

**汇编语言与逆向技术课程实验报告**

**实验六：ImportExportTable**

****

学 院 网络空间安全学院

专 业 信息安全

学 号 2211044

姓 名 陆皓喆

班 级 信息安全

**一、实验目的**

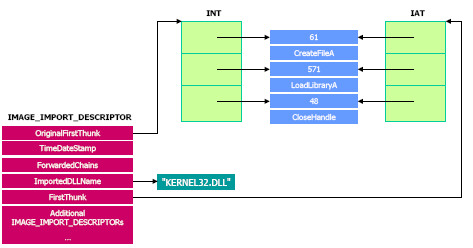
1、熟悉PE文件的输入表和输出表结构。

**二、实验原理**

**（1）输入表**

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输入表。

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。输入表的结构如图1所示。

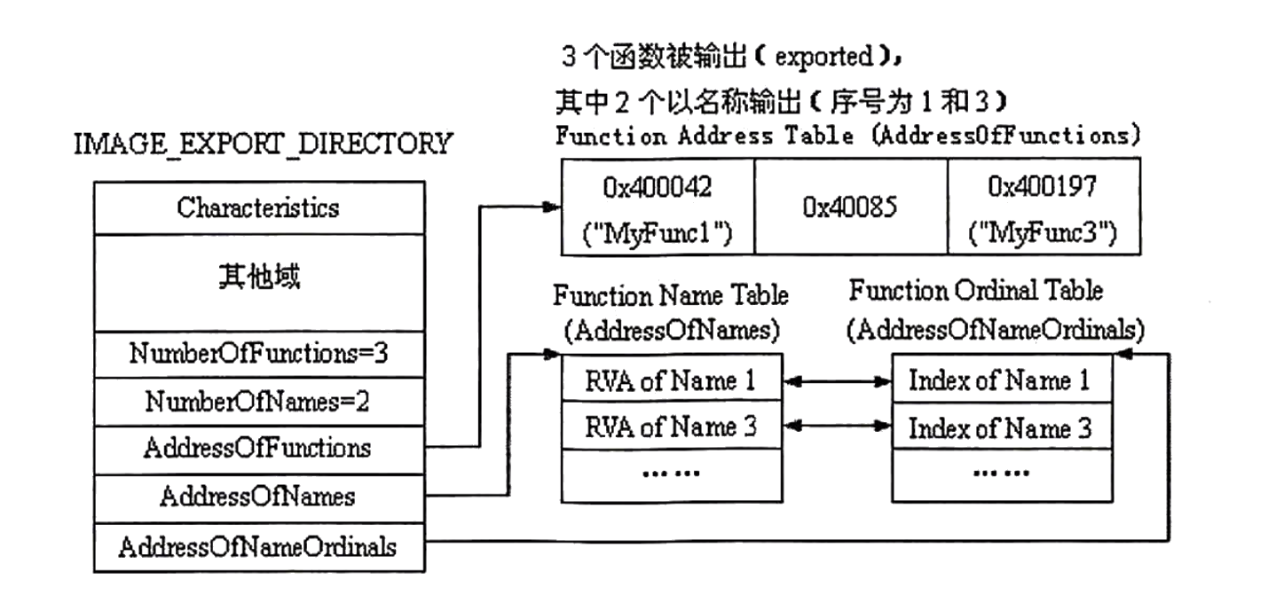


**图 1 输入表结构**

**（2）输出表**

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输出表。

输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输出表中。输出表的结构如图2所示。

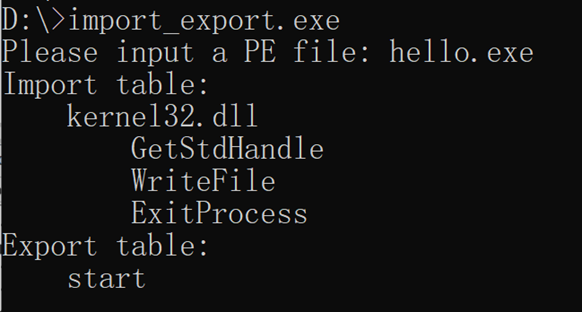
****

**图 2 输出表的数据结构**

**三、实验环境**

Windows操作系统，MASM32编译环境。

**四、实验内容**



**图 3 输入表和输出表实验演示**

**主要步骤：**

（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文件的输入表，显示输入表中引入的DLL文件名和对应的库函数名字；

（3）读入PE文件的输出表，显示导出函数的函数名；

**五、实验报告**

**5.1描述PE文件的输入表的作用和数据结构**

导入表中保存的内容与Windows操作系统的核心进程、内存、DLL结构等密切相关。IAT 是一种表格，用来记录程序中正在使用哪些库中的哪些库函数。

数据结构为**IMAGE\_IMPORT\_DESCRIPTOR**

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;

其中比较重要的数据成员为：

数据成员 含义

OriginalFirstThunk INT的地址（RVA）

Name 库名称字符串的地址（RVA）

FirstThunk IAT的地址（RVA）

**5.2描述PE文件的输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 能够准确求得从相应库中导出函数的起始地址。

数据结构为**IMAGE\_EXPORT\_DIRECTORY**

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;

其中比较重要的数据成员为：

数据成员 含义

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

**5.3程序的汇编语言源代码和注释**

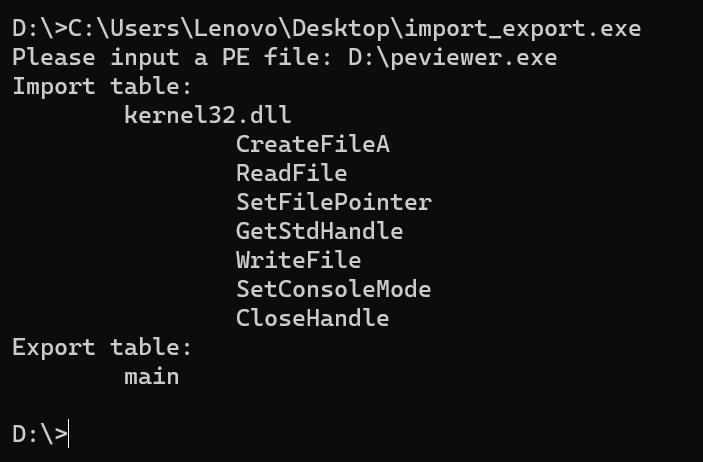
先解释一下该程序的主要思路：

**程序源代码**如下所示：

1. .386
2. .model flat, stdcall
3. option casemap :none
4. include \masm32\include\windows.inc
5. include \masm32\include\kernel32.inc
6. include \masm32\include\masm32.inc
7. includelib \masm32\lib\kernel32.lib
8. includelib \masm32\lib\masm32.lib
10. .data  ;数据段
11. str\_Input\_Cmd BYTE "Please input a PE file: ",0
12. str\_Import BYTE "Import table:",0ah,0dh,0
13. str\_Export BYTE "Export table:",0ah,0dh,0  ;一些输出的内容
15. inputBuf BYTE 256 DUP(0)
16. outputBuf BYTE 256 DUP(0)
17. fileBuf BYTE 4000 DUP(0)
19. n BYTE 0ah,0dh,0
20. tab BYTE 9h,0
22. .code  ;代码段
24. rva\_to\_raw PROC
25. push ebp
26. MOV ebp,esp
27. sub esp,8
28. MOV eax,[ebp+8]  ; 移动8位，目的是保存nt头地址
29. movzx ecx,word ptr [eax+14h]  ; 为了保存SizeOfOptionalHeader
30. add eax,18h
31. add eax,ecx  ; 此时eax指向节表
32. MOV [ebp-4],eax  ; 在eax中保存节表的起始地址
33. MOV eax,[ebp+8]  ; 获得nt头地址
34. movzx ecx,word ptr [eax+6h]  ; 目的是获得NumberOfSections
35. MOV [ebp-8],ecx  ; 向前移动8位，用于保存NumberOfSections
36. MOV ebx,[ebp-4]  ; 设置遍历基地址
37. MOV edx,[ebp+12]  ; 获取rva
39. find\_loop:
40. MOV eax,[ebx+0ch]  ; 获取到VirtualAddress
41. cmp edx,eax
42. jb not\_in\_bound  ; 如果小于就跳走
43. add eax,[ebx+10h]  ; 加上SizeOfRawData
44. cmp edx,eax
45. jae not\_in\_bound  ; 如果大于等于也跳走
46. ;此时满足条件，计算raw
47. MOV eax,edx
48. sub eax,[ebx+0ch]
49. add eax,[ebx+14h]
50. jmp find\_end
52. not\_in\_bound:
53. add ebx,28h  ; IMAGE\_SECTION\_HEADER大小
54. loop find\_loop
55. xor eax,eax  ; 都没匹配上，返回0
57. find\_end:
58. MOV esp,ebp
59. pop ebp
60. ret
62. rva\_to\_raw ENDP
64. printImport PROC
65. push ebp
66. MOV ebp,esp
67. sub esp,12
69. MOV eax,[ebp+12]  ; 检测导入表rva是否是0
70. cmp eax,0
71. je import\_end
73. invoke StdOut, addr str\_Import
75. push [ebp+12]  ; 参数2：导入表rva
76. push [ebp+8]  ; 参数1：nt头
77. call rva\_to\_raw
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
83. loop\_IID:
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\_IID\_end
89. invoke StdOut,addr tab
90. MOV eax,[ebp-8]
91. MOV ebx,[eax+0ch]  ; 从导入表中读取name的rva
92. push ebx  ; 参数2：导入表rva
93. push [ebp+8]  ; 参数1：nt头
94. call rva\_to\_raw
95. add esp,8
96. add eax,offset fileBuf
97. invoke StdOut, eax  ; 输出DLL的name
98. invoke StdOut, addr n
100. MOV eax,[ebp-8]
101. MOV eax,[eax]  ; 获得OriginalFirstThunk的rva
102. push eax  ; 参数2：OriginalFirstThunk的rva 入栈
103. push [ebp+8]  ; 参数1：nt头 入栈
104. call rva\_to\_raw
105. add esp,8
106. add eax,offset fileBuf  ; 转到到文件中的地址
107. MOV [ebp-12],eax  ; 保存INT起始地址
109. loop\_int:
110. MOV eax,[ebp-12]  ; 获取INT[i]地址
111. MOV eax,[eax]  ; 取INT[i]的IMAGE\_IMPORT\_BY\_NAME的RVA
112. cmp eax,0
113. je loop\_int\_end
115. invoke StdOut,addr tab
116. invoke StdOut,addr tab
117. MOV eax,[ebp-12]  ; 获取INT[i]地址
118. MOV eax,[eax]  ; 取INT[i]的IMAGE\_IMPORT\_BY\_NAME的RVA
119. push eax  ; 参数2：INT[i]的rva
120. push [ebp+8]  ; 参数1：nt头
121. call rva\_to\_raw  ; 获取函数名在文件中的RAW
122. add esp,8  ; 栈平衡
123. add eax,offset fileBuf  ; 此时得到IMAGE\_IMPORT\_BY\_NAME[i]在文件中的地址
124. add eax,2
125. invoke StdOut, eax  ; 输出导入函数名称
126. invoke StdOut, addr n
128. MOV ecx,4
129. add [ebp-12],ecx
130. jmp loop\_int
132. loop\_int\_end:
133. MOV ecx,14h
134. add [ebp-8],ecx  ; 遍历下一个
135. jmp loop\_IID
136. loop\_IID\_end:
137. import\_end:
138. MOV esp,ebp
139. pop ebp
140. ret
141. printImport ENDP
143. printExport PROC
144. push ebp
145. MOV ebp,esp
146. sub esp,8
148. MOV eax,[ebp+12]  ; 检测导出表rva是否是0
149. cmp eax,0
150. je export\_end  ; 如果是0就跳出
151. invoke StdOut, addr str\_Export
152. push [ebp+12]  ; 参数2：导出表rva
153. push [ebp+8]  ; 参数1：nt头
154. call rva\_to\_raw
155. add esp,8
156. add eax,offset fileBuf
157. MOV ebx,eax
158. MOV eax,[eax+24]
159. MOV [ebp-4],eax
160. MOV eax,[ebx+32]
162. push eax  ; 参数2：AddressOfNames rva
163. push [ebp+8]  ; 参数1：nt头
164. call rva\_to\_raw
165. add esp,8  ; 栈平衡
166. add eax,offset fileBuf
167. MOV [ebp-8],eax
168. MOV ecx,[ebp-4]
169. export\_loop:
170. MOV [ebp-4],ecx
171. invoke StdOut, addr tab
172. MOV ebx,[ebp-8]
173. push [ebx]  ; 参数2为函数名字的rva
174. push [ebp+8]  ; 参数1为nt头
175. call rva\_to\_raw  ;调用
176. add esp,8
177. add eax,offset fileBuf
178. invoke StdOut, eax
179. invoke StdOut, addr n
180. MOV edx,4
181. add [ebp-8],edx
182. MOV ecx,[ebp-4]
183. loop export\_loop
185. export\_end:
186. MOV esp,ebp
187. pop ebp
188. ret
189. printExport ENDP
191. main PROC
192. push ebp
193. MOV ebp,esp
194. sub esp,12
196. invoke StdOut, addr str\_Input\_Cmd  ; 提示用户输入文件名  ，与上次实验的内容差不多
197. invoke StdIn, addr inputBuf, 255  ; 获得文件名
198. invoke CreateFile, addr inputBuf,\  ; 打开文件
199. GENERIC\_READ,\
200. FILE\_SHARE\_READ,\
201. 0,\
202. OPEN\_EXISTING,\
203. FILE\_ATTRIBUTE\_ARCHIVE,\
204. 0
205. 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
206. invoke SetFilePointer, [ebp-4], 0, 0, FILE\_BEGIN  ; 将文件指针置到文件头部
207. invoke ReadFile, [ebp-4], addr fileBuf, 4000, 0, 0  ; 将文件读入缓冲区
209. ;计算NtHeader地址并保存在栈中
210. MOV eax,offset fileBuf  ; 取的IMAGE\_DOS\_HEADER地址
211. add eax,[eax+3ch]  ; 加上偏移量IMAGE\_DOS\_HEADER::e\_lfanew
212. MOV [ebp-8],eax
213. add eax,78h
215. ; 导入表
216. MOV [ebp-12],eax  ; DataDirectory
217. MOV ebx,[eax+8]  ; 获得导入表RVA
218. push ebx  ; 参数2：导入表RVA 入栈
219. push [ebp-8]  ; 参数1：nt头 入栈
220. call printImport  ; 调用函数printImport
221. add esp,8  ;给esp寄存器后移8位
223. ; 通过移动eax来实现导出表
224. MOV eax,[ebp-12]  ; DataDirectory
225. MOV ebx,[eax]  ; 通过移动寄存器ebx来获得导出表RVA
226. push ebx  ; 参数2的作用是导出表RVA 入栈
227. push [ebp-8]  ; 参数1的作用是nt头 入栈
228. call printExport  ; 函数调用
229. add esp,8
231. invoke CloseHandle, [ebp-4]  ; 关闭文件
232. invoke ExitProcess, 0  ; 结束程序
233. main ENDP
234. END main  ;结束主程序

**5.4程序测试时，输出结果的截图**

对上一次实验的**peviewer.exe**进行检验：



得出正确的结果——Import table和Export table.

**5.5讨论输入表的安全问题**

由于输入表中记录着程序需要用的函数的地址，所以如果有人在程序装载期间或者执行期间修改了输入表中的相关内容，便可以使我们的程序在调用某些函数的时候，被引到攻击者注入的恶意函数的位置，进而执行一些破坏操作。因此，我们需要注重保护好程序的导入表。

我们可以对导入表进行加密操作。

**加密**

1 找到导入表

2 提取导入表的数据结构，自己用***自己的数据结构***来存储，把原本的导入表的数据结构加密，***不让操作系统来处理***

3 把存储的操作系统的数据结构保存在区段里面

4 擦除原来的导入表，弄一个假的导入表给系统读入

5 自己修复导入表

我们还可以进行进一步的加密，对API明文进行加密。

我们利用hash值，给API的名称算出一个hash值，然后利用hash碰撞来实现加密，得到API。

**检验**

如果我们需要按时检验导入表的安全性问题，我们可以每间隔一段时间对导入表进行依次hash校验，如果校验值出现问题，则说明导入表遭到了修改，程序很有可能会出错，应当进行一定的处理，来保证导入表的安全性。