《软件安全》实验报告

姓名：张丛 学号：2113662 班级：信安一班

**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.实验代码及注释**

|  |
| --- |
| #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;  } |

1. **实现API自搜索的关键部分**

（1）定位kerne132.dll

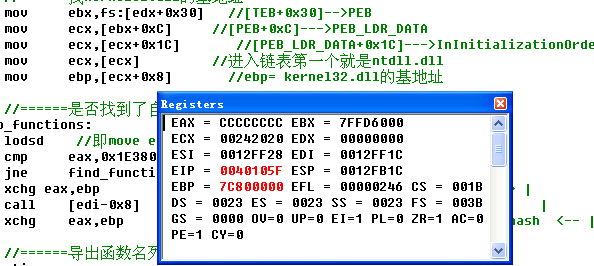
（2）定位kerne132.Dll的导出表

（3）搜索定位LoadLibrary，MessageBoxA,ExitProcess这些目标函数

（4）根据定位的函数地址，完成Shellcode的编写

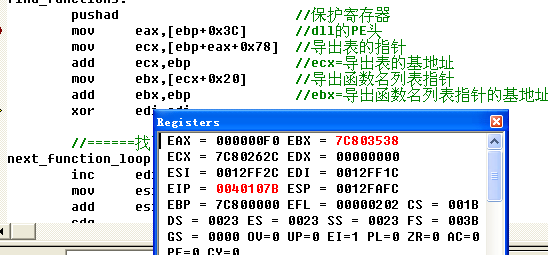
实现（1）定位kernel.32.dll：

1. mov ebx,fs:[0x30] 找到PEB(当前线程环境块）的地址
2. mov ecx,[ebx+0x0c] 找到PEB\_LDR\_DATE结构体的地址
3. mov ecx,[ecx+0x1c] 找到模块初始化链表头指针
4. mov ebp,[ecx+0x08] 找到kernel32.dll的地址



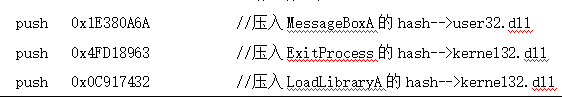
实现（2）定位kernel32.dll的导出表：

1. Mov eax,[ebp+0x3C] 找到dll的PE头指针(ebp储存kernel32.dll的基地址）
2. Mov ecx,[ebp+eax+0x78] 找到导出表的指针
3. Add ecx,eax 计算导出表基地址
4. Mov ebx,[ecx+0x20] 找到函数名列表指针
5. Add ebx,ebp 得到函数名列表指针的基地址



在实现（3）搜索目标函数时，会对目标函数名进行hash运算

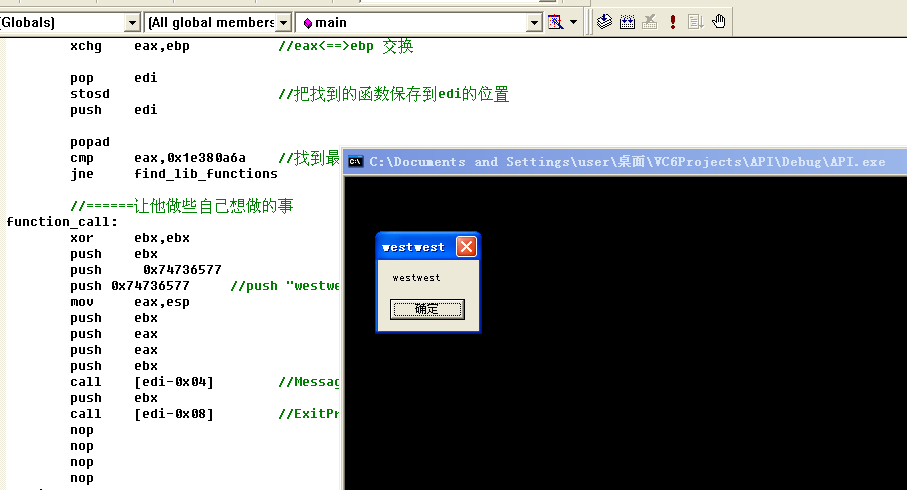
而三个函数名对应的hash:



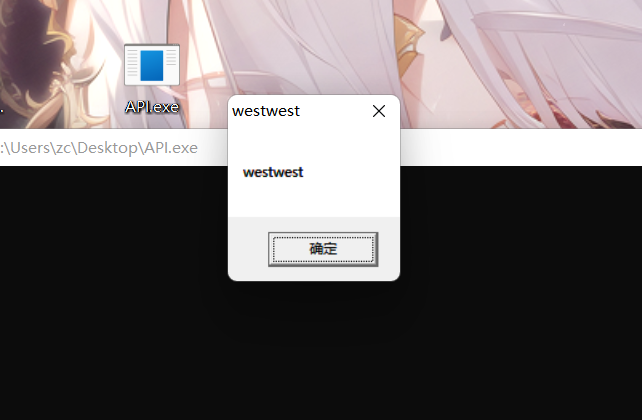
当找到所需的API函数后，基于找到的函数地址，编写shellcode.



1. **API自搜索实验的结果：**



将exe程序复制到windows操作系统的运行结果：



能成功运行便是因为Shellcode具备动态的自动搜索API函数地址的能力，即便版本系统变化。

**心得体会：**

**亲自运行了具有API自搜索技术的Shellcode代码，观察了在实现API函数自搜索时的寄存器值变化。**

**熟悉了API自搜索技术的步骤和关键。**

**通过在不同的操作系统运行exe程序，实践了API函数自搜索技术，进一步掌握了Shellcode编写。**