**《软件安全》实验报告**

姓名：禹相祐 学号：2312900 班级：计算机科学与技术

**实验名称：**

**API函数自搜索**

**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：**

本次实验需要编写能在不同系统中通用的shellcode代码，让其不仅能在x86同时也要在windows 11下运行。要想实现这个功能，就要求我们的shellcode代码能有动态API函数地址自搜索的功能。

1. **梳理全程的思路**
2. MessageBoxA位于user32.dll中，能拿来弹出对话框;
3. ExitProcess位于kernel32.dll中，能拿来退出程序。需要注意的是，所有程序都会自动加载kernel32.dll这个库，所以也就成为了我们实现定位的切入点。
4. 而LoadLibraryA位于kernel32.dll中。所以我们需要先通过kernel32.dll实现定位，再定位LoadLibraryA等函数，最终实现shellcode编写。
5. **定位 kernel32.dll**

代码如下：

|  |  |  |
| --- | --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13 | **//** 压入user32.dll  mov bx,0x3233  push ebx  push 0x72657375  push esp  xor edx,edx  **//** 找kernel32.dll的基址  mov ebx,fs:[edx**+**0x30]  **//**[TED**+**0x30]**--**>PEB  mov ecx,[ebx**+**0xC]      **//**[PEB**+**0xC]**---**>PEB\_LDR\_DATA  mov ecx,[ecx**+**0x1C]  **//**[PEB\_LDR\_DATA**+**0x1C]**---**>InInitializationOrderMoudleList  mov ecx,[ecx]         **//** 进入链表第一个就是ntdll.dll  mov ebp,[ecx**+**0x8]     **//** ebp**=** kernel32.dll的基址 |  |

**理解：**

首先将user32.dll的地址压入栈中，并且通过xor将edx设置为0，然后通过fs段寄存器定位到当前的线程块TEB，加上0x30的偏移量得到PEB并保存在ebx内，然后再偏移0xC到PEB\_LDR\_DATA，然后再是0x1C的偏移量到InInitializationOrderMoudleList。直到找到这个链表后，第一个就是节点ntdll.dll，再偏移8位就是kernel32.dll，至此实现定位。

1. **定位kernel32.dll的导出表**

代码如下：

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8 | 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 |  |

**理解：**

首先将ebp地址偏移0x3C指向PE头指针，而PE头指针偏移0x78处存放着导出表的指针，所以将ebp+eax+0x78就得到了导出表的基地址，然后再偏移0x20指向函数名的指针，最后再加上ebp去获得函数名列表的基本地址。后续只需要一个个对比hash值就能找到我们所需要的特定函数。

1. **定位LoadLibrary等特定目标函数**

代码如下：

|  |  |  |
| --- | --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | #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);  } |  |

然后将三个函数名称的hash值入栈：

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6 | 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] **//**为了兼容性空出8字节 |  |
|  |  |  |

然后通过三个函数find\_lib\_functions & find\_functions & next\_function\_loop 进行循环从而找到我们需要的三个函数的地址。

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | 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    **//======**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 |

**理解：**

其实这些函数就如名字一样：Find\_lib\_functions通过调用find\_functions实现寻找函数，next\_function\_loop即如果不符合要求，那就一直往后寻找，找到就跳出循环。

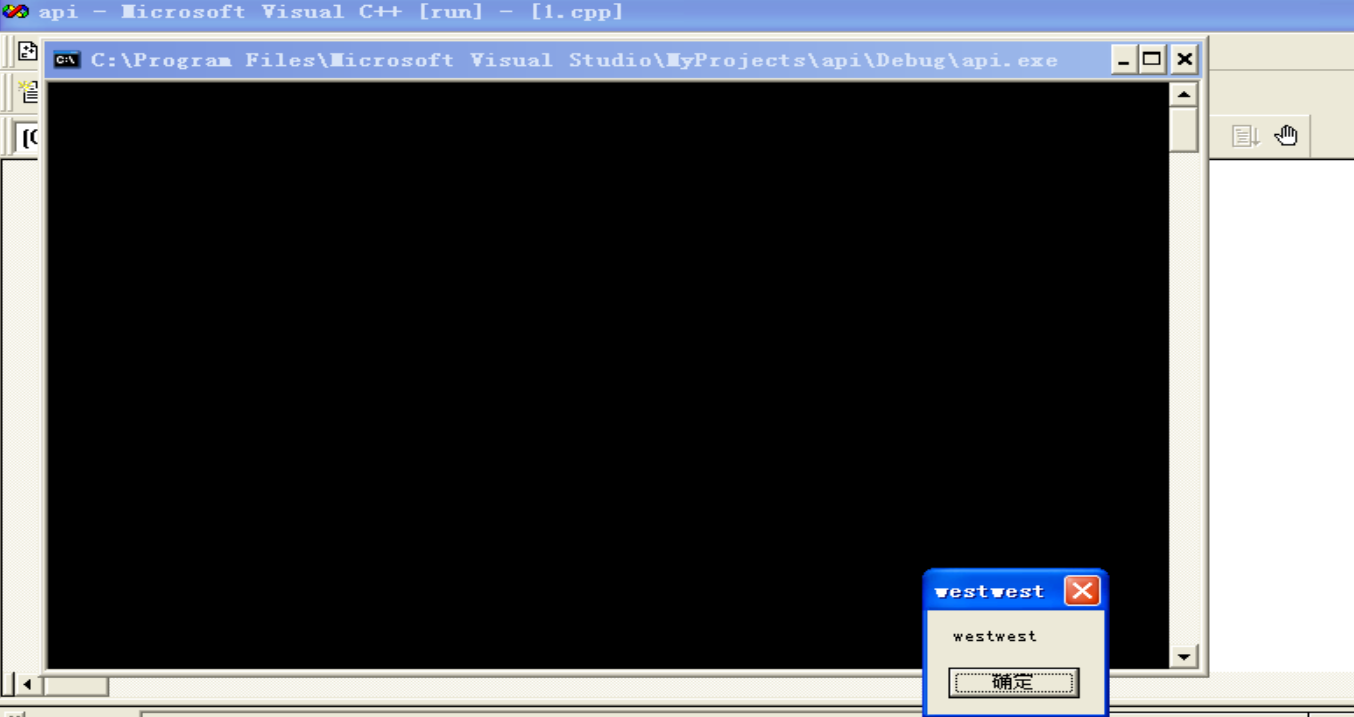
Hash 循环和 Hash值对比的函数如下：

|  |  |  |
| --- | --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | 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    **//======**比较当前函数的hash是否是自己想找的hash  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 **//** 保存到edi的位置      push edi      popad      cmp eax,0x1e380a6a **//**找到MessageBox后，跳出循环      jne find\_lib\_functions |  |

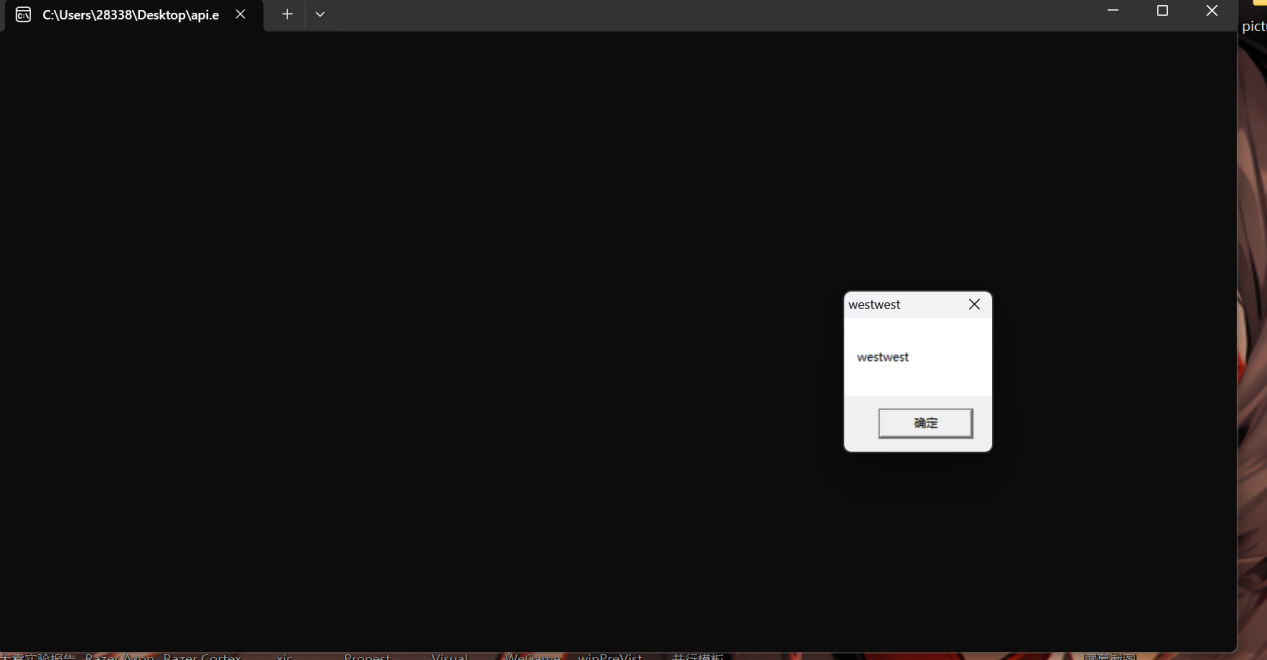
1. **完成shellcode代码的编写：**

|  |  |  |
| --- | --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20 | 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;  } |  |

运行如图：



在windows11上运行：



**心得体会：**

此次实验让我掌握了API函数的子搜索技术，学会了通过TEB、PEB等逐步通过加偏移量实现定位，并通过比较hash值最终确定需要的函数；另外，通过此次实验，也让我对汇编指令有了更深的理解。