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。

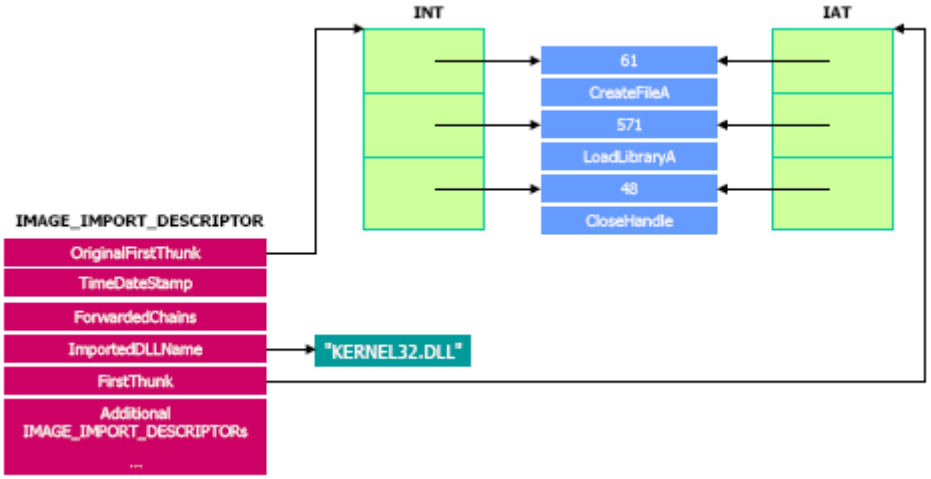

图 1 输入表结构

（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。

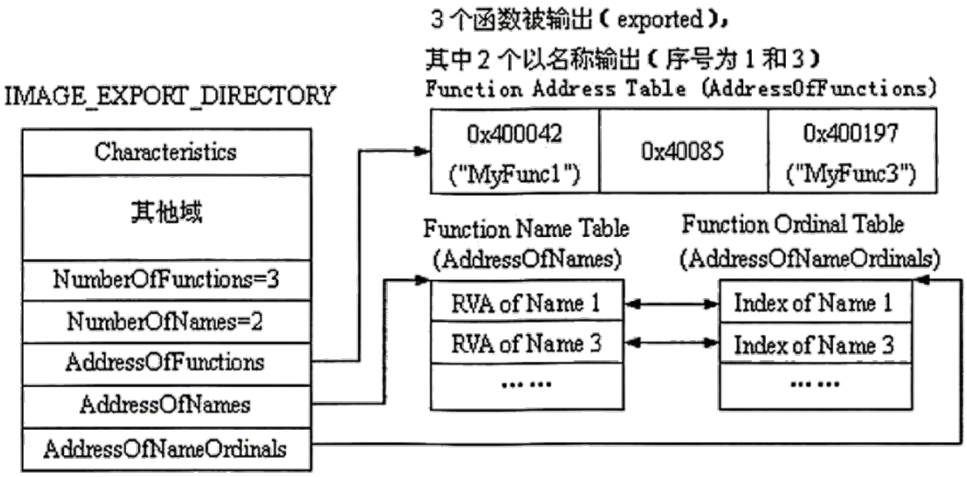

图 2 输出表的数据结构

三、实验环境

Windows 操作系统，MASM32 编译环境。

四、实验内容

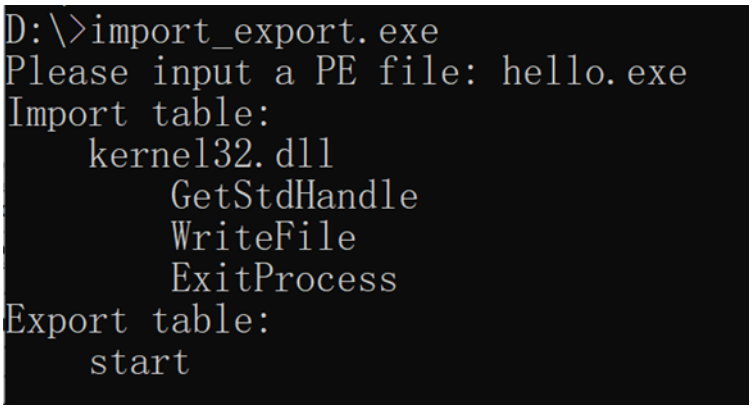

图 3 输入表和输出表实验演示

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.
5.  include \masm32\include\windows.inc
```

```
include \masm32\include\kernel32.inc
include \masm32\include\masm32.inc
includelib \masm32\lib\kernel32.lib
includelib \masm32\lib\masm32.lib

.data ;数据段
    str_Input_Cmd BYTE "Please input a PE file: ",0
    str_Import BYTE "Import table:",0ah,0dh,0
    str_Export BYTE "Export table:",0ah,0dh,0 ;一些输出的内容

    inputBuf BYTE 256 DUP(0)
    outputBuf BYTE 256 DUP(0)
    fileBuf BYTE 4000 DUP(0)

    n BYTE 0ah,0dh,0
    tab BYTE 9h,0

.code ;代码段

rva_to_raw PROC
    push ebp
    MOV ebp,esp
    sub esp,8
    MOV eax,[ebp+8] ；移动 8 位，目的是保存 nt 头地址
    movzx ecx,word ptr [eax+14h] ；为了保存 SizeOfOptionalHeader
    add eax,18h
    add eax,ecx ；此时 eax 指向节表
    MOV [ebp-4],eax ；在 eax 中保存节表的起始地址
    MOV eax,[ebp+8] ；获得 nt 头地址
    movzx ecx,word ptr [eax+6h] ；目的是获得 NumberOfSections
    MOV [ebp-8],ecx ；向前移动 8 位，用于保存 NumberOfSections
    MOV ebx,[ebp-4] ；设置遍历基地址
    MOV edx,[ebp+12] ；获取 rva

find_loop:
    MOV eax,[ebx+0ch] ；获取到 VirtualAddress
    cmp edx,eax
    jb not_in_bound ；如果小于就跳走
    add eax,[ebx+10h] ；加上 SizeOfRawData
    cmp edx,eax
    jae not_in_bound ；如果大于等于也跳走
    ;此时满足条件，计算 raw
    MOV eax,edx
    sub eax,[ebx+0ch]
```

```
    add eax,[ebx+14h]
    jmp find_end

not_in_bound:
    add ebx,28h ; IMAGE_SECTION_HEADER 大小
    loop find_loop
    xor eax,eax ; 都没匹配上，返回 0

find_end:
    MOV esp,ebp
    pop ebp
    ret

rva_to_raw ENDP

printImport PROC
    push ebp
    MOV ebp,esp
    sub esp,12

    MOV eax,[ebp+12] ; 检测导入表 rva 是否是 0
    cmp eax,0
    je import_end

    invoke StdOut, addr str_Import

    push [ebp+12] ; 参数 2：导入表 rva
    push [ebp+8] ; 参数 1：nt 头
    call rva_to_raw
    add esp,8
    add eax,offset fileBuf
    MOV [ebp-4],eax ; 保存导入表在文件中的偏移
    MOV [ebp-8],eax

loop_IID:
    MOV eax,[ebp-8] ; 取当前导入 dll 记录信息地址
    MOV ecx,[eax] ; 取起始部分，看是否为 0，为 0 则跳出外层循环
    cmp ecx,0
    je loop_IID_end

    invoke StdOut,addr tab
    MOV eax,[ebp-8]
    MOV ebx,[eax+0ch] ; 从导入表中读取 name 的 rva
    push ebx ; 参数 2：导入表 rva
```

```
94.     push [ebp+8] ；参数 1：nt 头
95.     call rva_to_raw
96.     add esp,8
97.     add eax,offset fileBuf
98.     invoke StdOut, eax ；输出 DLL 的 name
99.     invoke StdOut, addr n
100.
101.    MOV eax,[ebp-8]
102.    MOV eax,[eax] ；获得 OriginalFirstThunk 的 rva
103.    push eax ；参数 2：OriginalFirstThunk 的 rva 入栈
104.    push [ebp+8] ；参数 1：nt 头 入栈
105.    call rva_to_raw
106.    add esp,8
107.    add eax,offset fileBuf ；转到到文件中的地址
108.    MOV [ebp-12],eax ；保存 INT 起始地址
109.
110. loop_int:
111.    MOV eax,[ebp-12] ；获取 INT[i]地址
112.    MOV eax,[eax] ；取 INT[i]的 IMAGE_IMPORT_BY_NAME 的 RVA
113.    cmp eax,0
114.    je loop_int_end
115.
116.    invoke StdOut,addr tab
117.    invoke StdOut,addr tab
118.    MOV eax,[ebp-12] ；获取 INT[i]地址
119.    MOV eax,[eax] ；取 INT[i]的 IMAGE_IMPORT_BY_NAME 的 RVA
120.    push eax ；参数 2：INT[i]的 rva
121.    push [ebp+8] ；参数 1：nt 头
122.    call rva_to_raw ；获取函数名在文件中的 RAW
123.    add esp,8 ；栈平衡
124.    add eax,offset fileBuf ；此时得到 IMAGE_IMPORT_BY_NAME[i]在文件中的地址
125.    add eax,2
126.    invoke StdOut, eax ；输出导入函数名称
127.    invoke StdOut, addr n
128.
129.    MOV ecx,4
130.    add [ebp-12],ecx
131.    jmp loop_int
132.
133. loop_int_end:
134.    MOV ecx,14h
135.    add [ebp-8],ecx ；遍历下一个
136.    jmp loop_IID
137. loop_IID_end:
```

```
import_end:
    MOV esp,ebp
    pop ebp
    ret
printImport ENDP

printExport PROC
    push ebp
    MOV ebp,esp
    sub esp,8

    MOV eax,[ebp+12] ；检测导出表 rva 是否是 0
    cmp eax,0
    je export_end ；如果是 0 就跳出
    invoke StdOut, addr str_Export
    push [ebp+12] ；参数 2：导出表 rva
    push [ebp+8] ；参数 1：nt 头
    call rva_to_raw
    add esp,8
    add eax,offset fileBuf
    MOV ebx,eax
    MOV eax,[eax+24]
    MOV [ebp-4],eax
    MOV eax,[ebx+32]

    push eax ；参数 2：AddressOfNames rva
    push [ebp+8] ；参数 1：nt 头
    call rva_to_raw
    add esp,8 ；栈平衡
    add eax,offset fileBuf
    MOV [ebp-8],eax
    MOV ecx,[ebp-4]
export_loop:
    MOV [ebp-4],ecx
    invoke StdOut, addr tab
    MOV ebx,[ebp-8]
    push [ebx] ；参数 2 为函数名字的 rva
    push [ebp+8] ；参数 1 为 nt 头
    call rva_to_raw ;调用
    add esp,8
    add eax,offset fileBuf
    invoke StdOut, eax
    invoke StdOut, addr n
    MOV edx,4
```

```
182.    add [ebp-8],edx
183.    MOV ecx,[ebp-4]
184.    loop export_loop
185.
186. export_end:
187.    MOV esp,ebp
188.    pop ebp
189.    ret
190. printExport ENDP
191.
192. main PROC
193.    push ebp
194.    MOV ebp,esp
195.    sub esp,12
196.
197.    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
198.    invoke StdIn, addr inputBuf, 255 ；获得文件名
199.    invoke CreateFile, addr inputBuf,\ ；打开文件
200.                GENERIC_READ,\
201.                FILE_SHARE_READ,\
202.                0,\
203.                OPEN_EXISTING,\
204.                FILE_ATTRIBUTE_ARCHIVE,\
205.                0
206.    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
207.    invoke SetFilePointer, [ebp-4], 0, 0, FILE_BEGIN ；将文件指针置到文件头部
208.    invoke ReadFile, [ebp-4], addr fileBuf, 4000, 0, 0 ；将文件读入缓冲区
209.
210.    ;计算 NtHeader 地址并保存在栈中
211.    MOV eax,offset fileBuf ；取的 IMAGE_DOS_HEADER 地址
212.    add eax,[eax+3ch] ；加上偏移量 IMAGE_DOS_HEADER::e_lfanew
213.    MOV [ebp-8],eax
214.    add eax,78h
215.
216.    ; 导入表
217.    MOV [ebp-12],eax ；DataDirectory
218.    MOV ebx,[eax+8] ；获得导入表 RVA
219.    push ebx ；参数 2：导入表 RVA 入栈
220.    push [ebp-8] ；参数 1：nt 头 入栈
221.    call printImport ；调用函数 printImport
222.    add esp,8 ;给 esp 寄存器后移 8 位
223.
224.    ; 通过移动 eax 来实现导出表
225.    MOV eax,[ebp-12] ；DataDirectory
```

```
226.    MOV ebx,[eax] ；通过移动寄存器 ebx 来获得导出表 RVA
227.    push ebx ；参数 2 的作用是导出表 RVA 入栈
228.    push [ebp-8] ；参数 1 的作用是 nt 头 入栈
229.    call printExport ；函数调用
230.    add esp,8
231.
232.    invoke CloseHandle, [ebp-4] ；关闭文件
233.    invoke ExitProcess, 0 ；结束程序
234. main ENDP
235.     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：

```
D:\>C:\Users\Lenovo\Desktop\import_export.exe
Please input a PE file: D:\peviewer.exe
Import table:
        kernel32.dll
                CreateFileA
                ReadFile
                SetFilePointer
                GetStdHandle
                WriteFile
                SetConsoleMode
                CloseHandle
Export table:
        main

D:\>
```

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。