

《软件安全》实验报告

姓名：刘星宇 学号：2212824 班级：信息安全法学双学位班

实验名称：

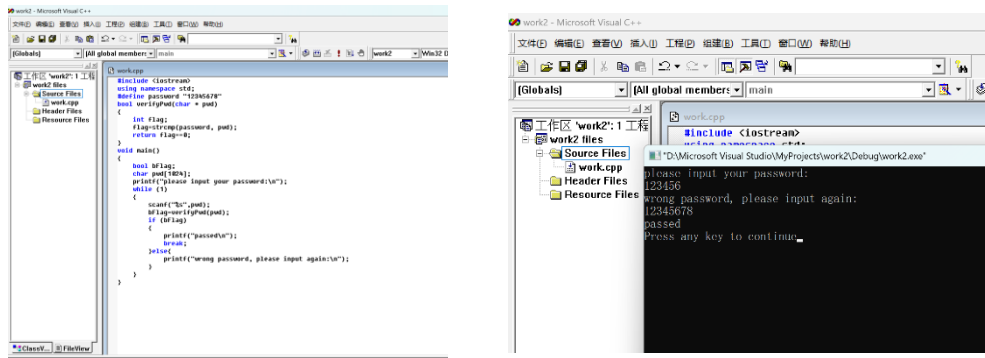
OLLYDBG 软件破解

实验要求：

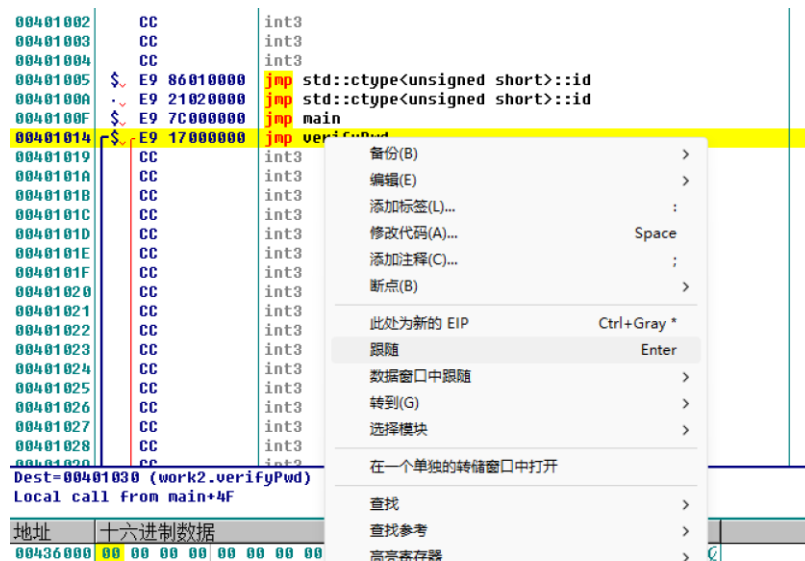
1. 请在 XP VC6 生成课本第三章软件破解的案例(DEBUG 模式，示例 3-1)。进而，使用 OllyDBG 进行单步调试，获取 verifyPWD 函数对应 flag==0 的汇编代码，并对这些汇编代码进行解释。
2. 对生成的 DEBUG 程序进行破解，复现课本上提供的两种破解方法。

实验过程：

1. 在 XP VC6 生成课本第三章软件破解的案例



2. 使用 OllyDBG 进行单步调试，获取 verifyPWD 函数对应 flag==0 的汇编代码对 verifyPWD 函数进行跟踪找到函数所在位置：



CPU - main thread, module work2			
00401030	> 55	push ebp	work2.verifyPwd(void)
00401031	- 8BEC	mov ebp,esp	
00401033	- 83EC A4	sub esp,44	
00401036	- 53	push ebx	
00401037	- 56	push esi	
00401038	- 57	push edi	
00401039	- 8D7D 0C	lea edi,[ebp-44]	
0040103C	- 09 11000000	mov ecx,11	
00401041	- 8B CCCCCCCC	mov eax,CCCCCCCC	
00401046	- F3:AB	rep stos dword ptr [edi]	
00401048	- 8B45 08	mov eax,dword ptr [ebp+8]	
0040104B	- 50	push eax	
0040104C	- 68 1C30A300	push offset 0043301C	
00401051	- E8 C6710000	call strcmp	ASCII "12345678"
00401056	- 83C4 08	add esp,8	[strcmp
00401059	- 8945 FC	mov dword ptr [ebp-4],eax	
0040105C	- 33C0	xor eax,eax	
0040105E	- 837D FC 00	cmp dword ptr [ebp-4],0	
00401062	- 0F94C0	sete al	
00401065	- 5F	pop edi	
00401066	- 5E	pop esi	
00401067	- 5B	pop ebx	
00401068	- 83C4 A4	add esp,44	
0040106B	- 3BEC	cmp ebp,esp	
0040106D	- E8 3E720000	call _chkesp	
00401072	- 8BE5	mov esp,ebp	
00401074	- 50	pop ebp	
00401075	- C3	ret	
地址	十六进制数据	ASCII	寄存器
00436000	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	CCCCCCCCCCCCCCCC	0019FF70 76A97B09
00436010	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	CCCCCCCCCCCCCCCC	0019FF7C 00362000
00436020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	CCCCCCCCCCCCCCCC	0019FF80 76A97B09
00436030	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	CCCCCCCCCCCCCCCC	0019FF84 0019FFDC
00436040	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	CCCCCCCCCCCCCCCC	0019FF88 0019FF88
00436050	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	CCCCCCCCCCCCCCCC	0019FF8C 00362000

3. 对 verifyPWD 函数对应 flag==0 的汇编代码进行解释。

push ebp	work2.verifyPwd(void)
mov ebp,esp	
sub esp,44	
push ebx	
push esi	
push edi	
lea edi,[ebp-44]	
mov ecx,11	
mov eax,CCCCCCCC	
rep stos dword ptr [edi]	
mov eax,dword ptr [ebp+8]	
push eax	
push offset 0043301C	ASCII "12345678"
call strcmp	[strcmp
add esp,8	
mov dword ptr [ebp-4],eax	
xor eax,eax	
cmp dword ptr [ebp-4],0	
sete al	
pop edi	
pop esi	
pop ebx	
add esp,44	
cmp ebp,esp	
call _chkesp	
mov esp,ebp	
pop ebp	
ret	

push ebp //将 ebp (将基指针寄存器) 压栈

mov ebp,esp //将 esp (栈指针寄存器) 的当前值复制到 ebp。此时, ebp 指向当前栈帧的底部。

sub esp,44 //从 esp 中减去 44 (十进制), 为局部变量和可能的临时值在栈上预留空间。这 44 个字节的空间通常用于存储局部变量。

push ebx

push esi

push edi

//将 ebx、esi 和 edi 寄存器的当前值压入栈中

lea edi,[ebp-44] //使用 lea (加载有效地址) 指令计算 ebp-44 的地址, 并将结果存储在 edi 寄存器中。设置一个指向栈帧中局部变量区域的指针。

mov ecx,11//循环

mov eax CCCCCCCC//将立即数 CCCCCCCC 加载到 eax 寄存器中

rep stos dword ptr[edi]//重复指令, 用于初始化

mov eax,dword ptr [ebp+8]//将栈中第一个参数加载到 eax 寄存器

```

push eax
push offset 0043301C
//将 eax 寄存器的值和一个偏移地址 0043301C 压入栈中，作为后续 strcmp 函数的参数。
call strcmp//调用 strcmp 函数，比较两个字符串（由栈上的参数指定）
Add esp,8//从栈中弹出两个之前压入的参数，即 eax 的值和偏移地址 0043301C，通过增加 esp 寄存器来释

```

放这些空间

```

mov dword ptr [ebp-4],eax
//将 strcmp 的比较结果存储到栈帧中的一个局部变量位置（ebp-4）
xor eax,eax
//将 eax 寄存器清零
cmp dword ptr [ebp-4],0
sete al
//比较栈帧中存储的 strcmp 返回值是否为 0（即两个字符串是否相等）。如果是，则设置 al（eax 寄存器的最低字节）
为 1，否则为 0

```

```

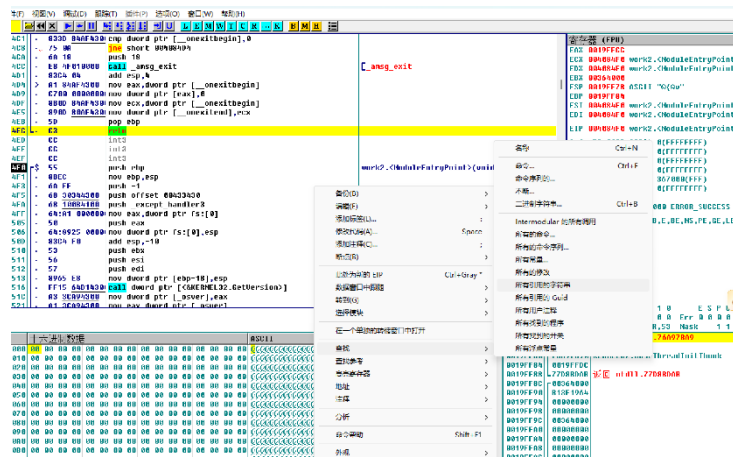
pop edi
pop esi
pop ebx
//出栈
add esp,44
cmp ebp,esp
call chkesp
//比较 ebp 和 esp 是否相等，以检查栈是否平衡。如果不平衡，则调用 chkesp 函数
mov esp,ebp
//将 esp 设置为 ebp 的值，这通常是函数结束时的标准操作，以确保栈帧正确释放
pop ebp
//出栈
Retn//结束

```

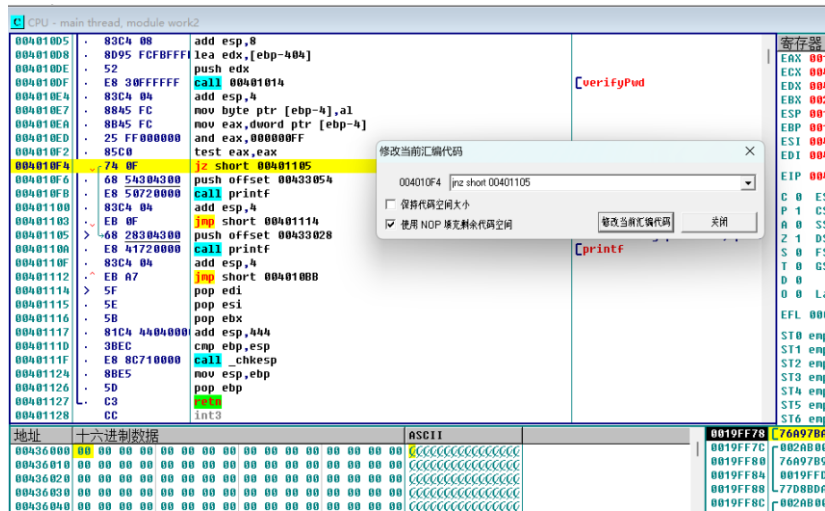
4. 对生成的 DEBUG 程序进行破解，复现课本上提供的两种破解方法。

（1）将 jz 该指令改为 jnz，则程序截然相反。输入了错误密码，将进入验证成功的分支中。

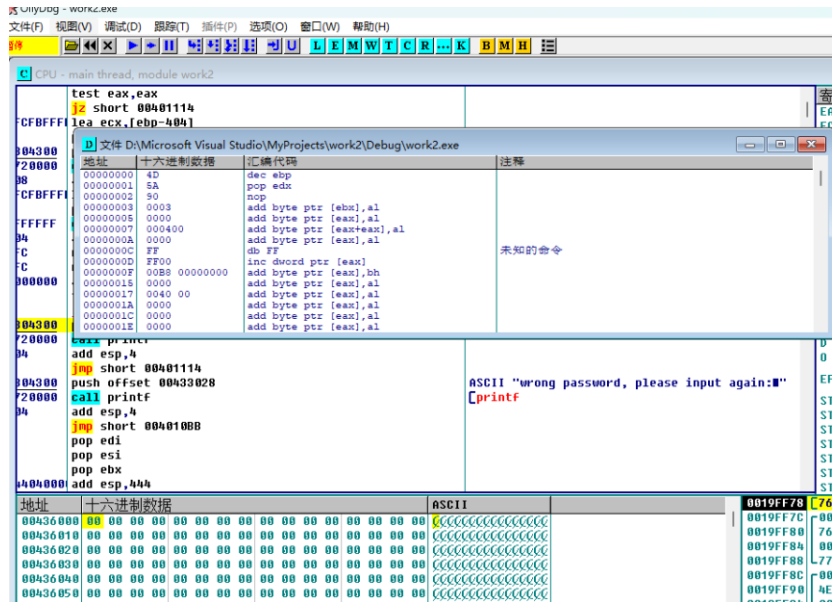
通过查找找到 wrong password, please input again:



将 jz 修改为 jnz



在反汇编窗口，点右键，选择“编辑->复制当前修改到可执行文件”。保存后的可执行文件，将是破解后的文件

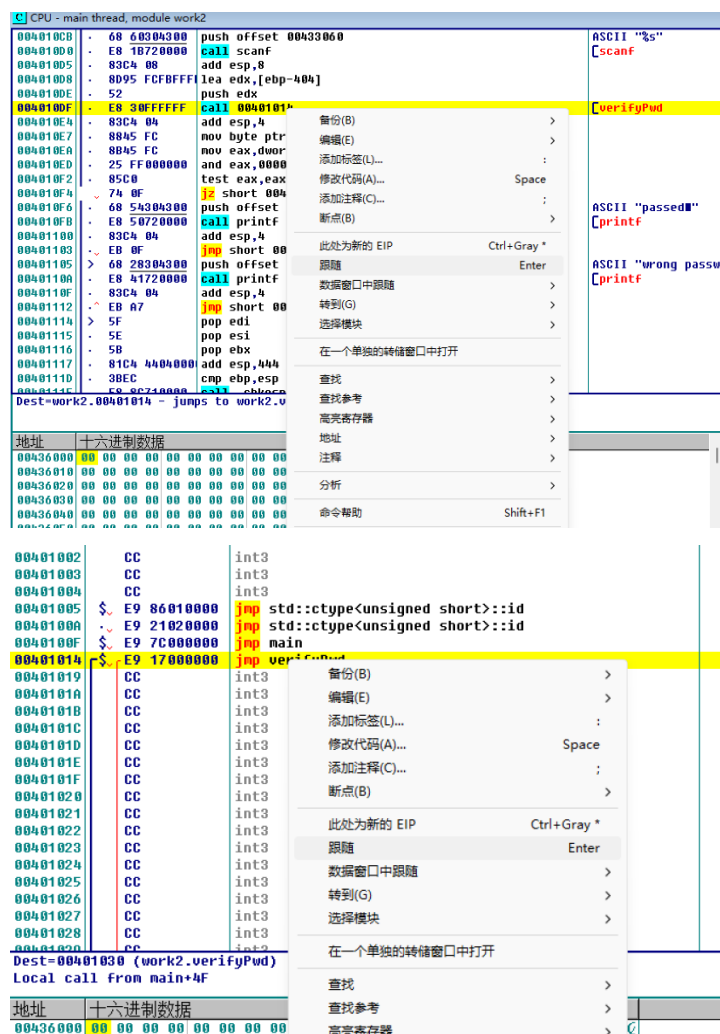


得到结果与原来相反，输入错误密码能够进行，正确密码则会显示错误

```
C:\Users\liuxi\Desktop\work2 X + v

please input your password:
12345678
wrong password, please input again:
12|
```

(2) 更改函数。通过分析汇编语句，可知，验证命令使用的是 verifyPwd 函数，点右键选择跟随，逐步进入该函数。



函数的返回值通过 eax 寄存器来完成的，核心语句即 sete al。

CPU - main thread, module work2			
00401037	56	push esi	
00401038	57	push edi	
00401039	8D7D BC	lea edi,[ebp-44]	
0040103C	B9 11000000	mov ecx,11	
00401041	B8 CCCCCCCC	mov eax,CCCCCCCC	
00401046	F3AB	rep stos dword ptr [edi]	
00401048	8B45 08	mov eax,dword ptr [ebp+8]	
00401048	50	push eax	
0040104C	68 1C304300	push offset 0043301C	ASCII "12345678"
00401051	E8 CA710000	call strcmp	[strcmp
00401056	83C4 08	add esp,8	
00401059	8945 FC	mov dword ptr [ebp-4],eax	
0040105C	33C0	xor eax,eax	
0040105E	80 01	mov al,1	
00401060	90	nop	
00401061	90	nop	
00401062	90	nop	
00401063	90	nop	
00401064	90	nop	
00401065	5F	pop edi	
00401066	5E	pop esi	
00401067	5B	pop ebx	
00401068	83C4 44	add esp,44	
0040106B	3BEC	cmp ebp,esp	
0040106D	E8 3E720000	call _chkesp	
00401072	8B5C	mov ebx,ebp	

Mov dword ptr [ebp-8], eax //将 strcmp 函数调用后的返回值（存在 eax 中）赋值给变量 flag

Xor eax, eax //将 eax 的值清空

Cmp dword ptr [ebp-8], 0 //将 flag 的值与 0 进行比较，即 flag==0;

//注意 cmp 运算的结果只会影响一些状态寄存器的值

Sete al //sete 是根据状态寄存器的值，如果相等，则设置，如果不等，则不设置

在 cmp dword ptr [ebp-8], 0 处开始更改，将其更改为：mov al,01。取消保持代码空间大小，如果新代码超长，将无法完成更改。

并将 sete al 改为 NOP。

得到结果如下：

300	push eax	
000	push offset 0043301C	ASCII "12345678"
	call strcmp	[strcmp
	add esp,8	
	mov dword ptr [ebp-4],eax	
	xor eax,eax	
	mov al,0	
	nop	
	nop	
	nop	
	nop	
	pop edi	
	pop esi	
	pop ebx	
	add esp,44	
	cmp ebp,esp	
000	call _chkesp	
	mov esp,ebp	
	pop ebp	
	call	

修改当前汇编代码

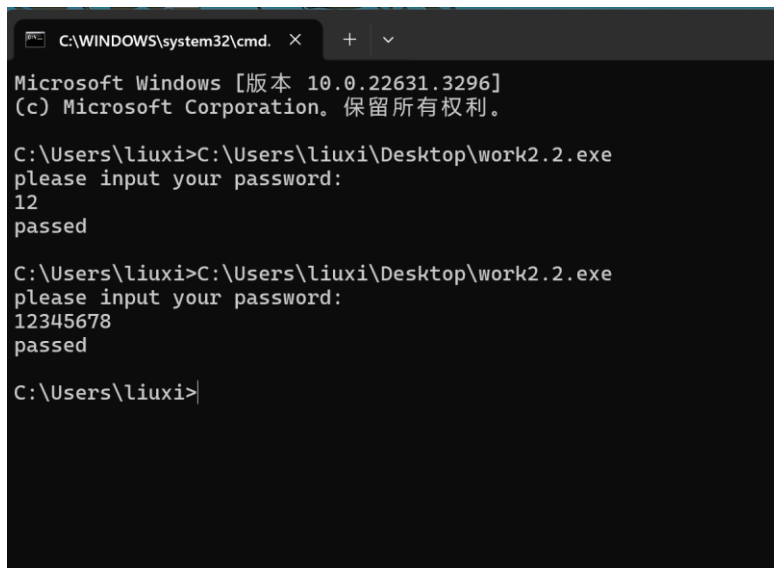
0040105E mov al,01

☐ 保持代码空间大小

☒ 使用 NOP 填充剩余代码空间

修改当前汇编代码 关闭

运行结果校验破解正确性



```
C:\WINDOWS\system32\cmd. X + v
Microsoft Windows [版本 10.0.22631.3296]
(c) Microsoft Corporation。保留所有权利。

C:\Users\liuxi>C:\Users\liuxi\Desktop\work2.2.exe
please input your password:
12
passed

C:\Users\liuxi>C:\Users\liuxi\Desktop\work2.2.exe
please input your password:
12345678
passed

C:\Users\liuxi>
```

心得体会：

通过实验，掌握了 OllyDbg 的基本使用方法，通过 Trace（跟踪）功能可以记录调试过程中执行的指令，用于分析前序执行指令，并注意到了修改是在原始文件副本里修改的，如果要保存修改，需要“编辑→复制所有修改到可执行文件”，会弹出一个对话框，包含所有修改后的代码；在这个对话框空白处继续点右键“编辑→保存文件”，弹出保存文件的界面，在这个里面选择保存类型为“可执行文件或 DLL”，输入新的文件名，然后保存。

在这次实验中，我进一步熟悉了汇编语言的各种用法，通过运行程序，观察关键信息，通过对关键信息定位，来得到关键分支语句，通过对该分支语句进行修改，达到破解的目的。