《软件安全》实验报告

姓名: 陆皓喆 学号: 2211044 班级: 信息安全

实验名称:

OLLYDBG软件破解

实验要求:

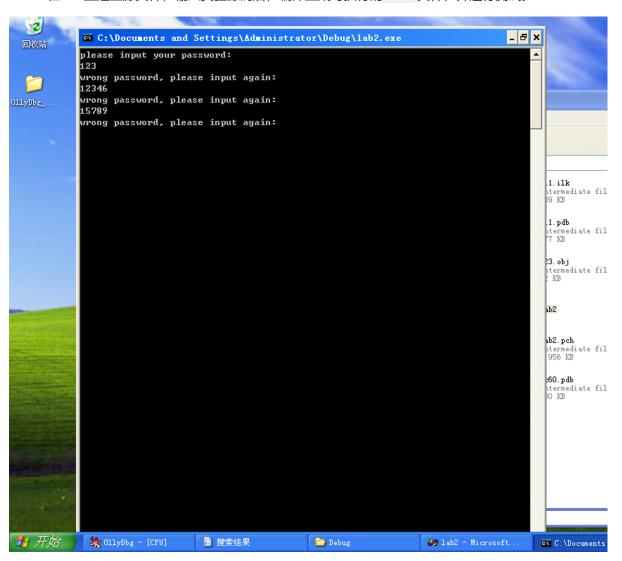
1.请在XP VC6生成课本第三章软件破解的案例(DEBUG模式,示例3-1)。进而,使用OllyDBG进行单步调试,获取 verifyPwD 函数对应 f1ag==0 的汇编代码,并对这些汇编代码进行解释。

2.对生成的DEBUG程序进行破解,复现课本上提供的两种破解方法。

实验过程:

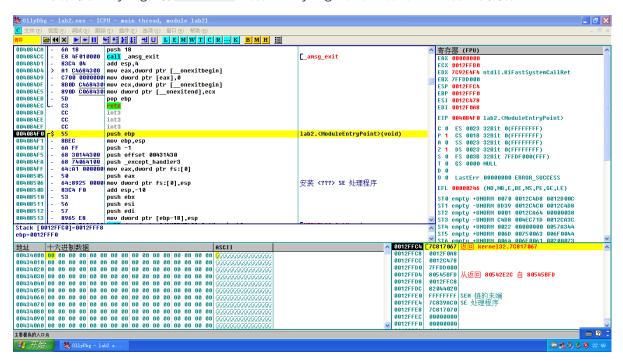
一、初步测试

1.在VC6上建立源文件,输入实验源码后,编译生成可执行的.exe文件,并进行测试。



如图可看出当输入的密码不正确时,会输出"wrong password",并持续重复输入,无法进入核心逻辑(即程序跳出)。那么我们如何通过修改程序的汇编语言代码来实现进入核心逻辑呢?我们需要进行以下的一些步骤。

2.我们使用ollydbg,将 1ab2.exe 文件拖入ollydbg进行动态调试。导入后如下图所示。



下面我们就对其进行破解操作。

二、对verifyPwd函数的具体解释

首先,我们知道该函数肯定是决定着我们能否进入核心过程的一个重要函数,所以我们先进行关键字查询,看看有没有出现"wrong"之类的关键词。

```
main thread, module lab2
004010DE
                   52
                    E8 30FFFFFF
                                                                                                                              [verifyPwd
                                         add esp,4
mov byte ptr [ebp-4],al
mov eax,dword ptr [ebp-4]
004010E4
                    8304 04
                   8845 FC
8B45 FC
004010F7
004010EA
ពព្ធភព។ RED
                    25 FF000000
                                         and eax,000000FF
test eax,eax
                                         push offset 00401105
push offset 00431054
call printf
add esp,4
jmp short 00401114
004010F4
                    74 ØF
                   68 <u>54104300</u>
E8 50720000
                                                                                                                               ASCII "passed∎"
004010FB
00401100
00401103
                   83C4 04
EB 0F
                   68 28104300
E8 41720000
                                         push offset 00431028
call printf
00401105
                                                                                                                               ASCII "wrong password, please input again:∎"
                                        add esp,4
<mark>jmp</mark> short 004010BB
0040110F
                    8304 84
```

经过搜索,我们发现 verifyPwd 函数就是从此处开始的。而且,这个密码正确或者是错误的判断是从上面的一句汇编语句 jz short 00401105 来决定的。

为了找到 verifyPwd 函数的位置,我们右键选择"跟随",在跟随两次后,我们找到了该函数的位置。

```
0040102F
            cc
                           int3
                                                                                        lab2.verifyPwd(void)
00401030
                            push ebp
00401031
              8BEC
                            mov ebp,esp
00401033
              83EC 44
                            sub esp,44
00401036
              53
                            push ebx
00401037
                            push esi
              56
                            .
push edi
00401038
              57
00401039
              8D7D BC
                            lea edi,[ebp-44]
                            mov ecx,11
0040103C
             B9 11000000
             B8 CCCCCCCC mov eax,CCCCCCCC rep stos dword ptr [edi]
00401041
00401046
00401048
              8B45 08
                            mov eax, dword ptr [ebp+8]
0040104B
              50
                            push eax
             68 1C104300 push offset 0043101C
E8 CA710000 call strcmp
                                                                                        ASCII "12345678"
00401040
00401051
                                                                                        [strcmp
00401056
              8304 08
                            add esp,8
00401059
              8945 FC
                            mov dword ptr [ebp-4],eax
00401050
             3300
                            xor eax,eax
              837D FC 00
0040105E
                           cmp dword ptr [ebp-4],0
00401062
              0F94C0
                           sete al
```

核心代码:

第一行将输入的 password 的首地址移入 eax 中, 空出8个位置

第二行将 eax 入栈

第三行将待比对的正确密码入栈 (12345678)

第四行调用 strcmp 函数比对两个字符串是否相等,如果两个字符串相等,则eax的值为0,如果两个字符串不相等,则 eax 的值为00000001

第五行做栈平衡

第六行将 strcmp 的结果从 eax 中移送到局部变量 flag 中

第七行将 eax 寄存器的值清空

第八行对 flag 的值和0进行比较,如果相等的话就会设置 ZF 为1,否则 ZF 为0

第九行根据ZF的值设置 eax 的低八位,如果 ZF 为1,就设置 a1 为1,反之 a1 被置0

三、两种破解方法

3.1 修改主函数中的条件跳转指令

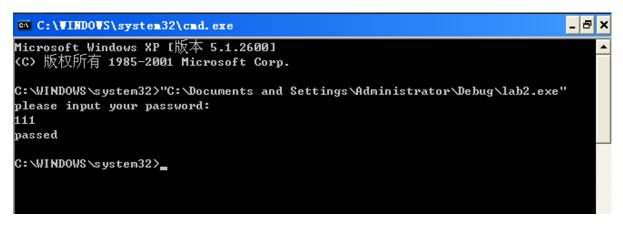
我们回到上面的 verifyPwd 函数部分的判定密码是否正确的部分,可以发现,基本上是由一个 jz 的判定语句构成的。

CPU - main thread, module lab2				
004010DE		52	push edx	
004010DF		E8 30FFFFFF	call 00401014	[verifyPwd
004010E4		8304 04	add esp,4	
004010E7		8845 FC	mov byte ptr [ebp-4],al	
004010EA	٠.	8B45 FC	mov eax,dword ptr [ebp-4]	
004010ED	٠.	25 FF000000	and eax,000000FF	
004010F2	٠.	8500	test eax,eax	
004010F4	٠,	74 OF	<mark>jz</mark> short 00401105	
004010F6		68 54104300	push offset 00431054	ASCII "passed∎"
004010FB		E8 50720000	call printf	[printf
00401100		8304 04	add esp,4	
00401103	٠.,	EB 0F	jmp short 00401114	
00401105	>	68 28104300	push offset 00431028	ASCII "wrong password, please input again:∎"
0040110A		E8 41720000	call printf	[printf
0040110F		8304 04	add esp,4	
00401112	• ^	EB A7	L <mark>jmp</mark> short 004010BB	

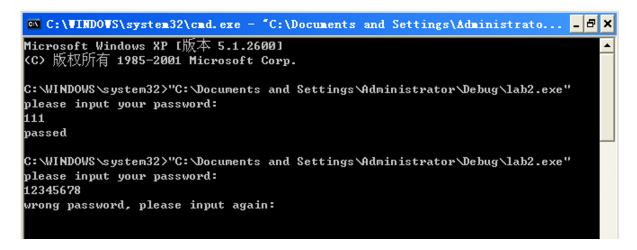
我们跟踪 00401105 语句,发现该语句是用于输出密码错误的,反之 004010F6 语句是用于输出密码通过测试。

所以,实际上程序的逻辑应该是:如果密码符合,那么跳转到"passed"语句,否则跳转到密码错误的语句。因此,我们可以修改逻辑判断语句 jz short 00401105 为 jnz short 00401105 , 这样就可以反方向跳转,输入错误的密码会跳转到passed界面。

我们双击该语句,进行修改,然后右键选择"编辑",选择"复制当前修改到所有可执行文件",这样才是真正修改了我们的 lab.exe 文件。我们修改完毕后,重新打开 exe 文件,输入原本是错误的密码"111",发现输出了passed,说明我们的修改起了作用。



为了进一步验证,我们输入原本正确的密码"12345678",发现输出的是wrong,说明程序的核心判断语句已经被我们修改为反逻辑了。



3.2 修改flag==0的返回值指令,令其永远返回1

接下来我们尝试第二种方法来进行破解。基于我们在第一部分中讨论的 f1ag==0 的对应区域汇编代码,我们可以想到另一种更有效地破解方法,那就是让 f1ag==0 的返回值永远为1,这样可以避免当输入正确密码而提示密码错误的情况。这样我们就可以实现,不管输入什么密码,都会输出passed的结果。

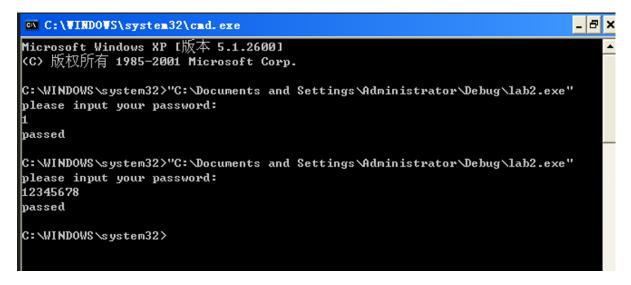
根据上面所说的,我们找到 flag==0的 语句,发现该语句的功能是靠 cmp dword ptr [ebp-4],0 来实现的。我们直接将其判断函数删除,修改为直接赋值,即直接将 al 寄存器的值修改为1即可。

```
C CPU - main thread, module lab2
00401033
             83EC 44
                           sub esp,44
00401036
             53
                           push ebx
00401037
             56
                           .
push esi
00401038
                           push edi
00401039
             8D7D BC
                           lea edi,[ebp-44]
0040103C
             B9 11000000
                           mov ecx,11
                           mov eax,CCCCCCCC
00401041
             B8 CCCCCCCC
00401046
             F3:AB
                           rep stos dword ptr [edi]
00401048
             8B45 08
                           mov eax, dword ptr [ebp+8]
0040104B
             50
                           push eax
0040104C
             68 10104300
                           push offset 0043101C
                                                                                     ASCII "12345678"
00401051
             E8 CA710000
                           call strcmp
                                                                                    [strcmp
00401056
             8304 08
                           add esp,8
00401059
             8945 FC
                           mov dword ptr [ebp-4],eax
0040105C
             3300
                           xor eax.eax
0040105E
             B0 01
                           mov al.1
00401060
              90
                           nop
00401061
              98
00401062
              0F94C0
                           sete
```

进一步,我们再来修改下一条语句 sete al。我们已经给 al 寄存器永久赋值为1了,所以自然不需要这一步了。我们直接将该语句使用 nop 指令进行填充即可。

```
0040104B
                           push eax
0040104C
             68 10104300
                                                                                     ASCII "12345678"
                           push offset 0043101C
00401051
             E8 CA710000
                           call strcmp
                                                                                     [strcmp
00401056
             8304 08
                           add esp,8
00401059
             8945 FC
                           mov dword ptr [ebp-4],eax
0040105C
             3300
                           xor eax,eax
0040105F
             B0 01
                           mov al,1
00401060
             98
                           nop
00401061
             98
                           nop
00401062
             90
                           nop
00401063
00401064
                           nop
             90
```

我们同样右键选择编辑,复制当前修改到可执行文件,然后运行可执行文件。我们发现无论输入什么样的密码,窗口都一闪而过,证明我们成功破解了程序。



心得体会:

通过本次实验,我学会了ollydbg的动态调试方式,包括一些很简单的操作,比如说在整个栈内存中通过跟随操作来跟踪程序的进程;通过查找关键词来定位关键语句;通过修改程序汇编语言代码来实现程序的简单破解。

通过实验,我熟悉了很多关键核心逻辑语句的寻找与判定,学会了修改核心语句来使代码错误跳转从而达到目的,对汇编语言代码的认识更加深刻了。这个实验和上学期的《汇编语言与逆向技术》中的最后一个实验十分的相似,上学期我们也是通过修改汇编语言中的核心代码来实现boss血量的修改,boss数量的修改等等。这和本次实验也是共通的,其实理解汇编语言的核心是找到准确的核心逻辑语句的位置。比如说本次实验,我们通过查找关键词的方式,在很大的代码块中找到了自己需要的东西。在CTF比赛中的某些题型,这样的分析技能是最基本的,我还需要多做该方面的练习来进一步熟练掌握该项技能。