PE结构总览

PE结构总览

```
IMAGE_DOS_HEADER
IMAGE_NT_HEADERS
  |-- IMAGE_FILE_HEADER
  |-- IMAGE_OPTIONAL_HEADER
        |-- IMAGE_DATA_DIRECTORY
              |-- IMAGE_DIRECTORY_ENTRY_EXPORT      // index 0 导出表目录
              |-- IMAGE_DIRECTORY_ENTRY_IMPORT      // index 1 导入表目录
                    |-- IMAGE_IMPORT_DESCRIPTOR
                          |-- IMAGE_THUNK_DATA
                                |-- IMAGE_IMPORT_BY_NAME
              |-- IMAGE_DIRECTORY_ENTRY_RESOURCE        // index 2 资源
                    |-- IMAGE_RESOURCE_DIRECTORY
                          |-- IMAGE_RESOURCE_DIRECTORY_ENTRY
                                |-- RT_ICON // 宏
                          |-- IMAGE_RESOURCE_DATA_ENTRY
              |-- IMAGE_DIRECTORY_ENTRY_BASERELOC       // index 5 重定位
                    |-- IMAGE_BASE_RELOCATION
              |-- IMAGE_DIRECTORY_ENTRY_TLS         // index 9 线程局部存储
                    |-- IMAGE_TLS_DIRECTORY
             |-- IMAGE_DIRECTORY_ENTRY_IAT          // index 12 导入地址表
IMAGE_SECTION_HEADER
...
IMAGE_SECTION_HEADER
SECTION_DATA
...
SECTION_DATA
user data   // 附加数据，用户自定义（安全领域）
```

IMAGE_DOS_HEADER

```
#define IMAGE_DOS_SIGNATLURE    0x5A4D  // MZ

typedef struct _IMAGE_DOS_HEADER {
    WORD e_magic;       √   //  Magic number
    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;              //  Inital (relative) SS value
    WORD e_sp;              //  Inital SP value
    WORD e_csum;            //  Checksum
    WORD e_ip;              //  Inital IP value
    WORD e_cs;              //  Inital (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 headers
} IMAGE_DOS_HEADER, *PIMAGE_DOS_HEADER;
```

DOS头一共**64byte**，其中第一个字段`e_magic`和最后一个字段`e_lfanew`最为重要

- `e_magic`
  DOS头标志位，其值恒为**0x5A4D** (“MZ”，系统中可用宏`IMAGE_DOS_SIGNATURE`)
- `e_lfanew`
  表示NT头部在文件中的偏移，(其NT头位置与DOS头中间间隔为“垃圾数据”，可自行填充)

IMAGE_NT_HEADERS

```
#define IMAGE_NT_SIGNATURE  0x00004550

typedef struct _IMAGE_NT_HEADERS {
    DWORD Signature;                            √   //  A 4-byte signature
identifying the file as a PE image. The bytes are "PE\0\0".
    IMAGE_FILE_HEADER FileHeader;               √   //  An IMAGE_FILE_HEADER
structure that specifies the file header.
    IMAGE_OPTIONAL_HEADER32 OptionalHeader;     √   //  An IMAGE_OPTIONAL_HEADER
structure that specifies the optional file header.
} IMAGE_NT_HEADERS32, *PIMAGE_NT_HEADERS32
```

三个字段都重要，第一个为标志，后两个为嵌套的结构体

- `Signature`
  类似DOS头的`e_magic`字段，其值为**0x00004550**，ASCII码为**“PE\0\0”**（系统中可用宏`IMAGE_NT_SIGNATURE`）
- `FileHeader`
  IMAGE_FILE_HEADER结构，存储着PE文件的基本信息

  ```
  typedef struct _IMAGE_FILE_HEADER {
      WORD Machine;                   √    //  Architecture type of the
  computer
      WORD NumberOfSections;          √    //  The number of sections
      DWORD TimeDateStamp;            √    //  The low 32 bits of the time
  stamp of the image
      DWORD PointerToSymbolTable;          //  The offser of the symbol
  table
      DWORD NumberOfSymbols;               //  The number of symbols in the
  symbol table
      WORD SizeOfOptionalHeader;      √    //  The size of the optional header
      WORD Characteristics;           √    //  The characteristics of the image
  } IMAGE_FILE_HEADER, *PIMAGE_FILE_HEADER;
  ```

  共20 bytes

  - `Machine`
    标识运行平台

- `NumberOfSections`
  区段数量
- `TimeDateStamp`
  时间戳，文件创建的时间，可参考但不可信任
- `SizeOfOptionalHeader`
  可选头结构的大小，大小不固定
- `Characteristics`
  该可执行文件的特征属性

- `OptionalHeader`
  IMAGE_OPTIONAL_HEADER32结构，存储着PE文件加载的信息

```
#define IMAGE_NUMBEROF_DIRECTORY_ENTRIES    16

typedef struct _IMAGE_OPTIONAL_HEADER {
    WORD Magic;
                √   //  The state of the image file
    BYTE MajorLinkerVersion;
                    //  The major version number of the linker
    BYTE MinorLinkerVersion;
                    //  The minor version number of the linker
    DOWRD SizeOfCode;
                √   //  The size of the code section
    DWORD SizeOfInitializedData;
               √   //  The size of the initialilzed data section
    DOWRD SizeOfUninitializedData;
               √   //  The size of the uninitialized data section
    DWORD AddressOfEntryPoint;
               √   //  A pointer to the entry point function, relative to
the image base address
    DWORD BaseOfCode;
                √   //  A pointer to the beginning of the code section,
relative to the image base
    DWORD BaseOfData;
                √   //  A pointer to the beginning of the data section,
relative to the image base
    DWORD ImageBase;
               √   //  The preferred address of the first byte of the image
when it loaded in memory
    DWORD SectionAlignment;
                √   //  The alignment of sections loaded in memory
    DWORD FileAlignment;
               √   //  The alignment of the raw data of section in the image
file
    WORD MajorOperatingSystemVersion;
                    //  The major version number of the required operating
system
    WORD MinorOperatingSystemVersion;
                    //  The minor version number of the required operating
system
    WORD MajorImageVersion;
                    //  The major version number of the image
    WORD MinorImageVersion;
                    //  The minor version number of the imgae
    WORD MajorSubsystemVersion;
                    //  The major version number of the subsystem
```

```
    WORD MinorSubsystemVersion;
                    //  The minor version number of the subsystem
    DWORD Win32VersionValue;
                   //  This member is reserved and must be 0
    DWORD SizeOfImage;
              √    //  The size of the image
    DWORD SizeOfHeaders;
              √    //  The combined size of the items
    DWORD CheckSum;
               √    //  The image file checksum
    WORD Subsystem;
                    //  The subsystem required to run this image
    WORD DllCharacteristics;
                   //  The DLL characteristics of the image
    DWORD SizeOfStackReserve;
               √    //  The number of bytes to reserve for the stack
    DWORD SizeOfStackCommit;
              √    //  The number of bytes to commit for the stack
    DWORD SizeOfHeapReserve;
              √    //  The number of bytes to reserve for the local heap
    DWORD SizeOfHeapCommit;
               √    //  The number of bytes to commit for the loacl heap
    DWORD LoaderFlags;
                   //  This member is obsolete
    DWORD NumberOfRvaAndSizes;
                   //  The number of directory entries in the remainder of
the optional header
    IMAGE_DATA_DIRECTORY DataDirectory[IMAGE_NUMBEROF_DIRECTORY_ENTRIES];
                √    //  A pointer to the first IMAGE_DATA_DIRECTORY
structure in the data directory
} IMAGE_OPTIONAL_HEADER, *PIMAGE_OPTIONAL_HEADER;
```

- `Magic`
  文件状态
- `SizeofCode`
  代码区段的总大小，**说明性字段**
- `SizeOfInitializedData`
  已初始化数据的总大小，**说明性字段**
- `SizeOfUninitializedData`
  未初始化数据的总大小，在磁盘中不占用空间，加载如内存后会预留空间，一般没存储在.bss段中，说明性字段
- `AddressOfEntryPoint`
  指向入口点函数的指针，相对于基地址（也称为EP，OEP为原始入口点）。对于可执行文件这是起始地址，对于设备文件这是初始化函数地址，对于DLL入口点函数指针是可选的。当没有入口函数指针时，这个成员的值为0，**PE下数两行半**
- `BaseOfCode`
  指向代码区段开始的指针，相对于基地址，**说明性字段**
- `BaseOfData`
  指向数据区段开始的指针，相对于基地址，**说明性字段**

- `ImageBase`
  载入到内存时该image文件第一个字节的地址即基地址，这个值是64K bytes的整数倍。DLL默认为0x10000000，应用程序默认为0x00400000，除了在Windows CE它是0x00010000，建议装载地址
- `SectionAlignment`
  映射到内存中段的对齐方式，以字节为单位。该值必须大于或等于FileAlignment成员。默认值为系统的页面大小
- `FileAlignment`
  在磁盘中的段的对齐方式，以字节为单位。该值应为512到64K（含）之间的2的幂。默认值为512.如果SectionAlignment成员小于系统页面大小，则此成员必须与SectionAlignment相同
- `MajorSubsystemVersion`

  子系统版本，这个值是必须的
- `SizeOfImage`
  载入内存后image的大小（进行了内存对齐之后），操作系统依赖，必须对齐，必须不多不少
- `SizeOfHeaders`
  以下项目的总大小：（按文件对齐后）
  - `e_lfanew` member of IMAGE_DOS_HEADER
  - 4 byte `signature`
  - size of `IMAGE_FILE_HEADER`
  - size of `optional header`
  - size of all `section headers`
- `CheckSum`
  文件校验和，驱动相关，R3不检查
- `SizeOfStackReserve`
  初始化时栈的大小
- `SizeOfStackCommit`
  初始化时实际提交的栈的大小
- `SizeOfHeapReserve`
  初始化时保留的堆的大小
- `SizeOfHeapCommit`
  初始化时实际提交的堆的大小
- `DataDirectory`
  数据目录`IMAGE_DATA_DIRECTORY`结构的数组，总大小为`IMAGE_NUMBEROF_DIRECTORY_ENTRIES`，实际可用大小为`NumberOfRvaAndSizes`字段

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

// 数组按索引定义每项含义
#define IMAGE_DIRECTORY_ENTRY_EXPORT          0   // Export Directory
#define IMAGE_DIRECTORY_ENTRY_IMPORT          1   // Import Directory
#define IMAGE_DIRECTORY_ENTRY_RESOURCE        2   // Resource Directory
#define IMAGE_DIRECTORY_ENTRY_EXCEPTION       3   // Exception
Directory
```

```
#define IMAGE_DIRECTORY_ENTRY_SECURITY        4   // Security Directory
#define IMAGE_DIRECTORY_ENTRY_BASERELOC       5   // Base Relocation
Table
#define IMAGE_DIRECTORY_ENTRY_DEBUG           6   // Debug Directory
//      IMAGE_DIRECTORY_ENTRY_COPYRIGHT       7   // (X86 usage)
#define IMAGE_DIRECTORY_ENTRY_ARCHITECTURE    7   // Architecture
Specific Data
#define IMAGE_DIRECTORY_ENTRY_GLOBALPTR       8   // RVA of GP
#define IMAGE_DIRECTORY_ENTRY_TLS             9   // TLS Directory
#define IMAGE_DIRECTORY_ENTRY_LOAD_CONFIG    10   // Load Configuration
Directory
#define IMAGE_DIRECTORY_ENTRY_BOUND_IMPORT   11   // Bound Import
Directory in headers
#define IMAGE_DIRECTORY_ENTRY_IAT            12   // Import Address
Table
#define IMAGE_DIRECTORY_ENTRY_DELAY_IMPORT   13   // Delay Load Import
Descriptors
#define IMAGE_DIRECTORY_ENTRY_COM_DESCRIPTOR 14   // COM Runtime
descriptor
```

- `VirtualAddress`
  所指向表的相相对虚拟地址
- `Size`
  所指向表的大小

IMAGE_SECTION_HEADER

长度两行半

```
#define IMAGE_SIZEOF_SHORT_NAME 8

typedef struct _IMAGE_SECTION_HEADER {
  BYTE  Name[IMAGE_SIZEOF_SHORT_NAME];  √   //  An 8-byte, null-padded UTF-8
string
  union {
    DWORD PhysicalAddress;              √   //  The file address
    DWORD VirtualSize;                  √   //  The total size of the section
when loaded into memory
  } Misc;
  DWORD VirtualAddress;                 √   //  The address of the first byte of
the section when loaded into memory
  DWORD SizeOfRawData;                  √   //  The size of the initialized data
on disk
  DWORD PointerToRawData;               √   //  A file pointer to the first page
within the COFF file
  DWORD PointerToRelocations;               //  A file pointer to the beginning
of the relocation entries for the section
  DWORD PointerToLinenumbers;               //  A file pointer to the beginning
of the line-number entries for the section
  WORD  NumberOfRelocations;                //  The number of relocation entries
for the section
  WORD  NumberOfLinenumbers;                //  The number of line-number
entries for the section
```

```
  DWORD Characteristics;                √   //  The characteristics of the image
} IMAGE_SECTION_HEADER, *PIMAGE_SECTION_HEADER;
```

- `Name`
  节表名，8字节的零填充UTF-8字符串。超过8字节则结尾没有填充零字符，类似注释
- `Misc`
  - `PhysicalAddress`
    文件（物理）地址
  - `VirtualSize`
    载入到内存中节的总大小（真实长度）
- `VirtualAddress`
  载入到内存中该节的首地址，相对于基址，即RVA
- `SizeOfRawData`
  该区块在磁盘中所占的大小。在可执行文件中，该字段已对齐
- `PointerToRawData`
  在文件中的偏移量，从文件头开始算起的偏移量，即FA
- `Characteristics`
  内存属性，WRES写读执行共享对应最高位的二进制位