汇编语言与逆向技术课程实验报告

实验六：ImportExportTable

学院：密码与网络空间安全学院

专业：信息安全

学号：2412950

姓名：路浩斌

班级：信安一班

一、实验目的

1、熟悉 PE 文件的输入表和输出表结构

二、实验原理

(1) 导入表

在 PE 文件头的 IMAGE_OPTIONAL_HEADER 结构中的 DataDirectory(数据目录表) 的第二个成员就是指向导入表。

每个被链接进来的 DLL 文件都分别对应一个 IMAGE_IMPORT_DESCRIPTOR (简称 IID) 数组结构。导入表的结构如图 1 所示。

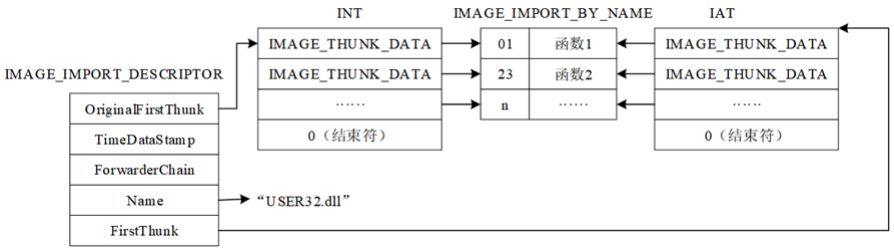

图 1: 导入表结构

(2) 导出表

在 PE 文件头的 IMAGE_OPTIONAL_HEADER 结构中的 DataDirectory(数据目录表) 的第一个成员就是指向导出表。

导出表是用来描述模块中导出函数的数据结构。如果一个模块导出了函数，那么这个函数会被记录在导出表中。导出表的结构如图 2 所示。

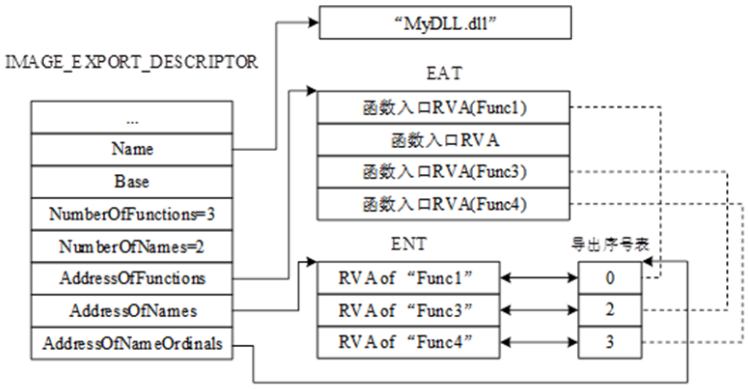

图 2: 导出表结构

三、实验环境

Windows 操作系统，MASM32 编译环境。

四、实验内容

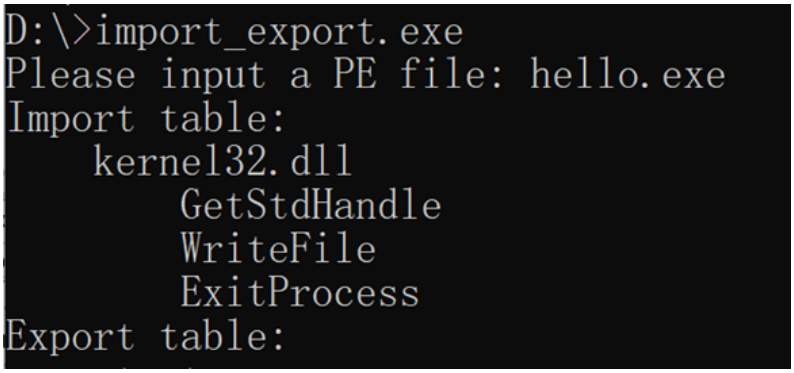

图 3: 导入表和导出表实验演示

(1) 输入 PE 文件的文件名，调用 Windows API 函数，打开指定的 PE 文件；

(2) 读取 PE 文件的导入表，显示导入表中引入的 DLL 文件名和对应的库函数名字；

(3) 读入 PE 文件的导出表，显示导出函数的函数名。

五、实验报告

1.PE 文件的导入表的作用和数据结构

IAT 中的内容与 Windows 操作系统的核心进程、内存、DLL 结构等有关。IAT 是一种表结构，标记程序需要使用哪些库中的哪些函数。

数据结构为 _IMAGE_IMPORT_DESCRIPTOP

```
typedef struct _IMAGE_IMPORT_DESCRIPTOR {
union {
DWORD Characteristics;
DWORD OriginalFirstThunk;
} DUMMYUNIONNAME;
DWORD TimeDateStamp;
DWORD ForwarderChain;
DWORD Name;
DWORD FirstThunk;
```

```
} IMAGE_IMPORT_DESCRIPTOR;
typedef IMAGE_IMPORT_DESCRIPTOR UNALIGNED *PIMAGE_IMPORT_DESCRIPTOR;
```

其中关键字段为：

字段名称	含义
Name	库文件名字符串的地址（RVA）
OriginalFirstThunk	INT（Import Name Table）的地址（RVA）
FirstThunk	IAT（Import Address Table）的地址（RVA）

表 1: IMAGE_IMPORT_DESCRIPTOR 结构中的关键字段

2.PE 文件的导出表的作用和数据结构

Windows 操作系统中，“库”是为了方便其他程序调用而计中包含相关函数的文件。EAT 是一种 DLL 的核心机制，他使得不同的应用程序可以调用库文件中提供的函数。通过 EAT 能够求得从相应库文件中导出函数的入口地址。

```
typedef struct _IMAGE_EXPORT_DIRECTORY {
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;
```

其中关键字段为：

字段名称	含义
Name	库文件名字符串地址
NumberOfFunctions	实际 Export 函数的个数
NumberOfNames	Export 函数中具有名字的函数个数
AddressOfFunctions	Export 函数地址数组
AddressOfNames	函数名称地址数组
AddressOfNameOrdinals	Ordinal 地址数组

表 2: IMAGE_EXPORT_DIRECTORY 结构关键字段说明

3. 汇编语言源代码和注释

```
.386
.model flat,stdcall
option casemap:none

include C:\masm32\include\windows.inc
include C:\masm32\include\kernel32.inc
include C:\masm32\include\masm32.inc
includelib C:\masm32\lib\kernel32.lib
includelib C:\masm32\lib\masm32.lib

.data
;提示信息
str1 BYTE "Please␣input␣a␣PE␣file:",0
str2 BYTE "Import␣table:",0ah,0dh,0
str3 BYTE "Export␣table:",0ah,0dh,0
str4 BYTE "----------------------",0ah,0dh,0

inputBuf BYTE 256 DUP(0);输入的路径
fileBuf BYTE 4000 DUP(0);将文件的前4000字节读入缓存
endl BYTE 0ah,0dh,0

.code
;过程, 将RVA装换成RAW
rvatoraw PROC
    push ebp
    mov ebp,esp
    sub esp,8
```

```
28
29      mov eax,[ebp+8]  ; 保存nt头地址
30      movzx ecx,word ptr [eax+14h]  ; 保存SizeOfOptionalHeader
31      add eax,18h
32      add eax,ecx  ; 此时eax指向节表
33      mov [ebp-4],eax  ; 保存节表起始地址
34      mov eax,[ebp+8]  ; 获得nt头地址
35      movzx ecx,word ptr [eax+6h]  ; 获得NumberOfSections
36      mov [ebp-8],ecx  ; 保存NumberOfSections
37      mov ebx,[ebp-4]  ; 设置遍历基地址
38      mov edx,[ebp+12]  ; 获取RVA
39
40  find_loop:
41      mov eax,[ebx+0ch]  ; 获取VirtualAddress
42      cmp edx,eax
43      jb not_in_bound  ; 小于就跳走
44      add eax,[ebx+10h]  ; 加上SizeOfRawData
45      cmp edx,eax
46      jae not_in_bound  ; 大于等于也跳走
47      ;此时满足条件，计算raw
48      mov eax,edx
49      sub eax,[ebx+0ch]
50      add eax,[ebx+14h]
51      jmp find_end
52
53  not_in_bound:
54      add ebx,28h  ; IMAGE_SECTION_HEADER大小
55      loop find_loop
56      xor eax,eax  ; 都没匹配上，返回0
57
58  find_end:
59      mov esp,ebp
60      pop ebp
61      ret
62  rvatoraw ENDP
63
64  printImport PROC
65      push ebp
66      mov ebp,esp
67      sub esp,12
```

```
68
69      mov eax,[ebp+12]  ; 检测导入表rva是否是0
70      cmp eax,0
71      je import_end
72
73      invoke StdOut, addr str2
74
75      push [ebp+12]  ; 导入表rva
76      push [ebp+8]  ; nt头
77      call rvatoraw
78      add esp,8
79      add eax,offset fileBuf
80      mov [ebp-4],eax  ; 保存导入表在文件中的偏移
81      mov [ebp-8],eax
82
83  loop_dll:
84      mov eax,[ebp-8]  ; 取当前导入dll记录信息地址
85      mov ecx,[eax]  ; 取起始部分, 看是否为0, 为0则跳出外层循环
86      cmp ecx,0
87      je loop_dll_end
88
89      mov eax,[ebp-8]
90      mov ebx,[eax+0ch]  ; 从导入表中读取name的rva
91      push ebx  ;导入表rva
92      push [ebp+8]  ;nt头
93      call rvatoraw
94      add esp,8
95      add eax,offset fileBuf
96      invoke StdOut, eax  ; 输出DLL的name
97      invoke StdOut, addr endl
98
99      mov eax,[ebp-8]
100     mov eax,[eax]  ; 获得OriginalFirstThunk的rva
101     push eax  ; 参数2: OriginalFirstThunk的rva 入栈
102     push [ebp+8]  ; 参数1: nt头 入栈
103     call rvatoraw
104     add esp,8
105     add eax,offset fileBuf  ; 转到到文件中的地址
106     mov [ebp-12],eax  ; 保存INT起始地址
107
```

```
108 loop_int:
109     mov eax,[ebp-12]  ; 获取INT[i]地址
110     mov eax,[eax]  ; 取INT[i]的IMAGE_IMPORT_BY_NAME的RVA
111     cmp eax,0
112     je loop_int_end
113 
114     mov eax,[ebp-12]  ; 获取INT[i]地址
115     mov eax,[eax]  ; 取INT[i]的IMAGE_IMPORT_BY_NAME的RVA
116     push eax  ; INT[i]的rva
117     push [ebp+8]  ;nt头
118     call rvatoraw  ; 获取函数名在文件中的RAW
119     add esp,8
120     add eax,offset fileBuf  ; 此时得到IMAGE_IMPORT_BY_NAME[i]在文件中
            的地址
121     add eax,2
122     invoke StdOut, eax  ; 输出导入函数名称
123     invoke StdOut, addr endl
124 
125     mov ecx,4
126     add [ebp-12],ecx
127     jmp loop_int
128 
129 loop_int_end:
130     mov ecx,14h
131     add [ebp-8],ecx  ; 遍历下一个
132     jmp loop_dll
133 loop_dll_end:
134 import_end:
135     mov esp,ebp
136     pop ebp
137     ret
138 printImport ENDP
139 
140 printExport PROC
141     push ebp
142     mov ebp,esp
143     sub esp,8
144 
145     mov eax,[ebp+12]  ; 检测导出表rva是否是0
146     cmp eax,0
```

```
147     je export_end  ; 如果是0就跳出
148     invoke StdOut, addr str3
149     push [ebp+12]  ; 导出表rva
150     push [ebp+8]  ; nt头
151     call rvatoraw
152     add esp,8
153     add eax,offset fileBuf
154     mov ebx,eax
155     mov eax,[eax+24]
156     mov [ebp-4],eax
157     mov eax,[ebx+32]
158
159     push eax  ; AddressOfNames rva
160     push [ebp+8]  ; nt头
161     call rvatoraw
162     add esp,8
163     add eax,offset fileBuf
164     mov [ebp-8],eax
165     mov ecx,[ebp-4]
166 export_loop:
167     mov [ebp-4],ecx
168     mov ebx,[ebp-8]
169     push [ebx]  ;函数名字的rva
170     push [ebp+8]  ;nt头
171     call rvatoraw
172     add esp,8
173     add eax,offset fileBuf
174     invoke StdOut, eax
175     invoke StdOut, addr endl
176     mov edx,4
177     add [ebp-8],edx
178     mov ecx,[ebp-4]
179     loop export_loop
180
181 export_end:
182     mov esp,ebp
183     pop ebp
184     ret
185 printExport ENDP
186
```

```
;主函数
main PROC
   push ebp
   mov ebp,esp
   sub esp,12

   invoke StdOut,addr str1 ;输出提示信息
   invoke StdIn,addr inputBuf,255 ;输入文件路径
   invoke CreateFile,addr inputBuf,\   ;打开文件
                  GENERIC_READ,\
                  FILE_SHARE_READ,\
                  0,\
                  OPEN_EXISTING,\
                  FILE_ATTRIBUTE_ARCHIVE,\
                  0
    mov [ebp-4],eax  ; 保存文件句柄, 程序结束时要通过此句柄关闭文件
    invoke SetFilePointer, [ebp-4], 0, 0, FILE_BEGIN  ; 将文件指针置
       到文件头部
    invoke ReadFile, [ebp-4], addr fileBuf, 4000, 0, 0  ; 将文件读入
       缓冲区

    ;计算NtHeader地址并保存在栈中
    mov eax,offset fileBuf
    add eax,[eax+3ch]
    mov [ebp-8],eax
    add eax,78h

    ; 导入表
    mov [ebp-12],eax  ; DataDirectory
    mov ebx,[eax+8]  ; 获得导入表RVA
    push ebx  ; 导入表RVA 入栈
    push [ebp-8]  ; nt头 入栈
    call printImport  ;输出
    add esp,8

    invoke StdOut,addr str4
    ; 导出表
    mov eax,[ebp-12]  ; DataDirectory
    mov ebx,[eax]  ; 获得导出表RVA
    push ebx  ;导出表RVA 入栈
```

```
225     push [ebp-8]   ; nt头 入栈
226     call printExport   ;输出
227     add esp,8
228
229     invoke CloseHandle, [ebp-4]   ; 关闭文件
230     invoke ExitProcess,0
231 main ENDP
232 END main
```

4. 运行截图

对自己编写的 test.exe 文件进行检验：得出正确的结果–Import table 和 Export table

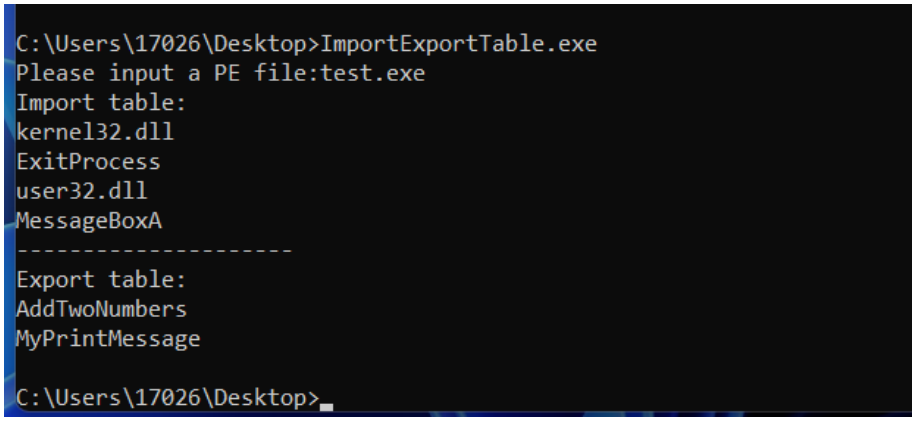

图 4: 运行截图

5. 导入表安全问题的讨论

导入表会有以下的安全问题：

- DLL 注入或内存修改技术，替换 IAT 中的函数地址
- DLL 劫持
- 导入函数伪造，伪造导入表中的函数名称或提示值
- 内存篡改

保护方式:

- 在程序运行时定期对 IAT 的函数地址进行检验，如果检验值出现问题，则导入表被修改，在进行一定的处理，保证导入表的安全性。

- 采用代码签名技术，先对文件计算哈希值，然后使用发布者私钥对哈希加密，并将签名信息写入 PE 文件安全目录。系统在软件运行前会基于证书链验证签名合法性，并对文件重新计算哈希值进行一致性检查。
- 对导入表函数进行加密操作，对导入函数在 PE 文件中以密文方式储存，在程序运行过程中在动态解密，得到函数地址，在写入 IAT 或直接调用，可以让攻击者无法直接得到真实函数名。