

# 《软件安全》实验报告

## 实验名称：

OLLYDBG软件破解

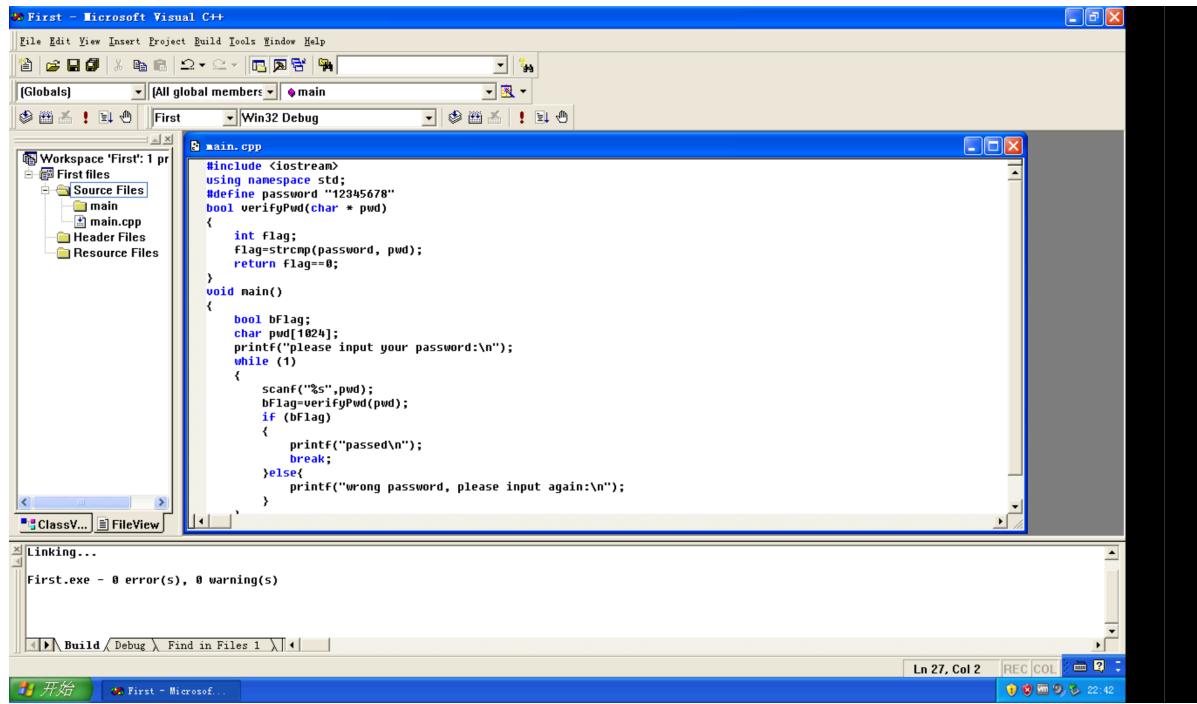
## 实验要求：

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。

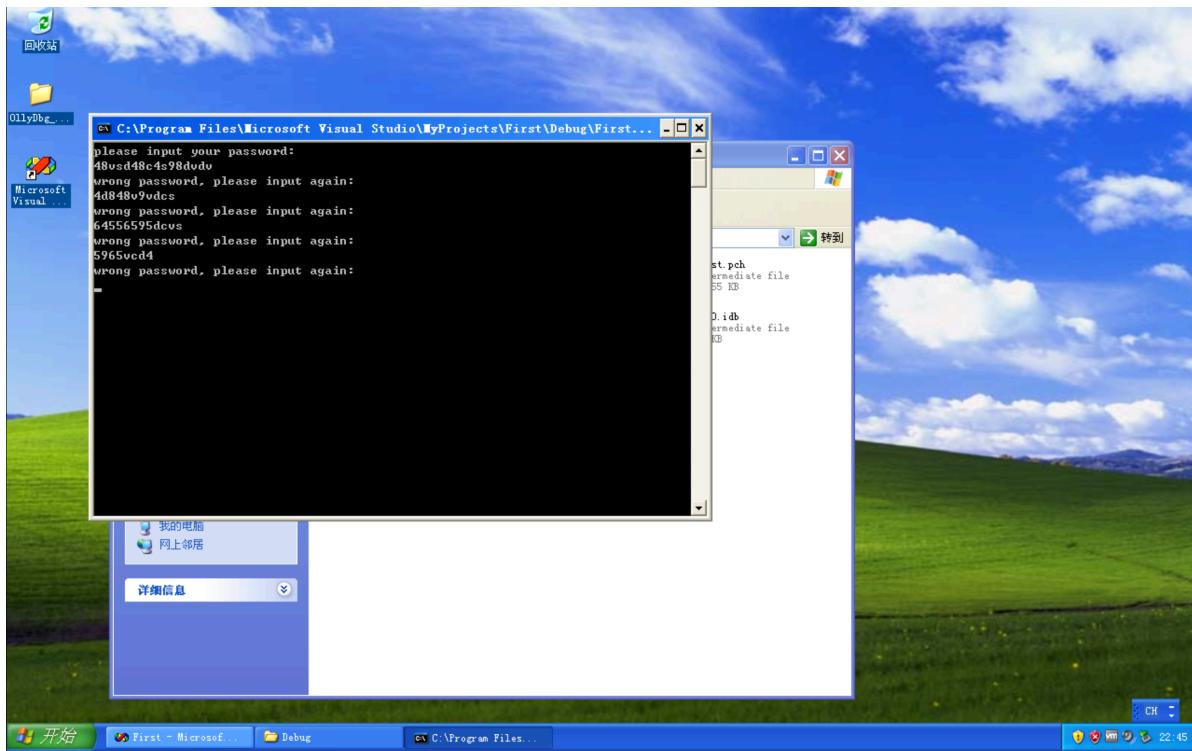
## 实验过程：

### 一、初步调试

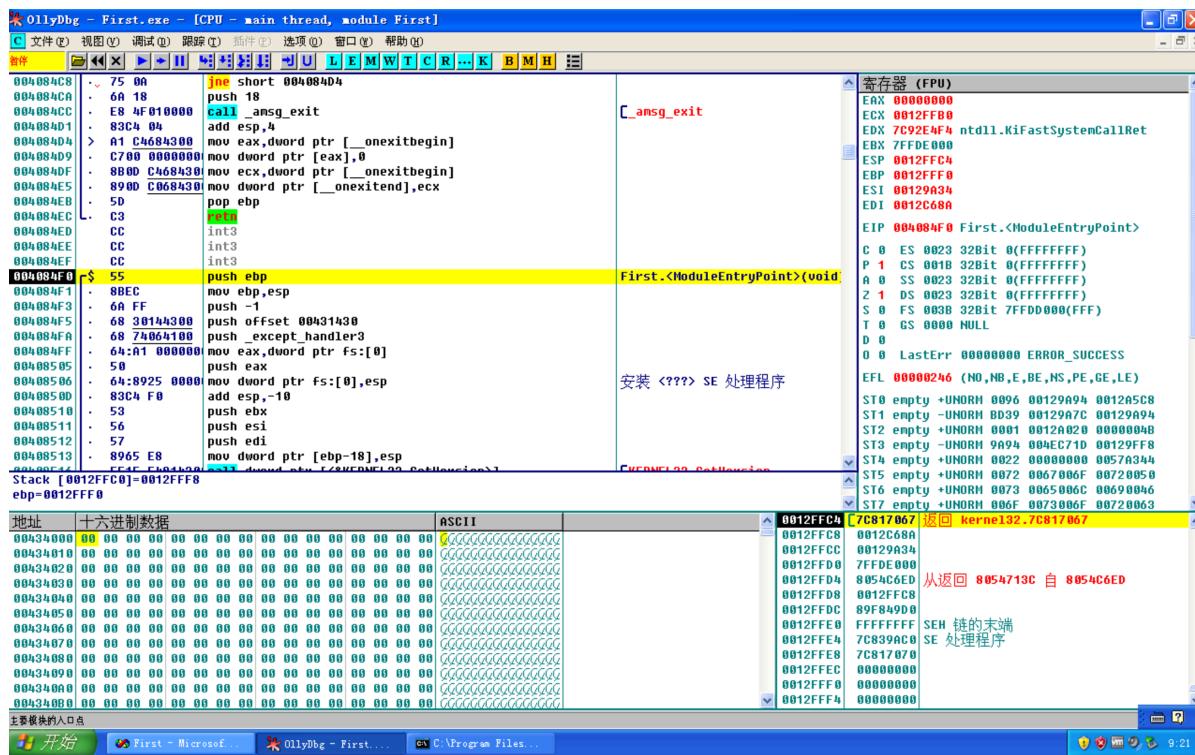
1. 新建project并进行编译处理进入 debug 模式，在文件所在文件夹中打开debug模式中的 .exe 运行文件。



如图所示，当输入不正确的密码时，会输出“wrong password”，并允许再次输入，界面不消失。当正确输入密码时 .exe 文件结束运行，跳出程序。那么我们如何通过修改汇编代码进入核心逻辑呢，我们将使用OLLYDBG来进行实现。

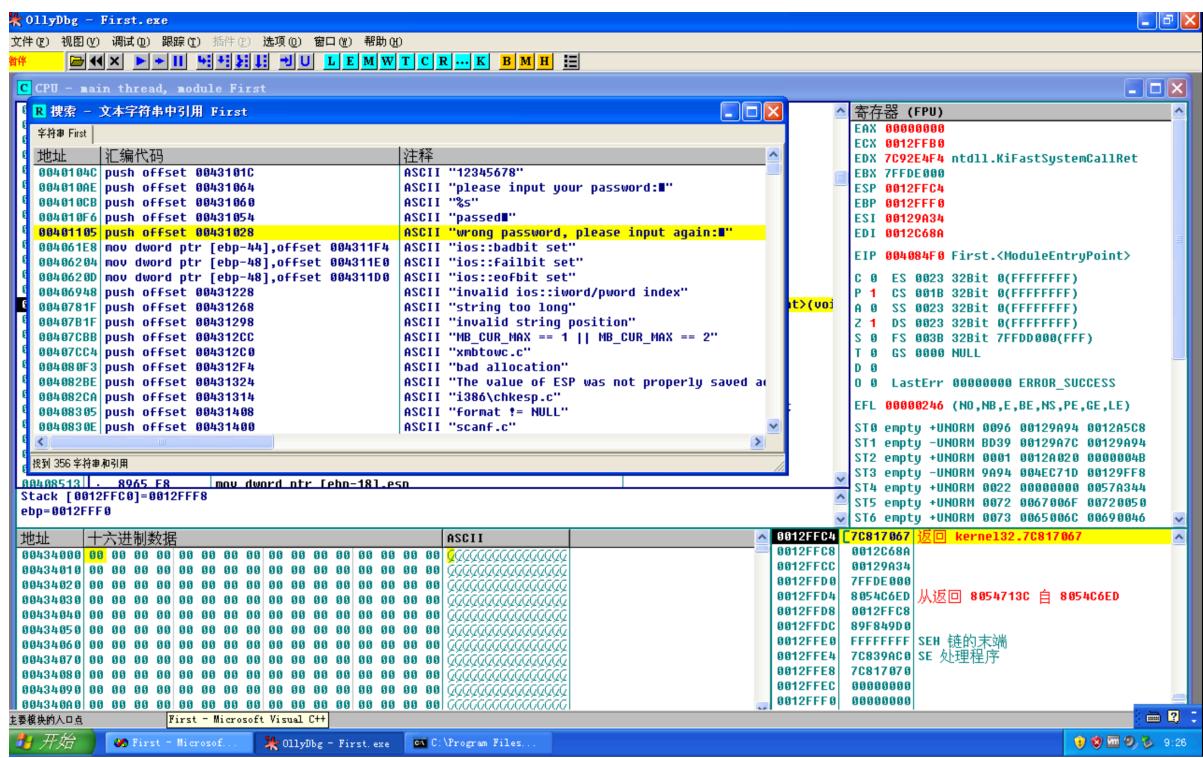


在 windows XP 上打开 ollyDbg，并打开刚才文件夹所在的 debug 文件夹中的 .exe 文件，为接下来的调试工作做准备。

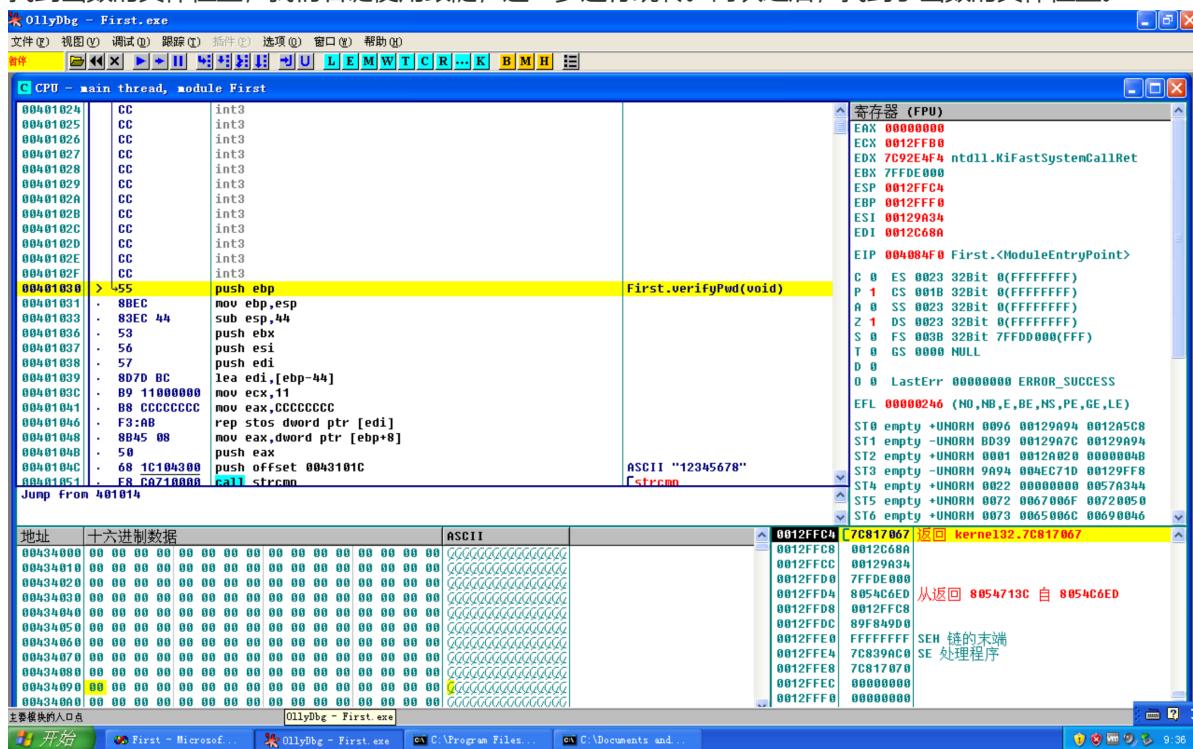


## 二、定位关键语句

如果要破解密码，首先需要定位到执行跳转的关键语句。所以我们先进行关键字的查询，找到了“wrong password”所在的汇编行，并进行双击进入主程序中该行代码所在的具体位置。



由红色下箭头我们可以知晓，该条件进行判断的语句是由 `jz short 00401114` 汇编语句所决定的。为了找到函数的具体位置，我们右键使用跟随，进一步进行跳转。两次之后，找到了函数的具体位置。



代码如下所示：

```
mov eax,dword ptr [ebp+8]
push eax
push offset 0043101C ;                                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
```

具体解释：

`mov eax, dword ptr [ebp+8]`

将 `[ebp+8]` 地址处的值（通常是一个函数参数）加载到寄存器 `eax` 中。`[ebp+8]` 通常是函数的第一个参数（在32位调用约定中）。

`push eax`

将 `eax` 的值压入栈中。此时 `eax` 中存储的是第一个字符串的地址。

`push offset 0043101C ;`

将字符串 `"12345678"` 的地址（`0043101C`）压入栈中。这是第二个字符串。

`call strcmp ;`

调用 `strcmp` 函数，比较栈中的两个字符串。`strcmp` 的返回值存储在 `eax` 中：

如果两个字符串相等，返回 `0`。如果第一个字符串小于第二个字符串，返回负数。如果第一个字符串大于第二个字符串，返回正数。

```
add esp,8
```

清理栈空间。因为之前压入了两个参数（每个4字节），所以需要将栈指针 `esp` 增加8。

```
mov dword ptr [ebp-4],eax
```

将 `strcmp` 的返回值（`eax`）存储到 `[ebp-4]` 地址处。这是一个局部变量。

```
xor eax, eax
```

将 `eax` 清零。

```
cmp dword ptr [ebp-4],0
```

比较 `[ebp-4]` 中的值（即 `strcmp` 的返回值）与 `0`。

```
sete al
```

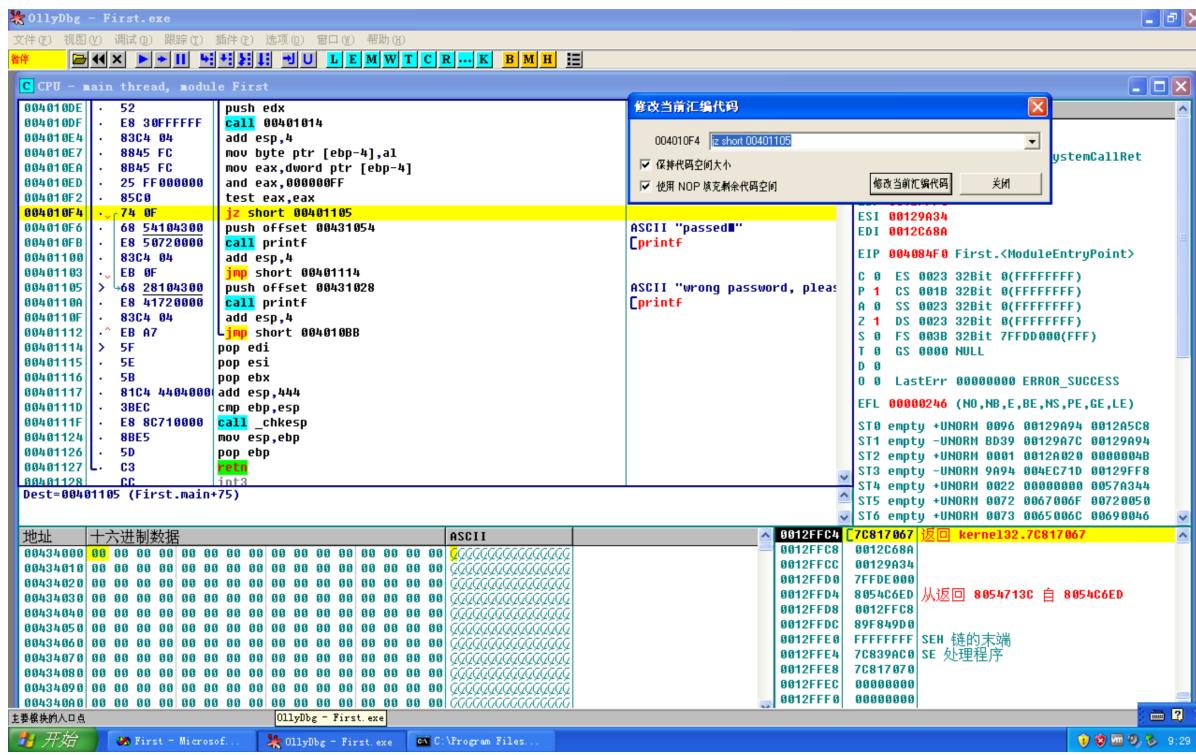
如果 `cmp` 的结果为相等（即 `strcmp` 返回 `0`），则将 `al`（`eax` 的低8位）设置为 `1`，否则设置为 `0`。

## 三、破解方法

### (1) 逆转主函数中的跳转条件：

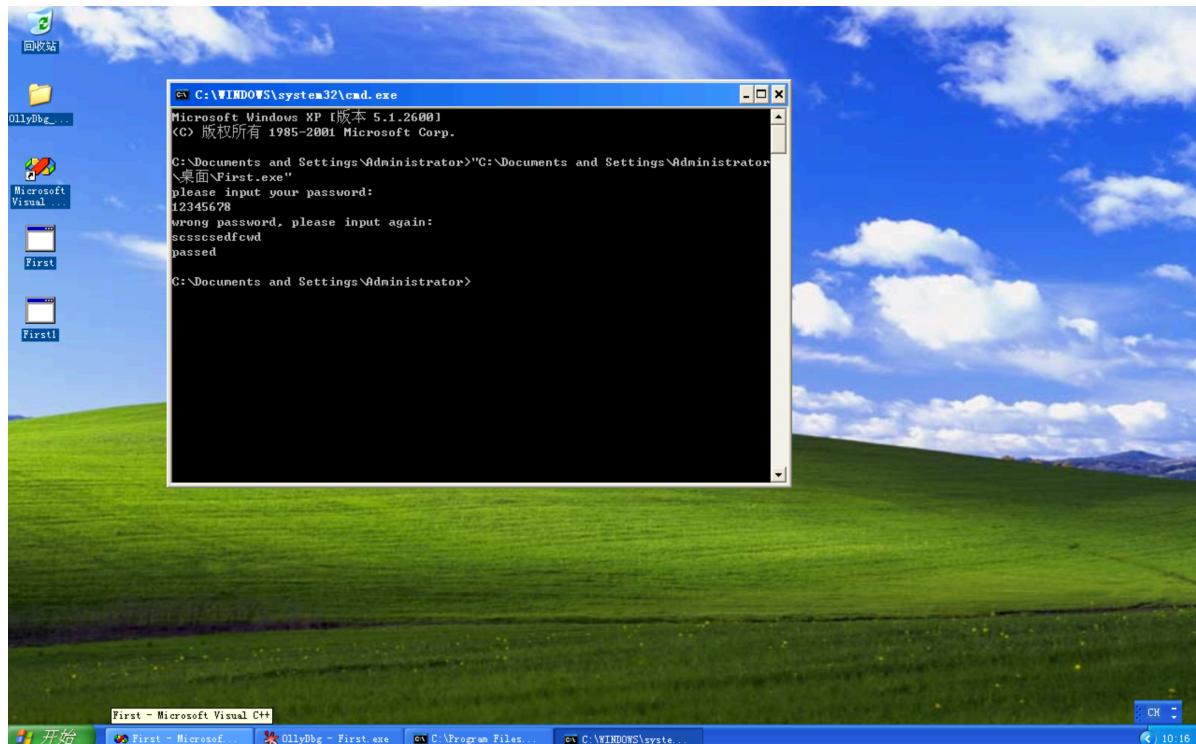
使得错误的代码能够进一步跳转，而正确的代码不能够进入。

从上面的分析我们可以看出，程序的判定部分主要是由 `jz short 00401114` 语句所决定的。



通过观察我们可以发现 00401105 语句是用来执行错误密码的， 004010F6 语句是用来执行正确语句的跳转的。所以程序的逻辑是如果密码正确，则进入 004010F6 语句，并执行 “pass” 语句，否则跳转到 00401105 语句。所以我们可以通过逆转逻辑来实现输入错误密码则进入 pass 语句。即将 jz short 00401114 变为 jnz short 00401114 。这样如果输入错误的密码则可以进入 pass 界面。

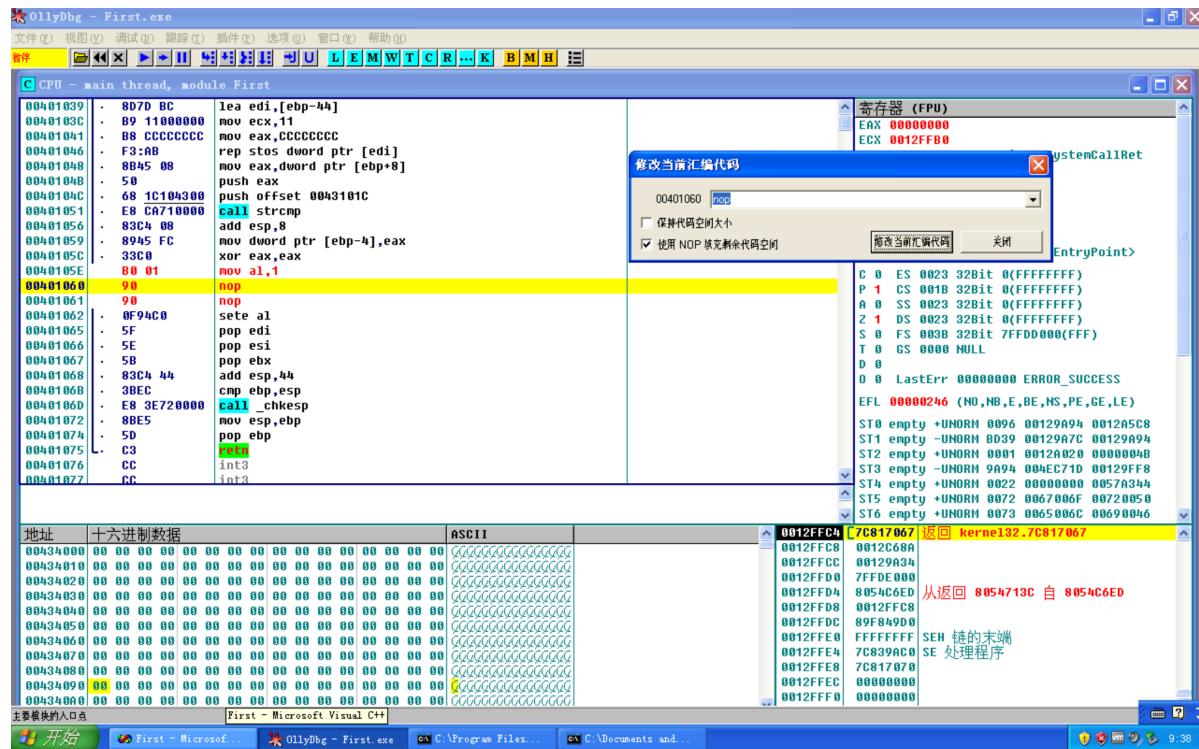
双击 jz short 00401114 ，进行修改。然后右键 编辑 ，选择 复制当前修改到所有可执行文件 ，并保存到桌面。



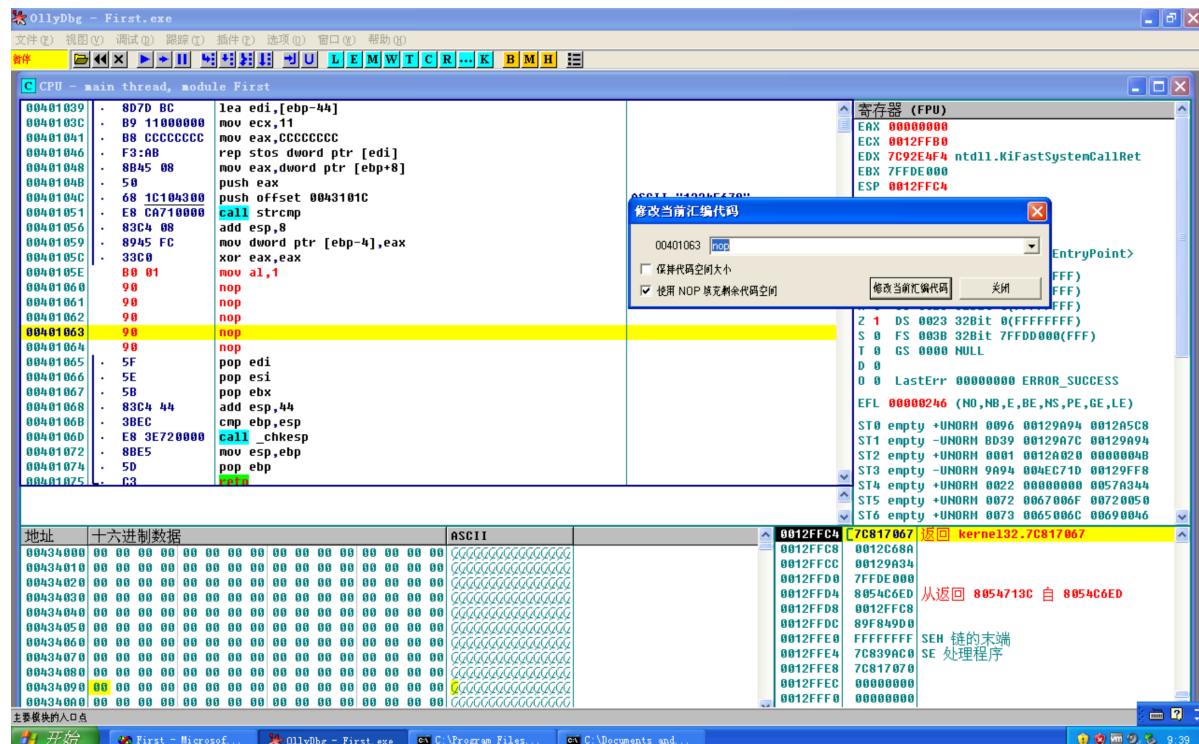
我们可以看到如果我们输入正确的密码则跳转到了 wrong 语句。输入错误的代码则进入 pass 语句，并结束进程。说明我们的修改起到了作用。

### (3) 将判断返回变量的值恒定设置为1

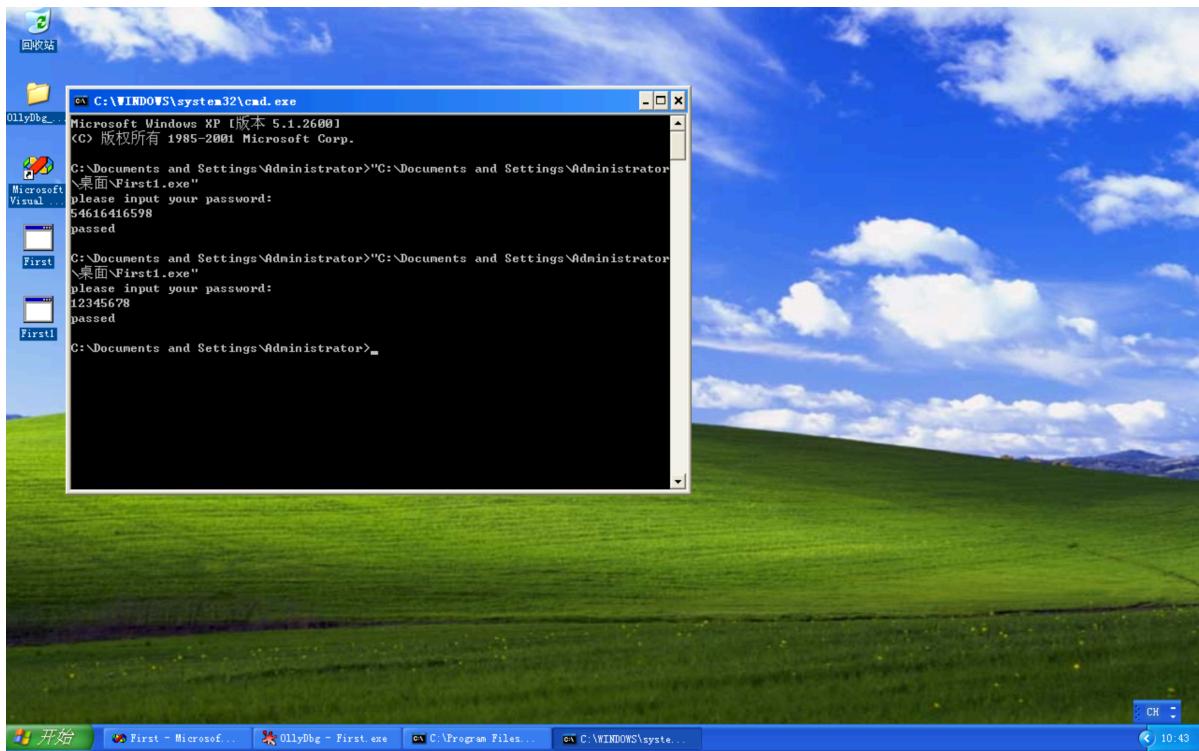
接下来我们尝试第二种方法来进行破解。基于我们在第一部分中讨论的 `flag==0` 的对应区域汇编代码，我们可以想到另一种更有效地破解方法，那就是让 `flag==0` 的返回值永远为1，这样可以避免当输入正确密码而提示密码错误的情况。这样我们就可以实现，不管输入什么密码，都会输出 `passed` 的结果。



我们将 `cmp dword ptr [ebp-4], 0` 语句直接改成 `mov a1, 01` 并且将后续可能影响 `a1` 的值的语句统一改成 `nop`，使得后续操作不再对 `a1` 做出改变。



在命令行中打开 exe 文件，发现不管输入什么都能够进入 pass 语句。



证明我们已经成功破解了程序。

## 心得体会：

通过本次实验，学会了ollydebug的动态调试方法，和一些简单的调试操作。比如说在整个栈内存中通过跟随操作来跟随程序的进程；通过查找关键字来定位关键语句和通过修改关键代码来实现程序的简单破解。

通过实验，我了解到了许多关键核心逻辑语句的寻找和判定。学习了通过修改核心代码使代码错误跳转从而达到目的，对汇编语言进行了更加深刻的认识。通过本次实验，我深刻体会到逆向工程在软件安全分析中的重要性。通过调试工具 OllyDBG，可以直观地看到程序的执行流程和关键逻辑，从而找到破解的突破口。汇编代码是理解程序底层逻辑的关键。通过对 `verifyPWD` 函数的汇编代码分析，我学会了如何跟踪函数的执行流程，理解条件判断和跳转指令的作用。通过修改 `JNZ` 指令和 `strcmp` 返回值，我成功实现了对程序的破解。这让我认识到，软件的安全性不仅依赖于高级语言的逻辑，还需要考虑底层的实现细节。本次实验让我意识到，简单的字符串比较逻辑很容易被破解。在实际开发中，需要采用更复杂的保护机制（如加密、混淆等）来提高软件的安全性。通过使用 OllyDBG，我掌握了基本的调试技巧，如单步执行、断点设置、寄存器查看和内存修改等。这些技能对后续的软件分析和漏洞挖掘非常有帮助。

总之，本次实验让我对软件破解和逆向工程有了更深入的理解，同时也让我认识到软件安全的重要性。在今后的学习和工作中，我会更加注重代码的安全性和抗逆向能力。