南開大學

汇编语言与逆向技术课程实验报告

实验七：ImportExportTable

学　　院　网络空间安全学院

专　　业　信息安全

学　　号　2313546

姓　　名　蒋枘言

班　　级　信息安全班

一、实验目的

1.熟悉 PE 文件的输入表和输出表结构。

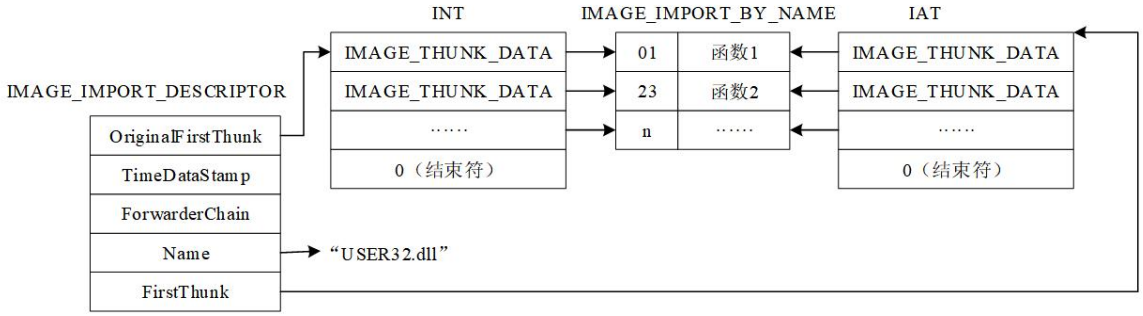
二、实验环境

Windows 操作系统，MASM32 编译环境。

三、实验原理

1.导入表

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导入表。每个被链接进来的 DLL 文件都分别对应一个 IMAGE_IMPORT_DESCRIPTOR（称 IID）数组结构。导入表的结构如图所示。



数据结构为 **IMAGE_IMPORT_DESCRIPTOR**

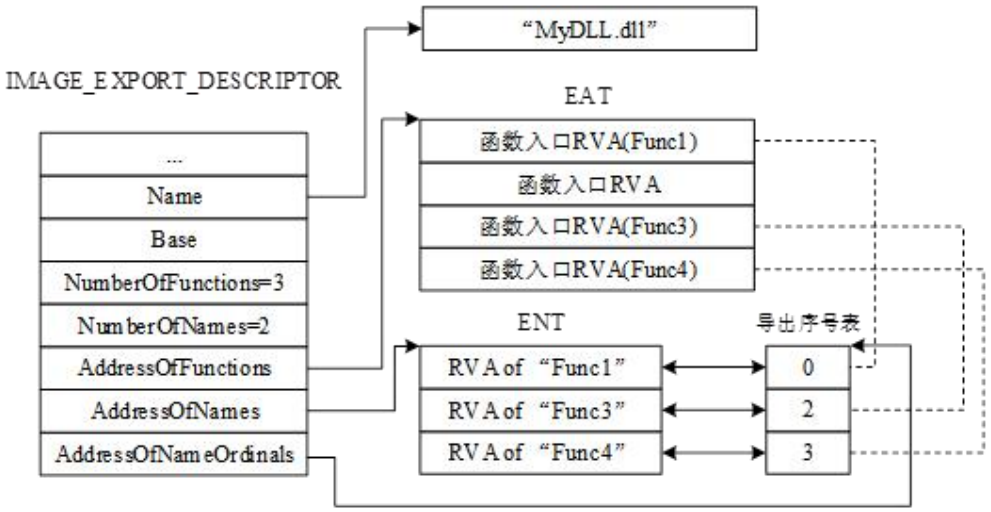
```
typedef struct _IMAGE_IMPORT_DESCRIPTOR {
    union {
        DWORD     Characteristics;
        DWORD     OriginalFirstThunk;
    } DUMMYUNIONNAME;
    DWORD     TimeDateStamp;
    DWORD     ForwarderChain;
    DWORD     Name;
    DWORD     FirstThunk;
} IMAGE_IMPORT_DESCRIPTOR;
typedef IMAGE_IMPORT_DESCRIPTOR 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）

2.导出表

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导出表。导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导出表中。导出表的结构如图所示。



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

四、实验内容

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、代码解释

```
.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\kernel32.lib
includelib \masm32\lib\masm32.lib

.data

   ; 一些输出的内容
   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

   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
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
    push [ebp+8]                    ;参数 1：NT 头
    call rva_to_raw
    add esp,8
    add eax,offset fileBuf
    invoke StdOut, eax              ;输出 DLL 的 name
    invoke StdOut, addr n

    MOV eax,[ebp-8]
    MOV eax,[eax]                   ;获得 OriginalFirstThunk 的 RVA
    push eax                        ;参数 2：OriginalFirstThunk 的 RVA 入栈
    push [ebp+8]                    ;参数 1：NT 头 入栈
    call rva_to_raw
    add esp,8
    add eax,offset fileBuf          ;转到到文件中的地址
    MOV [ebp-12],eax                ;保存 INT 起始地址

loop_int:
    MOV eax,[ebp-12]                ;获取 INT[i]地址
    MOV eax,[eax]                   ;取 INT[i]的 IMAGE_IMPORT_BY_NAME 的 RVA
    cmp eax,0
    je loop_int_end

    invoke StdOut,addr tab
    invoke StdOut,addr tab
```

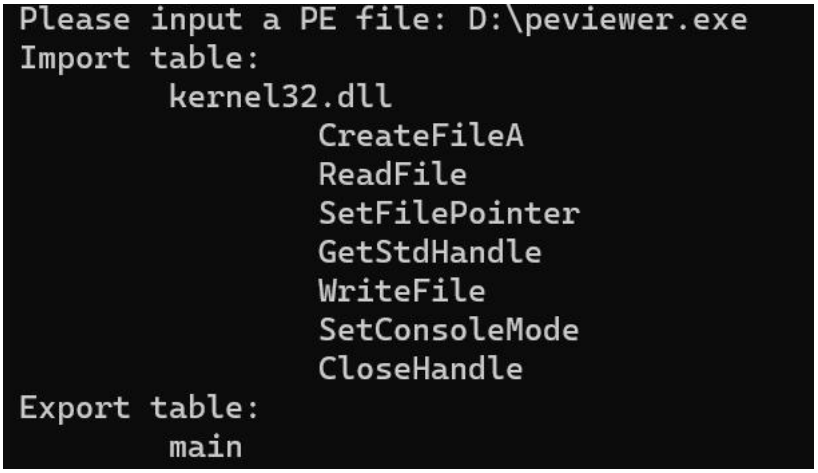
```
    MOV eax,[ebp-12]                    ;获取 INT[i]地址
    MOV eax,[eax]                       ;取 INT[i]的 IMAGE_IMPORT_BY_NAME 的 RVA
    push eax                            ;参数 2：INT[i]的 RVA
    push [ebp+8]                        ;参数 1：NT 头
    call rva_to_raw                     ;获取函数名在文件中的 RAW
    add esp,8                           ;栈平衡
    add eax,offset fileBuf          ;此时得到 IMAGE_IMPORT_BY_NAME[i]在文件中的地址
    add eax,2
    invoke StdOut, eax                  ;输出导入函数名称
    invoke StdOut, addr n

    MOV ecx,4
    add [ebp-12],ecx
    jmp loop_int

loop_int_end:
    MOV ecx,14h
    add [ebp-8],ecx                     ;遍历下一个
    jmp loop_IID
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
    add esp,8
    add eax,offset fileBuf
    invoke StdOut, eax
    invoke StdOut, addr n
    MOV edx,4
    add [ebp-8],edx
    MOV ecx,[ebp-4]
    loop export_loop

export_end:
    MOV esp,ebp
    pop ebp
    ret
printExport ENDP

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

    invoke StdOut, addr str_Input_Cmd            ;提示用户输入文件名
    invoke StdIn, addr inputBuf, 255             ;获得文件名
    invoke CreateFile, addr inputBuf,\           ;打开文件
                GENERIC_READ,\
                FILE_SHARE_READ,\
                0,\
                OPEN_EXISTING,\
                FILE_ATTRIBUTE_ARCHIVE,\
                0
    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
    invoke SetFilePointer, [ebp-4], 0, 0, FILE_BEGIN       ;将文件指针置到文件头部
    invoke ReadFile, [ebp-4], addr fileBuf, 4000, 0, 0     ;将文件读入缓冲区

    ;计算 NtHeader 地址并保存在栈中
    MOV eax,offset fileBuf                              ;取的 IMAGE_DOS_HEADER 地址
    add eax,[eax+3ch]                          ;加上偏移量 IMAGE_DOS_HEADER::e_lfanew
    MOV [ebp-8],eax
    add eax,78h

    ; 导入表
    MOV [ebp-12],eax                            ; DataDirectory
    MOV ebx,[eax+8]                            ;获得导入表 RVA
    push ebx                                   ;参数 2：导入表 RVA 入栈
    push [ebp-8]                               ;参数 1：nt 头 入栈
    call printImport                            ;调用函数 printImport
    add esp,8                                  ;给 esp 寄存器后移 8 位

    ; 通过移动 eax 来实现导出表
    MOV eax,[ebp-12]                            ; DataDirectory
    MOV ebx,[eax]                              ;通过移动寄存器 ebx 来获得导出表 RVA
    push ebx                                   ;参数 2 的作用是导出表 RVA 入栈
    push [ebp-8]                               ;参数 1 的作用是 nt 头 入栈
    call printExport                            ;函数调用
    add esp,8

    invoke CloseHandle, [ebp-4]
    invoke ExitProcess, 0
```

```
main ENDP
END main
```

六、运行结果

输入上次实验 6 的 peviewer.exe：



七、导入表的安全问题

1.DLL 劫持：攻击者通过替换或修改目标程序依赖的 DLL 文件，使其加载恶意 DLL，从而执行恶意代码。攻击者可能会将恶意 DLL 文件放置在与目标程序相同的目录下，或者修改系统的搜索路径，使目标程序在加载 DLL 时优先找到恶意文件。

2.未经验证的外部函数调用：程序可能从不可信的源加载 DLL，并调用其中的函数，导致安全风险。如果这些函数包含恶意代码，那么程序在调用它们时就会执行恶意操作。

加固方案：

1.数字签名验证：对 DLL 文件进行数字签名，并在程序加载时验证签名的有效性。这可以确保只有经过认证和授权的 DLL 文件才能被加载和执行。

2.地址空间布局随机化（ASLR）：通过随机化 DLL 的加载地址，增加攻击者预测和劫持 DLL 的难度。这使得攻击者难以准确找到并替换目标 DLL 文件。

3.安全的 DLL 搜索路径：限制程序搜索 DLL 文件的路径，只从可信的目录加载 DLL。通过设置系统的环境变量或修改程序的配置，可以确保程序只从指定的、安全的路径加载 DLL 文件。