南开大学

**PE Viewer 2**

**（汇编语言与逆向技术实验）**

****

**姓名：申宗尚**

**学号：2213924**

**专业：信息安全**

1. **实验目的**

1、熟悉PE文件的输入表和输出表结构

1. **实验环境**

1、Windows记事本的汇编语言编写环境

2、MASM32编译环境

3、Windows命令行窗口

1. **实验原理**

**（1）输入表数据结构**

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。

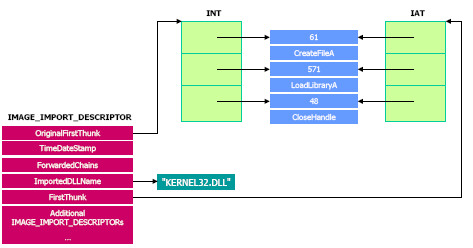


图 1 输入表结构

**（2）输出表数据结构**

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。

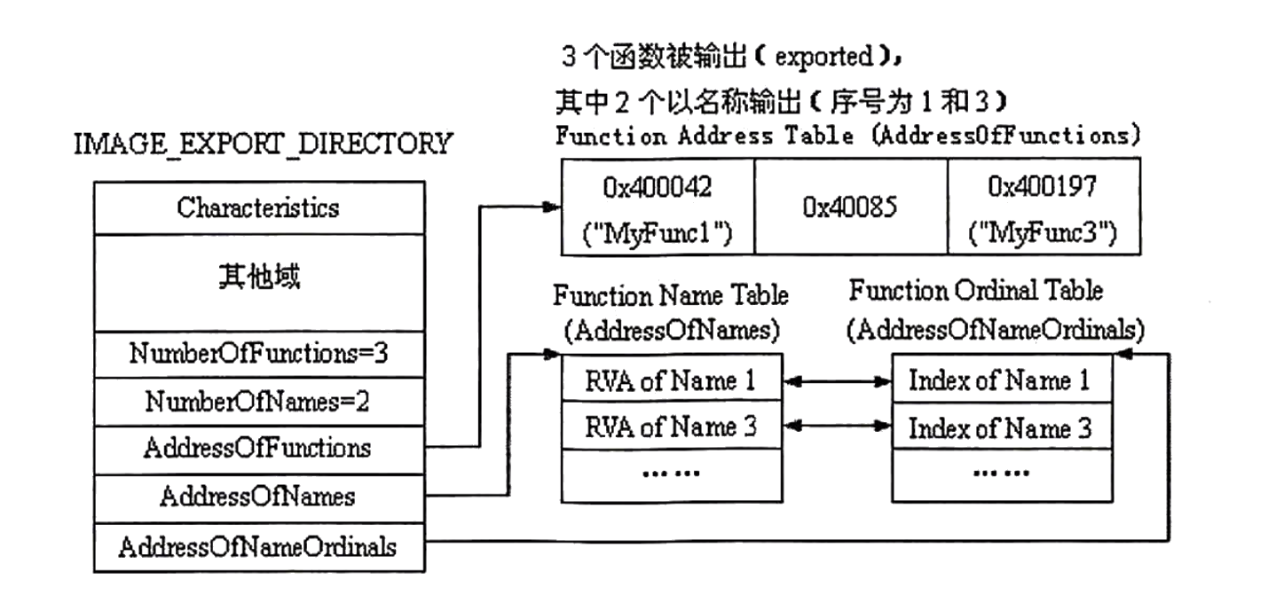
****

图 2 输出表的数据结构

1. **实验内容**

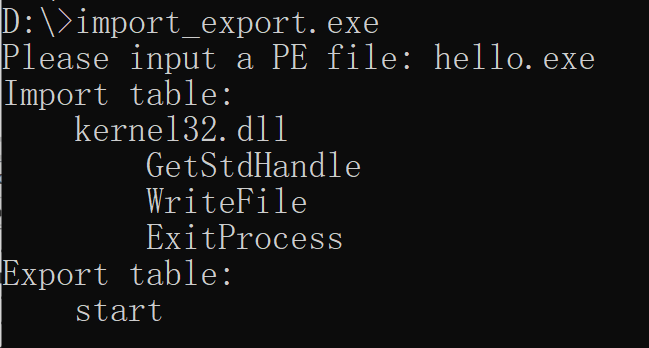


图 3 输入表和输出表实验演示

* + 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、实验程序调试**

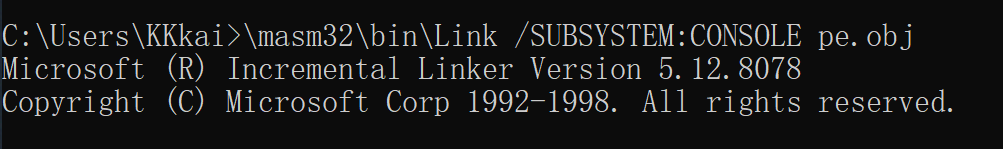
**1.编译：使用ml将peviewer2.asm文件汇编到peviewer2.obj目标文件。**

编译命令：“\masm32\bin\ml /c /Zd /coff pe2.asm”

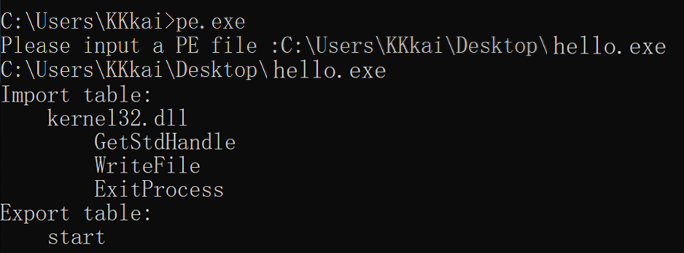


**2. 链接：使用link将目标文件peviewer2.obj链接成peviewer2.exe可执行文件。**

链接命令：“\masm32\bin\Link /SUBSYSTEM:CONSOLE pe2.obj”



**3. 测试：直接执行peviewer2.exe可执行文件。(以hello.exe为例)**

****

1. **实验源代码及注释解释**

.386

.model flat, stdcall

option casemap :none

include \masm32\include\windows.inc

include \masm32\include\kernel32.inc

include \masm32\include\masm32.inc

includelib \masm32\lib\masm32.lib

includelib \masm32\lib\kernel32.lib

.data

output db 100 DUP(0) ; 最后输出的字符串

fileName db 100 DUP(0) ; 文件名

hFile HANDLE 0

content db 4000 DUP(0)

e\_lfnew dd 0

; 定义的待输出字符串

str1 db "Please input a PE file:", 0

str2 db "Import table:", 0

str3 db "Export table:", 0

.code

start:

; 输出提示信息

invoke StdOut, ADDR str1

; 输入文件名

invoke StdIn, ADDR fileName, 100

; 输出文件名

invoke StdOut, ADDR fileName

; 调用函数CreateFile来打开文件

invoke CreateFile, ADDR fileName,\

GENERIC\_READ,\

FILE\_SHARE\_READ,\

0,\

OPEN\_EXISTING,\

FILE\_ATTRIBUTE\_ARCHIVE,\

0

; 调用函数SetFilePointer和ReadFile读取

mov hFile, eax

invoke SetFilePointer, hFile,\

0,\

0,\

FILE\_BEGIN

invoke ReadFile, hFile,\

ADDR content,\

4000,\

0,\

0

; 获取PE文件头

mov eax, DWORD PTR [content+3Ch] ; 偏移0x3C处是PE文件头的偏移

mov e\_lfnew, eax

; 处理导入表

invoke ImportTable, hFile, e\_lfnew

; 处理导出表

invoke ExportTable, hFile, e\_lfnew

; 调用函数CloseHandle关闭句柄

invoke CloseHandle, hFile

invoke ExitProcess, 0

ImportTable PROC hFile:HANDLE, e\_lfnew:DWORD

; 输出导入表提示

invoke StdOut, ADDR str2

; 计算导入表的地址

lea ebx, content

add ebx, e\_lfnew

add ebx, DWORD PTR [ebx+0Ch] ; DataDirectory[1] 是导入表的 RVA

; 获取导入表的首个导入描述符

mov esi, DWORD PTR [ebx]

; 循环处理导入描述符

ImportLoop:

cmp dword ptr [esi], 0 ; 判断是否是导入描述符表的结束

je ImportDone

; 获取DLL名称

lea edi, [esi+2]

invoke StdOut, ADDR edi

; 获取导入地址表

mov eax, DWORD PTR [esi+0Ch] ; OriginalFirstThunk

add eax, e\_lfnew

mov edi, DWORD PTR [eax]

invoke StdOut, ADDR edi

add esi, SIZEOF IMAGE\_IMPORT\_DESCRIPTOR

jmp ImportLoop

ImportDone:

ret

ImportTable ENDP

ExportTable PROC hFile:HANDLE, e\_lfnew:DWORD

; 输出导出表提示

invoke StdOut, ADDR str3

; 计算导出表的地址

lea ebx, content

add ebx, e\_lfnew

add ebx, DWORD PTR [ebx+0Ch] ; DataDirectory[0] 是导出表的 RVA

; 获取导出表的首个导出函数地址表

mov esi, DWORD PTR [ebx+24h]

; 循环处理导出函数

ExportLoop:

cmp dword ptr [esi], 0 ; 判断是否是导出函数表的结束

je ExportDone

; 获取导出函数名

lea edi, [esi+2]

invoke StdOut, ADDR edi

add esi, SIZEOF DWORD ; 导出函数名在地址表中的下一个 DWORD 处

jmp ExportLoop

ExportDone:

ret

ExportTable ENDP

END start

1. **描述PE文件的输入表、输出表的作用**

**输入表（Import Table）：**

作用： 输入表用于存储程序运行时所需的外部函数或模块的信息，这些外部函数和模块通常存储在动态链接库（DLL）中。输入表告诉操作系统和运行时环境程序需要哪些外部资源，并在程序执行时将它们加载到内存中。

内容： 输入表包含了DLL模块的名称、函数的名称或序号、以及函数在内存中的地址等信息。

**输出表（Export Table）：**

作用： 输出表用于描述程序中可以被其他模块调用的函数和数据。它提供了对程序中可导出资源的命名、位置和属性的信息，允许其他程序或模块在运行时使用这些资源。

内容： 输出表包含了可导出的函数和数据的名称、地址、以及其他属性信息。

在具体执行流程中，当一个PE文件被加载到内存中并开始执行时，输入表和输出表发挥以下作用：

1. **讨论输入表的安全问题**

输入表在动态链接库（DLL）加载和函数解析方面，会产生一些安全隐患，以下是安全隐患和防范措施的举例。

**一、DLL劫持（DLL Hijacking）：**

恶意攻击者可能会尝试将恶意的DLL文件放置在系统路径或程序可执行文件所在的目录，以替代正常的DLL文件。这可以导致程序加载恶意DLL而不是预期的系统或第三方DLL。

防范措施： 使用绝对路径加载DLL、使用安全的加载方式（如使用SafeDLLSearchMode）或使用数字签名等方法可以减少DLL劫持的风险。

**二、DLL注入（DLL Injection）：**

恶意攻击者可能尝试将恶意DLL注入到运行的进程中，以执行恶意代码。

防范措施： 使用代码签名、运行时检测DLL完整性、采用安全的加载方式，以及实施进程完整性保护等措施可以帮助防止DLL注入。

**三、DLL篡改（DLL Tampering）：**

恶意攻击者可能尝试修改或替换DLL文件，以执行潜在的恶意功能。

防范措施： 使用数字签名验证DLL的完整性、定期检查DLL文件的一致性，以及限制DLL文件的写入权限都是防范DLL篡改的方法。

**四、不安全的导入函数解析：**

如果程序使用不安全的函数解析方式，例如使用字符串而不是函数序号来标识导入的函数，可能容易受到缓冲区溢出攻击。

防范措施： 使用安全的导入函数解析方式，如使用函数序号而非字符串，可以减少缓冲区溢出攻击的风险。

**五、过度依赖外部DLL版本**：

程序可能过度依赖特定版本的外部DLL，而不提供足够的向后兼容性，导致在系统上不存在或版本不匹配的DLL被加载。

防范措施： 使用明确的DLL版本依赖、提供适当的错误处理机制，以及在可能的情况下提供向后兼容性可以减轻这种问题。