《API函数自搜索实验》

姓名：田晋宇 学号：2212039 班级：物联网工程

实验名称

API函数子搜索定位实验

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

编写实现 API 自搜索实现对硬编址 shellcode 调用 API 的改写，该实验以 MessageBoxA 函数的调用的 shellcode 为例，大体流程为先使用 LoadLibrary 装载 user32.dll，定位 kernel32.dll，解析 kernel32.dll 的导表，搜索定位 LoadLibrary 等目标函数，再基于找到的函数地址，完成 Shellcode 的编写。

1. 在vc6中完成代码的编写。参照课本，完整代码如下：

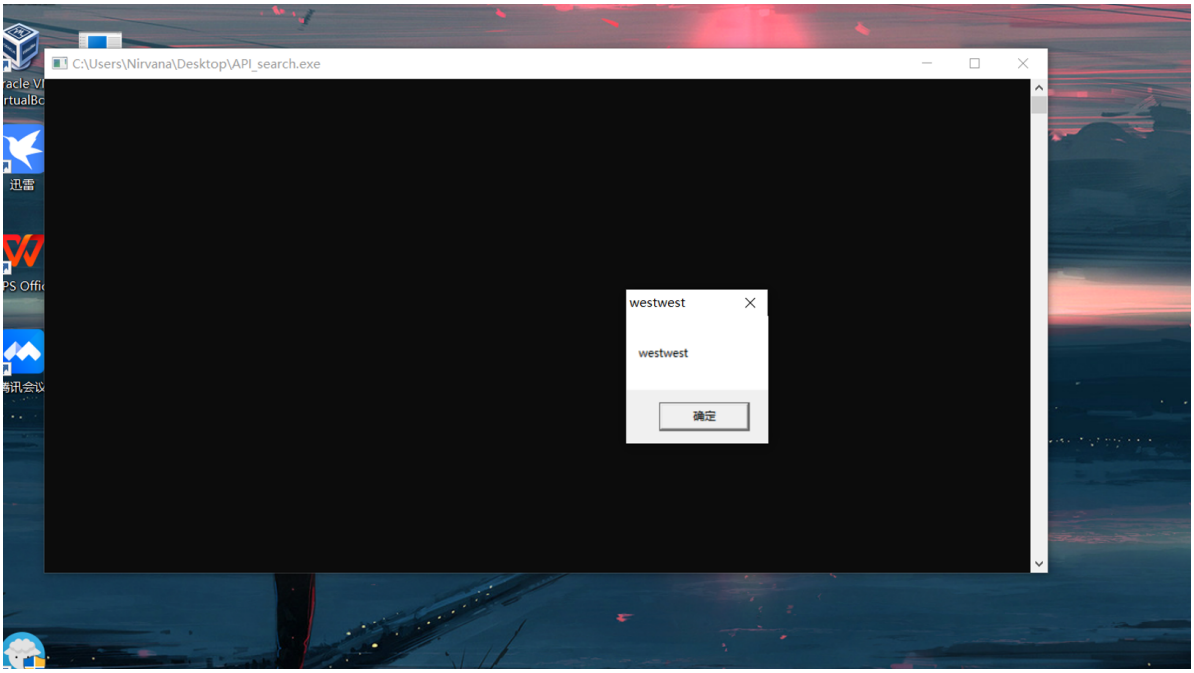
```
#include <stdio.h>
#include <windows.h>

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;
}
```

2. 在vc6环境下进行编译：

3.由于本次实验利用的是自搜索技术，地址是动态计算的，因此将程序放在win10上运行也没问题。将exe放在win10上运行的结果：



4. 代码分析：

```
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
```

这部分代码将 MessageBoxA、ExitProcess、LoadLibraryA 的哈希值压入栈中，然后用 esi 保存 esp，用于指向这三个函数的哈希值，此时 esi 为 0x0012FF28。

```
        //======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
```

这部分用于开辟占空间和压入 user32.dll，因为后续调用 MessageBoxA 函数的时 候首先要先加载出 user32.dll，所以要将其入栈，可以看到，此时执行完 push esp 后栈内 的数值为 323372657375 和 0012FB20，说明 user32.dll 和 esp 已经成功入栈。

```
 //======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
```

然后是找到 kernel32.dll 的基地址，由代码和寄存器可以得出 EBP 此时的值 7C800000 就是 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
```

首先用 loadsd 把 esi 中保存的哈希值赋值给 eax，第一次 eax 取的是 LoadLibraryA 的哈希值，然后和 MessageBoxA 的函数值比较，如果不等于则跳转至 find_functions 继续找，在 find_functions 函数里可以定位到导出表，PE 文件头，导出函 数名表等。在经过两次循环后，eax 存储的值变为 MessageBoxA 的哈希值，此时不发生跳转， 顺序执行，调用 LoadLibrary（“user32.dll”），然后 ebp 的值变为 user32 的基地址。

```
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
```

首先从函数列表取函数名，然后计算函数名的哈希值，然后将哈希值与目标 函数的哈希值比较，如果不相等则返回 next_function_loop 函数继续寻找，直到相等为止， 继续运行后面的代码。

```
        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
```

然后把找到的函数保存到 edi，然后继续返回 find_lib_functions 函数继续循环，直 到找到最后一个函数 MessageBox 为止跳出循环。

```
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
```

找到 MessageBoxA 之后，跳出循环，进入 function_call 函数。这部分代码是用于编写 shellcode，然后把字符串“westwest”压入栈中。此时 ebi 中存储了 3 个函数，在 find_lib_functions中 ebi 中第一个函数 LoadLibraryA 已经调用，所以只需要在 shellcode 中只需调用 MessageBoxA 和 ExitProcess 即可。调用 ExitProcess 后，代码结束。

心得体会

1. 通过本次实验的复现，对vc6实验环境更加了解。
2. API自搜索的方法主要是利用的PE文件头等的相关知识，实现在程序加载时对相对位置的查找、定位，更加适用于现在复杂的应用环境中，同时也能够相应的绕开一些简单的防护机制，达到自己的目的。