**电子科技大学信息与软件工程学院**

**标 准 实 验 报 告**

**（实验）课程名称：软件安全**

**电子科技大学教务处制表**

**学生姓名：周玉川 学 号：2017221302006 指导教师：钱伟中**

**实验地点：信软学院**  **实验时间：**

**一、实验室名称：信软学院实验室**

**二、实验项目名称： 搜索API函数地址**

1. **实验学时：4学时**

**四、实验原理：**

编写shellcode时，一般需使用一些API函数，例如CreateProcess(),socket()等，这些函数的入口地址位于系统的动态链接库中，由于不同操作系统的动态链接库的加载地址不同，shellcode中需增加API函数自搜索功能。

**五、实验目的：**

1）理解API函数搜索原理；

2）掌握搜索kernel32.dll地址的方法；

3）掌握搜索LoadLibrary（）及GetProcAddress()地址的方法；

**六、实验内容：**

1) 编程实现求动态链接库kernel32.dll的虚拟地址。

2）编程实现求LoadLibrary（）及GetProcAddress()虚拟地址。

1. **实验器材（设备、元器件）：**

PC机

* 1. **实验步骤：**

1. 定位kernel32.dll地址
   1. 通过段选择字FS在内存中找到当前的线程控制模块TEB
   2. 线程控制块中偏移位置为0x30的地方存放着指向进程控制块PEB的指针
   3. 进程控制块中偏移地址0x0c的地址存放着指向PEB\_LDA\_DATA结构体的指针，其中存放着已经被装载的动态链接库信息。
   4. PEB\_LDA\_DATA结构体偏移地址为0x1c的地方存放着指向模块初始化链表的头指针InInitializationOrderModulelist。
   5. 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信息，第一个链表节点是ntdll.dll，第二个链表节点就是kernel32.dll。
   6. 找到属于kernel32.dll的节点后，在此基础上再便宜0x08就是kernel32.dll在内存中的价值基地址。
2. 定位LoadLibrary（）及GetProcAddress()地址
   1. 从kernel32.dll的加载基地址开始偏移0x3c的地方就是其PE头
   2. PE头偏移0x78的地方存放着指向函数导出表的指针。
   3. 按以下方式在导出表中算出所需函数的入口地址

（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

（3）函数的RVA地址和名称按顺序放在RVA列表及函数名列表中，根据函数名在函数名称列表中搜索函数序号，再根据函数序号在RVA列表中搜索函数对应的RVA

（4）函数对应的RVA加上动态链接库的加载地址得到该函数的虚拟地址。

**九、实验数据及结果分析：**

**（1） 输出**kernel32.dll，LoadLibrary（）和GetProcAddress()地址

思路分析：

通过fs获得kernelbase地址，然后找到导出表，然后获得导出表中函数名称表，对比字符串获得所找函数在Names表的索引，根据索引找到在NameOrdinal表中的值，这个值就是所求函数在函数地址表中的索引下标，由此得到所求函数的偏移地址，加上kernel32Base即位所求函数的RVA。

代码

|  |
| --- |
| .486  ;告诉汇编器应该生成486处理器（或更高）的伪代码。  .model flat, stdcall  ;使用平坦内存模式并使用stdcall调用习惯,函数的参数从右往左压入  option casemap:none  ;控制字符的映射为大写。为了Windows.inc文件能正常工作，这个应该为”none”  includelib \masm32\lib\kernel32.lib  includelib \masm32\lib\user32.lib  includelib \masm32\lib\gdi32.lib  includelib \masm32\lib\msvcrt.lib  includelib \masm32\lib\masm32.lib  include \masm32\include\kernel32.inc  include \masm32\include\user32.inc  include \masm32\include\gdi32.inc  include \masm32\include\windows.inc  include \masm32\include\msvcrt.inc  include \masm32\include\masm32.inc  include \masm32\macros\macros.asm  ;为了使用来自WindowsAPI的函数，你需要导入dll和包含文件  .data  kernel dd 00 ; 记录kernel的基地址  L dd 00 ; 记录LoadLibraryA的地址  G dd 00 ; 记录GetProcAdress的地址  kernelTip db "The address of Kernel32.dll base is %x",0  load db "The address of LoadLibraryA is %x",0  get db "The address of GetProcAdress is %x",0  .code  start:  assume fs:nothing  push eax  push esi  push ebx  push edi  push ecx  push ebp  push 00000000h  push 41797261h ; 字符串LoadLibraryA的16进制asicc码  push 7262694ch  push 64616f4ch  mov L, esp ; 记录字符LoadLibraryA的地址  push 00007373h ; GetProcAddress  push 65726464h  push 41636f72h  push 50746547h  mov G, esp ; 记录字符GetProcAdress的地址    xor eax, eax  mov eax, fs:30h ;PEB  mov eax, dword ptr [eax+0ch] ; PROCESS\_ MODAULE\_INFO  mov esi, dword ptr [eax+1ch] ; InInitOrder.flink  lodsd ; eax=InInitOrder.flink  mov ebx, [eax+08h] ; ebx=kernel32.dll base address  mov kernel,ebx ; 获得kernel的基地址  push eax  push ebx  invoke crt\_printf,addr kernelTip  add esp,4  print chr$(" ",13,10)  pop eax  push L ; 把字符串LoadLibrary的首地址作为参数传入  push 0bh ; 把字符串的长度作为参数传入  call GetProAddess\_fun ; 调用函数  mov dword ptr L, eax ; 把函数结果存下来  add esp,8 ; 平衡栈  push eax  invoke crt\_printf,addr load  add esp,4  print chr$(" ",13,10)  push G ; 把字符串GetProcAddress的首地址作为参数传入  push 0dh ; 传入字符串GetProcAdress的长度  call GetProAddess\_fun ; 调用函数  mov dword ptr G,eax ; 获得函数返回值  add esp,8 ; 平衡栈  push eax  invoke crt\_printf,addr get  add esp,4  print chr$(" ",13,10)  add esp,20h ; 收回临时变量空间，平衡栈  pop ebp ;  pop ecx ; 恢复现场  pop edi  pop ebx  pop esi  pop eax  ret  GetProAddess\_fun:  mov esi,[ebx+03Ch] ; 获得PE头相对地址  mov esi,[ebx+esi+078h]; 获得导出表偏移地址  add esi,ebx ; 获得导出表的VA    push esi ; 存起来导出表VA  push ebp  mov ecx, dword ptr [esi + 014h] ; 导出表偏14h存放着函数的个数    mov eax, [esi+020h] ; 获得函数名称表偏移地址  add eax, ebx ; 获得函数名称表VA    mov ebp, eax  xor edx, edx ; 用edx记录索引    find\_start:  push ecx    mov eax, dword ptr [eax] ; 获得函数名称表中存放的字符串的偏移地址  add eax, ebx ; 获得函数的名称的实际地址  mov edi, eax ; 把字符串的VA赋值给edi，做比较准备    mov esi, dword ptr [esp + 14h] ; 获得传入的参数，即要需要的字符串的字符首地址  mov ecx, dword ptr [esp + 10h] ; 获得字符个数    cld ;  repz cmpsb ; 调用伪指令进行比较  je find ; 如果两个字符相等就代表找到了  ; 若找不到  inc edx ; 计数加1  add ebp, 4 ; 函数名称表的地址加4  mov eax, ebp ; 把值赋给eax  pop ecx  loop find\_start  find:  add esp,4 ; 平衡栈  pop ebp ; 恢复ebp值  pop esi ; 恢复esi即导出表地址    mov edi, [esi+24h] ; AddressofNameOrdinals  add edi, ebx  shl edx, 1  movzx ecx,word ptr [edi+edx] ; 获得表NameOrdinals中存放的函数序号  mov edi, [esi+1Ch] ; 获得表AdressofFunction的偏移地址  add edi, ebx ; 获得表AdressofFunction的VA  shl ecx, 2  mov eax, [edi+ecx] ; 获得所寻找函数的偏移地址  add eax, ebx ; 获得所寻找函数的RVA，并存放在eax中作为返回值  ret  end start |

输出结果

cmd输出如图1-1

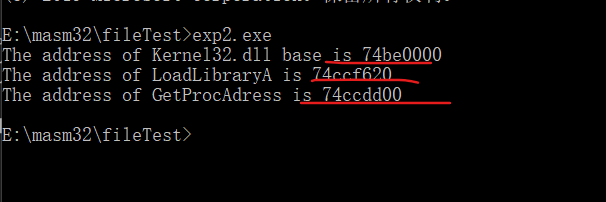


图1-1

odbg调试关键图如图1-2，1-3

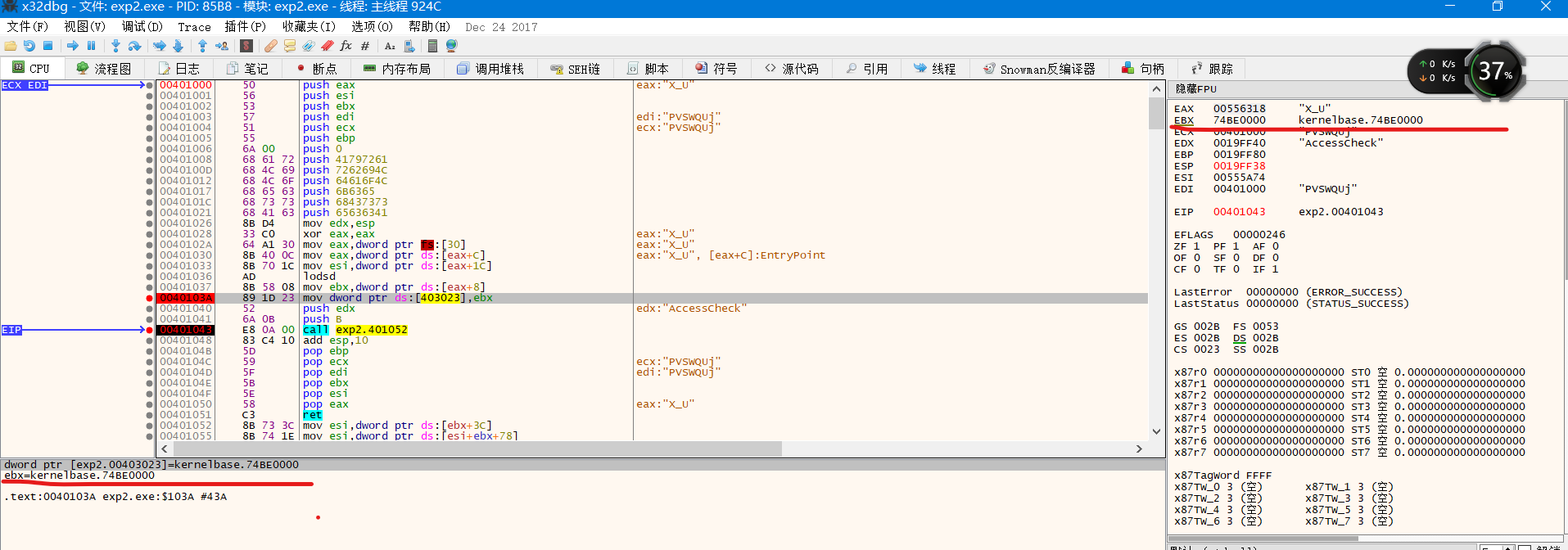


图1-2kernelbase

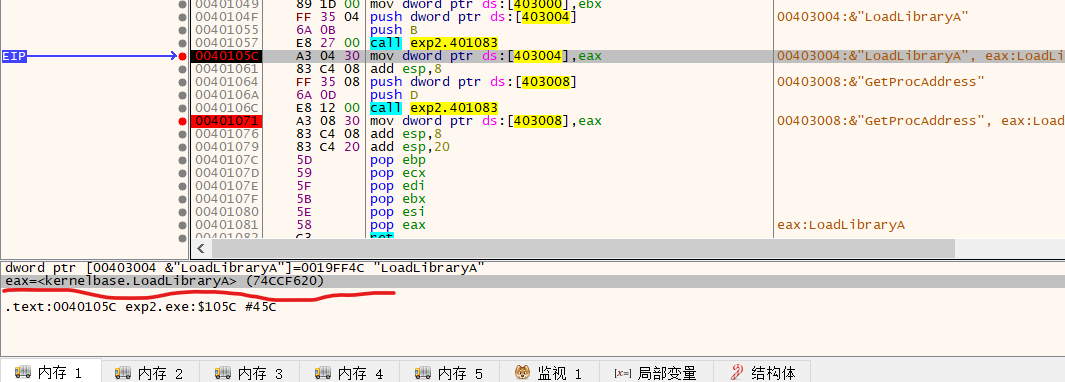


图1-3 LoadLibraryA地址

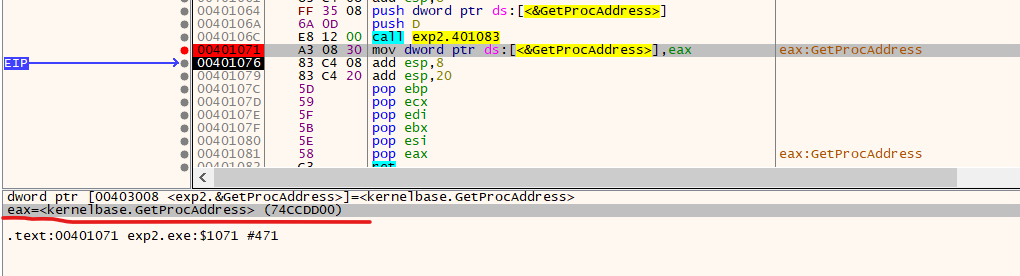


图1-4 GetProcAdress地址

**（2） 实验分析**

导出表定义的数据结构如下表

|  |
| --- |
| DWORD Characteristics;  DWORD TimeDateStamp;  WORD MajorVersion;  WORD MinorVersion;  DWORD Name;  DWORD Base;  DWORD NumberOfFunctions;  DWORD NumberOfNames;  DWORD AddressOfFunctions; // RVA from base of image  DWORD AddressOfNames; // RVA from base of image  DWORD AddressOfNameOrdinals; // RVA from base of image  } IMAGE\_EXPORT\_DIRECTORY, \*PIMAGE\_EXPORT\_DIRECTORY; |

|  |  |
| --- | --- |
| NumberOfFunctions | 所有导出函数的数量。 |
| NumberOfNames | 按名字导出函数的数量。 |
| AddressOfFunctions | 一个RVA，指向一个DWORD数组，数组中的每一项是一个导出函数的RVA，顺序与导出序号相同。 |
| AddressOfNames | 一个RVA，依然指向一个DWORD数组，数组中的每一项仍然是一个RVA，指向一个表示函数名字。 |
| AddressOfNameOrdinals | 一个RVA，还是指向一个WORD数组，数组中的每一项与AddressOfNames中的每一项对应，表示该名字的函数在AddressOfFunctions中的序号。 |

大致流程是，

* 1. 通过fs获得kernelbase地址
  2. 找到导出表。
  3. 获得导出表中函数名称表，函数地址表，nameOrdinal表。
  4. 对比字符串获得所找函数在Names表的索引。
  5. 根据索引找到在所求NameOrdinal表中的内容。
  6. 这个值就是所求函数在函数地址表中的索引下标。
  7. 由此得到所求函数的偏移地址。
  8. 加上kernel32Base即位所求函数的RVA。

**十、实验结论：**

api自搜索技术大致步骤为

1. 通过fs获得kernelbase地址
2. 找到导出表。
3. 获得导出表中函数名称表，函数地址表，nameOrdinal表。
4. 对比字符串获得所找函数在Names表的索引。
5. 根据索引找到在所求NameOrdinal表中的内容。
6. 这个值就是所求函数在函数地址表中的索引下标。
7. 由此得到所求函数的偏移地址。
8. 加上kernel32Base即位所求函数的RVA。

在odbg中动态调试，发现结果正确，实验成功。

**十一、总结及心得体会：**

* + 1. 汇编语言是本实验课的基础，如果汇编学的不扎实是不可能独立把完整代码写出来。
    2. 通过本实验更加理解老师所讲授的api自搜索的实例，对每一句话每一个步骤都有了更加清晰的认值和理解。

**十二、对本实验过程及方法、手段的改进建议：**

1. 实验挺难的，每次都要花费大量的时间在odbg上调试。
2. 软件安全的先修课应该是汇编语言，但是汇编语言是限选课不是必修，所以有些同学不会汇编，根本无从下手。

**报告评分：**

**指导教师签字：**