《软件安全》实验报告

姓名： 冯思程 学号： 2112213 班级： 1120（张健）

**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. 获取到目标API函数，

**i定位kernel32.dll**

方法如下：

（1）首先通过段选择字FS在内存中找到当前的线程环境块TEB。

（2）线程环境块偏移地址为0x30的地址存放着指向进程环境块PEB的指针。

（3）进程环境块中偏移地址为0x0c的地方存放着指向PEB\_LDR\_DATA结构体的指针，其中，存放着已经被进程装载的动态链接库的信息。

（4）PEB\_LDR\_DATA结构体偏移位置为0x1C的地址存放着指向模块初始化链表的头指针InInitializationOrderModuleList。

（5）模块初始化链表InInitializationOrderModuleList中按顺序存放着PE装入运行时初始化模块的信息，第一个链表结点是ntdll.dll，第二个链表结点就是kernel32.dll。

（6）找到属于kernel32.dll的结点后，在其基础上再偏移0x08就是kernel32.dll在内存中的加载基地址。

上述复杂的操作可以用如下简单的代码来实现：

int main()

{

\_asm

{

mov eax, fs:[0x30] ;PEB的地址

mov eax, [eax + 0x0c] ; PEB\_LDR\_DATA结构体的地址

mov esi, [eax + 0x1c] ; 指针InInitializationOrderModuleList

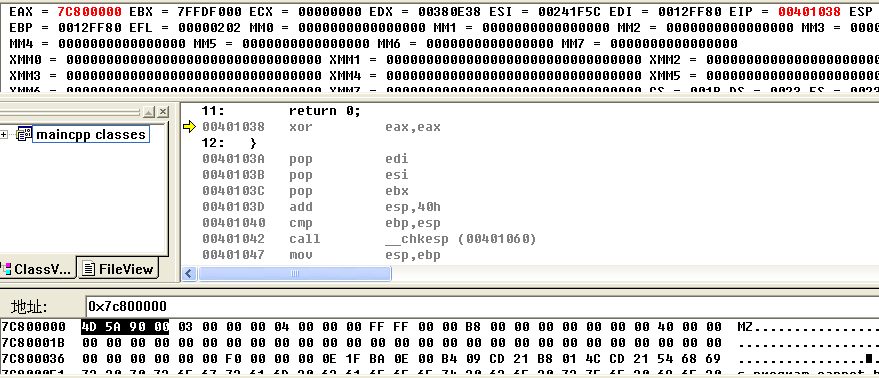
lodsd

mov eax, [eax + 0x08] ;eax就是kernel32.dll的地址

}

return 0;

}



加载上面的代码，在\_asm部分的第一句设置断点，然后F5进入反汇编模式，步过运行到上图中的位置，查看eax寄存器中的值是0x00905a4d，即是kernel32.dll在内存中的加载基地址。

**ii定位kernel32.dll的导出表**

找到了kernel32.dll，由于它也是属于PE文件，那么我们可以根据PE文件的结构特征，定位其导出表，进而定位导出函数列表信息，然后进行解析、遍历搜索，找到我们所需要的API函数。定位导出表及函数名列表的步骤如下：

（1）从kernel32.dll加载基址算起，偏移0x3c的地方就是其PE头的指针。

* 1. PE头偏移0x78的地方存放着指向函数导出表的指针。

（3）获得导出函数偏移地址（RVA）列表、导出函数名列表：

①导出表偏移0x1c处的指针指向存储导出函数偏移地址（RVA）的列表。

②导出表偏移0x20处的指针指向存储导出函数函数名的列表。

定位kernel32.dll导出表及其导出函数名列表的代码如下：

mov ebp, eax //将kernel32.dll基地址赋值给ebp

mov eax,[ebp+0x3C] //dll的PE头的指针（相对地址）

mov ecx,[ebp+eax+0x78] //导出表的指针（相对地址）

add ecx,ebp //ecx=0x78C00000+0x262c 得到导出表的内存地址

mov ebx,[ecx+0x20] //导出函数名列表指针

add ebx,ebp //导出函数名列表指针的基地址

**iii搜索定位目标函数**

至此，可以通过遍历两个函数相关列表，算出所需函数的入口地址：

（1）函数的RVA地址和名字按照顺序存放在上述两个列表中，我们可以在名称列表中定位到所需的函数是第几个，然后在地址列表中找到对应的RVA。

（2）获得RVA后，再加上前边已经得到的动态链接库的加载地址，就获得了所需API此刻在内存中的虚拟地址，这个地址就是最终在ShellCode中调用时需要的地址。

按照这个方法，就可以获得kernel32.dll中的任意函数。

**iv完整API函数自搜索代码**

为了让shellcode更加通用，能被大多数缓冲区容纳，总是希望shellcode尽可能短。因此，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。代码如下：

#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

digest+= \*fun\_name ; //累加

fun\_name++;

}

return digest;

}

main()

{

DWORD hash;

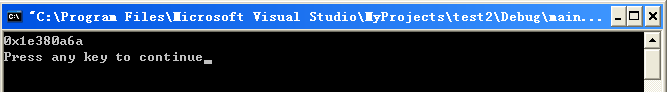
hash= GetHash("MessageBoxA");

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

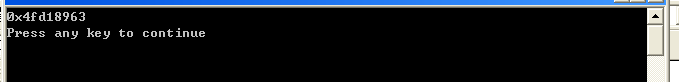
}

利用这个代码分别去获取到本实验所需要的API函数的hash值。截图如下：

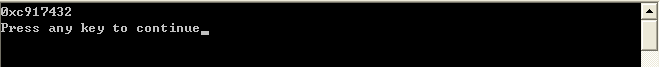
MessageBoxA函数：



ExitProcess函数：



LoadLibraryA函数：



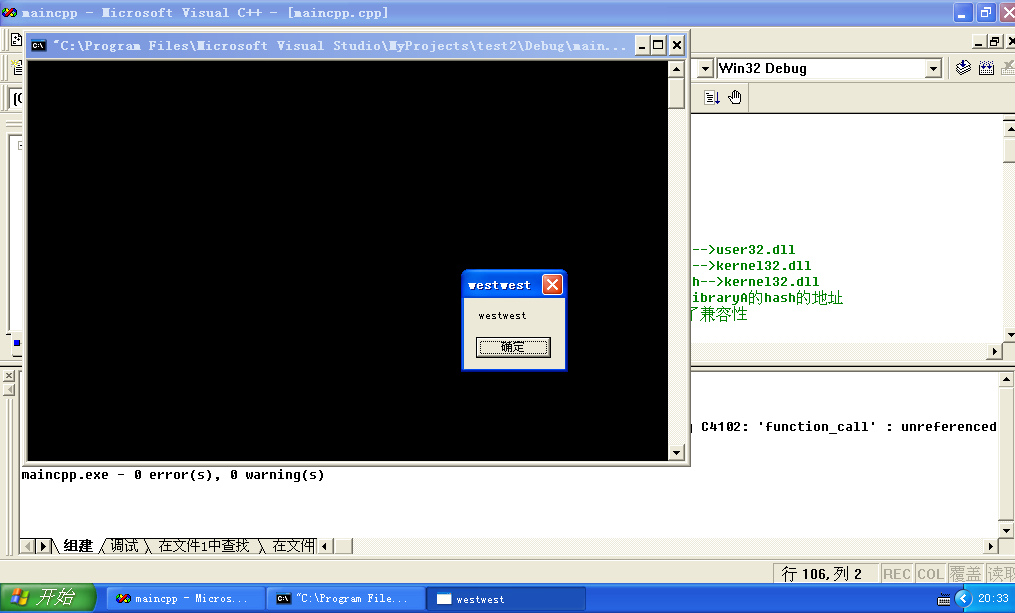
1. 根据示例5-11的完整API函数自搜索代码（代码过于冗长由于篇幅限制这里不直接给出），在xp系统下的VC6中运行，截图如下：（窗口的文字设置成westwest（嘻嘻））

给出小部分利用上文获取到的所需函数的hash值的代码：

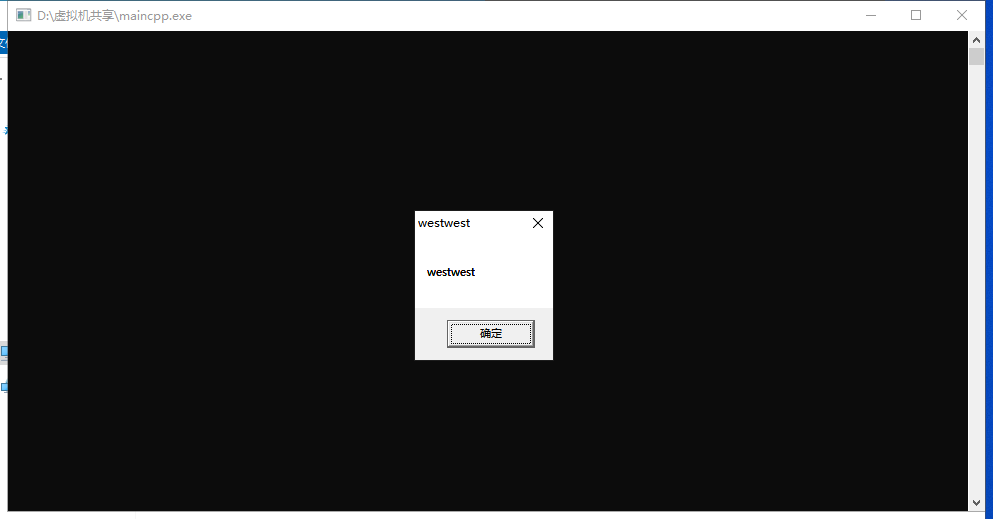
push 0x1E380A6A //压入 MessageBoxA 的 hash-->user32.dll

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

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



1. 将编译生成的exe程序应用到本机windows10操作系统下，结果如下：



可以成功的在windows10操作系统下进行调用窗口函数输出。成功实现了目标。这与我们实现通用型shellcode代码的目标匹配，即在任何操作系统下，都可以成功实现预设计出来的窗口弹出。

**心得体会：**

在软件安全领域，API 函数自搜索是一项非常重要的技能。通过 API 函数自搜索，可以帮助我们分析恶意代码、发现漏洞和进行软件逆向工程。这个实验让我了解了 API 函数自搜索的原理和方法，并且让我能够自己实现一个简单的 API 函数自搜索工具。

在实验过程中，我遇到了一些困难。最主要的困难是如何定位和读取 DLL 文件中的导出表。为了解决这个问题，我查阅了教材，并且阅读了一些现有的工具的实现代码。最终，我成功地实现了一个简单的 API 函数自搜索工具（本实验中是以弹出窗口为例），并且能够使用它来分析目标二进制文件中的 API 函数。

这个实验让我深入了解了软件安全中的 API 函数自搜索技术，并且让我学习到了如何通过实践来加深自己的理解和掌握。同时，这个实验也让我认识到软件安全是一个非常广泛而又深奥的领域，需要不断地学习和实践才能够真正掌握。