《漏洞利用及渗透测试基础》实验报告

姓名：赵悦蛟 学号：2313650 班级： 1071

**实验名称：**

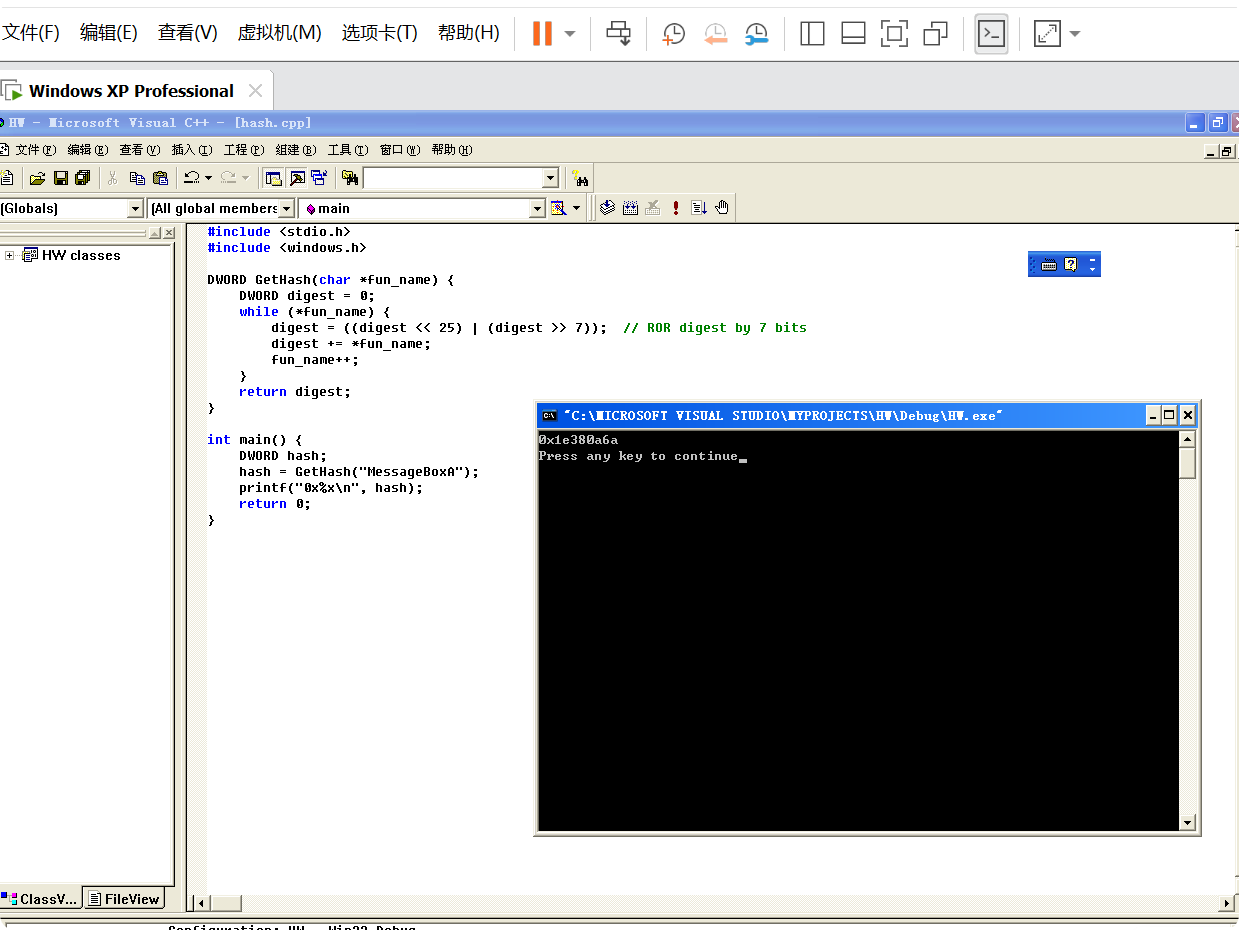
API 函数自搜索实验

**实验要求：**

复现第五章实验七，基于示例5-11，完成API函数自搜索的实验，将生成的exe程序，复制到windows 10操作系统里验证是否成功。

**实验过程：**

1. 编写压缩函数名的hash算法

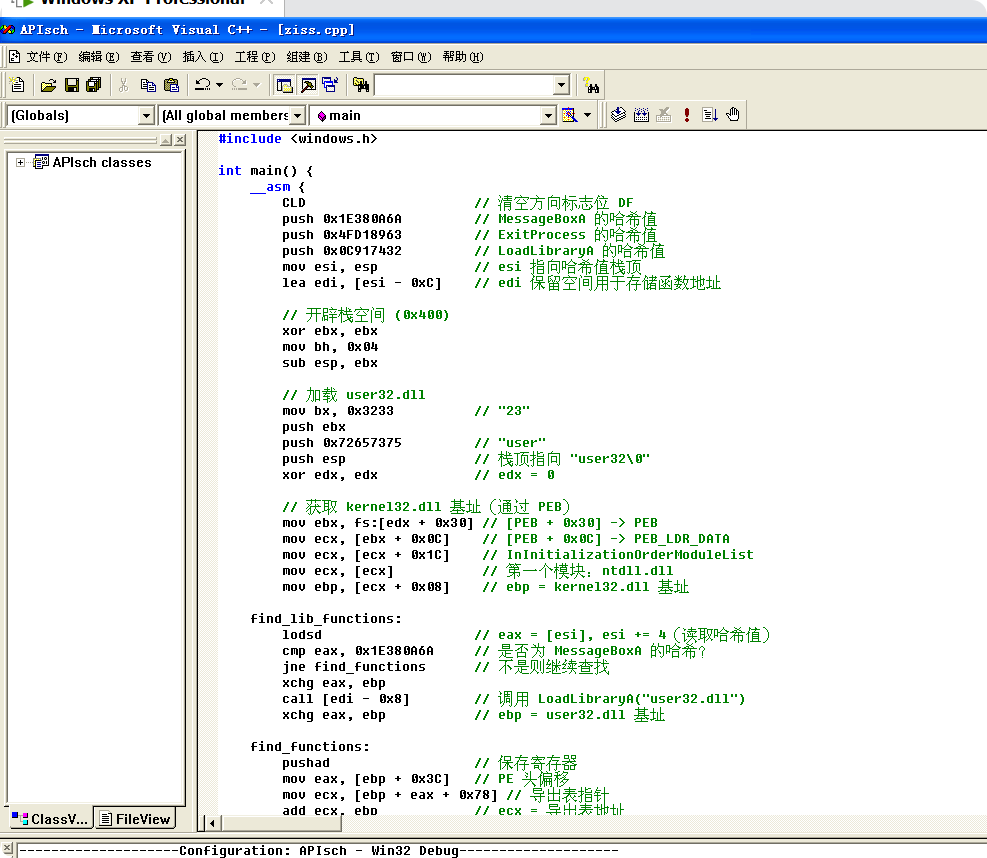


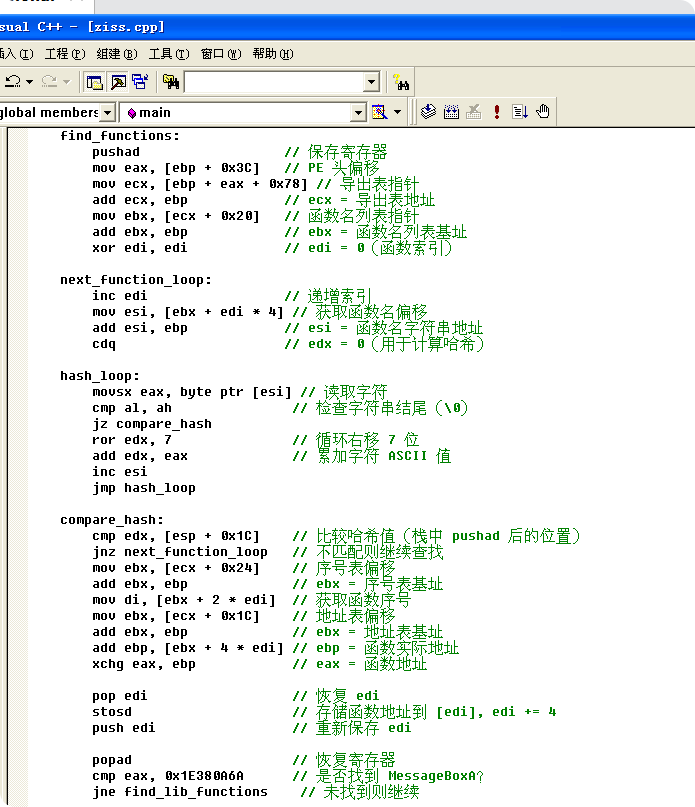
这个函数可以计算字符串的函数值，在本次实验中用于获取API函数名。

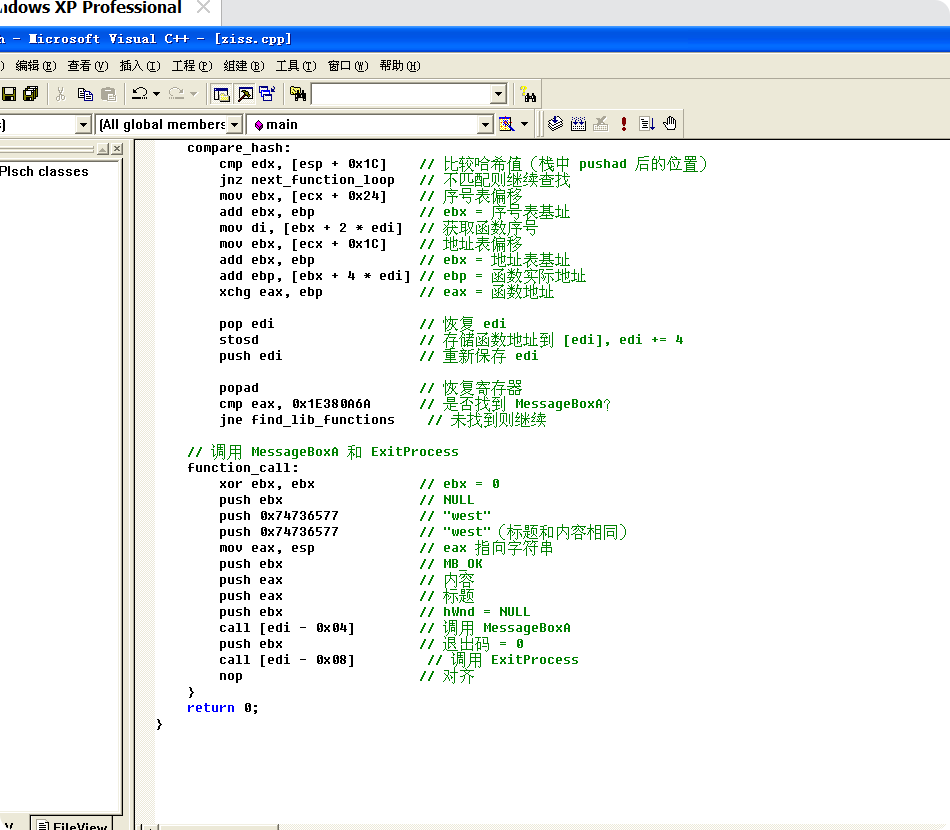
输入函数名，输出32位无符号整数表示计算出的哈希值。其中while循环语句将32位哈希值循环右移七位并将当前字符的ASCLL值加到digest中，确保函数名的每个字符都会影响到哈希值，遍历字符串直到遇到字符串终止字符\0时停止循环。通过这样的操作从而混淆哈希值避免线性相关性。

运行代码得到哈希值：0×1e380a6a

1. 编写API函数自搜索代码







这段代码是一个 动态解析并调用 Windows API 的函数自搜索 Shellcode，主要用于在不知道函数地址的情况下，通过哈希值动态定位并调用关键 API。

push 0x1E380A6A ; MessageBoxA 的哈希值

push 0x4FD18963 ; ExitProcess 的哈希值

push 0x0C917432 ; LoadLibraryA 的哈希值

mov esi, esp ; esi 指向哈希值栈顶

这些语句将需要调用的API哈希值压栈，esi指向栈顶，后续通过lodsd依次读取。

接下来获取kernel32.dll 的基址，加载 user32.dll，用find\_functions中的语句遍历导出表查找函数。

hash\_loop:

movsx eax, byte ptr [esi] ; 读取字符

ror edx, 7 ; 循环右移 7 位

add edx, eax ; 累加字符 ASCII 值

inc esi

cmp al, ah ; 检查字符串结尾（\0）

jnz hash\_loop

然后计算函数名的哈希值，采用对每个字符循环右移 7 位后累加的算法，生成唯一哈希值。

compare\_hash:

cmp edx, [esp + 0x1C] ; 比较目标哈希

jnz next\_function\_loop ; 不匹配则继续

mov ebx, [ecx + 0x24] ; 序号表偏移

mov di, [ebx + 2 \* edi] ; 获取函数序号

mov ebx, [ecx + 0x1C] ; 地址表偏移

add ebp, [ebx + 4 \* edi] ; ebp = 函数实际地址

接下来匹配哈希并获取函数地址，通过序号表找到函数序号，通过地址表获取函数实际地址。

最后调用MessageBoxA 和 ExitProcess。

3.运行代码，得到的结果如图：

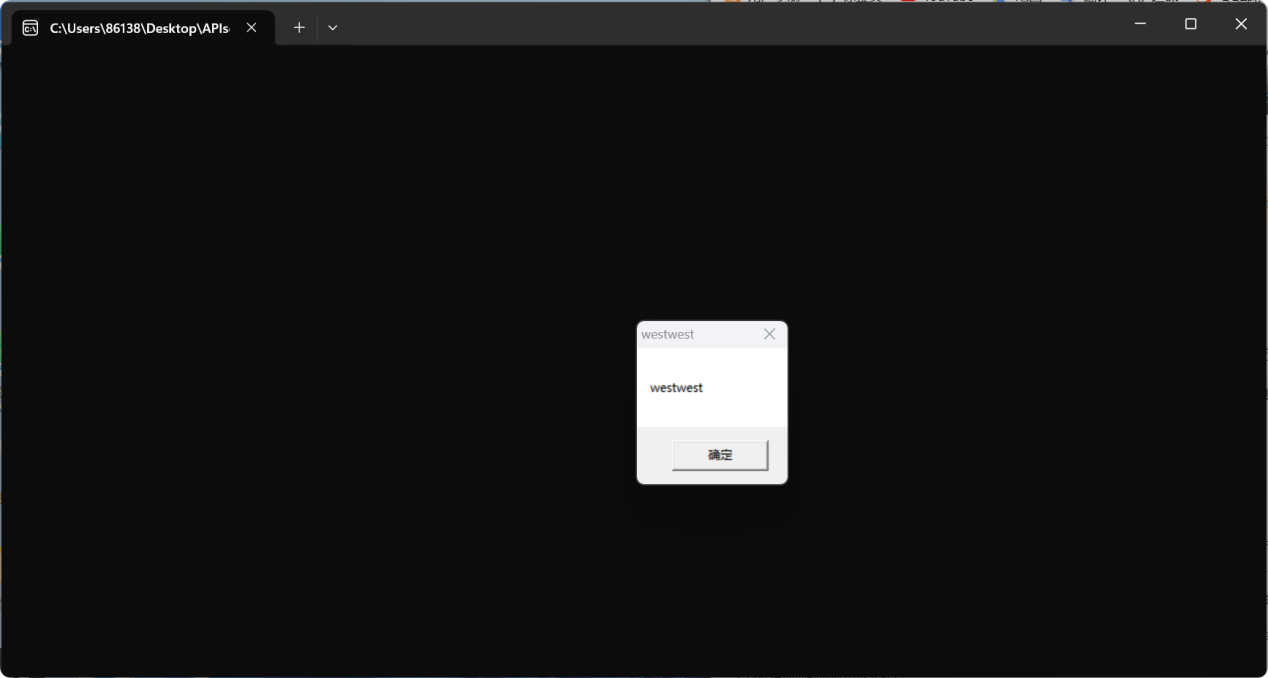
弹出窗口显示westwest



4.将生成的exe程序，复制到windows 10操作系统里进行验证

（由于我的电脑版本是Windows 11，所以就在Windows 11中进行验证）

运行exe文件时弹出：



成功显示一个消息框，内容为

"westwest"，标题也为"westwest"。在关闭消息框后，程序使用 ExitProcess 函数正常退出。

这证明了 API 函数自搜索功能在 Windows 11 操作系统上成功执行。

**心得体会：**

通过本次实验，我对Windows系统底层运行机制有了更加深刻的认识。在实验过程中，我重点研究了PEB（进程环境块）这一关键数据结构，它作为进程信息的核心载体，记录了包括模块列表在内的诸多重要信息。通过实际操作，我掌握了如何利用PEB结构遍历已加载模块，并准确定位关键DLL的基址，这一过程让我对Windows系统的模块管理机制有了直观理解。

在技术实践层面，我深入探索了动态解析API函数的实现方法。与传统调用方式不同，该方法无需依赖导入表，而是通过特定算法实现对目标函数的自主定位与调用。这种技术不仅适用于需要动态加载库的复杂场景，也为分析各类安全威胁提供了新的视角。在研究过程中，我特别注意到了该技术在安全领域的双面性：既可作为开发者的实用工具，也可能被恶意代码利用。

实验中的汇编语言实践环节让我获益匪浅。通过编写和调试内联汇编代码，我对寄存器操作、堆栈管理等底层概念有了更扎实的掌握。