软件安全实验报告

学号: 2310764 姓名: 王亦辉 班次: 计科一班

1 实验名称 ：

API函数自搜索

2 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。

3 实验过程 ：

3.1 思路

所用到的函数

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。

利用 API 自搜索方法进行的通用型 shellcode 编写的步骤如下：

1. 定位kernel32.dll。
2. 定位kernel32.dll的导出表。
3. 搜索定位LoadLibrary等目标函数。
4. 基于找到的函数地址，完成Shellcode的编写。

3.2 完整代码如下

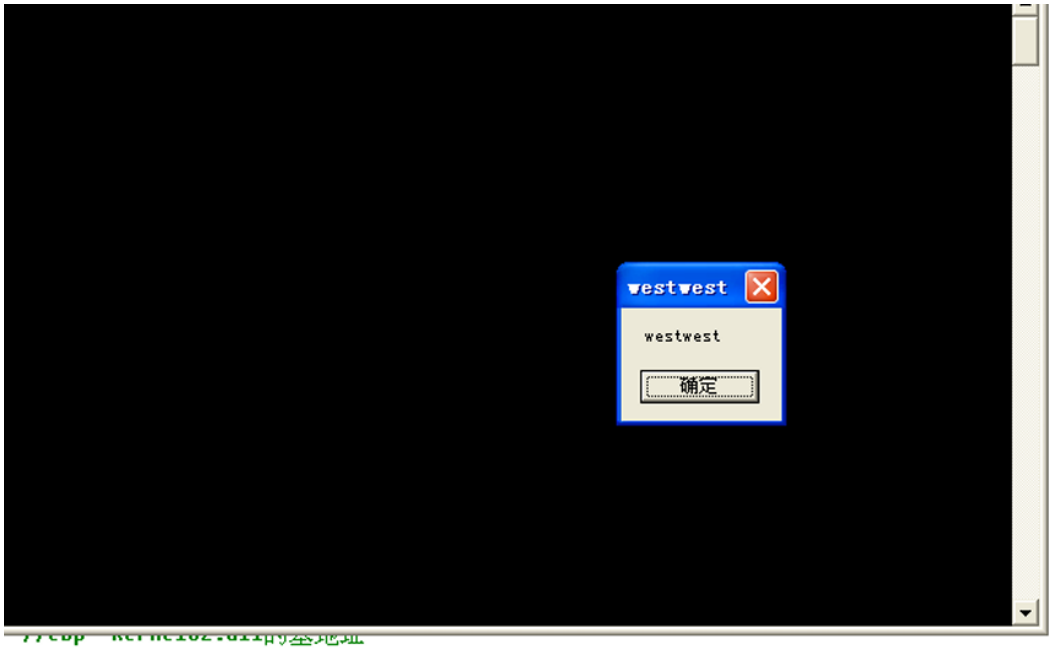
```
#include <stdio.h>
#include <windows.h>

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=user32.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;
}

```

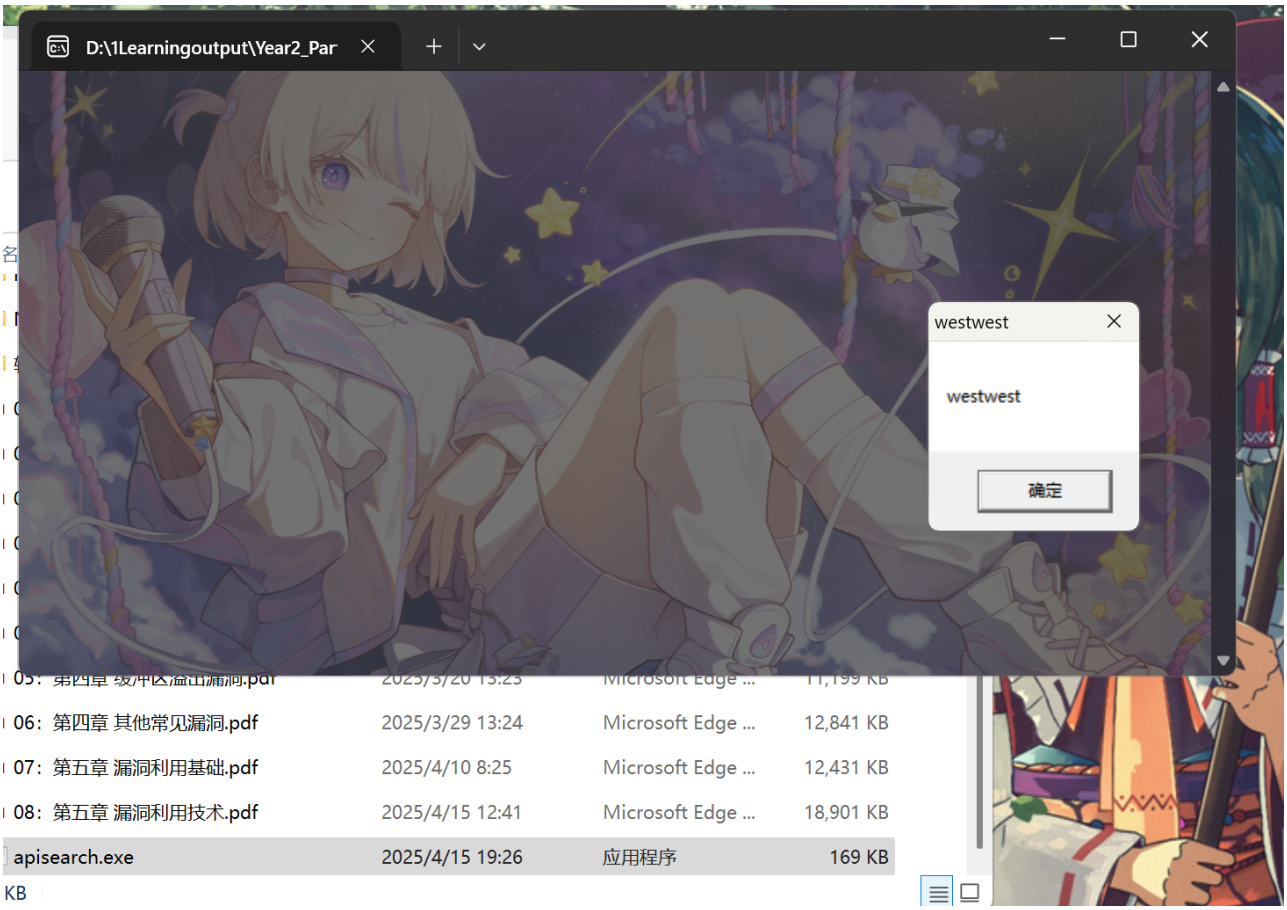
3.3 winxp 虚拟机运行结果



可以看到，我们编写的 Shellcode 能够成功找到目标函数 MessageBoxA 并调用。

3.4 将生成的 exe 复制到 win11 尝试运行

为了验证 API 函数自搜索这种方法写出的 shellcode 的通用性，我们将程序复制到 win11，运行结果如下。



可以看到相应 shellcode 能正常在 win11 中运行，说明这种方法在 win11 中也能成功找到相应的 LoadLibrary、MessageBoxA 等函数并调用。因此，即使在不同系统版本下，函数地址有变动，我们仍通过 API 自搜索能动态地获取到想要的函数。

4 心得体会：

1. 本次实验中，我更加熟悉了汇编语言，比如更深入理解了 ESI、EDI 的作用，以及一些字符串操作。
2. 理解了 API 自搜索这种技术的思想。我们是通过函数名(为了减少长度而使用 hash)这个在各系统版本中相对而言不变的东西，去编写通用的 shellcode。通用的东西往往是抓住了一些不变量。