十、PE文件

PE文件简介

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

- 可执行文件的格式是操作系统本身执行机制的反映
- 掌握可执行文件的数据结构及其运行原理，是研究软件安全的必修课

- Win16平台上，可执行文件格式是NE
- Win32平台上，可执行文件格式是PE，Win32可执行文件，如*.EXE、*.DLL、*.OCX等
- PE的意思就是Portable Executable(可移植、可执行)，是Win32可执行文件的标准格式

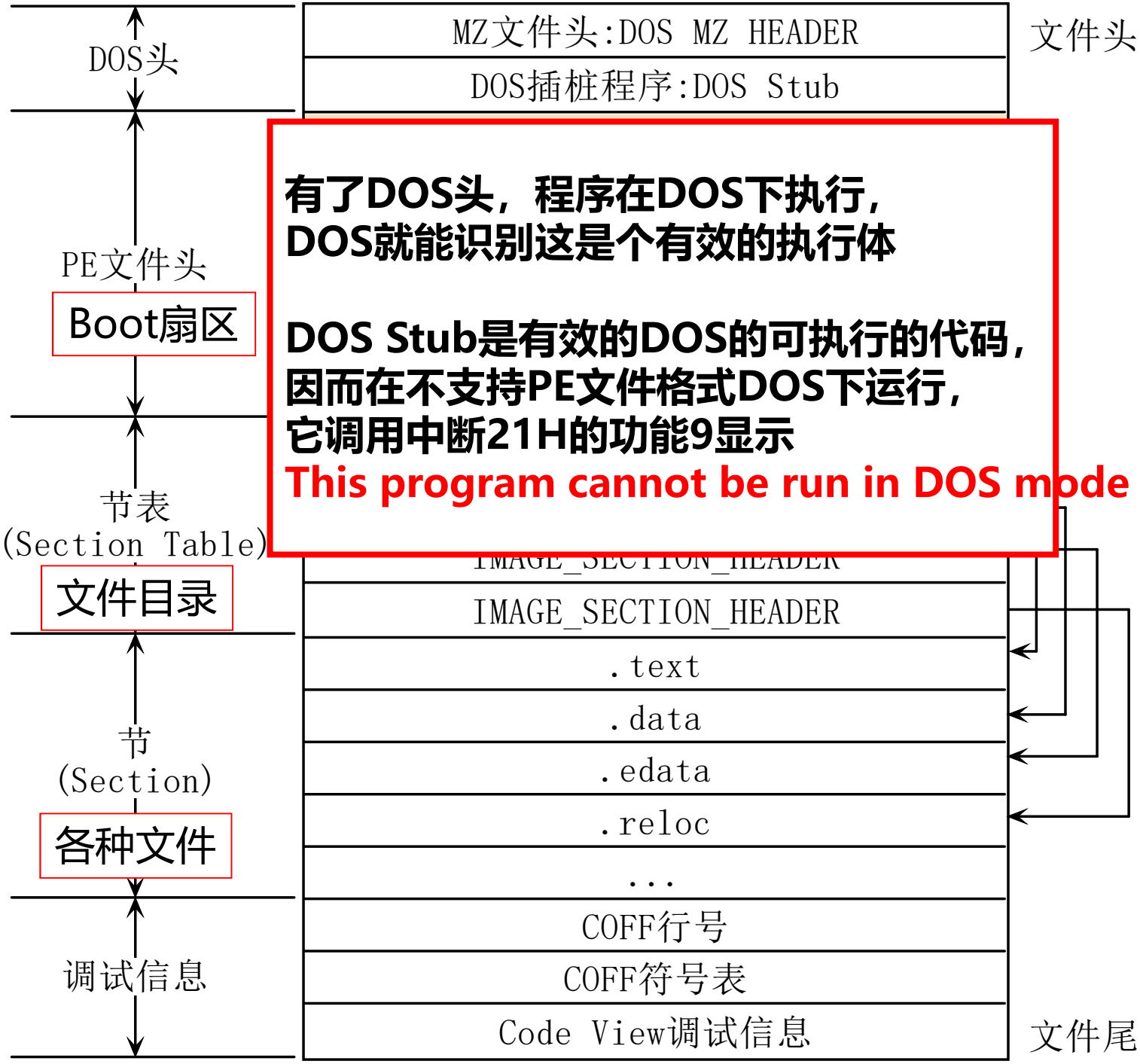
PE文件简介

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

- 描述PE文件格式的地方主要是winnt.h，有你想知道的everything

```
#ifndef _MAC

#include "pshpack4.h"                   // 4 byte packing is the default

#define IMAGE_DOS_SIGNATURE                 0x5A4D      // MZ
#define IMAGE_OS2_SIGNATURE                 0x454E      // NE
#define IMAGE_OS2_SIGNATURE_LE              0x454C      // LE
#define IMAGE_VXD_SIGNATURE                 0x454C      // LE
#define IMAGE_NT_SIGNATURE                  0x00004550  // PE00

#include "pshpack2.h"                   // 16 bit headers are 2 byte packed

#else

#include "pshpack1.h"

#define IMAGE_DOS_SIGNATURE                 0x4D5A      // MZ
#define IMAGE_OS2_SIGNATURE                 0x4E45      // NE
#define IMAGE_OS2_SIGNATURE_LE              0x4C45      // LE
#define IMAGE_NT_SIGNATURE                  0x50450000  // PE00
#endif

typedef struct _IMAGE_DOS_HEADER {      // DOS .EXE 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;                        // Initial (relative) SS value
```

PE



有了DOS头，程序在DOS下执行，DOS就能识别这是个有效的执行体

DOS Stub是有效的DOS的可执行的代码，因而在不支持PE文件格式DOS下运行，它调用中断21H的功能9显示

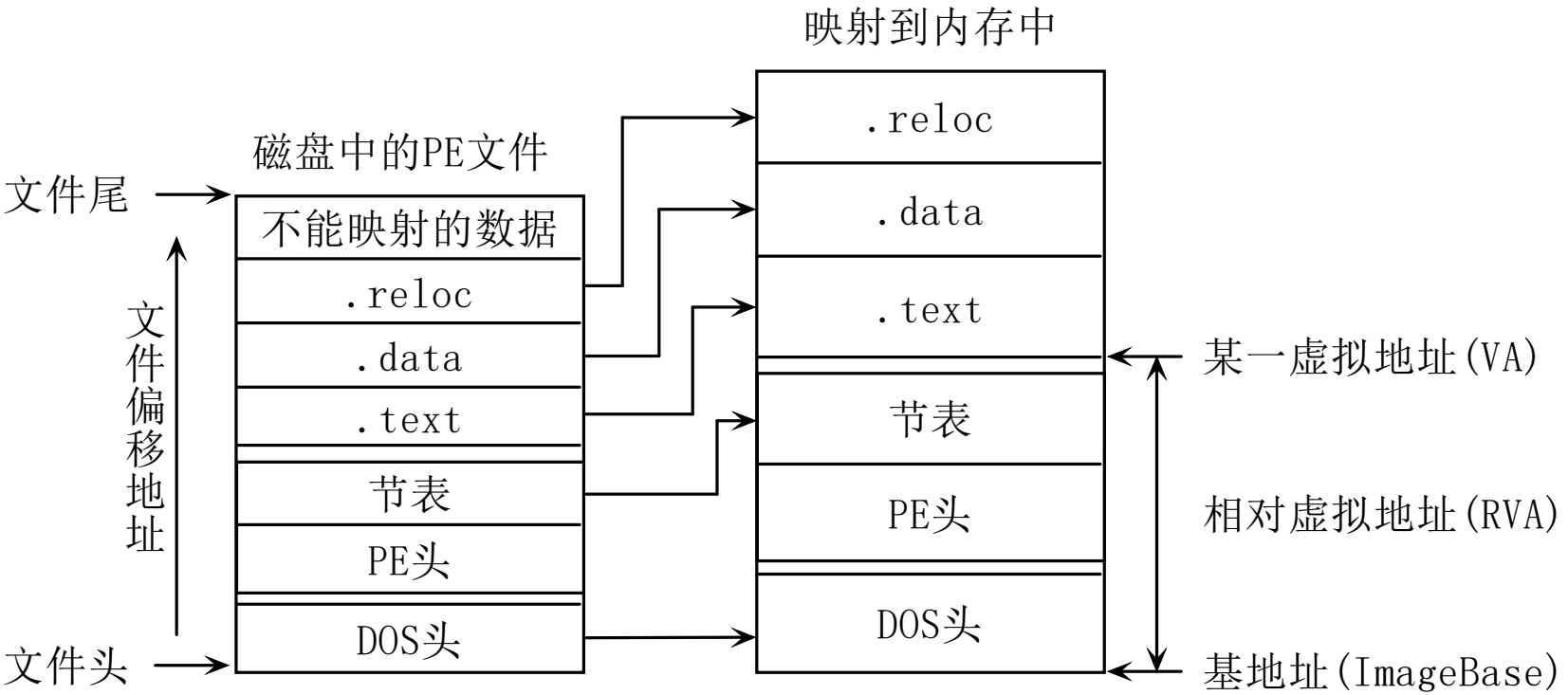
This program cannot be run in DOS mode

PE的基本概念

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

- PE文件使用一个平面地址空间，所有代码和数据合并在一起
- 文件的内容分割为不同的区块（Section）
- 区块中包含代码或数据，各个区块按页边界对齐
- 区块没有大小限制，是一个连续结构
- 每个区块有自己的属性，例如是否包含代码，是否只读或可读/写等

PE的基本概念 – 地址


映射到内存中
磁盘中的PE文件
文件尾
文件偏移地址
文件头
不能映射的数据
.reloc
.data
.text
节表
PE头
DOS头
.reloc
.data
.text
节表
PE头
DOS头
某一虚拟地址(VA)
相对虚拟地址(RVA)
基地址(ImageBase)

PE的基本概念 – 基地址

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

- 当PE文件通过Windows加载器载入内存后，内存中的版本称为模块（Module）
- 映射文件的起始地址称为模块句柄（hModule），可以通过模块句柄访问内存中的其他数据结构
- PE文件加载到内存后，初始内存地址即为基地址

PE的基本概念 – 相对虚拟地址

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

- 在Windows系统中，PE文件被系统加载器映射到内存中。每个程序都有自己的虚拟空间，这个虚拟空间的内存地址称为虚拟地址
- 相对虚拟地址是内存中的一个简单的、相对于PE文件载入地址的偏移位置，是一个相对地址（或称偏移量）
- 例如，假设一个EXE文件从400000h处载入，而且它的代码区块开始于401000h处，代码区块的RVA计算方法如下：

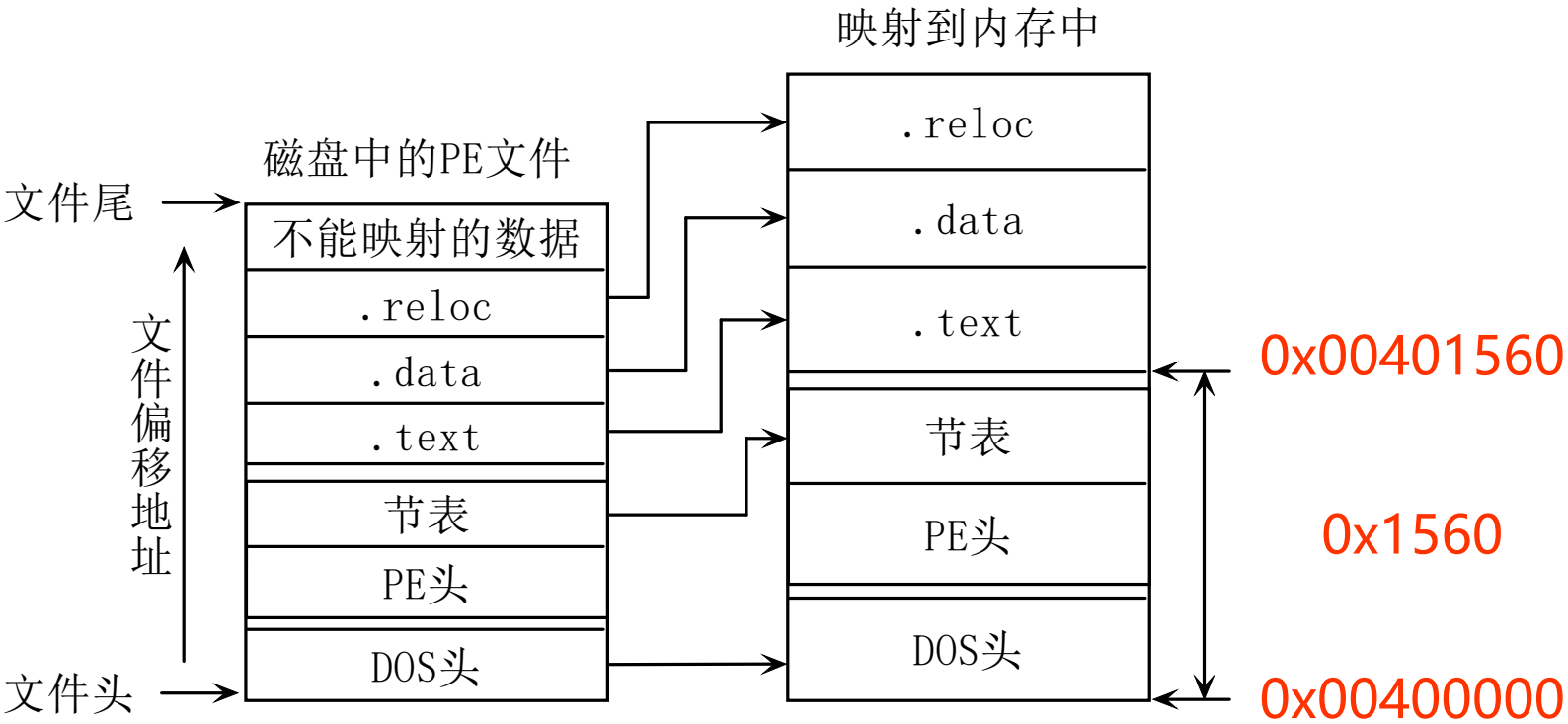
目标地址401000h - 载入地址400000h = RVA1000h

虚拟地址（VA）= 基地址（ImageBase）+ 相对虚拟地址（RVA）

PE的基本概念 – 文件偏移地址

- 当PE文件储存在磁盘中时，某个数据的位置相对于文件头的偏移量称为文件偏移地址（File Offset）或物理地址（RAW Offset）
- 文件偏移地址从PE文件的第1个字节开始计数，起始值为0
- 用十六进制工具（例如Hex Workshop、WinHex等）打开文件时所显示的地址就是文件偏移地址

PE的基本概念 – 地址


映射到内存中
磁盘中的PE文件
文件尾
文件偏移地址
文件头
不能映射的数据
.reloc
.data
.text
节表
PE头
DOS头
.reloc
.data
.text
节表
PE头
DOS头
0x00401560
0x1560
0x00400000

MS-DOS头部

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

- 每个PE文件都是以一个DOS程序开始的，有了它，一旦程序在DOS下执行，DOS就能识别出这是一个有效的执行体，然后运行紧随MZ header的DOS stub（DOS块）
- DOS stub实际上是一个有效的EXE，在不支持PE文件格式的操作系统中它将简单地显示一个错误提示，类似于字符串This program cannot be run in MS-DOS mode
- 程序员也可以根据自己的意图实现完整的DOS代码。用户通常对DOS stub不太感兴趣，因为在大多数情况下它是由汇编器/编译器自动生成的
- 通常把DOS MZ头与DOS stub合称为DOS文件头

MS-DOS头部

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

```
typedef struct _IMAE_DOS_HEADER
{                    //DOS .EXE header                                  位置
    WORD e_magic;    //Magic number;                                    0x00
    WORD e_cblp;     //Bytes on last page of file                       0x02
    WORD e_cp;       //Pages in file                                    0x04
    WORD e_crlc;     //Relocations                                      0x06
    WORD e_cparhdr;  //Size of header in paragraphs                     0x08
    WORD e_minalloc; //Minimum extra paragraphs needed                  0x0A
    WORD e_maxalloc; //Maximum extra paragraphs needed                  0x0C
    WORD e_ss;       //Initial (relative) SS value                      0x0E
    WORD e_sp;       //Initial SP value                                 0x10
    WORD e_csum;     //Checksum                                         0x12
    WORD e_ip;       //Initial IP value                                 0x14
    WORD e_cs;       //Initial (relative) CS value                      0x16
    WORD e_lfarlc;   //File address of relocation table                 0x18
    WORD e_ovno;     //Overlay number                                   0x1A
    WORD e_res[4];   //Reserved words                                   0x1C
    WORD e_oemid;    //OEM identifier (for e_oeminfo)                   0x24
    WORD e_oeminfo;  //OEM information; e_oemid specific                0x26
    WORD e_res2[10]; //Reserved words                                   0x28
    LONG e_lfanew;   //File address of new exe header                   0x3C
} IMAGE_DOS - HEADER, *PIMAGE_DOS_HEADER;
```

MS-DOS头部

- e_magic: 需要被设置为5A4Dh，名为IMAGE_DOS_SIGNATURE，在ASCII表示法里它的ASCII值为“MZ”，是MS-DOS的创建者之一Mark Zbikowski名字的缩写

- e_lfanew: 真正的PE文件头的相对偏移（RVA），指出真正的PE头的文件偏移位置，占用4字节，位于从文件开始偏移3Ch字节处

MS-DOS头部

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

- PE结构中紧随MZ文件头之后的DOS插桩程序(DOS Stub)
- 可以通过IMAGE_DOS_HEADER结构来识别一个合法的DOS头
- 可以通过该结构的e_lfanew(偏移60，32bits)成员来找到PE开始的标志0x00004550(“PE\0\0”)
- 病毒通过MZ、PE这两个标志，初步判断当前程序是否是目标文件——PE文件。如果要精确校验指定文件是否为一有效PE文件，则可以检验PE文件格式里的各个数据结构，或者仅校验一些关键数据结构
- 大多数情况下，没有必要校验文件里的每一个数据结构，只要一些关键数据结构有效，就可以认为是有效的PE文件

MS-DOS头部

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00000000	4D	5A	90	00	03	00	00	00	04	00	00	00	FF	FF	00	00	MZ▮.....yy...
00000010	B8	00	00	00	00	00	00	00	40	00	00	00	00	00	00	00	?.....@.....
00000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000030	00	00	00	00	00	00	00	00	00	00	00	00	B0	00	00	00	.....?...
00000040	0E	1F	BA	0E	00	B4	09	CD	21	B8	01	4C	CD	21	54	68	..?.???L?Th
00000050	69	73	20	70	72	6F	67	72	61	6D	20	63	61	6E	6E	6F	is program canno
00000060	74	20	62	65	20	72	75	6E	20	69	6E	20	44	4F	53	20	t be run in DOS
00000070	6D	6F	64	65	2E	0D	0D	0A	24	00	00	00	00	00	00	00	mode....$.....
00000080	5D	17	1D	DB	19	76	73	88	19	76	73	88	19	76	73	88	]..?vs?vs?vs??
00000090	19	76	73	88	0A	76	73	88	E5	56	61	88	18	76	73	88	.vs?vs堝Va?vs??
000000A0	52	69	63	68	19	76	73	88	00	00	00	00	00	00	00	00	Rich.vs?.....
000000B0	50	45	00	00	4C	01	03	00	2C	97	B8	3D	00	00	00	00	PE..L...,楼=....

PE文件头

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

- 紧接着DOS Stub的是PE header
- PE header是IMAGE_NT_HEADERS的简称，即NT映像头(PE文件头)，存放PE整个文件信息分布的重要字段，包含了许多PE装载器用到的重要域
- 执行体在支持PE文件结构的操作系统中执行时，PE装载器将从DOS MZ header中找到PE header的起始偏移量，从而跳过DOS Stub直接定位到真正的文件头PE header

PNTHeader = ImageBase + dosHeader→e_lfanew

PE文件头

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

- PE文件头的结构

```
IMAGE_NT_HEADERS STRUCT{
    +0h     DWORD                       Signature
    +4h     IMAGE_FILE_HEADER           FileHeader
    +18h    IMAGE_OPTIONAL_HEADER32     OptionalHeader
} IMAGE_NT_HEADERS ENDS
```

检验PE文件的有效性?

- 字符串“PE\0\0”(Signature)(4H字节)
- 映像文件头：关于PE文件物理分布的基本信息
- 可选映像头：PE文件逻辑分布的信息

PE文件头 - 映像文件头

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

- NT映像头的主要部分，包含有PE文件的基本信息
- 一个域指出了IMAGE_OPTIONAL_HEADER的大小

```
typedef struct _IMAGE_FILE_HEADER
{
    +04h WORD    Machine;                  // 运行平台
    +06h WORD    NumberOfSections;         // 文件的区块数目
    +08h DWORD TimeDateStamp;              // 文件创建日期和时间
    +0Ch DWORD PointerToSymbolTable;       // 指向符号表(主要用于调试)
    +10h DWORD NumberOfSymbols;            // 符号表中符号个数(同上)
    +14h WORD   SizeOfOptionalHeader;      // IMAGE_OPTIONAL_HEADER32 结构大小
    +16h WORD   Characteristics;           // 文件属性
} IMAGE_FILE_HEADER, *PIMAGE_FILE_HEADER;
```

PE文件头 - 映像文件头

- Machine：可执行文件的目标CPU类型。PE文件可以在多种机器上使用，不同平台上指令的机器码不同

机　器	标　志
Intel i386	14Ch
MIPS R3000	162h
MIPS R4000	166h
Alpha AXP	184h
Power PC	1F0h

- NumberOfSections：区块（Section）的数目，块表紧跟在IMAGE_NT_HEADERS后面

PE文件头 - 映像文件头

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

- TimeDateStamp：表示文件的创建时间。这个值是自1970年1月1日以来用格林威治时间（GMT）计算的秒数，是一个比文件系统的日期/时间更精确的文件创建时间指示器
- SizeOfOptionalHeader：紧跟IMAGE_FILE_HEADER，表示数据的大小
- Characteristics：文件属性，有选择地通过几个值的运算得到。普通EXE文件的这个字段的值一般是010h，DLL文件的这个字段的值一般是2102h

PE文件头 - 映像文件头

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

特 征 值	含 义
0001h	文件中不存在重定位信息
0002h	文件可执行。如果为 0，一般是链接时出问题了
0004h	行号信息被移去
0008h	符号信息被移去
0020h	应用程序可以处理超过 2GB 的地址。该功能是从 NT SP3 开始被支持的。因为大部分数据库服务器需要很大的内存，而 NT 仅提供 2GB 给应用程序，所以从 NT SP3 开始，通过加载 /3GB 参数，可以使应用程序被分配 2～3GB 区域的地址，而该处原来属于系统内存区
0080h	处理机的低位字节是相反的
0100h	目标平台为 32 位机器
0200h	.DBG 文件的调试信息被移去
0400h	如果映像文件在可移动介质中，则先复制到交换文件中再运行
0800h	如果映像文件在网络中，则复制到交换文件后才运行
1000h	系统文件
2000h	文件是 DLL 文件
4000h	文件只能运行在单处理器上
8000h	处理机的高位字节是相反的

```
typedef struct _IMAGE_OPTIONAL_HEADER
{
    WORD Magic;                         //幻数，一般为10BH
    BYTE MajorLinkerVersion;            //链接程序的主版本号
    BYTE MinorLinkerVersion;            //链接程序的次版本号
    DWORD SizeOfCode;                   //代码段大小
    DWORD SizeOfInitializedData;        //已初始化数据块的大小
    DWORD SizeOfUninitializedData;      //未初始化数据库的大小
    DWORD AddressOfEntryPoint;          //程序开始执行的入口地址,这是一个RVA（相对虚拟地址）
    DWORD BaseOfCode;                   //代码段的起始RVA 一般来说 是 1000h
    DWORD BaseOfData;                   //数据段的起始RVA

    DWORD ImageBase;                    //可执行文件默认装入的基地址
    DWORD SectionAlignment;             //内存中块的对齐值（默认的块对齐值为1000H，4KB个字节）
    DWORD FileAlignment;                //文件中块的对齐值（默认值为200H字节，为了保证块总是从磁盘的扇区开始的）
    WORD MajorOperatingSystemVersion;   //要求操作系统的最低版本号的主版本号
    WORD MinorOperatingSystemVersion;   //要求操作系统的最低版本号的次版本号
    WORD MajorImageVersion;             //该可执行文件的主版本号
    WORD MinorImageVersion;             //该可执行文件的次版本号
    WORD MajorSubsystemVersion;         //要求最低之子系统版本的主版本号 默认 0004
    WORD MinorSubsystemVersion;         //要求最低之子系统版本的次版本号 默认 0000
    DWORD Win32VersionValue;            //保留字                    默认00000000
    DWORD SizeOfImage;                  //映像装入内存后的总尺寸        一般为00004000 映射到内存中一个块1000内存
    DWORD SizeOfHeaders;                //部首及块表的大小
    DWORD CheckSum;                     //CRC检验和                     一般为00000000
    WORD Subsystem;                     //程序使用的用户接口子系统
    WORD DllCharacteristics;            //DLLmain函数何时被调用，默认为0
    DWORD SizeOfStackReserve;           //初始化时堆栈大小
    DWORD SizeOfStackCommit;            //初始化时实际提交的堆栈大小
    DWORD SizeOfHeapReserve;            //初始化时保留的堆大小
    DWORD SizeOfHeapCommit;             //初始化时实际提交的对大小
    DWORD LoaderFlags;                  //与调试有关，默认为0
    DWORD NumberOfRvaAndSizes;          //数据目录结构的数目
    IMAGE_DATA_DIRECTORY DataDirectory[IMAGE_NUMBEROF_DIRECTORY_ENTRIES]; //数据目录表
}
```

PE文件头 - 可选映像头

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

- Magic：这是一个标记字，说明文件是ROM映像（0107h）还是普通可执行的映像（010Bh），一般是010Bh
- AddressOfEntryPoint：程序执行入口RVA。对于DLL，这个入口点在进程初始化和关闭时及线程创建和毁灭时被调用
- ImageBase：文件在内存中的首选载入地址
- DataDirectory：数据目录表，由数个相同的IMAGE_DATA_DIRECTORY结构组成，指向输出表、输入表、资源块等数据

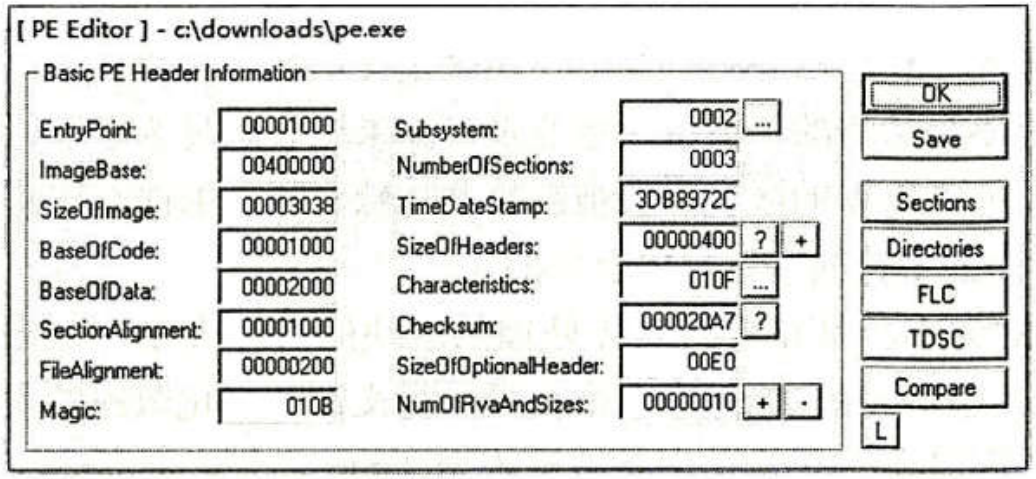
```
typedef struct _IMAGE_DATA_DIRECTORY
{
    DWORD VirtualAddress;
    DWORD Size;
} IMAGE_DATA_DIRECTORY, *PIMAGE_DATA_DIRECTORY;
```

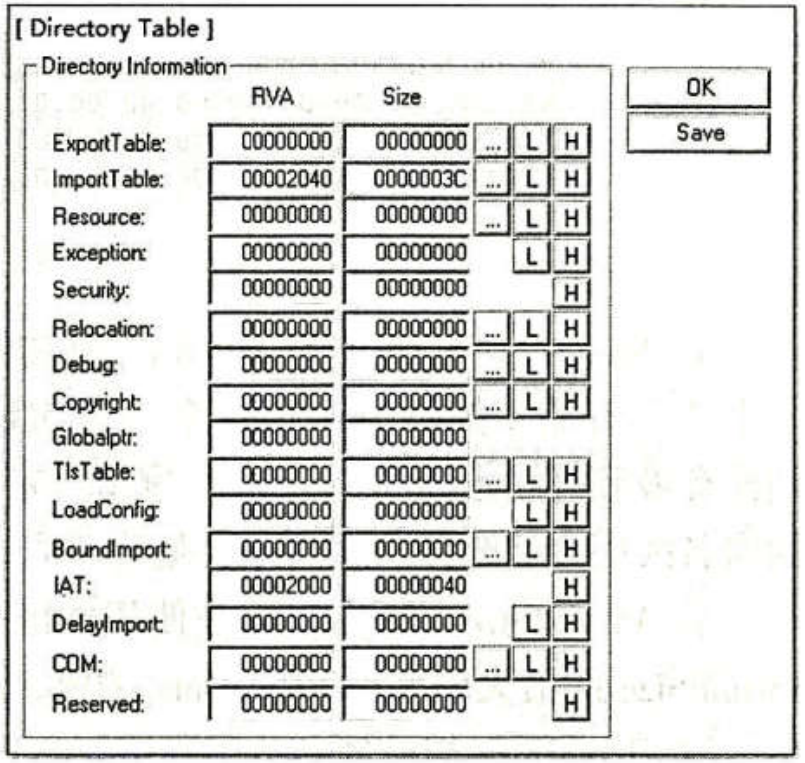
PE文件头 - 可选映像头

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

序　号	成　员	结　构	偏移量（PE/PE32+）
0	Export Table	IMAGE_DIRECTORY_ENTRY_EXPORT	78h / 88h
1	Import Table	IMAGE_DIRECTORY_ENTRY_IMPORT	80h / 90h
2	Resources Table	IMAGE_DIRECTORY_ENTRY_RESOURCE	88h / 98h
3	Exception Table	IMAGE_DIRECTORY_ENTRY_EXCEPTION	90h / A0h
4	Security Table	IMAGE_DIRECTORY_ENTRY_SECURITY	98h / A8h
5	Base relocation Table	IMAGE_DIRECTORY_ENTRY_BASERELOC	A0h / B0h
6	Debug	IMAGE_DIRECTORY_ENTRY_DEBUG	A8h / B8h
7	Copyright	IMAGE_DIRECTORY_ENTRY_COPYRIGHT	B0h / C0h
8	Global Ptr	IMAGE_DIRECTORY_ENTRY_GLOBALPTR	B8h / C8h
9	Thread local storage（TLS）	IMAGE_DIRECTORY_ENTRY_TLS	C0h / D0h
10	Load configuration	IMAGE_DIRECTORY_ENTRY_LOAD_CONFIG	C8h / D8h
11	Bound Import	IMAGE_DIRECTORY_ENTRY_BOUND_IMPORT	D0h / E0h
12	Import Address Table（IAT）	IMAGE_DIRECTORY_ENTRY_IAT	D8h / E8h
13	Delay Import	IMAGE_DIRECTORY_ENTRY_DELAY_IMPORT	E0h / F0h
14	COM descriptor	IMAGE_DIRECTORY_ENTRY_COM_DESCRIPTOR	E8h / F8h
15	保留，必须为 0		F0h / 100h

PE文件格式


[PE Editor] - c:\downloads\pe.exe
Basic PE Header Information
EntryPoint: 00001000
ImageBase: 00400000
SizeOfImage: 00003038
BaseOfCode: 00001000
BaseOfData: 00002000
SectionAlignment: 00001000
FileAlignment: 00000200
Magic: 010B
Subsystem: 0002
NumberOfSections: 0003
TimeDateStamp: 3DB8972C
SizeOfHeaders: 00000400
Characteristics: 010F
Checksum: 000020A7
SizeOfOptionalHeader: 00E0
NumOfRvaAndSizes: 00000010
OK
Save
Sections
Directories
FLC
TDSC
Compare
L


[Directory Table]
Directory Information
RVA
Size
OK
Save
ExportTable: 00000000 00000000
ImportTable: 00002040 0000003C
Resource: 00000000 00000000
Exception: 00000000 00000000
Security: 00000000 00000000
Relocation: 00000000 00000000
Debug: 00000000 00000000
Copyright: 00000000 00000000
Globalptr: 00000000 00000000
TlsTable: 00000000 00000000
LoadConfig: 00000000 00000000
BoundImport: 00000000 00000000
IAT: 00002000 00000040
DelayImport: 00000000 00000000
COM: 00000000 00000000
Reserved: 00000000 00000000

区块表（节表）

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

- 节表是紧挨着NT映像头的一结构数组，包含了它所关联的区块的信息，例如位置、长度、属性

```
typedef struct _IMAGE_SECTION_HEADER
{
    BYTE Name[IMAGE_SIZEOF_SHORT_NAME];
    union
    {
        DWORD PhysicalAddress; //不用关心，始终是NULL
        DWORD VirtualSize;     //指出实际的、被使用的区块的大小(也就是区块的数据没有对齐处理的实际大小)16H个
    } Misc;
    DWORD VirtualAddress;       //该块装载到内存中的RVA
    DWORD SizeOfRawData;        //该块在磁盘文件中所占的大小
    DWORD PointerToRawData;     //该块在磁盘文件中的偏移
    DWORD PointerToRelocations; //在EXE文件中无意义
    DWORD PointerToLinenumbers;
    WORD NumberOfRelocations; //由pointerToRelocations指向的重定位的数目
    WORD NumberOfLinenumbers;
    DWORD Characteristics; //块属性
};
```

区块表（节表）

字段值	地址	用途
IMAGE_SCN_CNT_CODE	00000020h	包含代码，常与 10000000h 一起设置
IMAGE_SCN_CNT_INITIALIZED_DATA	00000040h	该块包含已初始化的数据
IMAGE_SCN_CNT_UNINITIALIZED_DATA	00000080h	该块包含未初始化的数据
IMAGE_SCN_MEM_DISCARDABLE	02000000h	该块可被丢弃，因为它一旦被载入，进程就不再需要它了。常见的可丢弃块是 .reloc（重定位块）
IMAGE_SCN_MEM_SHARED	10000000h	该块为共享块
IMAGE_SCN_MEM_EXECUTE	20000000h	该块可以执行。通常当 00000020h 标志被设置时，该标志也被设置
IMAGE_SCN_MEM_READ	40000000h	该块可读。可执行文件中的块总是设置该标志
IMAGE_SCN_MEM_WRITE	80000000h	该块可写。如果 PE 文件中没有设置该标志，装载程序就会将内存映像页标记为可读或可执行

区块表（节表）

- Characteristics：块属性。该字段是一组指出块属性（例如代码/数据、可读/可写等）的标志。
- 多个标志值求或即为Characteristics的值
- 这些标志中的很多都可以通过链接器的/SECTION开关设置
- “E0000020h=20000000h | 40000000h | 80000000h | 00000020h”表示该块包含执行代码，可读、可写、可执行
- “C0000040h=40000000h | 80000000h | 00000040h”表示该块可读、可写，包含已初始化的数据
- “60000020h=20000000h | 40000000h | 00000020h”表示该块包含执行代码，可读、可执行

区块

- PE文件的真正内容划分成块，称之为Section(节)，紧跟在节表之后
- 每个节是一块拥有共同属性的数据，比如代码/数据、读/写等
- 可以把PE文件想象成一逻辑磁盘，PE header是磁盘的Boot扇区，节表就是根目录，而Section就是各种文件，每种文件自然就有不同属性如只读、系统、隐藏、文档等等
- 节的划分是基于各组数据的共同属性而不是逻辑概念——如果PE文件中的数据/代码拥有相同属性，它们就能被归入同一节中
- 节名称仅仅是个区别不同节的符号而已，类似“data”、“code”的命名只为了便于识别，惟有节的属性设置决定了节的特性和功能

常见区块

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

- 区块中的数据逻辑通常是关联的
- 至少两个区块：一个代码块，一个是数据块
- 每个区块都有特定的名字，表示区块的用途

常见区块

- .text
  - 默认的代码区块，它的内容全是指令代码
  - Windows NT默认的做法是将所有的可执行代码组成了一个单独的节，名为“.text”或“.code”
- .data
  - 默认的读/写数据区块。全局变量、静态变量一般放在这里
- .rdata
  - 默认的只读数据区块，但程序很少用到该块中的数据。
- .idata
  - 包含其他外来DLL的函数及数据信息，即输入表。将.idata区块合并到另一个区块已成为惯例，典型的是.rdata区块

常见区块

- .edata
  - 输出表
  - 引出函数节是本文件向其他程序提供的可调用函数列表
  - 这个节一般用在DLL中，EXE文件中也可以有这个节，但通常很少使用
  - 当PE装载器执行一个程序，它将相关DLLs都装入该进程的地址空间。然后根据主程序的引入函数信息，查找相关DLLs中的真实函数地址来修正主程序。PE装载器搜寻的是DLLs中的引出函数
- .rsrc
  - 资源，包含模块的全部资源，例如图标、菜单、位图等。这个区块是只读的，无论如何都不应该命名为.rsrc以外的名字，也不能被合并到其他区块里

常见区块

- .data
  - 已初始化数据节.data中存放的是在编译时已经确定的数据，这个节有时也叫DATA
- .bss
  - 未初始化数据节.bss存放的是没有初始化的全局变量和静态变量
- .reloc
  - 重定位节存放有一个重定位表。若装载器不是把程序装载到程序编译时默认的基地址时，就需要这个重定位表来做一些调整

区块对齐

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

- 区块的大小是要对齐的：磁盘文件内和内存中
- PE文件头指出了这两个值，它们可以不同
- 在PE文件头里，FileAlignment定义了磁盘区块的对齐值。每一个区块从对齐值的倍数的偏移位置开始，在不足的地方一般以00h来填充，形成区块间隙

例如，在PE文件中，一个典型的对齐值是200h，这样每个区块从200h的倍数的文件偏移位置开始。假设区块的第1个节在400h处，长度为90h，那么400h~490h为这一区块的内容，而文件对齐值是200h，为了使这一节的长度为FileAlignment的整数倍，490h~600h会被0填充，这段空间称为区块间隙，下一个区块的开始地址为600h

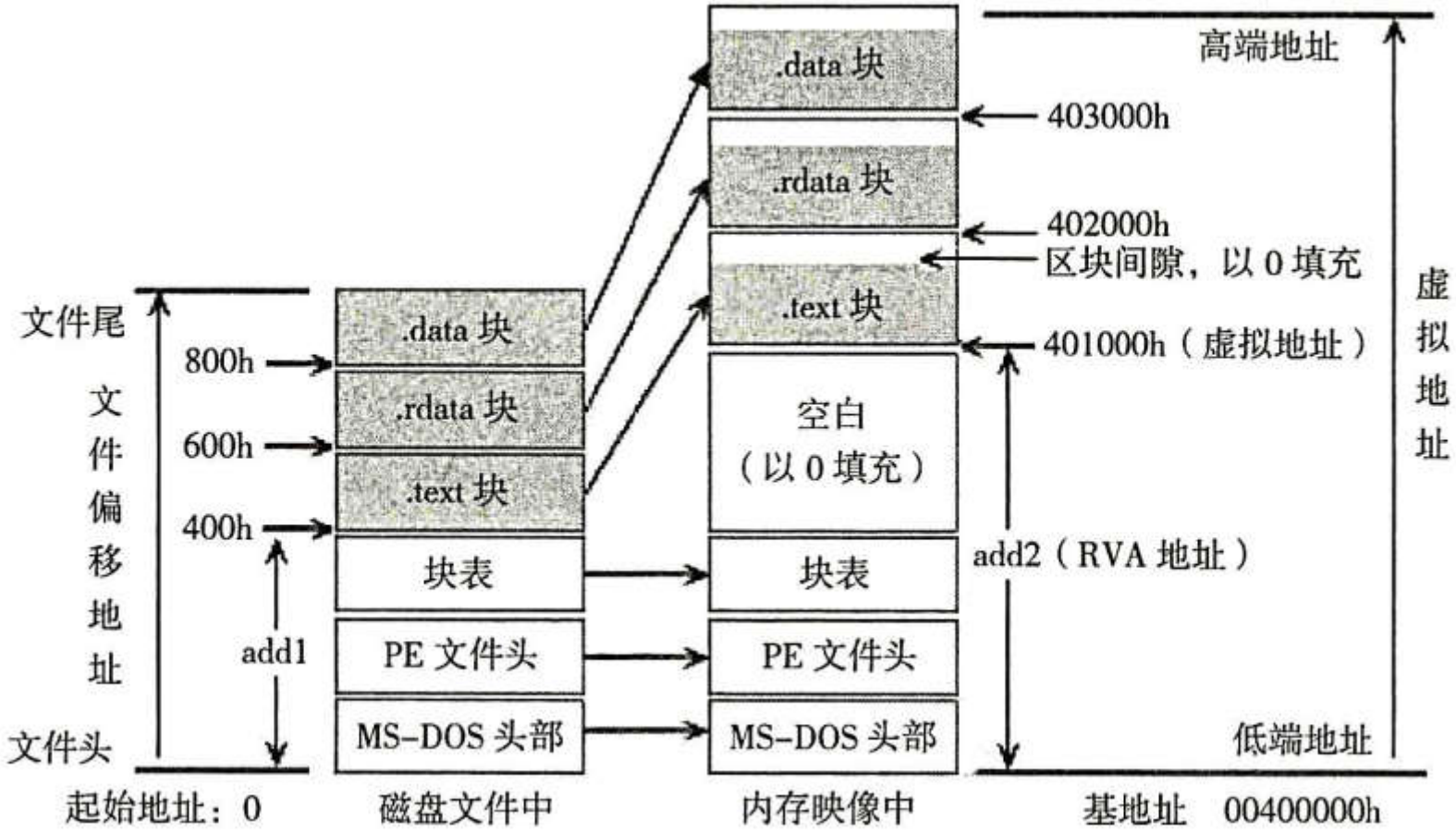
区块对齐

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

- 在PE文件头里，SectionAlignment定义了内存中区块的对齐值。当PE文件被映射到内存中时，区块总是至少从一个页边界处开始
- 当一个PE文件被映射到内存中时，每个区块的第1个字节对应于某个内存页
- 在x86系列CPU中，内存页是按4KB（1000h）排列的，在x86系统中，PE文件区块的内存对齐值一般为1000h，每个区块从1000h的倍数的内存偏移位置开始

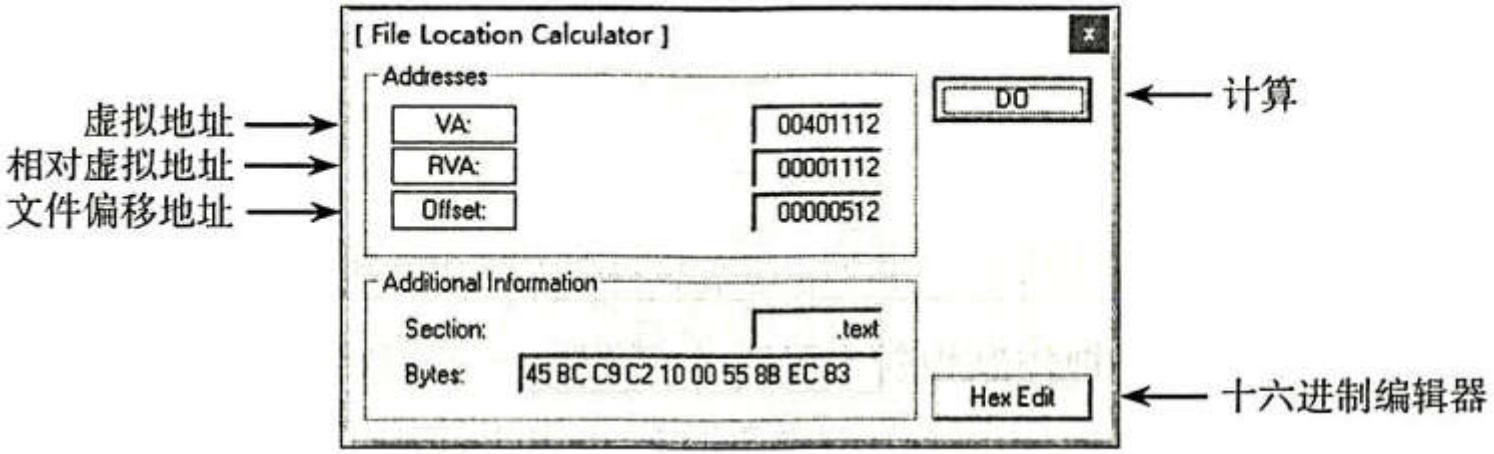
.text区块在磁盘文件中的偏移位置是400h，在内存中将是其载入地址之上的1000h字节处。.rdata区块在磁盘文件偏移的600h处，在内存中将是载入地址之上的2000h字节处

文件偏移与虚拟地址的转换



文件偏移与虚拟地址的转换

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY



输入表

- 可执行文件使用来自其他DLL的代码或数据的动作成为输入
- 当文件载入时，定位所有被输入的函数和数据，并让正在载入的文件可以使用那些地址
- 输入表保存的是函数名和其驻留的DLL名等动态链接所需的信息

输入表

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

- 输入函数就是被程序调用但其执行代码不在程序中的函数，位于相关的DLL文件中
- 在调用者程序中只保留相关的函数信息
- 当PE文件载入内存后，Windows加载器才将相关DLL载入
- 隐式链接和显式链接（LoadLibrary, GetProcAddress）

输入表

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

- 在PE文件内有一组数据结构，它们分别对应于被输入的DLL
- 每一个这样的结构都给出了被输入的DLL的名称并指向一组函数指针
- 这组函数指针称为输入地址表（Import Address Table，IAT）
- 每一个被引入的API在IAT里都有保留的位置，在那里它将被Windows加载器写入输入函数的地址
- 最后一点特别重要：一旦模块被载入，IAT中将包含所要调用输入函数的地址

输入表的结构

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

- 在PE文件头的可选映像头中，数据目录表的第2个成员指向输入表
- 输入表以一个IMAGE_IMPORT_DESCRIPTOR（IID）数组开始
- 每个被PE文件隐式链接的DLL都有一个IID
- 在这个数组中，没有字段指出该结构数组的项数，但它的最后一个单元是“NULL”

输入表的结构

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

```
typedef struct _IMAGE_IMPORT_DESCRIPTOR
{
    union
    {
        DWORD Characteristics;
        DWORD OriginalFirstThunk; //INT(Import Name Table) address (RVA)
    };
    DWORD TimeDateStamp;
    DWORD ForwarderChain;
    DWORD Name;       //library name string address (RVA)
    DWORD FirstThunk; //IAT(Import Address Table) address (RVA)
} IMAGE_IMPORT_DESCRIPTOR;

typedef struct _IMAGE_IMPORT_BY_NAME
{
    WORD Hint;    //ordinal
    BYTE Name[1]; //function name string
} IMAGE_IMPORT_BY_NAME, *PIMAGE_IMPORT_BY_NAME;
```

输入表的结构

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

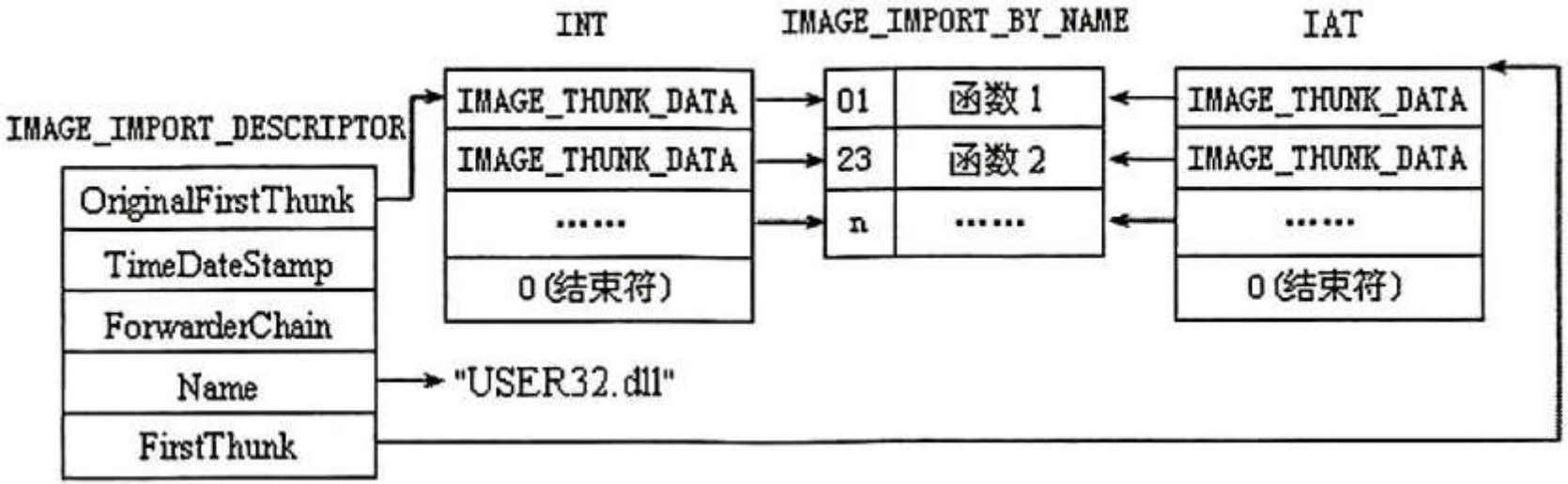
- OriginalFirstThunk（Characteristics）：包含指向输入名称表（INT）的RVA
  - INT是一个IMAGE THUNK_DATA结构的数组，数组中的每个IMAGE_THUNK_DATA结构都指向IMAGE_IMPORT_BY_NAME 结构，数组以一个内容为0的IMAGE_THUNK_DATA结构结束
- Name： DLL名字的指针。它是一个以“00”结尾的ASCII字符的RVA地址，该字符串包含输入的DLL名，例如“KERNEL32.DLL” “USER32.DLL”
- FirstThunk：包含指向输入地址表（IAT）的RVA。IAT是一个IMAGE_THUNK_DATA结构的数组

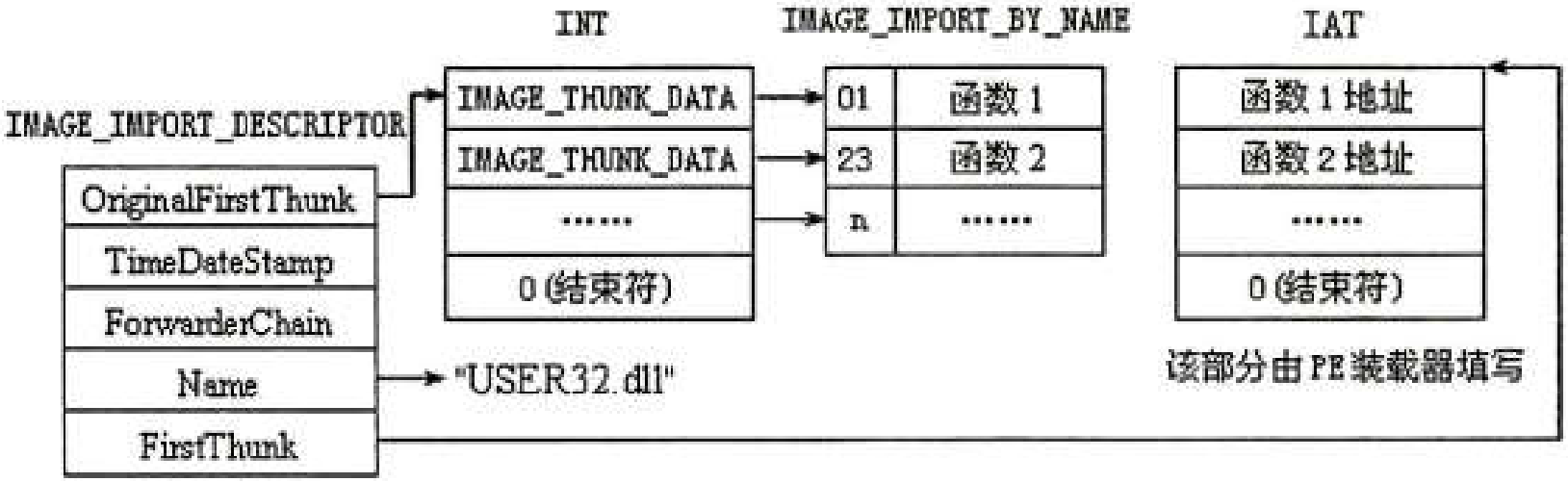
输入表

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

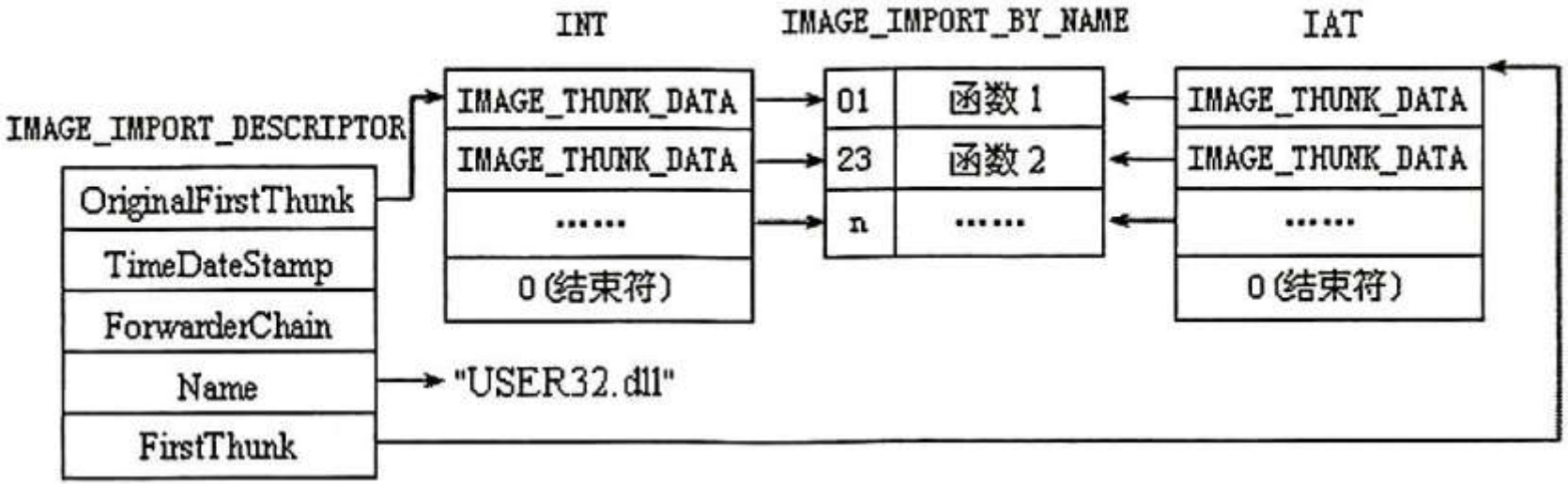
```
struct _IMAGE_THUNK_DATA32
{
    union
    {
        DWORD ForwarderString;
        DWORD Function;      //被输入的函数的内存地址
        DWORD Ordinal;       //高位为1则被输入的API的序数值
        DWORD AddressOfData; //高位为0则指向IMAGE_IMPORT_BY_NAME 结
构体二
    } u1;
} IMAGE_THUNK_DATA32;
//IMAGE_THUNK_DATA64与IMAGE_THUNK_DATA32的区别，仅仅是把
DWORD换成了64位整数。

struct _IMAGE_IMPORT_BY_NAME
{
    WORD Hint;    //指出函数在所在的dll的输出表中的序号
    BYTE Name[1]; //指出要输入的函数的函数名
} IMAGE_IMPORT_BY_NAME, *PIMAGE_IMPORT_BY_NAME;
```

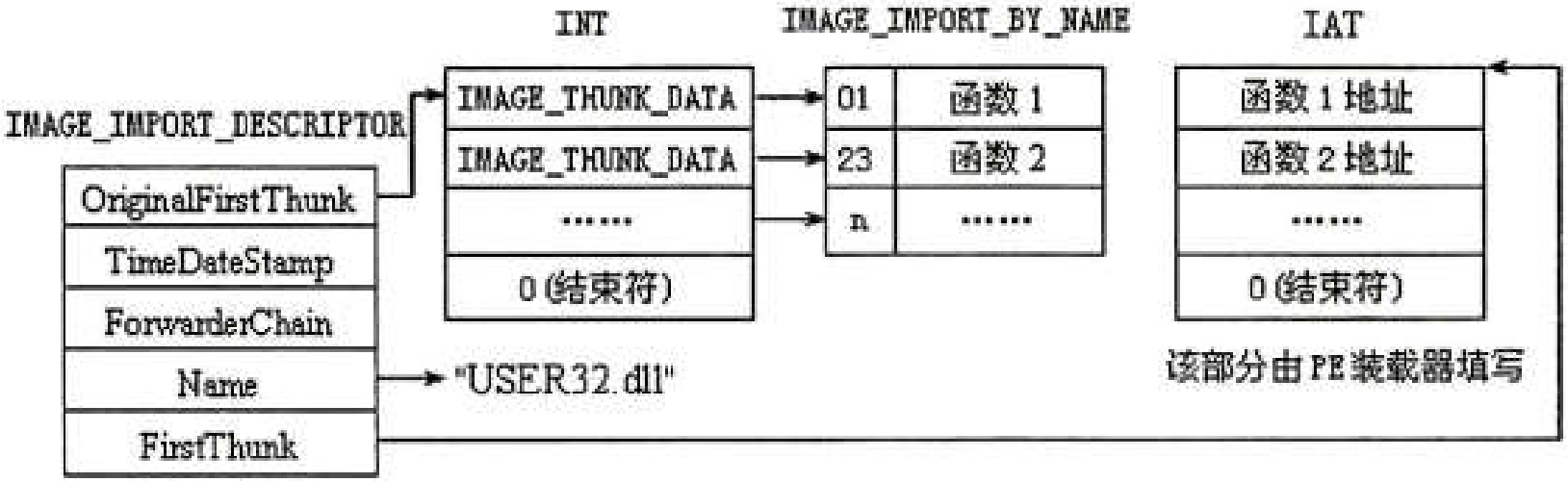
输入表


INT
IMAGE_IMPORT_BY_NAME
IAT
IMAGE_IMPORT_DESCRIPTOR
OriginalFirstThunk
TimeDateStamp
ForwarderChain
Name
FirstThunk
"USER32.dll"
IMAGE_THUNK_DATA
IMAGE_THUNK_DATA
……
0(结束符)
01
函数1
23
函数2
n
……
IMAGE_THUNK_DATA
IMAGE_THUNK_DATA
……
0(结束符)


INT
IMAGE_IMPORT_BY_NAME
IAT
IMAGE_IMPORT_DESCRIPTOR
OriginalFirstThunk
TimeDateStamp
ForwarderChain
Name
FirstThunk
"USER32.dll"
IMAGE_THUNK_DATA
IMAGE_THUNK_DATA
……
0(结束符)
01
函数1
23
函数2
n
……
函数1地址
函数2地址
……
0(结束符)
该部分由PE装载器填写

输入表


INT
IMAGE_IMPORT_BY_NAME
IAT
IMAGE_IMPORT_DESCRIPTOR
OriginalFirstThunk
TimeDateStamp
ForwarderChain
Name
FirstThunk
"USER32.dll"
IMAGE_THUNK_DATA
IMAGE_THUNK_DATA
……
0(结束符)
01
函数1
23
函数2
n
……
IMAGE_THUNK_DATA
IMAGE_THUNK_DATA
……
0(结束符)


INT
IMAGE_IMPORT_BY_NAME
IAT
IMAGE_IMPORT_DESCRIPTOR
OriginalFirstThunk
TimeDateStamp
ForwarderChain
Name
FirstThunk
"USER32.dll"
IMAGE_THUNK_DATA
IMAGE_THUNK_DATA
……
0(结束符)
01
函数1
23
函数2
n
……
函数1地址
函数2地址
……
0(结束符)
该部分由PE装载器填写

输入表实例

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00000600	A0	21	00	00	8E	21	00	00	B0	21	00	00	00	00	00	00	?..?..!!......
00000610	10	21	00	00	1C	21	00	00	F4	20	00	00	E0	20	00	00	.!...!..?..?..
00000620	50	21	00	00	64	21	00	00	D2	21	00	00	CE	20	00	00	P!..d!...!..?..
00000630	BC	20	00	00	2E	21	00	00	42	21	00	00	00	00	00	00	?...!..B!......
00000640	8C	20	00	00	00	00	00	00	00	00	00	00	74	21	00	00	?..........t!..
00000650	10	20	00	00	7C	20	00	00	00	00	00	00	00	00	00	00	. ..| ....
00000660	B4	21	00	00	00	20	00	00	00	00	00	00	00	00	00	00	?... ....
00000670	00	00	00	00	00	00	00	00	00	00	00	00	A0	21	00	00	.... !..
00000680	8E	21	00	00	B0	21	00	00	00	00	00	00	10	21	00	00	?..!!.......!..
00000690	1C	21	00	00	F4	20	00	00	E0	20	00	00	50	21	00	00	.!..?..?..P!..
000006A0	64	21	00	00	D2	21	00	00	CE	20	00	00	BC	20	00	00	d!...!..?..?..
000006B0	2E	21	00	00	42	21	00	00	00	00	00	00	58	00	43	72	.!..B!......X.Cr
000006C0	65	61	74	65	57	69	6E	64	6F	77	45	78	41	00	B3	00	eateWindowExA.?
000006D0	44	65	66	57	69	6E	64	6F	77	50	72	6F	63	41	00	00	DefWindowProcA..
000006E0	94	00	44	69	73	70	61	74	63	68	4D	65	73	73	61	67	?DispatchMessag
000006F0	65	41	00	00	28	01	47	65	74	4D	65	73	73	61	67	65	eA..(.GetMessage
00000700	41	00	97	01	4C	6F	61	64	43	75	72	73	6F	72	41	00	A.?LoadCursorA.

输入表实例

OrignalFirstThunk	TimeDateStamp	ForwardChain	Name	First Thunk
8C20 0000	0000 0000	0000 0000	7421 0000	1020 0000
7C20 0000	0000 0000	0000 0000	B421 0000	0020 0000
0000 0000	0000 0000	0000 0000	0000 0000	0000 0000

DLL 名称	OrignalFirstThunk	TimeDateStamp	ForwardChain	Name	First Thunk
USER32.dll	0000 208C	0000 0000	0000 0000	0000 2174	0000 2010
KERNEL32.dll	0000 207C	0000 0000	0000 0000	0000 21B4	0000 2000

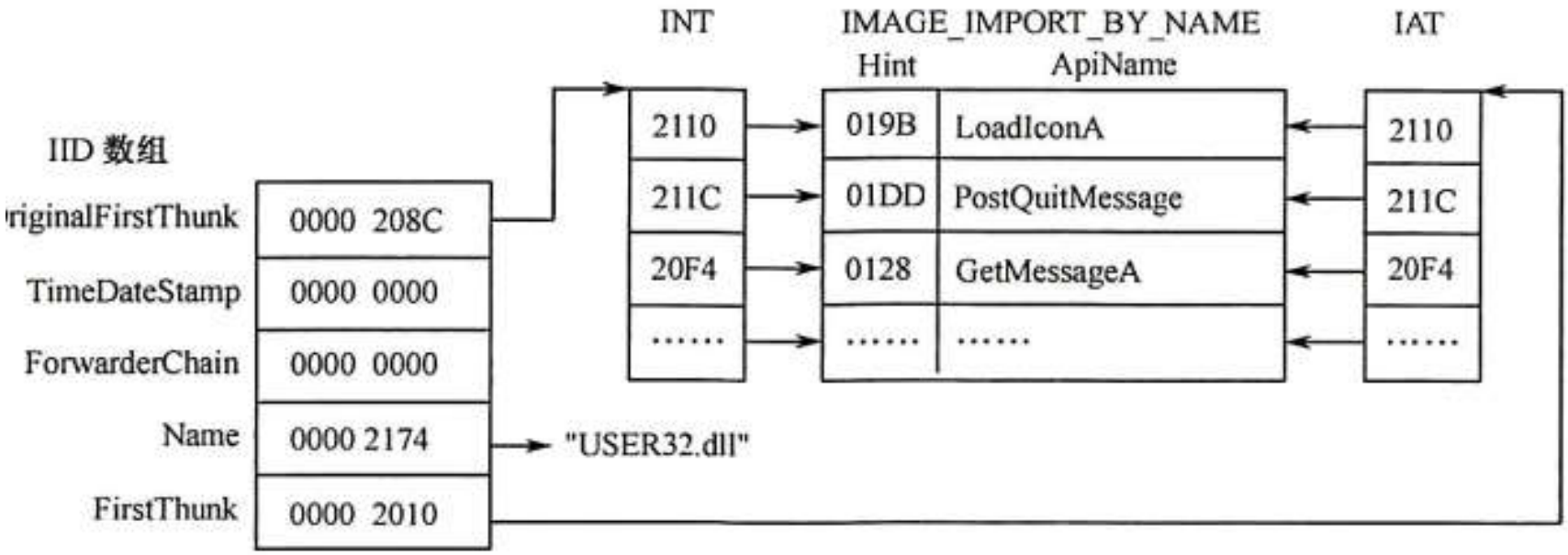
1021 0000	1C21 0000	F420 0000	E020 0000
5021 0000	6421 0000	0221 0000	CE20 0000
BC20 0000	2E21 0000	4221 0000	0000 0000

输入表实例

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

提示名表（RVA）	提示名表（File Offset）	Hint	ApiName
00002110h	710h	019Bh	LoadIconA
0000211Ch	71Ch	01DDh	PostQuitMessage
000020F4h	6F4h	0128h	GetMessageA
000020E0h	6E0h	0094h	DispatchMessageA
00002150h	750h	072Dh	TranslateMessage
00002164h	764h	028Bh	UpdateWindow
00002102h	702h	0197h	LoadCursorA
000020CEh	6CEh	0083h	DefWindowProcA
000020BCh	6BCh	0058h	CreateWindowExA
0000212Eh	72Eh	01EFh	RegisterClassExA
00002142h	742h	0265h	ShowWindow

输入表实例



Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00002000	D9	AC	E5	77	AB	E2	E5	77	63	98	E5	77	00	00	00	00	佻銓　銓c樺w....
00002010	DD	16	D2	77	77	F2	D1	77	18	91	D1	77	05	91	D1	77	?襴w蜒w.慷w.慷ww
00002020	BC	8A	D1	77	05	AB	D1	77	07	CA	D1	77	E0	AB	D1	77	紛祺.　w.恃w晌祺
00002030	F4	19	D1	77	72	EC	D1	77	DF	C1	D1	77	00	00	00	00	?祺r煆w吐祺....
00002040	8C	20	00	00	00	00	00	00	00	00	00	00	74	21	00	00	?.........t!..
00002050	10	20	00	00	7C	20	00	00	00	00	00	00	00	00	00	00	. ..| .....
00002060	B4	21	00	00	00	20	00	00	00	00	00	00	00	00	00	00	?... .....
00002070	00	00	00	00	00	00	00	00	00	00	00	00	A0	21	00	00	.....?..
00002080	8E	21	00	00	80	21	00	00	00	00	00	00	10	21	00	00	?..!!.....!..
00002090	1C	21	00	00	F4	20	00	00	E0	20	00	00	50	21	00	00	.!..?..?..P!..
000020A0	64	21	00	00	02	21	00	00	CE	20	00	00	BC	20	00	00	d!...!..?..?..
000020B0	2E	21	00	00	42	21	00	00	00	00	00	00	58	00	43	72	.!..B!.....X.Cr
000020C0	65	61	74	65	57	69	6E	64	6F	77	45	78	41	00	83	00	eateWindowExA.?

输出表

- 创建一个DLL时，实际上创建了一组能让EXE或其他DLL调用的函数，此时PE装载器根据DLL文件中输出的信息修正被执行文件的IAT

- 当一个DLL函数能被EXE或另一个DLL文件使用时，它就被“输出了”（Exported）

- 其中，输出信息被保存在输出表中，DLL文件通过输出表向系统提供输出函数名、序号和入口地址等信息

输出表

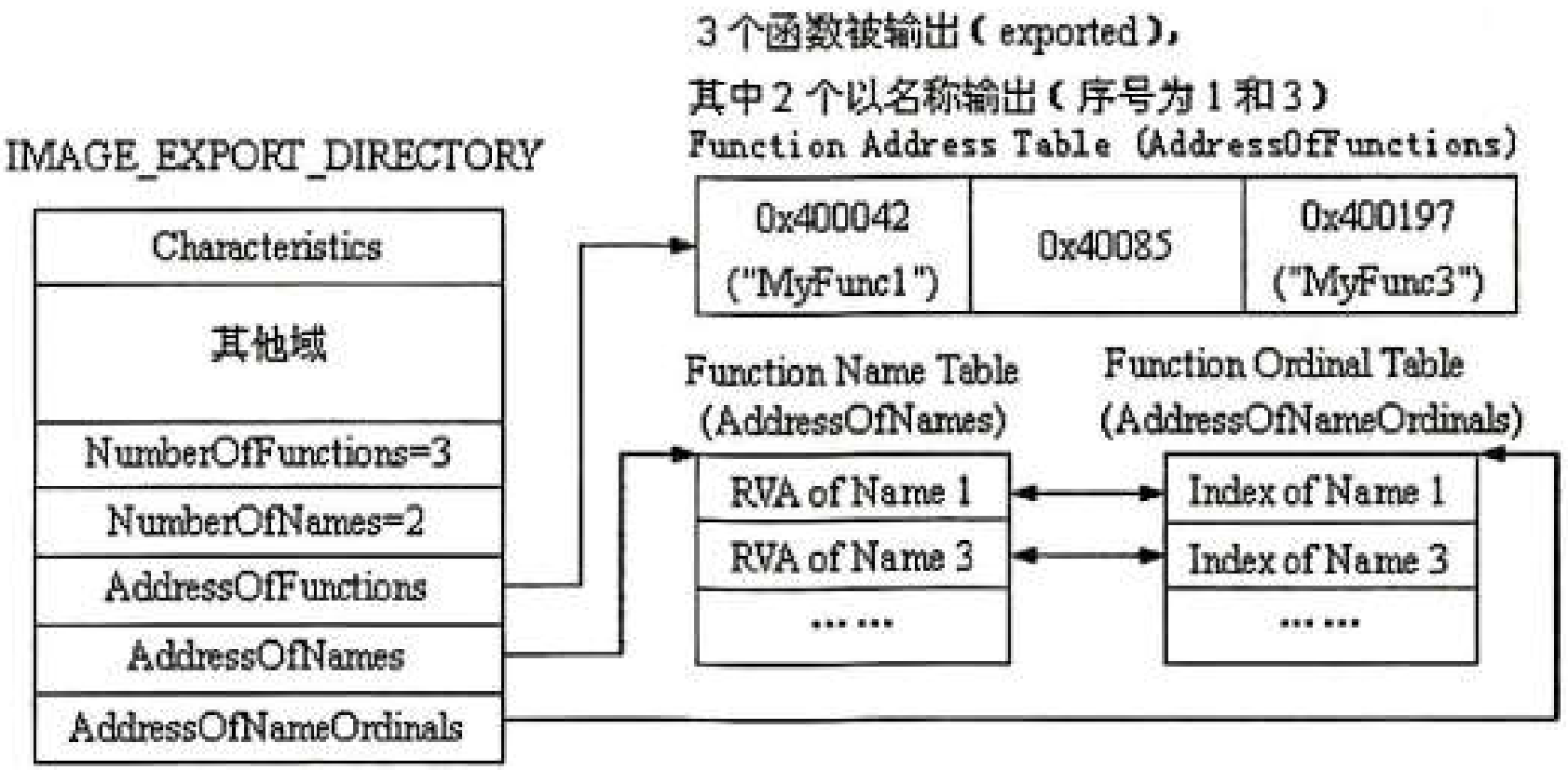
哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

```
typedef struct _IMAGE_EXPORT_DIRECTORY
{
    DWORD Characteristics;
    DWORD TimeDateStamp; //creation time date stamp
    WORD MajorVersion;
    WORD MinorVersion;
    DWORD Name;                 //address of library file name
    DWORD Base;                 //ordinal base
    DWORD NumberOfFunctions;    //number of functions
    DWORD NumberOfNames;        //number of names
    DWORD AddressOfFunctions;   //address of function start address array
    DWORD AddressOfNames;       //address of function name string array
    DWORD AddressOfNameOrdinals; //address of ordinal array
} IMAGE_EXPORT_DIRECTORYM, *pIMAGE_EXPORT_DIRECTORY;
```

输出表

- Name：指向一个ASCII字符串的RVA。这个字符串是与这些输出函数相关联的DLL的名字（例如KERNEL32.DLL）
- NumberOfFunctions：EAT中的条目数量
- NumberOfNames：输出函数名称表（ENT）里的条目数量
- AddressOfFunctions：EAT的RVA
- AddressOfNames：ENT的RVA

输出表


3个函数被输出（exported），
其中2个以名称输出（序号为1和3）
IMAGE_EXPORT_DIRECTORY
Function Address Table (AddressOfFunctions)
Characteristics
其他域
NumberOfFunctions=3
NumberOfNames=2
AddressOfFunctions
AddressOfNames
AddressOfNameOrdinals
0x400042
("MyFunc1")
0x40085
0x400197
("MyFunc3")
Function Name Table
(AddressOfNames)
Function Ordinal Table
(AddressOfNameOrdinals)
RVA of Name 1
RVA of Name 3
... ...
Index of Name 1
Index of Name 3
... ...

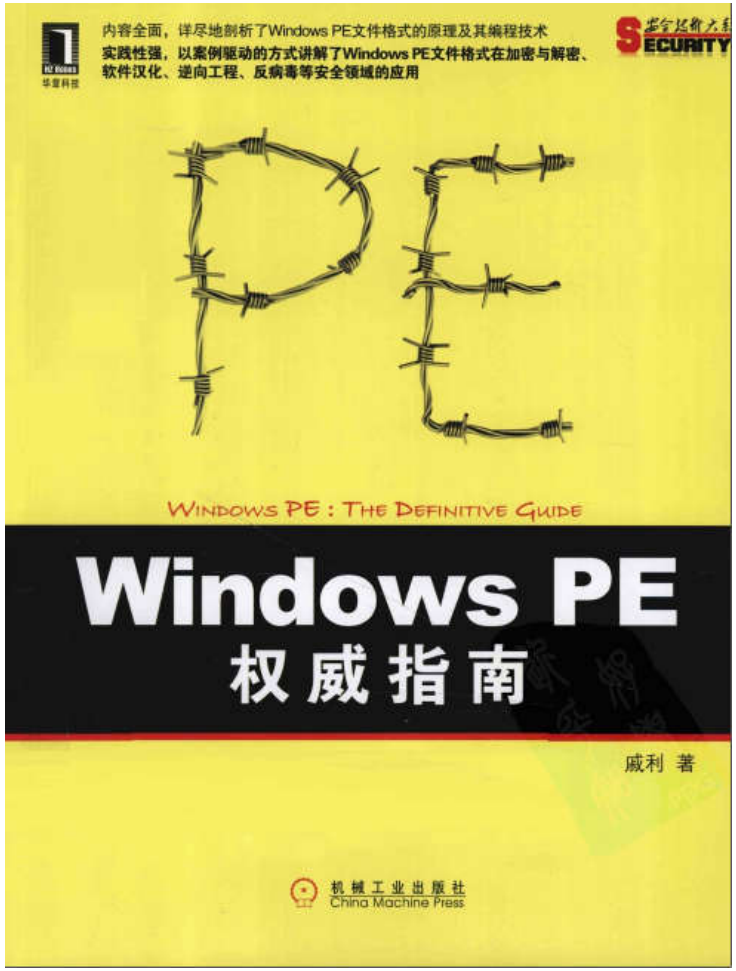
输出表实例

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00000C00	00	00	00	00	00	00	00	00	00	00	00	00	32	40	00	00	..........2@..
00000C10	01	00	00	00	01	00	00	00	01	00	00	00	28	40	00	00	..........(@..
00000C20	2C	40	00	00	30	40	00	00	08	10	00	00	3E	40	00	00	,@..0@......>@..
00000C30	00	00	44	6C	6C	44	65	6D	6F	2E	44	4C	4C	00	4D	73	..DllDemo.DLL.Ms
00000C40	67	42	6F	78	00	00	00	00	00	00	00	00	00	00	00	00	gBox..........

Characteristics	TimeDateStamp	MajorVersion	MinorVersion	Name	Base
0000 0000	0000 0000	0000	0000	3240 0000	0100 0000
NumberOfFunctions	**NumberOfNames**	**AddressOfFunctions**	**AddressOfNames**	**AddressOfNameOrdinals**	
0100 0000	0100 0000	2840 0000	2C40 0000	3040 0000	

PE文件


内容全面，详尽地剖析了Windows PE文件格式的原理及其编程技术
实践性强，以案例驱动的方式讲解了Windows PE文件格式在加密与解密、软件汉化、逆向工程、反病毒等安全领域的应用
SECURITY
PE
WINDOWS PE : THE DEFINITIVE GUIDE
Windows PE
权威指南
戚利 著
机械工业出版社
China Machine Press