南開大學

汇编语言与逆向技术课程实验报告

实验六： 读取 PE 文件的输入表和输出表

学　　院　网络空间安全学院
专　　业　信息安全、法学双学位
学　　号　2212000
姓　　名　宋奕纬
班　　级　1061

一、实验目的

1、使用 Windows API 函数读取文件内容

2、熟悉 PE 文件的输入表和输出表结构

二、实验原理

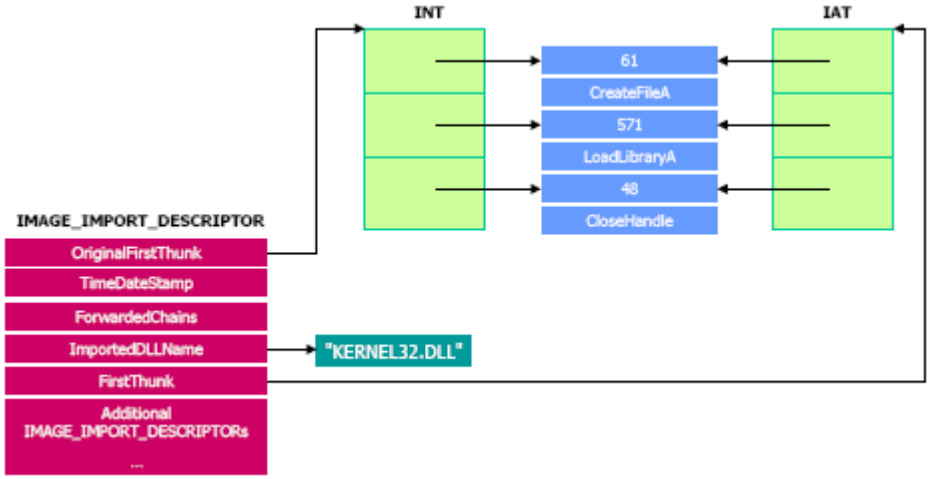
1、 实验环境：MASM32 编译环境、Windows 命令行窗口

2、实验原理：

（1） PE 文件输入表

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。输入表的结构如图所示。



功能：

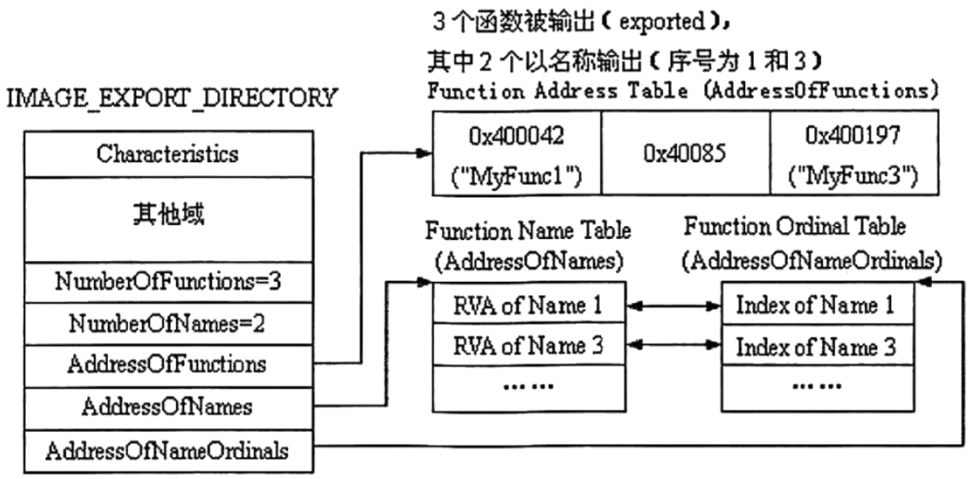
1. 导入函数：导入表记录了模块对其他模块的函数依赖关系，包括函数名、序号、模块名等信息，允许模块在运行时动态链接并调用其他模块的函数。
2. 导入变量：导入表还可以记录模块对其他模块的变量的依赖关系。

数据结构：见上图

（2） PE 文件输出表

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。输出表的结构如所示。



功能:

1.导出函数: 输出表记录了模块中导出的函数的名称、序号、内存地址等信息。其他程序可以通过导表中的函数名或号调用这些函数。

2.导出变量: 表还可以导出全局变量等数据，供其他模块使用。

3.动态链接: 输出表为动态链接供了必要的信息。当一个模块被加载时，其他模块可以通过输出表得所需的导出函数和变量，从而实模块间的调用和共享。

数据结构: 见上图

三、实验过程

1、指定处理器、指令集，指定内存模式，引入头文件，链接静态库文件。

```
.386
.model flat, stdcall
```

```
option casemap:none
;声明外部库的调用
include \masm32\include\windows.inc
include \masm32\include\masm32.inc
include \masm32\include\kernel32.inc
include \masm32\include\user32.inc
includelib \masm32\lib\masm32.lib
includelib \masm32\lib\kernel32.lib
includelib \masm32\lib\user32.lib
```

2、定义数据段，进行相关数据的定义、声名和初始化。

```
;数据段定义
.data

;常量字符串的定义
NoFind BYTE "No export table ERROR!",0 ; 未找到导出表时的提示信息
Please BYTE "Please input your PE file:",0 ; 提示用户所要查看的PE文件的提示语
HuanHang BYTE 0ah,00h ; 换行
;缓冲区定义
filename BYTE 20 DUP(0) ; 存储文件路径的缓冲区
hfile DWORD ? ; 文件句柄
buf BYTE 400000 DUP(0) ; 存储文件内容的缓冲区
;中间地址
NT_addr DWORD ? ; NT头地址
OP_addr DWORD ? ; 可选头地址
SE_addr DWORD ? ; 节表地址
EX_addr DWORD ? ; 导出表地址
IM_addr DWORD ? ; 导入表地址
INT_addr DWORD ? ; INT地址
SE_adds DWORD 30h DUP(0) ; 节表项地址数组（储存各个节的RVA起始地址）
EX_namebase DWORD ? ; 导出表函数名地址基址
SE_num WORD ?             ;节的数量
PT DWORD ? ; 指向第i个IMAGE_IMPORT_DESCRIPTOR的指针
LastPT DWORD ? ; 最后一个函数名指针地址
var1 DWORD ? ; 中间变量1
var2 DWORD ? ; 中间变量2
```

3、代码段定义

（1） 将 RVA 转化为 RAW 的过程

```
rva2raw PROC  rva : DWORD ;定义一个名为rva2raw的过程，传入一个DWORD类型的参数
     mov eax,rva    ;将rva的值存入eax寄存器中
     mov ecx,0      ;将0存入ecx寄存器中
L0:            ;开始一个名为L0的标签
```

```
        shr ecx,2       ;将ecx右移两位，相当于除以4，即左移两位
        cmp cx,SE_num   ;将SE_num与cx比较
        je L2           ;如果相等则跳转到L2标签
        shl ecx,2       ; 将ecx左移两位，相当于乘以4
        mov edx,DWORD PTR SE_adds[ecx4]      ;将SE_adds数组中ecx所指向的元素的值存
                                               入edx寄存器中
        mov edi,DWORD PTR SE_adds[ecx4+4h]   ;将SE_adds数组中ecx+1所指向的元素的值
                                               存入edi寄存器中
        add edx,edi                          ;将edx和edi寄存器中的值相加并存入edx
                                              中，相当于SE_adds[ecx]+SE_adds[ecx+1]
        cmp rva,edi                          ;将rva与edi进行比较
        JL L1                                ; 如果rva小于edi，则跳转到L1标签
        cmp rva,edx                          ;将rva与edx进行比较
        jg L1                                ;如果rva大于edx，则跳转到L1标签
        jmp L2                               ;如果rva在edi和edx之间跳转到L2标签
    L1:
        add ecx,4h                      ;将ecx加上4，指向SE_adds中的下一个元素
        jmp L0                          ;跳转到L0标签
    L2:
        sub eax,DWORD PTR SE_adds[ecx4+4h]  ;将eax减去SE_adds[ecx+1]的值并存入eax
        add eax,DWORD PTR SE_adds[ecx4+0ch] ;将eax加上SE_adds[ecx+3]的值并存入eax
        ret                                 ;返回eax中的值作为结果
    rva2raw ENDP
```

（2） 读取输入表的过程

```
ReadIID PROC
    mov eax,DWORD PTR [esi]  ;将esi指向的DWORD类型值存入eax中
    mov INT_addr,eax         ;将eax中的值存入INT_addr变量中
    add esi,0ch              ;将esi加上0Ch，相当于跳过12个字节
    mov eax,DWORD PTR [esi]  ;将esi指向的DWORD类型值存入eax中
    INVOKE rva2raw,eax       ;调用rva2raw过程将eax中的值转换为RAW地址并返回结果
    add eax,OFFSET buf       ;将buf的偏移地址加上eax中的值，即获取缓冲区中的地址
    INVOKE StdOut,eax        ;调用StdOut过程输出该地址所指向的字符串
    INVOKE StdOut,addr HuanHang  ;输出一个换行符
    add esi,4h               ;将esi加上4，跳过一个DWORD类型值
    mov eax,INT_addr         ;将INT_addr的值存入eax中
    ret                      ;返回eax中的值作为结果
ReadIID ENDP
```

（3） 读取输入表函数的过程

```
ReadIIDName PROC
        INVOKE rva2raw,eax      ;调用rva2raw过程将eax中的值转换为RAW地址并返回结果
        add eax,OFFSET buf      ;将buf的偏移地址加上eax的值，即获取缓冲区中的地址
        mov var2,eax            ;将eax中的值存入var2变量中
    L3:
```

```
        mov eax,var2              ;将var2的值存入eax中
        mov esi,DWORD PTR [eax]   ;将eax指向的DWORD类型值存入esi中
        cmp esi,0h                ;将esi与0进行比较
        je L5                     ;如果相等则跳转到L5标签
        mov edi,esi               ;将esi的值存入edi中
        and edi,0ff000000h        ;将edi与0FF000000h进行按位与操作
        cmp edi,80000000h         ;将edi与80000000h进行比较
        JNE L4                    ;如果不相等则跳转到L4标签
        add eax,4h                ;将eax加上4，跳过一个DWORD类型值
        mov esi,DWORD PTR [eax]   ;将eax指向的DWORD类型值存入esi中
    L4:
        INVOKE rva2raw,esi        ;调用rva2raw过程将esi中的值转换为RAW地址并返回结果
        add eax,OFFSET buf        ;将buf的偏移地址加上eax的值，即获取缓冲区中的地址
        add eax,2h                ;将eax加上2，跳过两个字节
        mov var1,eax              ;将eax中的值存入var1变量中
        INVOKE StdOut,var1        ;调用StdOut过程输出该地址所指向的字符串
        INVOKE StdOut,addr HuanHang   ;输出换行符
        add var2,4h               ;将var2加上4，指向下一个DWORD类型值
        JMP L3                    ;无条件跳转到L3标签
    L5:
        ret                       ;返回
ReadIIDName ENDP
```

（4） 读取输出表函数的过程

```
ReadIED PROC
      add esi,20h                 ;将esi加上20h，跳过32个字节
      mov eax,DWORD PTR [esi]     ;将esi指向的DWORD类型值存入eax中
     INVOKE rva2raw,eax         ;调用rva2raw过程将eax中的值转换为RAW地址并返回结果
      mov EX_namebase,eax         ;将eax中的值存入EX_namebase变量中
      mov LastPT,eax              ;将eax中的值存入LastPT变量中
      L6:                         ;开始一个名为L6的标签
         add eax,OFFSET buf      ;将buf的偏移地址加上eax中的值，即获取缓冲区中的地址
         mov ebx,DWORD PTR [eax]     ;将eax指向的DWORD类型值存入ebx中
         mov edi,ebx                 ;将ebx的值存入edi中
         sub edi,LastPT              ;将edi减去LastPT的值
         cmp edi,0h                  ;将edi与0进行比较
         JL L7                       ;如果小于0则跳转到L7标签
         INVOKE rva2raw,ebx       ;调用rva2raw过程将ebx中的值转换为RAW地址并返回结果
         add eax,OFFSET buf      ;将buf的偏移地址加上eax中的值，即获取缓冲区中的地址
         INVOKE StdOut,eax           ;调用StdOut过程输出该地址所指向的字符串
         INVOKE StdOut,addr HuanHang ;输出一个换行符
         add EX_namebase,4h          ;将EX_namebase加上4，指向下一个地址
         mov eax,EX_namebase         ;将EX_namebase的值存入eax中
         JMP L6                      ;无条件跳转到L6标签
    L7:                              ; 开始一个名为L7的标签
```

```
        ret                          ;返回
  ReadIED ENDP                       ;结束过程定义
```

（5） Main过程

```
main PROC
INVOKE StdOut, addr Please       ; 输出提示信息 "Please"
INVOKE StdOut, addr HuanHang     ; 输出换行符

INVOKE StdIn, addr filename, 20  ; 读取用户输入的文件名，最多20个字符长度

; 打开待分析的文件
INVOKE CreateFile, addr filename, \
                GENERIC_READ, \
                FILE_SHARE_READ, \
                0, \
                OPEN_EXISTING, \
                FILE_ATTRIBUTE_ARCHIVE, \
                0
mov hfile, eax                   ; 将文件句柄保存到hfile寄存器中

INVOKE SetFilePointer, hfile, \
                    0, \
                    0, \
                    FILE_BEGIN   ; 将文件指针设置到文件开头

INVOKE ReadFile, hfile, \
                addr buf, \
                217502, \
                0, \
                0            ; 读取文件内容到缓冲区buf中

mov esi, OFFSET buf              ; 将esi指向buf缓冲区的起始位置

add esi, 3ch                     ; 跳过PE文件的DOS头，找到NT头（PE FILE_HEADER）
mov eax, DWORD PTR [esi]         ; 从NT头中获取PE标志
mov esi, OFFSET buf              ; 将esi重新指向buf缓冲区的起始位置
add esi, eax                     ; 将esi指向PE标志之后的位置

mov ax, WORD PTR [esi+6h]        ; 获取节表数量
mov SE_num, ax                   ; 将节表数量保存到SE_num变量中

mov NT_addr, esi                 ; 将NT头的地址保存到NT_addr变量中
mov ebx, NT_addr                 ; 将ebx设置为NT头地址
add ebx, 18h                     ; 找到可选头的地址，并保存到OP_addr变量中
```

```
add esi, 14h                    ; 跳过PE标志和节表数量，指向第一个节头的位置
mov eax, 0                      ; 清零eax寄存器
mov ax, WORD PTR [esi]          ; 获取第一个节头的大小
mov ebx, OP_addr                ; 将ebx设置为可选头的地址
add ebx, eax                    ; 找到第一个节的地址，并保存到SE_addr变量中

mov esi, OP_addr                ; 将esi设置为可选头的地址
add esi, 60h                    ; 找到导入表的地址，并保存到EX_addr变量中
mov eax, DWORD PTR [esi]
mov EX_addr, eax

add esi, 8h                     ; 找到导出表的地址，并保存到IM_addr变量中
mov eax, DWORD PTR [esi]
mov IM_addr, eax

; 开始遍历节头表
mov esi, SE_addr                ; 将esi指向第一个节头的地址
mov ecx, 0                      ; 清零ecx寄存器
L8:
    mov edx, DWORD PTR [esi]        ; 检查是否存在节头内容
    cmp edx, 0
    je L9                           ; 如果没有节头内容，则跳转到L9并结束遍历
    add esi, 8h                     ; 跳过节头中的其他内容
    mov eax, DWORD PTR [esi]        ; 获取虚拟内存大小
    mov SE_adds[ecx], eax           ; 将虚拟内存大小保存到SE_adds数组中
    add esi, 4h                     ; 跳过其他内容
    mov eax, DWORD PTR [esi]        ; 获取虚拟内存基址
    mov SE_adds[ecx+4h], eax        ; 将虚拟内存基址保存到SE_adds数组中
    add esi, 4h                     ; 跳过其他内容
    mov eax, DWORD PTR [esi]        ; 获取文件节区大小
    mov SE_adds[ecx+8h], eax        ; 将文件节区大小保存到SE_adds数组中
    add esi, 4h                     ; 跳过其他内容
    mov eax, DWORD PTR [esi]        ; 获取文件节区基址偏移
    mov SE_adds[ecx+0ch], eax       ; 将文件节区基址偏移保存到SE_adds数组中
    add ecx, 10h                    ; 增加ecx的值，为下一个节头做准备
    add esi, 14h                    ; 指向下一个节头
    JMP L8                          ; 继续遍历

L9:
    mov ebx, IM_addr                ; 将ebx设置为导入表的地址
    INVOKE rva2raw, ebx             ; 找到导入表的文件偏移，返回值保存在eax中
    mov esi, OFFSET buf             ; 将esi重新指向buf缓冲区的起始位置
    add esi, eax                    ; 将esi指向导入表的位置
```

```
    mov PT, esi                     ; 将PT设置为每个IID元素的首地址
L10:
    mov esi, PT                     ; 将esi设置为当前IID元素的首地址（RAW）
    mov edi, DWORD PTR [esi+0ch]    ; 检查是否已经读完所有IID元素
    cmp edi, 0
    je ReadEnd                      ; 如果已经读完，则跳转到ReadEnd并结束遍历
    INVOKE ReadIID                  ; 读取当前IID元素
    cmp INT_addr, 0h
    jne L11                         ; 检查INT_addr是否为0
    mov eax, DWORD PTR [esi]        ; 如果INT_addr不为0，则读取INT_addr的值
L11:
    INVOKE ReadIIDName              ; 读取当前IID元素的函数名称
    add PT, 14h                     ; 指向下一个IID元素的首地址
    JMP L10                         ; 继续遍历

ReadEnd:
    INVOKE StdOut, addr HuanHang    ; 输出换行符
    mov ebx, EX_addr                ; 将ebx设置为导出表的地址
    cmp ebx, 0h
    je L12                          ; 检查导出表是否为0
    INVOKE rva2raw, ebx             ; 找到导出表的文件偏移，返回值保存在eax中
    mov esi, OFFSET buf             ; 将esi重新指向buf缓冲区的起始位置
    add esi, eax                    ; 将esi指向导出表的位置
    INVOKE ReadIED                  ; 读取导出表的内容，将esi设置为IED的首地
    JMP L13                         ; 跳转到L13并结束程序
L12:
    INVOKE StdOut, addr NoFind      ; 导出表为0时输出"No Find"

L13:
    INVOKE CloseHandle, hfile       ; 关闭文件句柄
    INVOKE ExitProcess, 0           ; 退出程序

main ENDP
END main
```

4、编译，链接，测试

```
C:\Users\宋奕纬>D:

D:\>masm32\bin\ml /c /coff pe.asm
Microsoft (R) Macro Assembler Version 6.14.8444
Copyright (C) Microsoft Corp 1981-1997.  All rights reserved.

 Assembling: pe.asm

***********
ASCII build
***********


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


D:\>pe.exe
Please input your PE file:
dec2hex.exe
kernel32.dll
GetStdHandle
WriteFile
ReadFile
SetConsoleMode
ExitProcess
```

四、实验结论及心得体会

1、收获：熟悉 PE 文件的输入表和输出表结构，感受汇编之美

2、对输入表安全问题的思考：

（1）从安全性的角度来看，输入表可能存在以下几个安全问题：

1.动态链接库劫持：攻击者可以通过替换或劫持被依赖的动态链接库，让程序加载并执行恶意代码。这种攻击方式利用了输入表中指定的模块和函数的特性。

2.导入函数劫持：攻击者可以修改输入表中的函数地址，将其指向恶意代码所在的地址。当程序调用这些导入函数时，实际上执行的是攻击者指定的恶意代码。

3.输入表欺骗：攻击者可以修改输入表中的模块名称或函数名称，以欺骗程序加载错误的模块或函数。这可以导致程序运行异常甚至崩溃。

（2）为了提高PE文件的安全性，可以采取以下措施：

1.数字签名：使用数字签名技术对PE文件进行签名，确保文件的完整性和来源可信。

2.文件完整性检查：在运行时对PE文件进行完整性检查，确保文件没有被篡改或损坏。

3.白名单验证：验证输入表中的模块和函数是否在一个可信的白名单列表中，以防止加载未经授权的代码。

4.使用编译器和工具的安全选项：使用编译器和工具提供的安全选项，如启用地址空间布局随机化（ASLR）和数据执行保护（DEP），以增加对输入表相关攻击的防护能力。