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

一．实验目的

1、熟悉 PE 文件的输入表和输出表结构

二．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

三．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。

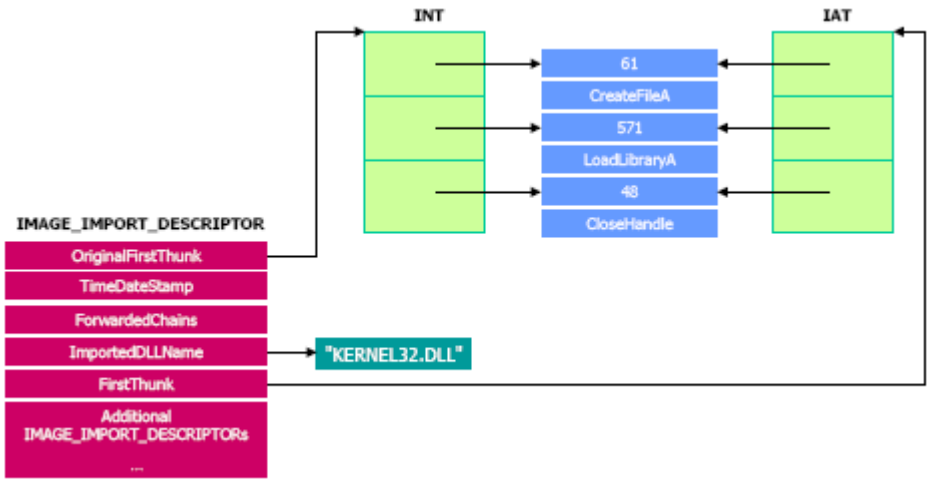

图 1 输入表结构

（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。

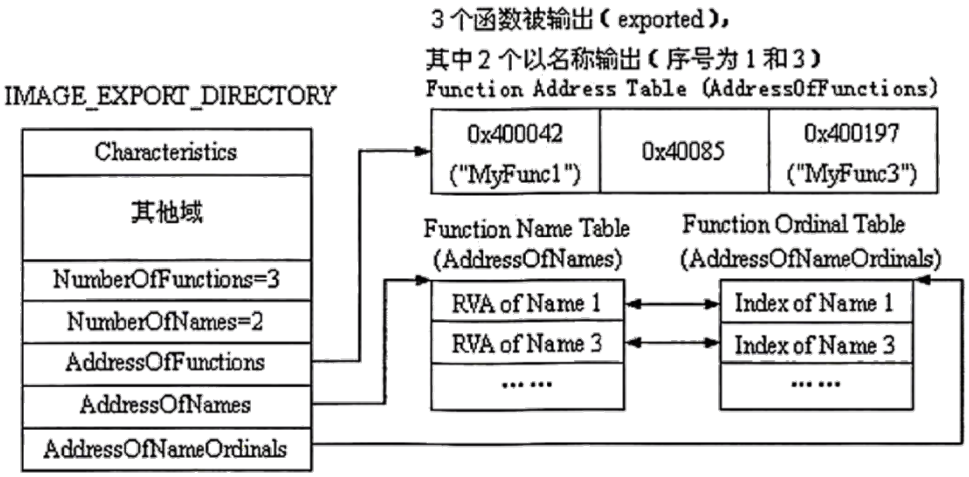

图 2 输出表的数据结构

四．实验内容

```
D:\>import_export.exe
Please input a PE file: hello.exe
Import table:
    kernel32.dll
        GetStdHandle
        WriteFile
        ExitProcess
Export table:
    start
```

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”

```
C:\Users\KKkai>\masm32\bin\ml /c /Zd /coff C:\Users\KKkai\Desktop\pe2.asm
Microsoft (R) Macro Assembler Version 6.14.8444
Copyright (C) Microsoft Corp 1981-1997.  All rights reserved.

 Assembling: C:\Users\KKkai\Desktop\pe2.asm

***********
ASCII build
***********
```

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”

```
C:\Users\KKkai>\masm32\bin\Link /SUBSYSTEM:CONSOLE pe.obj
Microsoft (R) Incremental Linker Version 5.12.8078
Copyright (C) Microsoft Corp 1992-1998. All rights reserved.
```

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

```
C:\Users\KKkai>pe.exe
Please input a PE file :C:\Users\KKkai\Desktop\hello.exe
C:\Users\KKkai\Desktop\hello.exe
Import table:
    kernel32.dll
        GetStdHandle
        WriteFile
        ExitProcess
Export table:
    start
```

六、 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
```

七、 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：

八、 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。