《API函数自搜索定位》实验报告

姓名：汤清云 学号：2013536 班级： 1075

**实验名称：**

API函数自搜索定位技术

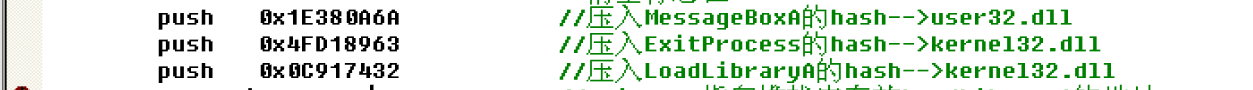
**实验要求：**

1. 定位kernal32.dll：
2. 通过段选择字FS在内存中找到当前的线程环境块TEB。
3. 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。
4. 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。
5. 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。
6. 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。
7. 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。
8. 找到kernal32.dll的导出表：
9. 从kernel32.dll加载基址算起，偏移0x3c的地方就是其PE头的指针。
10. PE头偏移0x78的地方存放着指向函数导出表的指针。
11. 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。

1. 搜索定位目标函数：
2. 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。
3. RVA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。

**实验过程：** 

**将MessageBoxA/ExitProcess/LoadLibraryA函数的字符串转为哈希值后先后压入栈内。在之后作函数名比较时也是使用字符串的哈希值进行比较。**



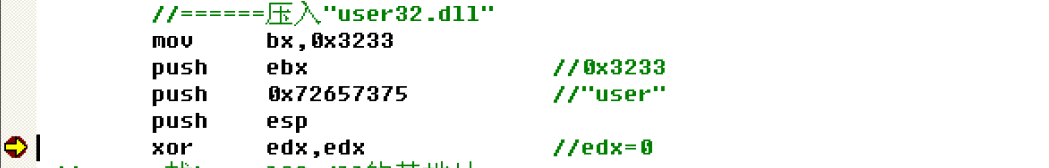
**将此时ESP的值赋值给ESI，使得ESI的值为0012FF28，用以标注三个函数名哈希值存放处。**



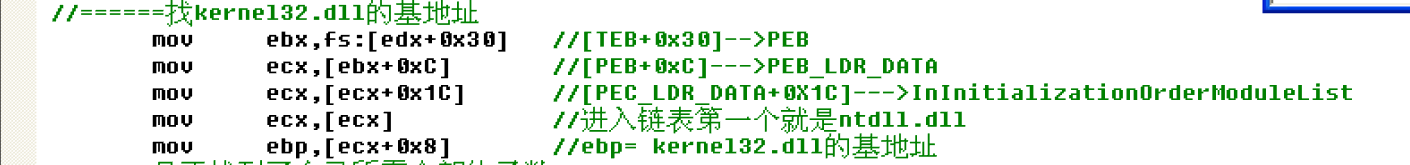
**EDI指向未压入三个函数名字哈希值之前的栈顶位置，即0012FF1C**



**将ebx清零后赋值为00000400（bh为bx的高八位），将栈顶抬高0x400，增加栈空间。**



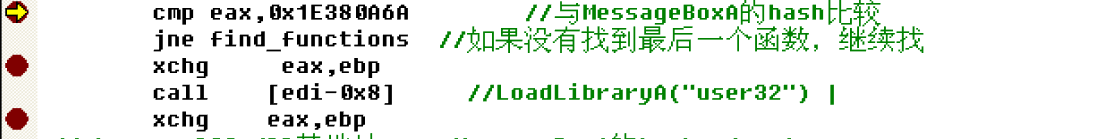
**将‘32’存放在ebx中，bx是ebx的低十六位，故ebx值为00003233，再将“user”的字符串转哈希值压入栈，此时esp处则存下了user.32的哈希值，再将esp压入栈，即将字符串“user32”字符串哈希值地址压入栈中。最后将edx归零。**



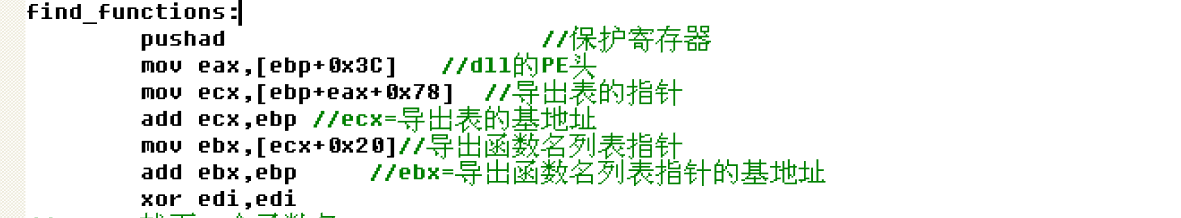
**由实验原理1所得出代码，此时ebp存放了kernel32的基地址，为7C800000**



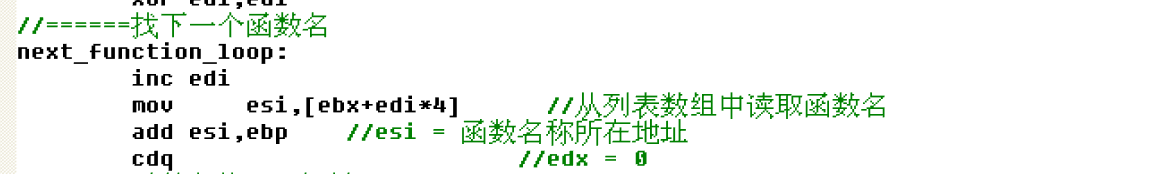
**此时eax值为：0C917432，为LoadLibrary的哈希值，ESI的值变为0012FF2C**



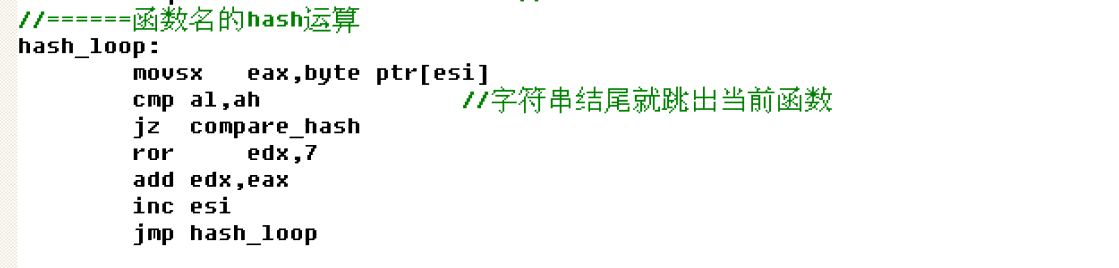
**与MessageBox的哈希值比较，查看是否为最后一个需要找的函数的哈希值，如果不是的话需要继续寻找，跳转到find\_functions函数处完成后续操作。如果是的话则将eax的值与ebp互换，调用LoadLibrary的user32.dll，并返回后再次交换eax与ebp的值，则此时ebp存放了user32的基地址，eax存放了messagebox的哈希值。**



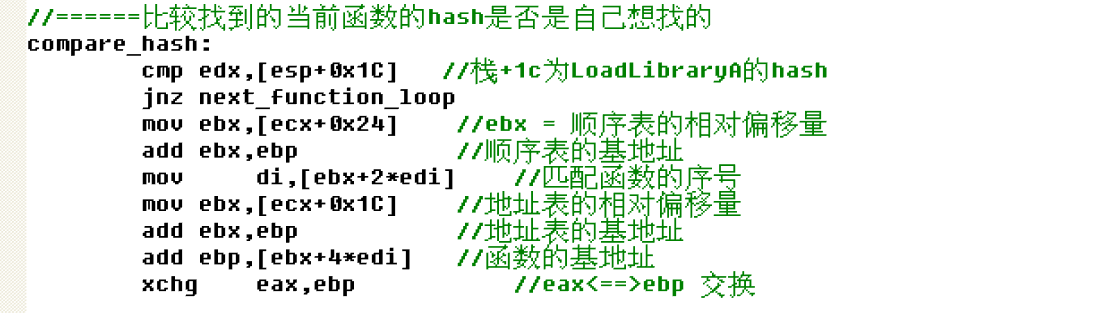
**函数逻辑为：定义到导出表，再定位到导出函数名列表。**



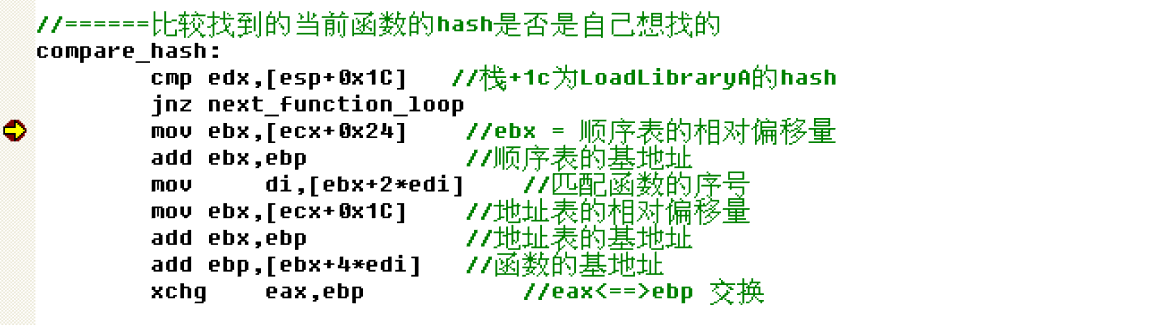
**依次取出下一个未访问过的函数。**



**使用hash\_loop计算出函数名字符串对应的哈希值。**



**使用compare\_hash来进行判别，比较当前找出的函数的哈希值是否是自己想要的，如果不是则跳转回next\_function\_loop寻找下一个函数名，继续进行上述操作，直到匹配成功。**



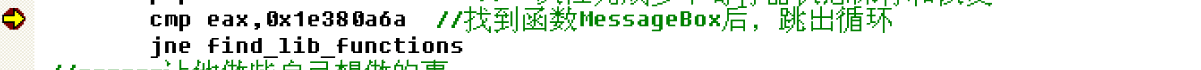
**找到之后同实验原理3，计算出其虚拟地址。**



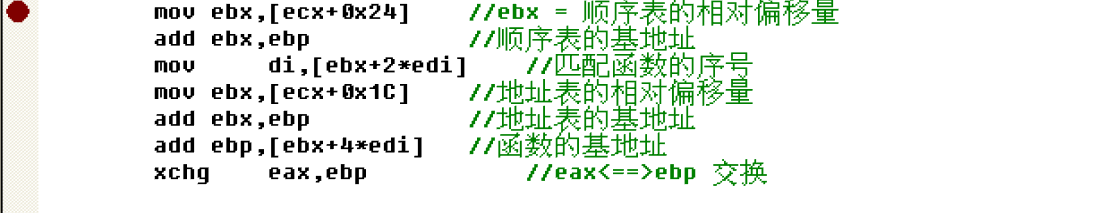
**EDI由00000244变为0012FF1C（存放着刚才找到的虚拟地址），再将其+4后压入栈中。**



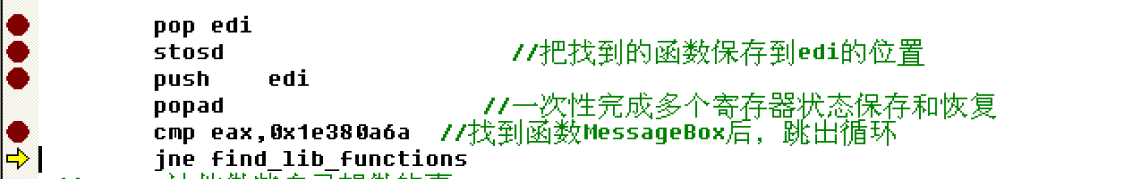
**保存多个寄存器的状态。**



**判断是否为我们需要找的最后一个函数MessageBox的哈希值，若不是则继续寻找，是则跳出循环。**



**找到messagebox函数后，通过以上语句计算出其地址。**



**将计算出的地址放入edi中，再次比较是否为messagebox函数。此时edi存放了三个函数的虚拟地址。**

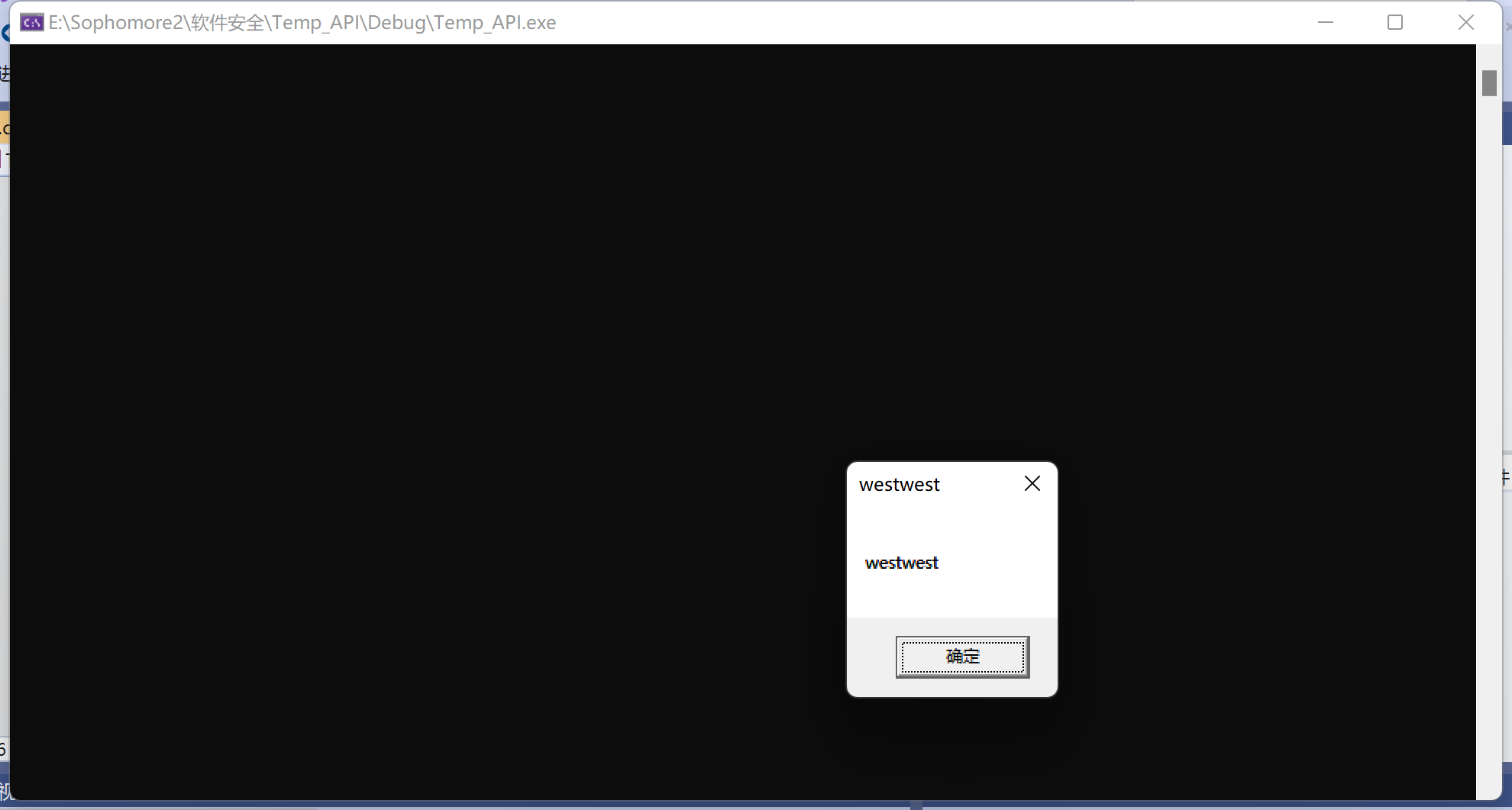


**此处进行shellcode编写，将ebx清零后压入栈中。将“westwest”的ascii码压入栈中，esp（westwest字符串的地址）赋值给eax，压入ebx，eax，eax，ebx，调用messagebox函数，进行调用时会自动调取栈中前四个函数，故对应了null,westwest,westwest,null**



**调用结果见上。之后再次压入ebx（0），调用exitprocess函数，退出程序。**

**在windows11 VS2019版本运行结果为：**



**总结心得：**

**学习了如何进行API函数的自搜索，了解了kernel32以及user32的dll的查询。**