实验：PE文件结构分析实验

一、 实验目的

1. 掌握PE文件基本结构
2. 掌握PE文件分析常用工具的使用

二、实验环境

操作系统：Windows 7/8/10

实验对象：PE文件（实验主机随机选取，如notepad.exe）

实验软件：HxD，ExeinfoPE，PEViewer，LordPE，PEiD

三、实验内容

1. 实验背景

PE文件是Windows操作系统中可执行的程序文件。PE是Portable Executable的缩写，意为“可移植的、可执行的”，表示一个可执行文件可以在多种操作系统中运行。PE文件格式是在UNIX可执行文件COFF（Common Object File Format，通用对象文件格式）基础上创建而成的。常见的EXE，DLL，OBJ，SYS等都是PE文件。

许多恶意程序都是PE文件，拥有PE文件结构，因此学习和掌握PE文件结构是深入学习恶意代码分析课程的基础知识储备。

2.实验内容

PE文件是二进制格式文件，具有复杂的格式定义，每个字节都赋予了特定的含义。为了方便描述，通常会借助C语言中结构体定义的方式来说明每个有意义的字段。

PE文件格式为了兼容DOS时代的可执行程序，因此在PE文件头部使用了完整的DOS头部数据结构，紧跟在DOS头后面就是PE数据结构。

PE头部数据结构可以分成两个部分，一个是PE头，另一个是PE节区。

PE头中包含文件执行时所需的一般信息，包括执行文件时最初执行代码的起始部分信息，驱动应用程序的平台信息等。

PE节区中包含组成程序的汇编代码，通常恶意代码分析中逆向分析的对象；源代码中声明的全局变量与static变量；以及程序中使用的图片、文档文件等资源。

PE文件结构如图 1所示。本次实验需要使用常用的PE文件查看分析软件工具，对Windows中的PE文件进行分析，查找主要的组成结构，对关键的数据，如IMAGE_DOS_HEADER和IMAGE_NT_HEADER结构体的数据进行查看，了解掌握关键字段，如**e_lfanew**，PE标志字段.....

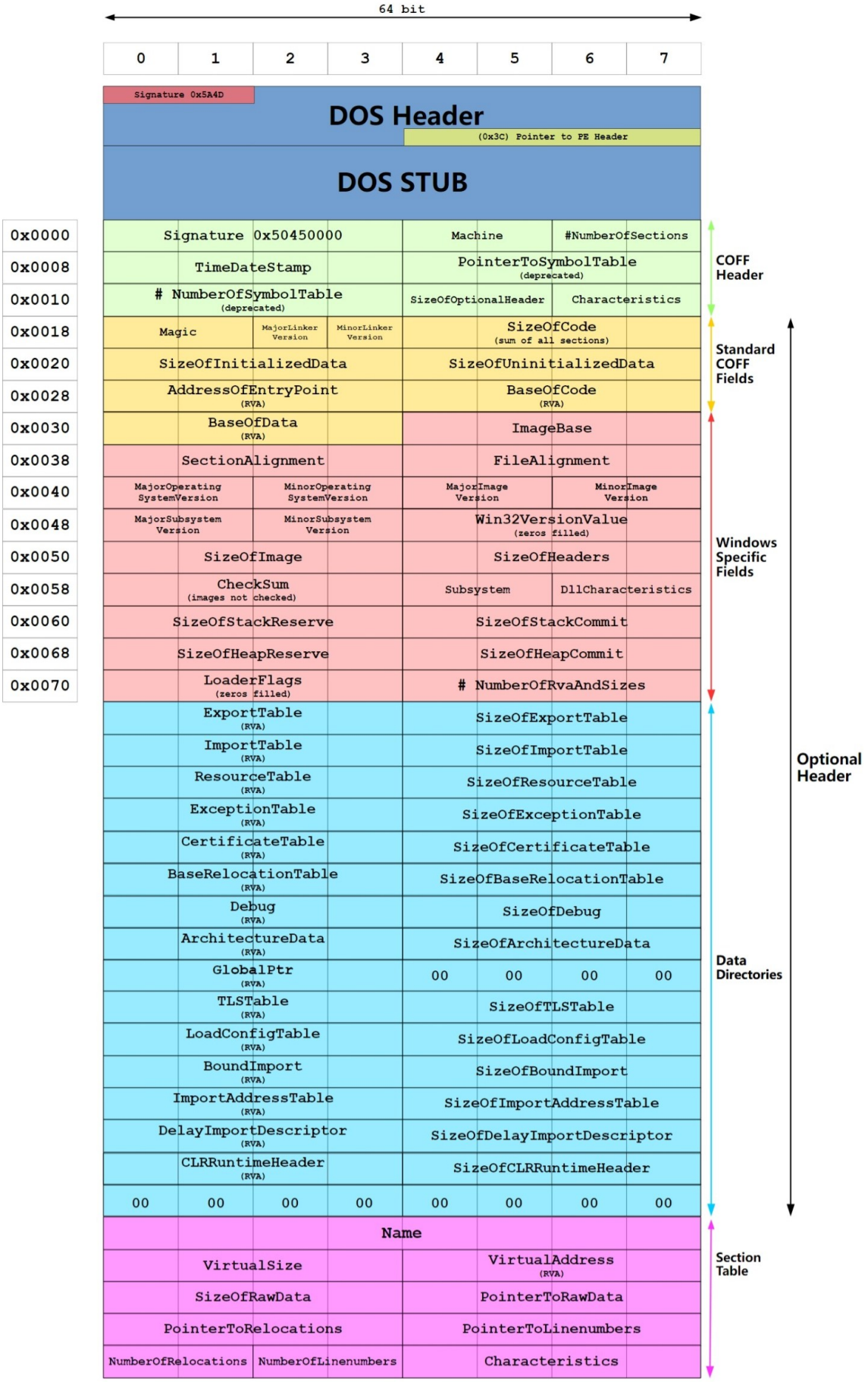

图1 PE文件结构

> 关于PE结构，还可以参考PE文件格式图解，PE文件结构样例图解，PE格式微软定义

四、实验步骤

1. 观察DOS头部数据

DOS头的作用是兼容MS-DOS操作系统中的可执行文件，对于32位PE文件来说，DOS所起的作用就是显示一行文字，提示用户：我需要在32位Windows上才可以运行，通常是显示为：“This program cannot be run in DOS mode”。

为了能清晰的说明DOS头的结构，通常会使用C语言结构体定义的方式表示。

```
typedef struct _IMAGE_DOS_HEADER {      // DOS .EXE header
    WORD   e_magic;                     // Magic number: 0x5a4d
    WORD   e_cblp;                      // Bytes on last page of file
    WORD   e_cp;                        // Pages in file
    WORD   e_crlc;                      // Relocations
    WORD   e_cparhdr;                   // Size of header in paragraphs
    WORD   e_minalloc;                  // Minimum extra paragraphs needed
    WORD   e_maxalloc;                  // Maximum extra paragraphs needed
    WORD   e_ss;                        // Initial (relative) SS value
    WORD   e_sp;                        // Initial SP value
    WORD   e_csum;                      // Checksum
    WORD   e_ip;                        // Initial IP value
    WORD   e_cs;                        // Initial (relative) CS value
    WORD   e_lfarlc;                    // File address of relocation table
    WORD   e_ovno;                      // Overlay number
    WORD   e_res[4];                    // Reserved words
    WORD   e_oemid;                     // OEM identifier (for e_oeminfo)
    WORD   e_oeminfo;                   // OEM information; e_oemid specific
    WORD   e_res2[10];                  // Reserved words
    LONG   e_lfanew;                    // File address of new exe header 0x3c
  } IMAGE_DOS_HEADER, *PIMAGE_DOS_HEADER;
```

以上定义的结构体中，第一个字段，e_magic，是2字节宽度的整数，值是常数0x4D5A，用文本编辑器查看该值为‘MZ’，如图 2使用“HxD”软件打开操作系统自带的“画图”（c:\windows\SysWoW64\mspaint.exe）程序文件开头显示，需要说明的是配图是在Windows7 32位环境下打开的文件，所以显示路径是system32文件夹，但现在Windows10 64位实验环境下，需要更改为SysWow64文件夹中。以下所有文档配图均按照这个规则查看。

> 本课程实验分析环境为Windows64位，如果分析32位系统自带的程序，例如记事本（notepad.exe）就应该打开`C:\Windows\SysWoW64\notepad.exe`，而不能打开System32文件夹内的64位程序版本。

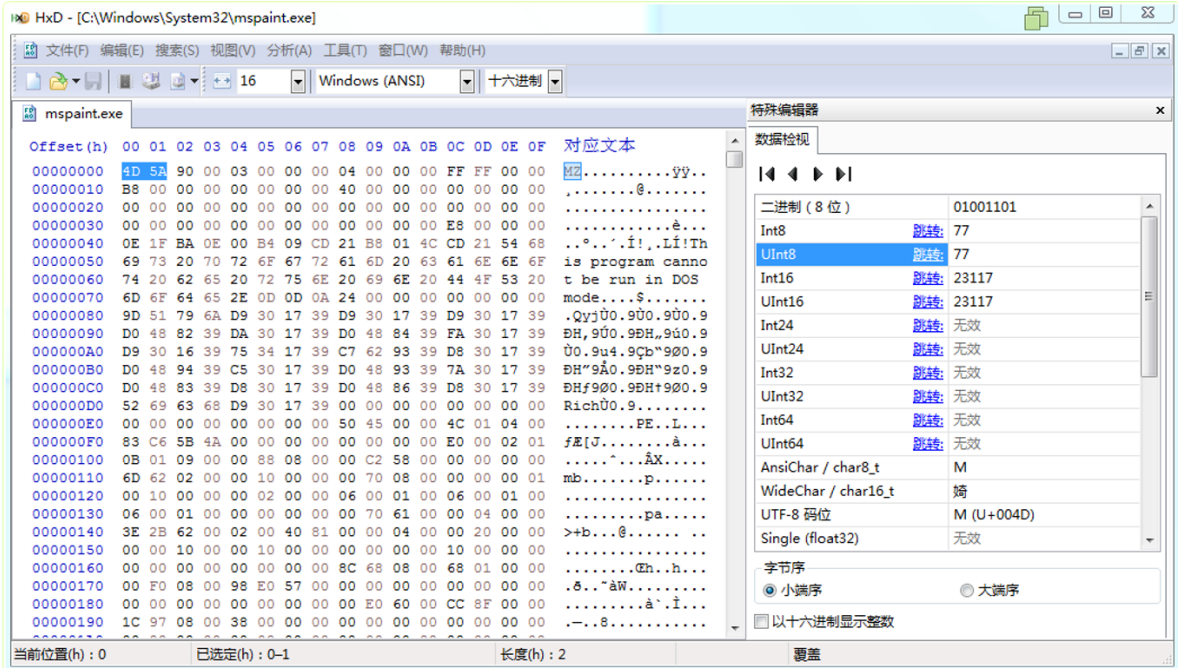

图2 mspaint.exe的e_magic字段

IMAGE_DOS_HEADER结构最后一个字段e_lfanew是4字节数据，用来表示DOS头之后的PE头的位置，即相对文件起始地址的偏移位置。**通过IMAGE_DOS_HEADER结构体各字段的长度可以计算出e_lfanew字段是位于文件头偏移0x3c位置**，图 2中其值为0x000000E8，这里请特别注意长度为4字节的数据，在文件中存储使用了小端存储，也就是32位数据中低位先存储，这样E8会先存储在0x3c位置，而且3个字节的0会以此向后保存。这是在文件中阅读数据时需要注意的。

> 通常的PE文件DOS头结构是稳定的，所以0x3C这个数据可以记忆下来，通过这个文件的偏移，能够快速定位PE头位置。
>
> PE文件中数组是按照低地址到高地址查看，其他数据都是按照小端存储查看，特别需要注意的是如果是整数数字，那么元素之间是按照从低地址到高地址，但是每个元素自身的4个字节是按照小端存储，所以需要倒过来看！

2. 查看PE头数据

PE头也称为NT头，微软公司在其开发文档winnt.h头文件中进行了如下定义。

```
typedef struct _IMAGE_NT_HEADERS {
    DWORD Signature;                            // Signature == 0x00004550,  4字节
    IMAGE_FILE_HEADER FileHeader;               // 20（0x14）字节
    IMAGE_OPTIONAL_HEADER32 OptionalHeader; // 96 + 8x16 = 224(0xE0)
} IMAGE_NT_HEADERS32, *PIMAGE_NT_HEADERS32; // 0xF8
```

PE头部也存在类似DOS头部的标记，直接翻译过来就是签名，也是一个常数，即 4字节的字符串"PE\0\0"，如图 3显示了画图程序的签名标记。

> 从以上定义的NT头结构分析可以得出结论，以NT头为基准，文件头(File Header)的偏移量为4，可选头(Optional Header)的偏移量为24，如果分析这两个头部数据信息，可以先找到NT头，然后定位到对应头部开始点，然后再继续分析各自头部的字段，分析起来会更方便。

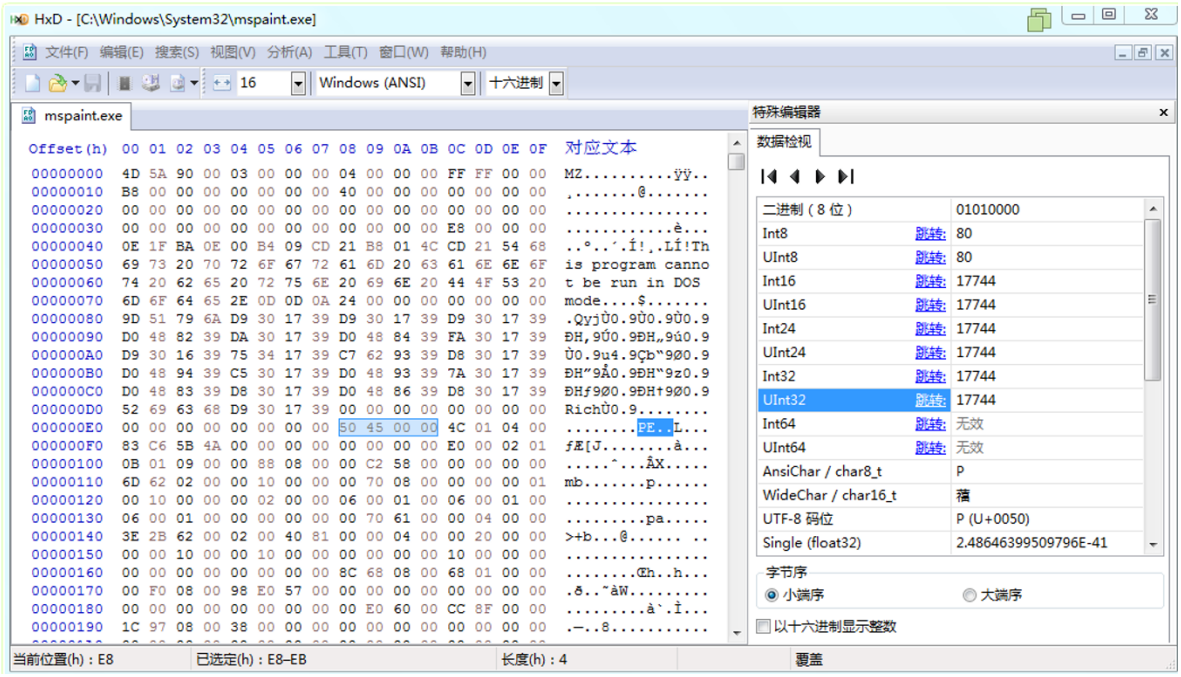

图3 PE头签名

PE头部包括了签名，文件头和选项头。除了签名，另外2个字段又分别是IMAGE_FILE_HEADER和IMAGE_OPTIONAL_HEADER32两个结构体。

```
typedef struct _IMAGE_FILE_HEADER {
    WORD    Machine;  //每个CPU都拥有的唯一的Machine码，兼容32位Intel x86芯片的Machine码为14C
    WORD    NumberOfSections; //★指出文件中存在的节区数量
    DWORD   TimeDateStamp;
    DWORD   PointerToSymbolTable;
    DWORD   NumberOfSymbols;
    WORD    SizeOfOptionalHeader; //指出结构体IMAGE_OPTIONAL_HEADER32（32位系统）的长度★
    WORD    Characteristics; //标识文件属性，是否可运行、是否为DLL等，以bit OR形式进行组合
} IMAGE_FILE_HEADER, *PIMAGE_FILE_HEADER;

typedef struct _IMAGE_OPTIONAL_HEADER {
    WORD     Magic;  //IMAGE_OPTIONAL_HEADER32为0x10B，IMAGE_OPTIONAL_HEADER64为0x20B
    BYTE     MajorLinkerVersion;
    BYTE     MinorLinkerVersion;
    DWORD    SizeOfCode;
    DWORD    SizeOfInitializedData;
    DWORD    SizeOfUninitializedData;
    DWORD    AddressOfEntryPoint; //★RVA值，指出程序最先执行的代码起始地址
    DWORD    BaseOfCode;
    DWORD    BaseOfData;
    DWORD    ImageBase;  //★指出文件的优先装入地址（32位进程虚拟内存范围为：0～7FFFFFFF）
    DWORD    SectionAlignment; //★节区在内存中的最小单位
    DWORD    FileAlignment;    //★节区在磁盘文件中的最小单位
    WORD     MajorOperatingSystemVersion;
    WORD     MinorOperatingSystemVersion;
    WORD     MajorImageVersion;
    WORD     MinorImageVersion;
    WORD     MajorSubsystemVersion;
```

```
    WORD     MinorSubsystemVersion;
    DWORD    Win32VersionValue;
    DWORD    SizeOfImage;  //指定了PE Image在虚拟内存中所占空间的大小
    DWORD    SizeOfHeaders;
    DWORD    CheckSum;
    WORD     Subsystem;   //区分系统驱动文件和普通可执行文件
    WORD     DllCharacteristics;
    DWORD    SizeOfStackReserve;
    DWORD    SizeOfStackCommit;
    DWORD    SizeOfHeapReserve;
    DWORD    SizeOfHeapCommit;
    DWORD    LoaderFlags;
    DWORD    NumberOfRvaAndSizes;   //指定Data Directory数组的大小
    IMAGE_DATA_DIRECTORY DataDirectory[IMAGE_NUMBEROF_DIRECTORY_ENTRIES]; //★数
据目录数组
} IMAGE_OPTIONAL_HEADER32, *PIMAGE_OPTIONAL_HEADER32;
```

> 以上定义中，IMAGE_FILE_HEADER头部中SizeOfOptionalHeader字段给出了可选头部的长度信息，这将会方便二进制级别的分析中，快速定位节表，因为可选头后面紧跟的就是节表Section Table。SizeOfOptionalHeader位于NT头部20字节偏移位置。
>
> 另外，以上定义中DataDirectory（距离NT头偏移120字节）是可选头中的重要数据，但数组大小由IMAGE_OPTIONAL_HEADER.NumberOfRvaAndSizes确定，因为次字段在DataDirectory前面4字节，所以距离NT头部116字节。
>
> AddressOfEntryPoint距离NT头40字节。

通过以上的结构定义可以了解PE文件头由IMAGE_NT_HEADERS，IMAGE_FILE_HEADER和IMAGE_OPTIONAL_HEADER结构体定义其内部的数据含义。具体含义可以通过查询微软的官方文档获得。

参考地址为：https://docs.microsoft.com/en-us/windows/win32/api/winnt/ns-winnt-image_file_header

这里对其中关键的几个属性进行说明（请重点关注标注★符号）：

（1）Machine 计算机架构。

该字段表示PE文件执行的计算机架构环境，0x014c表示x86架构，而0x8664表示x64计算机架构环境。

（2）NumberOfSections ★节数

该字段表示在PE头部后面紧跟的“节表”的大小，也就是“节数”，在Windows环境下最大节数为96。

（3）SizeOfOptionalHeader ★可选头的大小，单位是字节。

（4）Characteristics 镜像特性

该字段表示了文件镜像的一些属性特点，使用表格 1中列出了各种不同属性含义及其对应代码，多个属性可以使用“按位或”的方式计算得到最终值。反过来说，可以通过对Characteristics值使用按位或的计算，通过是否结果为零的方式判断是否具有对应属性。

表格 1 Characteristics含义

标志	值	说明
IMAGE_FILE_RELOCS_STRIPPED	0x0001	纯映像、Windows CE 和 Microsoft Windows NT 及更高版本。 这表示该文件不包含基填重定位，因此必须加载到其首选基址。 如果基址不可用，则加载程序将报告错误。 链接器的默认行为是从可执行文件 (EXE) 文件中去除基址重定位。
IMAGE_FILE_EXECUTABLE_IMAGE	0x0002	纯映像。 这表示映像文件有效并且可以运行。 如果未设置此标志，则表示有链接器错误。
IMAGE_FILE_LINE_NUMS_STRIPPED	0x0004	COFF 行号已删除。 此标志已被弃用，应为零。
IMAGE_FILE_LOCAL_SYMS_STRIPPED	0x0008	本地符号的 COFF 符号表条目已删除。 此标志已被弃用，应为零。
IMAGE_FILE_AGGRESSIVE_WS_TRIM	0x0010	已过时。 主动剪裁工作集。 已针对 Windows 2000 及更高版本弃用此标志，此标志必须为零。
IMAGE_FILE_LARGE_ADDRESS_AWARE	0x0020	应用程序可以处理 > 2 GB 地址。
	0x0040	此标志将保留以供将来使用。
IMAGE_FILE_BYTES_REVERSED_LO	0x0080	Little endian：最低有效位 (LSB) 位于内存中最高有效位 (MSB) 之前。 此标志已被弃用，应为零。
IMAGE_FILE_32BIT_MACHINE	0x0100	计算机基于 32 位字体系结构。
IMAGE_FILE_DEBUG_STRIPPED	0x0200	从映像文件中删除了调试信息。
IMAGE_FILE_REMOVABLE_RUN_FROM_SWAP	0x0400	如果映像位于可移动媒体上，请完全加载该映像并将其复制到交换文件。
IMAGE_FILE_NET_RUN_FROM_SWAP	0x0800	如果映像位于网络媒体上，请完全加载该映像并将其复制到交换文件。
IMAGE_FILE_SYSTEM	0x1000	映像文件是系统文件，而不是用户程序。
IMAGE_FILE_DLL	0x2000	图像文件是动态链接库 (DLL)。 虽然无法直接运行此类文件，但这些文件被视为几乎适用于所有用途的可执行文件。
IMAGE_FILE_UP_SYSTEM_ONLY	0x4000	该文件应仅在单处理器计算机上运行。

标志	值	说明
IMAGE_FILE_BYTES_REVERSED_HI	0x8000	Big endian：MSB 在内存中的 LSB 之前。此标志已被弃用，应为零。

（5）Magic魔术字

该字段表示了文件的状态，可以使用表格 2查询其具体含义。

表格 2 Magic字段含义

标识符	值	含义解释
IMAGE_NT_OPTIONAL_HDR_MAGIC		可执行文件映像。32位程序值为 0x10b；64位程序值为 0x20b
IMAGE_NT_OPTIONAL_HDR32_MAGIC	0x10b	可执行文件映像。
IMAGE_NT_OPTIONAL_HDR64_MAGIC	0x20b	可执行文件映像。
IMAGE_ROM_OPTIONAL_HDR_MAGIC	0x107	ROM映像。

（6）AddressOfEntryPoint 入口地址★★★

相对于PE文件加载进入内存中，代码的起始入口地址，这个地址使用了相对地址，即相对于Image base地址的偏移地址。

（7）BaseOfCode 代码节入口地址，该地址是代码节的开始位置，使用了相对地址表示。

（8）ImageBase 镜像基地址★★

镜像基地址就是以上叙述中提及的Image base地址，即当PE文件加载到内存中，PE文件内存中开始位置的地址。通常在PE头部中的各类地址都是相对这个地址的偏移量，所以这是其他地址的基础，称为基地址。这个地址是64K的整倍数，通常默认Windows可执行文件（EXE）使用0x00400000,动态链接库（DLL）使用0x10000000。

（9）SectionAlignment（节对齐）和FileAlignment（文件对齐）

前者制定了节区在内存中的最小单位，后者制定了节区在磁盘文件中的最小单位。节对齐和文件对齐分别是PE文件中数据在内存和文件中数据存储的分块尺寸规定，无论是PE文件数据加载到内存中还是保存在文件中，都需要按照一定的尺寸划分存储块，默认情况下，加载到内存中的“节”的数据必须按照0x1000大小存放，这实际上就是操作系统默认的内存页面的大小（4K）；而相同的数据如果保存在文件中，那么需要按照FileAlignment大小保存，这个数值通常在512 到64K之间的2的整数次幂，默认值为512，即0x200。

（10）Subsystem 镜像运行的子系统环境

由于PE文件设计的时候是Windows NT开发的年代，当时考虑到底层对当时主流的操作系统的兼容，因此使用了子系统的方式进一步区别PE运行的环境。可以使用表格 3中查询对应的运行环境。

表格 3 Subsystem取值表

常量	值	说明
IMAGE_SUBSYSTEM_UNKNOWN	0	未知子系统
IMAGE_SUBSYSTEM_NATIVE	1	设备驱动程序和本机 Windows 进程
IMAGE_SUBSYSTEM_WINDOWS_GUI	2	Windows 图形用户界面 (GUI) 子系统
IMAGE_SUBSYSTEM_WINDOWS_CUI	3	Windows 字符子系统
IMAGE_SUBSYSTEM_OS2_CUI	5	OS/2 字符子系统
IMAGE_SUBSYSTEM_POSIX_CUI	7	Posix 字符子系统
IMAGE_SUBSYSTEM_NATIVE_WINDOWS	8	本机 Win9x 驱动程序
IMAGE_SUBSYSTEM_WINDOWS_CE_GUI	9	Windows CE
IMAGE_SUBSYSTEM_EFI_APPLICATION	10	可扩展固件接口 (EFI) 应用程序
IMAGE_SUBSYSTEM_EFI_BOOT_ SERVICE_DRIVER	11	具有启动服务的 EFI 驱动程序
IMAGE_SUBSYSTEM_EFI_RUNTIME_ DRIVER	12	具有运行时服务的 EFI 驱动程序
IMAGE_SUBSYSTEM_EFI_ROM	13	EFI ROM 映像
IMAGE_SUBSYSTEM_XBOX	14	XBOX
IMAGE_SUBSYSTEM_WINDOWS_BOOT_APPLICATION	16	Windows 启动应用程序。

（11）DataDirectory 数据目录★

```
typedef struct _IMAGE_DATA_DIRECTORY {
    DWORD   VirtualAddress;
    DWORD   Size;
} IMAGE_DATA_DIRECTORY, *PIMAGE_DATA_DIRECTORY;
```

PE头部中的一些关键的数据使用了“表格”的方式存放，DataDirectory的值指向了这个表格首地址。表格中数据每一行使用IMAGE_DATA_DIRECTORY结构体定义表格列，包括了一个地址(VirtualAddress)信息和所指向内存的大小(Size)信息。虽然每一行都是IMAGE_DATA_DIRECTORY结构体，但是每一行都预先定义好了PE文件中特殊的数据含义。表格 4列出了每一行的含义，第一行是导出表目录，第二行是导入表目录……。

关于DataDirectory特别需要注意以下2点

① DataDirectory中每一行实际上是指向另一个表格的地址，如第0行IMAGE_DIRECTORY_ENTRY_EXPORT是导出表的首地址。

② DataDirectory使用表格 4指明了各行的数据含义，但是实际上DataDirectory最后一行是用于保留定义的，没有出现在表格 4的定义中，因此实际上DataDirectory的总行数最大值应该是16，实际表格行数会保存在NumberOfRvaAndSizes字段中，而因此DataDirectory所占用空间最大为8×16=0x80。

表格 4数据目录表各行含义

索引	偏移量 (PE/PE32+)	说明
0	96/112	导出表地址和大小 Export Table
1	104/120	导入表地址和大小 Import Table
2	112/128	资源表地址和大小 Resource Table
3	120/136	异常表地址和大小 Exception Table
4	128/144	证书表地址和大小 Certificate Table
5	136/152	基重定位表地址和大小 Base Relocation Table
6	144/160	调试信息起始地址和大小 Debug Table
7	152/168	特定于体系结构的数据地址和大小 Architecture Table
8	160/176	全局指针注册相对虚拟地址 Global Pointer Table
9	168/184	线程本地存储 (TLS) 表地址和大小 Thread Local Storage Table
10	176/192	加载配置表地址和大小 Load Configuration Table
11	184/200	绑定导入表地址和大小 Bound Import Table
12	192/208	导入地址表地址和大小 Import Address Table, IAT
13	200/216	延迟导入描述符地址和大小 Delay Import Table
14	208/224	CLR 标头地址和大小
15	216/232	预留

注意：偏移量PE32+是64位版本PE规范，以上偏移都是以NT头为基准。

3. 节区头和节区（Sections）

PE文件中的数据是按照节（Section）进行管理存储的，一般有代码节区和数据节区等，节区的名称和个数都是可变的，因此通常需要从节区头中获得以上节的相关信息。

节区头紧跟在PE头后面，因此可以通过计算获得其地址。节区头中定义了各节区的属性，包括不同的特性、访问权限等，结构体为IMAGE_SECTION_HEADER，重要成员有5个，以下使用代码注释标出。

```
typedef struct _IMAGE_SECTION_HEADER {
    BYTE    Name[IMAGE_SIZEOF_SHORT_NAME];
    union {
            DWORD   PhysicalAddress;
            DWORD   VirtualSize;      //内存中节区所占大小
    } Misc;
    DWORD   VirtualAddress;           //内存中节区起始地址（RVA）
    DWORD   SizeOfRawData;            //磁盘文件中节区所占大小
```

```
    DWORD   PointerToRawData;       //磁盘文件中节区的偏移地址
    DWORD   PointerToRelocations;
    DWORD   PointerToLinenumbers;
    WORD    NumberOfRelocations;
    WORD    NumberOfLinenumbers;
    DWORD   Characteristics;        //节区属性（bit OR）
} IMAGE_SECTION_HEADER, *PIMAGE_SECTION_HEADER;
```

节区头从本质上就是一个表格，表格的每一行是一个IMAGE_SECTION_HEADER结构体，表格长度就是PE头中**NumberOfSections**字段中的数值。

在Windows7操作系统画图程序（mspaint.exe）程序，使用HxD打开后，通过前面的分析，可以确定节区头的位置是*0xE8* +（**0x04+0x14+0xE0**）= 0x1E0,等式左边分别是*DOS头长度*，**PE签名长度，PE文件头长度和PE可选头长度(这3部分实际上是PE头长度)**，因此在文件0x1E0位置开始就是第一个IMAGE_SECTION_HEADER结构体，如图 4。

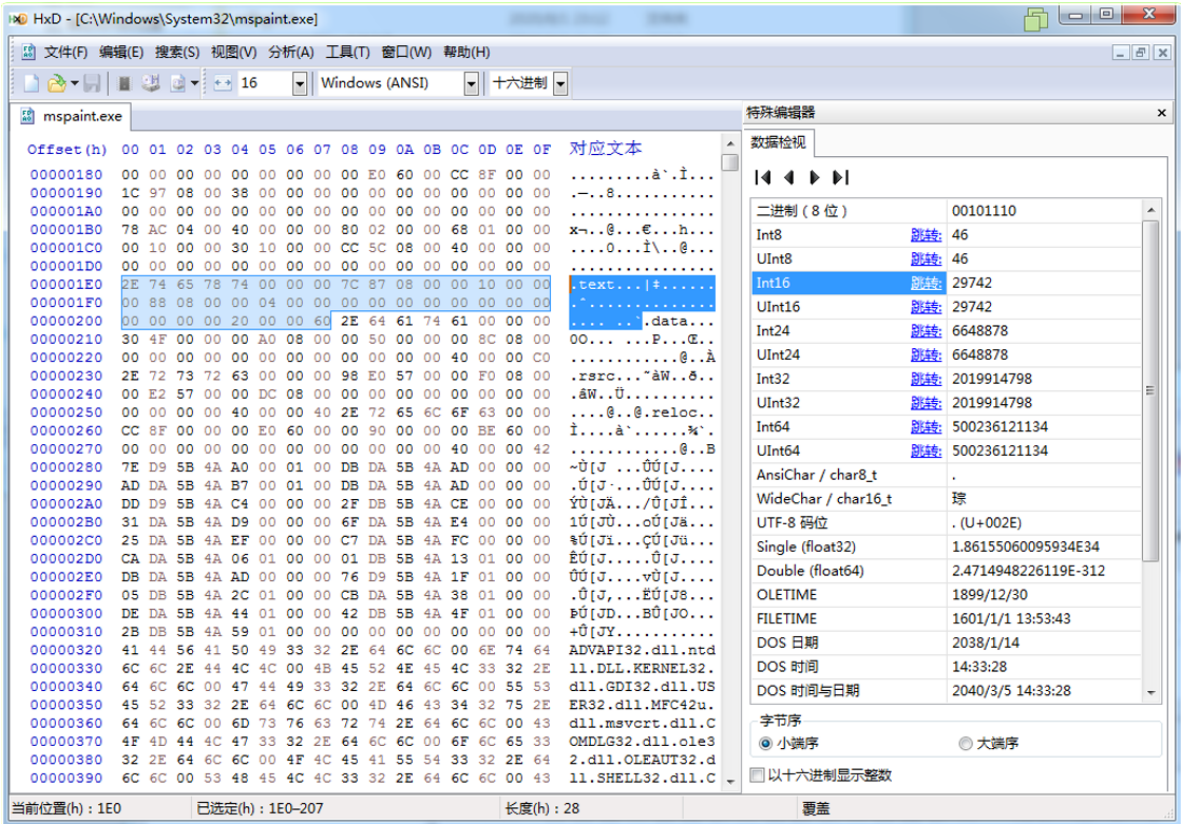

图 4 节区头的第一条记录

使用以上的方式获取节表的数据不够方便，因此实际工作中，都是使用工具软件读取，典型的可以使用ExeInfo PE，和x64dbg PE view插件等查阅。

4. 使用其他工具高效的查看以上实验步骤中的各类数据

通过前面的实验步骤，可以发现虽然直接使用二进制分析工具能从字节底层对PE文件进行分析，但是工作效率不高，如果仅仅需要查询某个字段的数据，可以使用更方便的专用工具。例如Exeinfo PE，PE Viewer，LordPE，PEiD……，这些工具都能快速的分析PE文件。

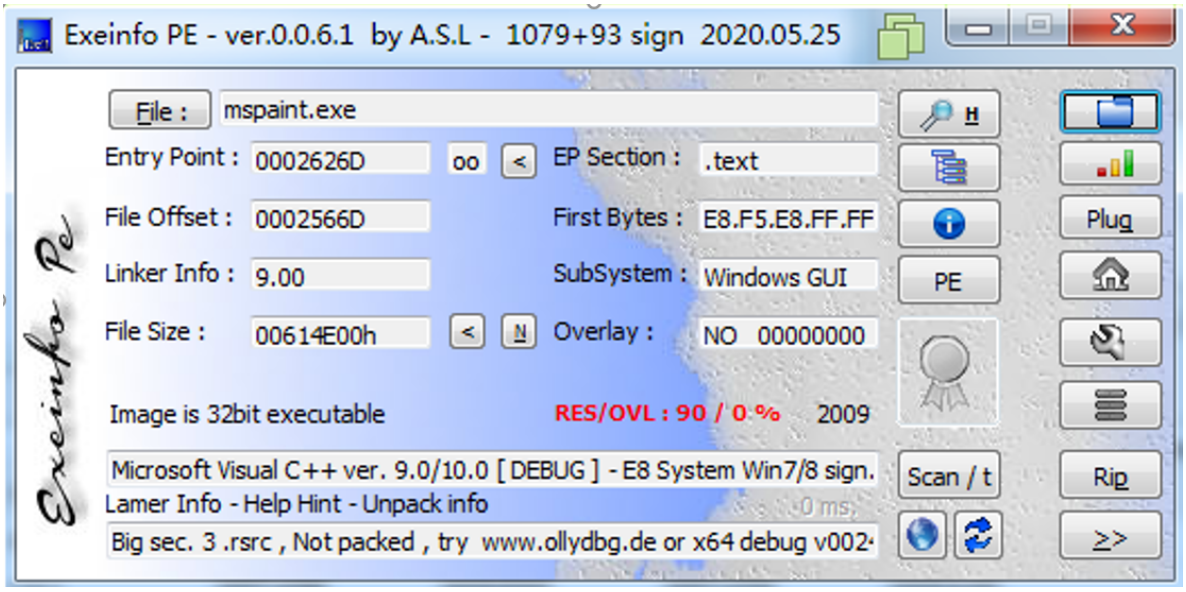

图5 使用Exeinfo PE打开画图程序

使用Exeinfo PE工具能方便的查看PE头中的各类信息，如图 6可以清晰的查看头部的各项数据。

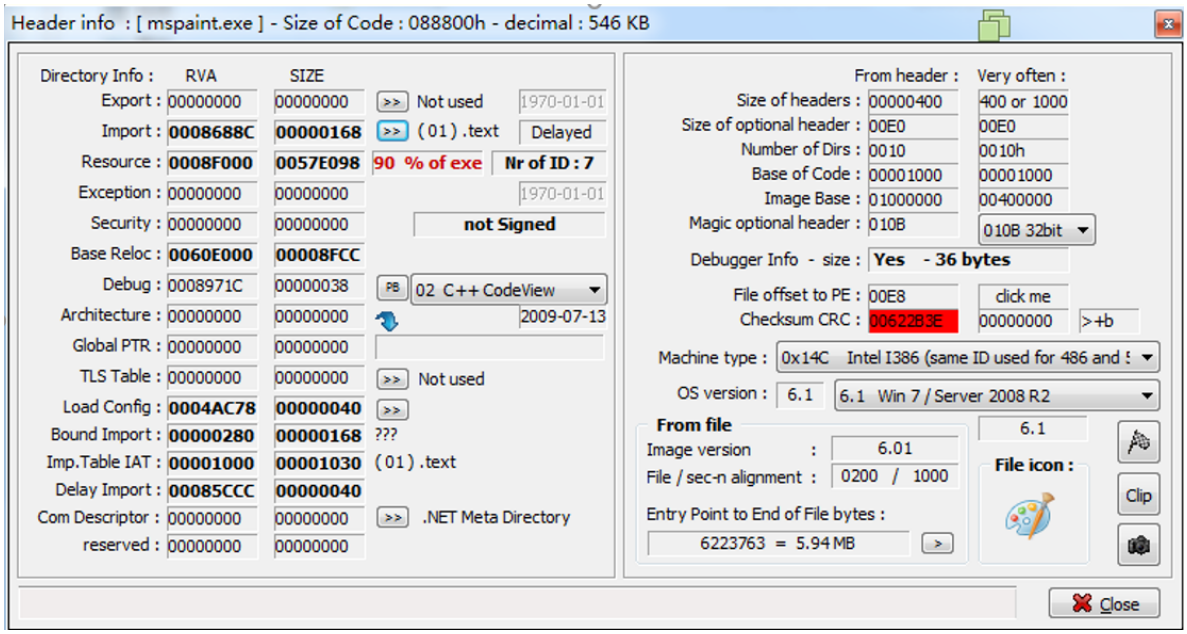

图 6 Exeinfo PE查看头部信息

也可以方便的查看节表中的信息，如图 7所示。

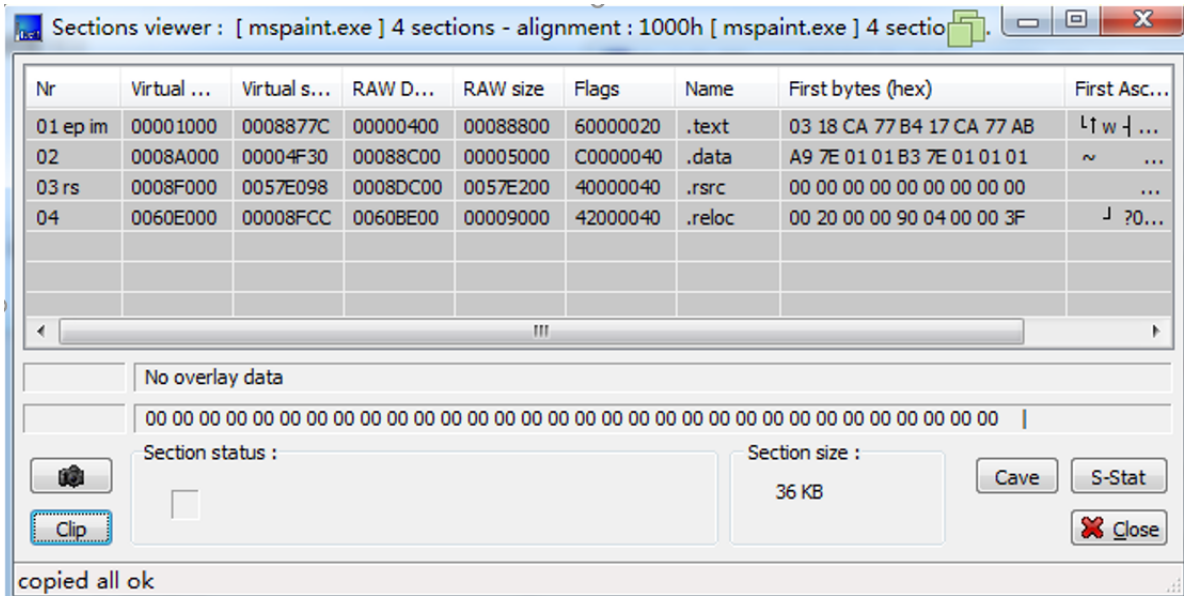

图 7 查看节表信息

在实践工作中，x32dbg工具的PE Viewer插件能比较直观的显示PE头部的各类数据，同学们也可以尝试使用。

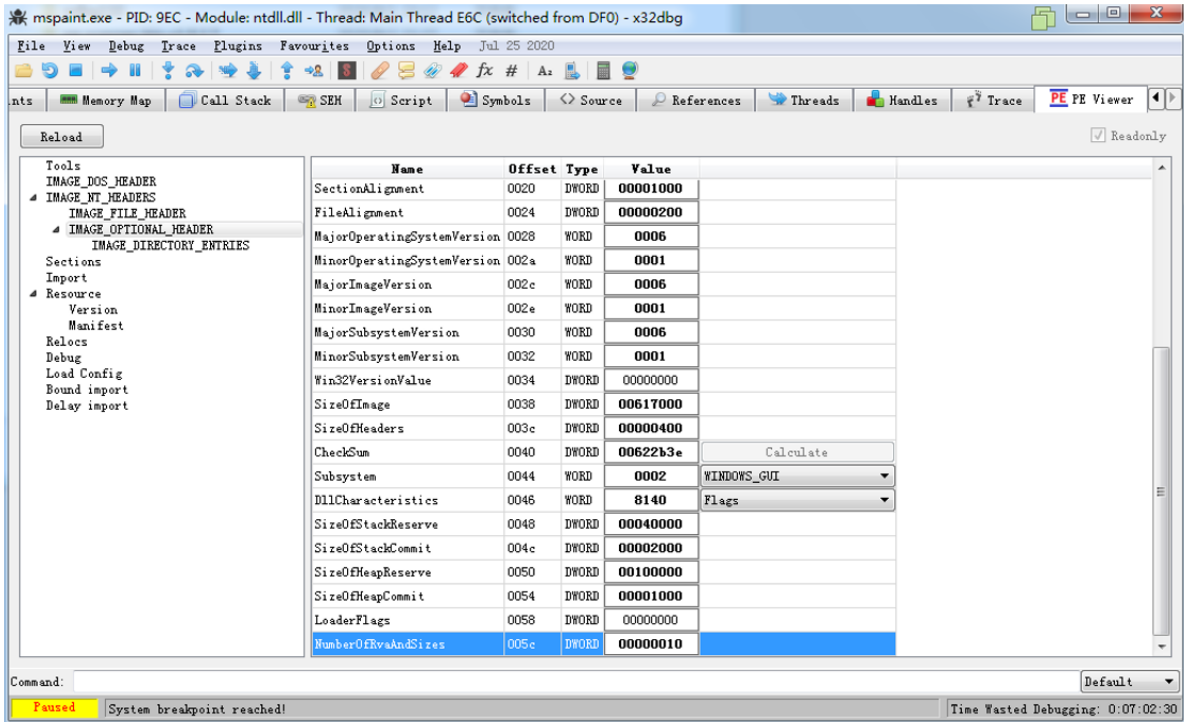

图 8 使用x32dbg工具查看PE可选头部信息

如图 8的界面中，使用x32dbg工具的PE Viewer插件能直观的查看PE文件中的数据字段，其界面左侧的树状导航窗格给出了PE文件的基本数据架构，单击其中数据项，右侧就能以表格的方式显示具体相关的数据，使用起来非常方便。此外x32dbg是x64dbg动态调试工具的32位版本，是今后学习中动态调试PE文件的重要工具。

五、实验结果

1. 请使用HxD打开Windows操作系统自带的记事本程序（c:\Windows\SysWOW64\notepad.exe），完成以下的实验内容。

（1）查找DOS头部e_lfanew字段的值（　　），并提供截图。

（2）查找PE头部AddressOfEntryPoint字段值（　　），并提供截图。

（3）查找PE头部Characteristics，并解析出其含义：（　　）

（4）程序加载到内存中有（　）个节，其中代码节（即.text节）的RAV地址是（　　）

2. 使用Exeinfo PE和x32dbg工具完成题目1中内容。
3. 其他实验中的问题和解决结论思考等。

六、思考题

尝试分析一个Windows下64位可执行程序，如Windows 10操作系统中的notepad.exe文件。请独自查阅资料完成，并给出关键的不同之处。（可以参考微软官方文档，如https://docs.microsoft.com/en-us/windows/win32/api/winnt/ns-winnt-image_optional_header32）