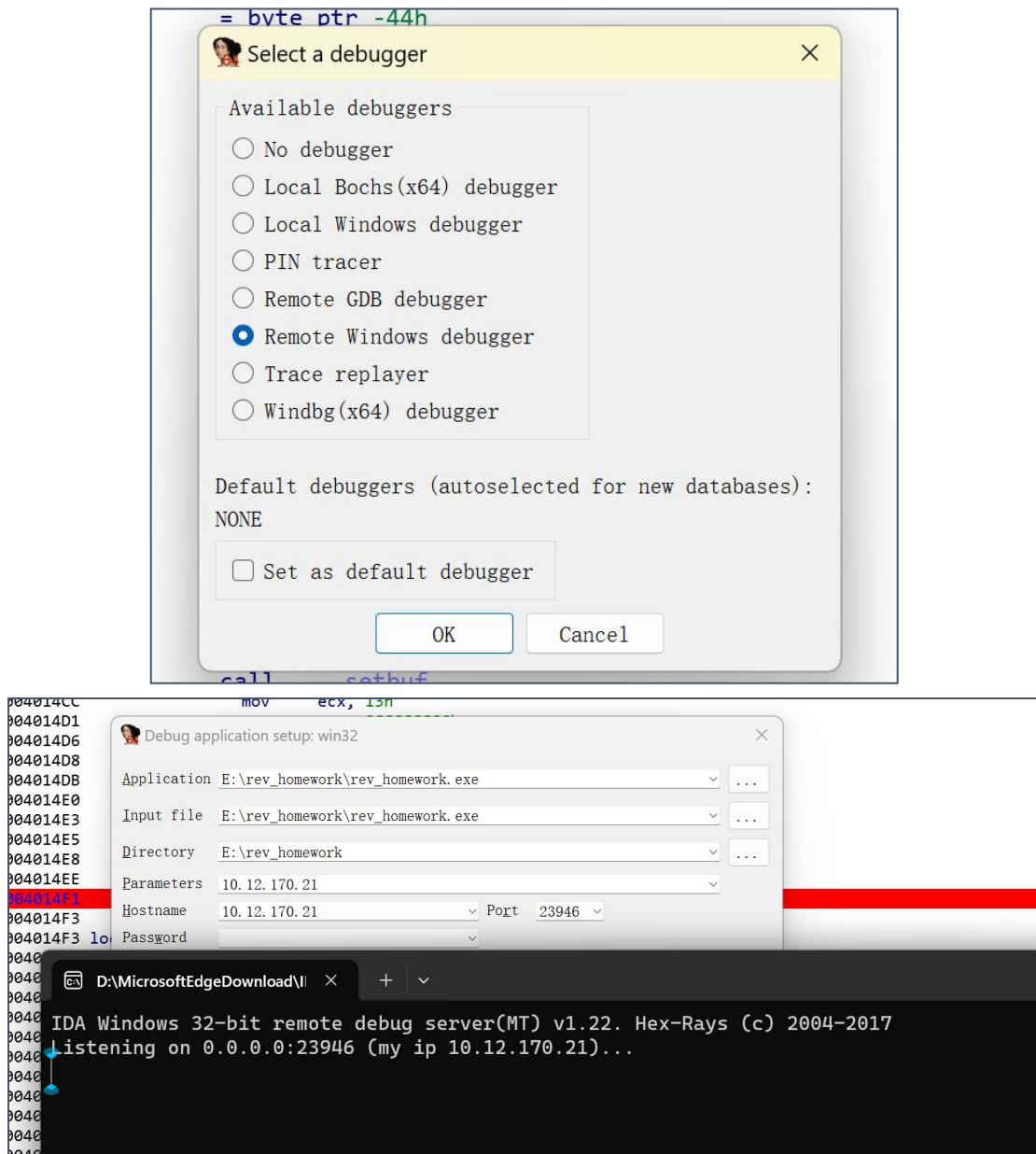
```

1 char *__cdecl sub_4014C0(char *Source, unsigned __int8 a2)
2 {
3     char Destination[8]; // [esp+4Ch] [ebp-Ch] BYREF
4
5     if ( a2 > 3u && a2 <= 8u )
6     {
7         fflush(&File);
8         return strcpy(Destination, Source);
9     }
10    else
11    {
12        puts("Invalid Password");
13        return (char *)fflush(&File);
14    }
15 }

```

观察上面的栈底栈顶位置，可知输入串将会放入栈底 12 字节处，读入输入串的长度是无符号数 unsigned\_int8，也就是单字节值。当长度满足大于 3 小于等于 8 的时候才会执行复制。如果我们输入一个大于 255 的数，在转换为 unsigned\_int8 时会直接高位截断，只留下低位，所以只要构造一个 255+x, 3<x<=8 的字符串，就可以实现溢出覆盖到返回地址，完成漏洞利用。观察栈基地址的位置，读入字符串 12 字节，加上保存的 ebp 值的 4 字节，一共 16 字节，也就是说构造的字符串从 16 字节开始填充目标跳转地址即可覆盖到 eip 返回地址。

下面进行动态调试，运行 win32\_remote.exe，在 IDA 中切换 debugger。



在 strcpy 函数内部打断点，开始调试。执行到断点处发现 ebp 和 eip 的值果然被覆盖成非正常值



由此修改二进制文件，将第 16-19 字节修改为目标函数地址

E: > rev\_homework > test.txt

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded Text	Data Inspector
00000000	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	1 1 1 1 1 1 1 1 1 1 1 1 1 1	binary 00110001
00000010	A0	11	40	00	32	32	32	32	32	32	32	32	32	32	32	32	. . @ . 2 2 2 2 2 2 2 2 2 2 2 2	octal 061
00000020	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	31	1 1 1 1 1 1 1 1 1 1 1 1 1 1	uint8 49

运行程序，成功获取 flag!!!

```

E:\rev_homework\rev_homew  X  +
-----
~~ Welcome to III-2! ~~
    1.Login by keybord
    2.Login by pw_file
    3.Exit
-----
Your choice:2
U202112131
Please input your account:
Please input your file_name:
test.txt
Success!
Flag: 4a2d2f2d2e2e2d2e2c2e

```