

# 汇编语言与逆向技术实验报告

## Lab4-peviewer

学号：2112492 姓名：刘修铭 专业：信息安全

### 一、实验目的

- 1.熟悉 PE 文件结构；
- 2.使用 Windows API 函数读取文件内容

### 二、实验原理

#### (1) PE 文件结构

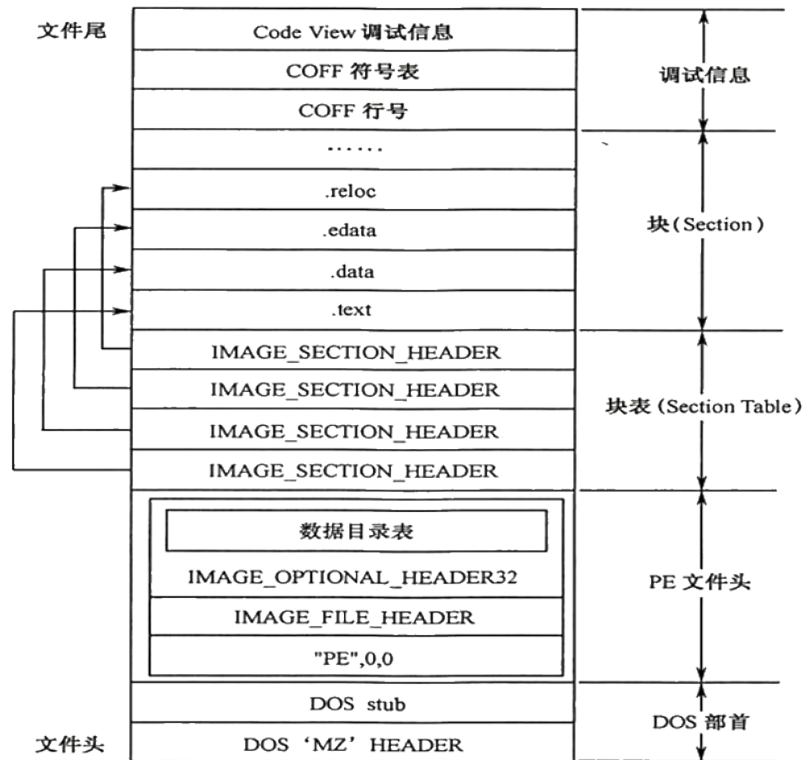


图 1 PE 文件结构

PE 文件结构如图 1 所示。二进制 PE 文件包括：DOS 部首、PE 文件头、块表 (Section Table)、块 (Section)、调试信息 5 个部分。

DOS 部首是 DOS 系统的残留内容，目的是防止 Windows 系统的可执行程序在 DOS 系统上执行时导致 DOS 系统崩溃。DOS 部首是一段 DOS 程序，输出一段提示信息，说明程序只能运行在 Windows 系统上，不能运行在 DOS 系统上。

IMAGE_DOS_HEADER_STRUCT{			
+0h	e_magic	WORD ?	;DOS 可执行文件标记 "MZ"
+2h	e_cblp	WORD ?	
+4h	e_cp	WORD ?	
+6h	e_crlc	WORD ?	
+8h	e_cparhdr	WORD ?	
+0ah	e_minalloc	WORD ?	
+0ch	e_maxalloc	WORD ?	
+0eh	e_ss	WORD ?	
+10h	e_sp	WORD ?	
+12h	e_csum	WORD ?	
+14h	e_ip	WORD ?	;DOS 代码入口 IP
+16h	e_cs	WORD ?	;DOS 代码入口 CS
+18h	e_lfarlc	WORD ?	
+1ah	e_ovno	WORD ?	
+1ch	e_res	WORD 4 dup(?)	
+24h	e_oemid	WORD ?	
+26h	e_oeminfo	WORD ?	
+28h	e_res2	WORD 10 dup(?)	
+3ch	e_lfanew	DWORD ?	;指向 PE 文件头 "PE", 0, 0
} IMAGE_DOS_HEADER_ENDS			

图 2 DOS 头的数据结构

PE 文件头记录了各种文件的装载信息，有映像的基地址（ImageBase）、程序的入口地址（EntryPoint）、数据块、编译时间、运行平台、数据目录表等信息。PE 文件头包括 Signature、FileHeader、OptionalHeader 三部分，数据结构如下所示：

IMAGE_NT_HEADERS STRUCT			
+0h	Signature	DWORD	?
+4h	FileHeader	IMAGE_FILE_HEADER	<>
+18h	OptionalHeader	IMAGE_OPTIONAL_HEADER32	<>
IMAGE_NT_HEADERS ENDS			

**Signature** 的定义是 IMAGE\_NT\_HEADER，值为 00004550h

**FileHeader** 的数据结构如下所示：

IMAGE_FILE_HEADER STRUCT			
+04h	Machine	WORD	?
+06h	NumberOfSections	WORD	?
+08h	TimeStamp	DWORD	?
+0Ch	PointerToSymbolTable	DWORD	?
+10h	NumberOfSymbols	DWORD	?
+14h	SizeOfOptionalHeader	WORD	?
+16h	Characteristics	WORD	?

**OptionalHeader** 的数据结构如下所示：

## IMAGE\_OPTIONAL\_HEADER STRUCT

+18h	Magic	WORD	?
+1Ah	MajorLinkerVersion	BYTE	?
+1Bh	MinorLinkerVersion	BYTE	?
+1Ch	SizeOfCode	DWORD	?
+20h	SizeOfInitializedData	DWORD	?
+24h	SizeOfUninitializedData	DWORD	?
+28h	AddressOfEntryPoint	DWORD	?
+2Ch	BaseOfCode	DWORD	?
+30h	BaseOfData	DWORD	?
+34h	ImageBase	DWORD	?
+38h	SectionAlignment	DWORD	?
+3Ch	FileAlignment	DWORD	?
+40h	MajorOperatingSystemVersion	WORD	?
+42h	MinorOperatingSystemVersion	WORD	?
+44h	MajorImageVersion	WORD	?
+46h	MinorImageVersion	WORD	?
+48h	MajorSubsystemVersion	WORD	?
+4Ah	MinorSubsystemVersion	WORD	?
+4Ch	Win32VersionValue	DWORD	?
+50h	SizeOfImage	DWORD	?
+54h	SizeOfHeaders	DWORD	?
+58h	Checksum	DWORD	?
+5Ch	Subsystem	WORD	?
+5Eh	DllCharacteristics	WORD	?
+60h	SizeOfStackReserve	DWORD	?
+64h	SizeOfStackCommit	DWORD	?
+68h	SizeOfHeapReserve	DWORD	?
+6Ch	SizeOfHeapCommit	DWORD	?
+70h	LoaderFlags	DWORD	?
+74h	NumberOfRvaAndSizes	DWORD	?
+78h	DataDirectory	[IMAGE_NUMBEROF_DIRECTORY_ENTRIES]	

## IMAGE\_DATA\_DIRECTORY <>

### IMAGE\_OPTIONAL\_HEADER ENDS

块表（Section Table）描述代码块、数据块、资源块等不同数据块的文件和内存的映射，数据块的各种属性。

块（Section）分别存储了程序的代码、数据、资源等信息。

## (2) Windows 文件读操作

读一个文件用到的 Windows API 函数有 CreateFile、SetFilePointer、ReadFile、CloseHandle。

**CreateFile** 的 MSDN 文档地址 [https://docs.microsoft.com/en-us/previous-versions/aa914735\(v=msdn.10\)](https://docs.microsoft.com/en-us/previous-versions/aa914735(v=msdn.10))，函数的原型如下：

```
HANDLE CreateFile(  
    LPCTSTR lpFileName,  
    DWORD dwDesiredAccess,  
    DWORD dwShareMode,  
    LPSECURITY_ATTRIBUTES lpSecurityAttributes,  
    DWORD dwCreationDisposition,  
    DWORD dwFlagsAndAttributes,  
    HANDLE hTemplateFile  
);
```

CreateFile 在 MASM 汇编语言中的应用实例如图 3 所示。



```
invoke CreateFile, ADDR buf2, \  
    GENERIC_READ, \  
    FILE_SHARE_READ, \  
    0, \  
    OPEN_EXISTING, \  
    FILE_ATTRIBUTE_ARCHIVE, \  
    0  
MOV hfile, EAX  
invoke SetFilePointer, hfile, 0, 0, FILE_BEGIN  
invoke ReadFile, hfile, ADDR buf3, 4000, 0, 0  
MOV EAX, DWORD PTR buf3  
invoke dw2hex, EAX, ADDR buf4  
invoke StdOut, ADDR buf4  
invoke CloseHandle, hfile
```

图 3 MASM 汇编中调用 CreateFile、SetFilePointer、ReadFile、CloseHandle 的示例

**SetFilePointer** 函数的 MSDN 文档地址 [https://docs.microsoft.com/en-us/previous-versions/aa911934\(v%3dmsdn.10\)](https://docs.microsoft.com/en-us/previous-versions/aa911934(v%3dmsdn.10))，函数原型如下：

```
DWORD SetFilePointer(  
    HANDLE hFile,  
    LONG lDistanceToMove,  
    PLONG lpDistanceToMoveHigh,  
    DWORD dwMoveMethod  
);
```

SetFilePointer 函数的 MASM 调用示例如图 3 所示。

**ReadFile** 函数的 MSDN 文档地址 [https://docs.microsoft.com/en-us/previous-](https://docs.microsoft.com/en-us/previous-versions/aa911934(v%3dmsdn.10))

versions/aa914377(v%3dmsdn.10), 函数原型如下:

```
BOOL ReadFile(  
    HANDLE hFile,  
    LPVOID lpBuffer,  
    DWORD nNumberOfBytesToRead,  
    LPDWORD lpNumberOfBytesRead,  
    LPOVERLAPPED lpOverlapped  
);
```

ReadFile 函数的 MASM 调用示例如图 3 所示。

**CloseHandle** 函数的 MSDN 文档地址 [https://docs.microsoft.com/en-us/previous-versions/aa914720\(v%3dmsdn.10\)](https://docs.microsoft.com/en-us/previous-versions/aa914720(v%3dmsdn.10)), 函数原型如下:

```
BOOL CloseHandle(  
    HANDLE hObject  
);
```

CloseHandle 函数的 MASM 调用示例如图 3 所示。

### 三、实验环境

MASM32 编译环境

Windows 操作系统

### 四、实验内容

(1) 输入 PE 文件的文件名, peviewer 程序调用 Windows API 函数, 打开指定的 PE 文件;

(2) 从文件的头部开始, 读取 IMAGE\_DOS\_HEADER 结构中的 **e\_magic** 和 **e\_lfanew** 字段的值, 按照实验演示的方式输出到命令行窗口;

(3) 继续读取 PE 文件的 IMAGE\_NT\_HEADER 结构中的 **Signature** 字段的值, 按照实验演示的方式输出到命令行窗口;

(4) 继续读取 IMAGE\_NT\_HEADER 结构中的 IMAGE\_FILE\_HEADER 结构, 从中读取字段 **NumberOfSections**、**TimeDateStamp**、**Characteristics** 的值, 按照实验演示的方式输出到命令行窗口;

(5) 继续读取 IMAGE\_NT\_HEADER 结构中的

IMAGE\_OPTIONAL\_HEADER 结构，从中读取字段 AddressOfEntryPoint、ImageBase、SectionAlignment、FileAlignment 的值，按照实验演示的方式输出到命令行窗口；

## 五、过程说明

### 1. 编辑：编写汇编程序 peviewer.asm。

#### ● 流程：

1.输入 PE 文件的文件名，peviewer 程序调用 Windows API 函数，打开指定的 PE 文件；

2.从文件的头部开始，读取 IMAGE\_DOS\_HEADER 结构中的 e\_magic 和 e\_lfanew 字段的值，按照实验演示的方式输出到命令行窗口；

3.继续读取 PE 文件的 IMAGE\_NT\_HEADER 结构中的 Signature 字段的值，按照实验演示的方式输出到命令行窗口；

4.继续读取 IMAGE\_NT\_HEADER 结构中的 IMAGE\_FILE\_HEADER 结构，从中读取字段 NumberOfSections、TimeDateStamp、Characteristics 的值，按照实验演示的方式输出到命令行窗口；

5.继续读取 IMAGE\_NT\_HEADER 结构中的 IMAGE\_OPTIONAL\_HEADER 结构，从中读取字段 AddressOfEntryPoint、ImageBase、SectionAlignment、FileAlignment 的值，按照实验演示的方式输出到命令行窗口；

#### ● 设计说明：

1.调用 CreateFile 函数，利用用户输入的文件名打开相应文件；

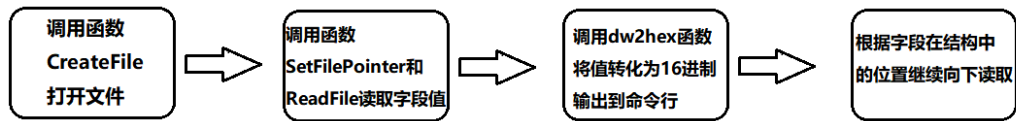
2.从文件的头部开始，调用 SetFilePointer 和 ReadFile 函数读取相应字段的值；

3.将读取出来的值通过 dw2hex 函数转化为 16 进制输出到命令行；

4.根据文件中其他字段与 e\_magic 和 e\_lfanew 的结构关系依次读取 Signature、NumberOfSections、TimeDateStamp、Characteristics、AddressOfEntryPoint、ImageBase、SectionAlignment、FileAlignment 的值并转化为 16 进制输出到命令行；

5.关闭。

- 控制流图：



2. 编译：使用 ml 将 peviewer.asm 文件汇编到 peviewer.obj 目标文件。

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

```
Microsoft Windows [版本 10.0.22623.875]
(c) Microsoft Corporation. 保留所有权利。

C:\Users\98712\OneDrive\桌面>\masm32\bin\ml /c /Zd /coff peviewer.asm
Microsoft (R) Macro Assembler Version 6.14.8444
Copyright (C) Microsoft Corp 1981-1997. All rights reserved.

Assembling: peviewer.asm

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

3. 链接：使用 link 将目标文件 peviewer.obj 链接成 peviewer.exe 可执行文件。

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

```
C:\Users\98712\OneDrive\桌面>\masm32\bin\Link /SUBSYSTEM:CONSOLE peviewer.obj
Microsoft (R) Incremental Linker Version 5.12.8078
Copyright (C) Microsoft Corp 1992-1998. All rights reserved.
```

4. 测试：直接执行peviewer.exe可执行文件。

```
C:\Users\98712\OneDrive\桌面>peviewer.exe
Please input a PE file :peviewer.exe
peviewer.exe
IMAGE_DOS_HEADER
  e_magic: 5A4D
  e_lfanew: 000000B8
IMAGE_NT_HEADERS
  Signature: 00004550
IMAGE_FILE_HEADER
  NumberOfSections: 0003
  TimeDateStamp: 6364ABAD
  Characteristics: 010F
IMAGE_OPTIONAL_HEADER
  AddressOfEntryPoint: 00001000
  ImageBase: 00400000
  SectionAlignment: 00001000
  FileAlignment: 00000200
```

## 五、源代码

```
.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_lfnw dd 0

;定义的待输出字符串
str1 db "Please input a PE file :",0
str2 db 0Ah,"IMAGE_DOS_HEADER",0Ah,"    e_magic: ",0
str3 db 0Ah,"    e_lfanew: ",0
str4 db 0Ah,"IMAGE_NT_HEADERS",0Ah,"    Signature: ",0
str5 db 0Ah,"IMAGE_FILE_HEADER",0Ah,"    NumberOfSections: ",0
str6 db 0Ah,"    TimeDateStamp: ",0
str7 db 0Ah,"    Charateristics: ",0
str8 db 0Ah,"IMAGE_OPTIONAL_HEADER",0Ah,"    AddressOfEntryPoint: ",0
str9 db 0Ah,"    ImageBase: ",0
str10 db 0Ah,"    SectionAlignment: ",0
str11 db 0Ah,"    FileAlignment: ",0

.code
start:
    invoke StdOut, ADDR str1
    invoke StdIn, ADDR fileName, 100
    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,\n0
```

;调用函数 SetFilePointer 和 ReadFile 读取

```
mov hFile, eax\ninvoke SetFilePointer, hFile,\n    0,\n    0,\n    FILE_BEGIN\ninvoke ReadFile, hFile,\n    ADDR content,\n    4000,\n    0,\n    0
```

;e\_magic 转化为 16 进制输出

```
mov eax, 0\nmov ax, WORD PTR content\ninvoke dw2hex, eax, ADDR output
```

```
invoke StdOut, ADDR str2\ninvoke StdOut, ADDR output+4
```

;e\_lfanew 转化为 16 进制输出

```
mov eax, 0\nmov eax, DWORD PTR [content+3ch]\nmov e_lfanew, eax\ninvoke dw2hex, eax, ADDR output
```

```
invoke StdOut, ADDR str3\ninvoke StdOut, ADDR output
```

;Signature 转化为 16 进制输出

```
lea ebx, content\nadd ebx, e_lfanew\nmov eax, 0\nmov eax, DWORD PTR [ebx]\ninvoke dw2hex, eax, ADDR output
```

```
invoke StdOut, ADDR str4\ninvoke StdOut, ADDR output
```

;NumberOfSections 转化为 16 进制输出

```
lea ebx, content
add ebx, e_lfnw
mov eax, 0
mov ax, WORD PTR [ebx+6h]
invoke dw2hex, eax, ADDR output
```

```
invoke StdOut, ADDR str5
invoke StdOut, ADDR output+4
```

```
;TimeStamp 转化为 16 进制输出
lea ebx, content
add ebx, e_lfnw
mov eax, 0
mov eax, DWORD PTR [ebx+8h]
invoke dw2hex, eax, ADDR output
```

```
invoke StdOut, ADDR str6
invoke StdOut, ADDR output
```

```
;Charateristics 转化为 16 进制输出
lea ebx, content
add ebx, e_lfnw
mov eax, 0
mov ax, WORD PTR [ebx+16h]
invoke dw2hex, eax, ADDR output
```

```
invoke StdOut, ADDR str7
invoke StdOut, ADDR output+4
```

```
;AddressOfEntryPoint 转化为 16 进制输出
lea ebx, content
add ebx, e_lfnw
mov eax, 0
mov eax, DWORD PTR [ebx+28h]
invoke dw2hex, eax, ADDR output
```

```
invoke StdOut, ADDR str8
invoke StdOut, ADDR output
```

```
;ImageBase 转化为 16 进制输出
lea ebx, content
add ebx, e_lfnw
mov eax, 0
mov eax, DWORD PTR [ebx+34h]
```

```
invoke dw2hex, eax, ADDR output

invoke StdOut, ADDR str9
invoke StdOut, ADDR output

;SectionAlignment 转化为 16 进制输出
lea ebx, content
add ebx, e_lfnew
mov eax, 0
mov eax, DWORD PTR [ebx+38h]
invoke dw2hex, eax, ADDR output

invoke StdOut, ADDR str10
invoke StdOut, ADDR output

;FileAlignment 转化为 16 进制输出
lea ebx, content
add ebx, e_lfnew
mov eax, 0
mov eax, DWORD PTR [ebx+3Ch]
invoke dw2hex, eax, ADDR output

invoke StdOut, ADDR str11
invoke StdOut, ADDR output

;调用函数 CloseHandle 关闭句柄
invoke CloseHandle, hFile
invoke ExitProcess, 0
END start
```