**《软件安全》实验报告**

姓名：胡博浩 学号：2212998 班级：信息安全

**一、实验名称：**

API 函数自搜索实验

**二、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。

**三、实验过程：**

（一）编写代码

在实际中为了编写通用shellcode，shellcode自身就必须具备动态的自动搜索所需 API 函数地址的能力，即 API 函数自搜索技术。

首先，总结一下我们将要用到的函数：

1、MessageBoxA 位于 user32.dll 中，用于弹出消息框。

2、ExitProcess 位于 kernel32.dll 中，用于正常退出程序。所有的 Win32 程序都会自动加载 ntdll.dll 以及 kernel32.dll 这两个最基础的动态链接库。

3、LoadLibraryA 位于 kernel32.dll 中，并不是所有的程序都会装载 user32.dll，所 以在调用 MessageBoxA 之前，应该先使用 LoadLibrary(“user32.dll”)装载 user32.dll

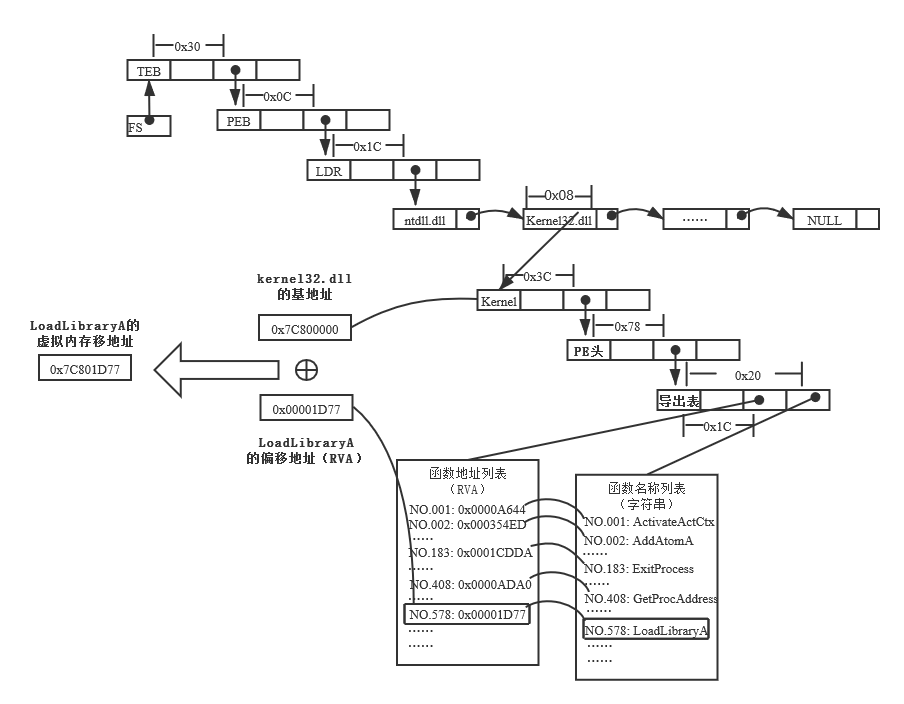
进而，按照以下步骤编写通用型shellcode：

（1）第一步：定位kernel32.dll。

（2）第二步：定位kernel32.dll的导出表。

（3）第三步：搜索定位LoadLibrary等目标函数。

（4）第四步：基于找到的函数地址，完成Shellcode的编写。



同时，一般情况下并不会“MessageBoxA”等这么长的字符串去进行直接比较。所以会对所需的 API 函数名进行 hash 运算，这样只要比较 hash 所得的摘要就能判定是不是我们所需的 API 了。我们使用如下代码计算 API 的 hash 值：

#include <stdio.h>

#include <windows.h>

DWORD GetHash(char \*fun\_name){

DWORD digest=0;

while(\*fun\_name){

digest=((digest<<25)|(digest>>7)); //循环右移7位

/\* movsx eax,byte ptr[esi]

cmp al,ah

jz compare\_hash

ror edx, 7 ; ((循环))右移,不是单纯的 >>7

add edx,eax

inc esi

jmp hash\_loop

\*/

digest+= \*fun\_name ; //累加

fun\_name++;}

return digest;}

main(){

DWORD hash;

hash= GetHash("MessageBoxA");

printf("%#x\n",hash);}

**综上所述，完整 API 函数自搜索代码如下图所示：**

#include <stdio.h>

#include <windows.h>

int main(){

\_\_asm{

CLD //清空标志位DF

push 0x1E380A6A //压入MessageBoxA的hash-->user32.dll

push 0x4FD18963 //压入ExitProcess的hash-->kernel32.dll

push 0x0C917432 //压入LoadLibraryA的hash-->kernel32.dll

mov esi,esp //esi=esp,指向堆栈中存放LoadLibraryA的hash的地址

lea edi,[esi-0xc] //空出8字节应该是为了兼容性

//======开辟一些栈空间

xor ebx,ebx

mov bh,0x04

sub esp,ebx //esp-=0x400

//======压入"user32.dll"

mov bx,0x3233

push ebx //0x3233

push 0x72657375 //"user"

push esp

xor edx,edx //edx=0

//======找kernel32.dll的基地址

mov ebx,fs:[edx+0x30] //[TEB+0x30]-->PEB

mov ecx,[ebx+0xC] //[PEB+0xC]--->PEB\_LDR\_DATA

mov ecx,[ecx+0x1C] //[PEB\_LDR\_DATA+0x1C]--->InInitializationOrderModuleList

mov ecx,[ecx] //进入链表第一个就是ntdll.dll

mov ebp,[ecx+0x8] //ebp= kernel32.dll的基地址

//======是否找到了自己所需全部的函数

find\_lib\_functions:

lodsd //即move eax,[esi], esi+=4, 第一次取LoadLibraryA的hash

cmp eax,0x1E380A6A //与MessageBoxA的hash比较

jne find\_functions //如果没有找到MessageBoxA函数，继续找

xchg eax,ebp //------------------------------------> |

call [edi-0x8] //LoadLibraryA("user32") |

xchg eax,ebp //ebp=userl32.dll的基地址,eax=MessageBoxA的hash <-- |

//======导出函数名列表指针

find\_functions:

pushad //保护寄存器

mov eax,[ebp+0x3C] //dll的PE头

mov ecx,[ebp+eax+0x78] //导出表的指针

add ecx,ebp //ecx=导出表的基地址

mov ebx,[ecx+0x20] //导出函数名列表指针

add ebx,ebp //ebx=导出函数名列表指针的基地址

xor edi,edi

//======找下一个函数名

next\_function\_loop:

inc edi

mov esi,[ebx+edi\*4] //从列表数组中读取

add esi,ebp //esi = 函数名称所在地址

cdq //edx = 0

//======函数名的hash运算

hash\_loop:

movsx eax,byte ptr[esi]

cmp al,ah //字符串结尾就跳出当前函数

jz compare\_hash

ror edx,7

add edx,eax

inc esi

jmp hash\_loop

//======比较找到的当前函数的hash是否是自己想找的

compare\_hash:

cmp edx,[esp+0x1C] //lods pushad后,栈+1c为LoadLibraryA的hash

jnz next\_function\_loop

mov ebx,[ecx+0x24] //ebx = 顺序表的相对偏移量

add ebx,ebp //顺序表的基地址

mov di,[ebx+2\*edi] //匹配函数的序号

mov ebx,[ecx+0x1C] //地址表的相对偏移量

add ebx,ebp //地址表的基地址

add ebp,[ebx+4\*edi] //函数的基地址

xchg eax,ebp //eax<==>ebp 交换

pop edi

stosd //把找到的函数保存到edi的位置

push edi

popad

cmp eax,0x1e380a6a //找到最后一个函数MessageBox后，跳出循环

jne find\_lib\_functions

//======让他做些自己想做的事

function\_call:

xor ebx,ebx

push ebx

push 0x74736577

push 0x74736577 //push "westwest"

mov eax,esp

push ebx

push eax

push eax

push ebx

call [edi-0x04] //MessageBoxA(NULL,"westwest","westwest",NULL)

push ebx

call [edi-0x08] //ExitProcess(0);

nop

nop

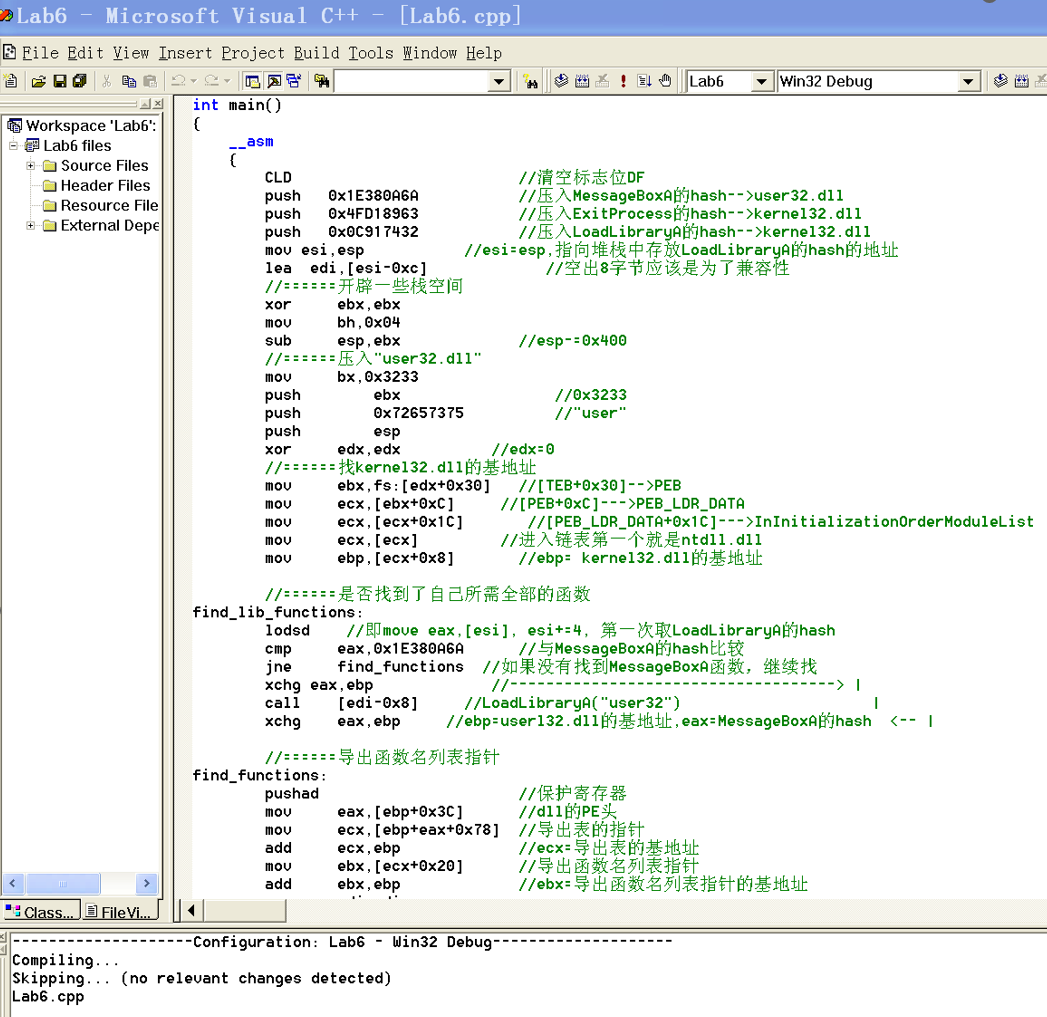
nop

nop}

return 0;}

（二）在XP系统下验证

先在XP系统下，使用VC6对代码文件进行编译：

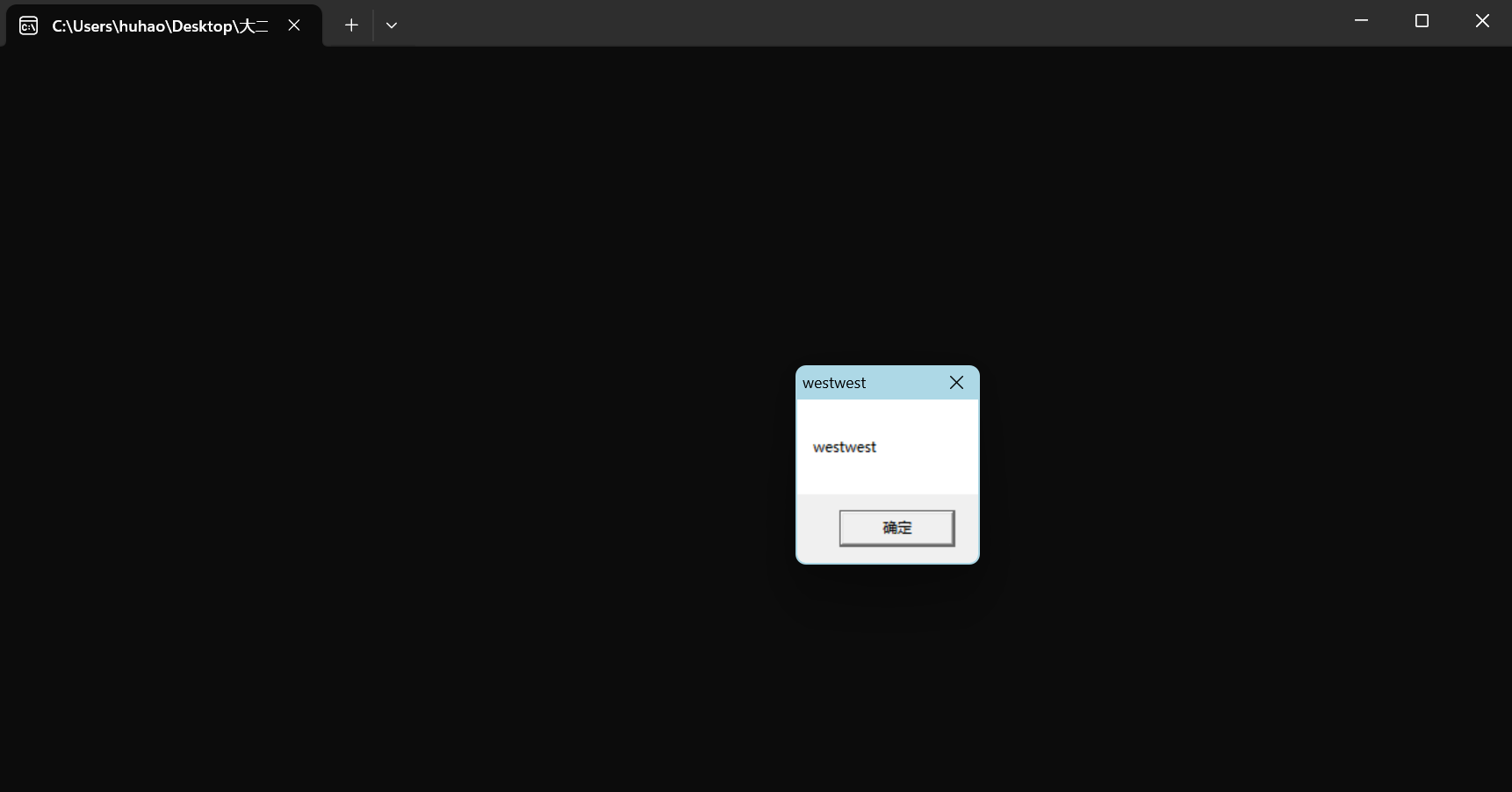


运行程序，结果如下，表明实验成功！



（三）在win10系统下验证

将exe程序复制到win10系统中，运行，结果如下，表明实验成功！



**四、心得体会：**

在本次实验中，我深刻领悟到了Shellcode编写中采用硬编址方式的局限性，特别是在面对环境变化时可能导致Shellcode失效的问题。

通过学习API函数自搜索技术，我了解到了如何动态地定位函数API，使得Shellcode具备了更强的适应性和稳定性。

结合之前对PE文件结构的了解，我对API函数自搜索技术有了更深入的理解，并能够灵活地应用于实际编程中。

这种经验不仅提升了我的技术水平，也加强了我对系统底层机制的理解，为我今后在安全领域的学习和工作打下了坚实的基础。我相信这些知识和技能将在未来的工作中发挥重要作用，让我能够更好地应对不同环境下的挑战。