PE文件格式解析

程艳芬[1]　翟海波[2]　钟　辉[3]

(武汉理工大学[1,3]　武汉 430063)(海军青岛装备部[2]　青岛 266071)

摘　要　文章介绍了操作系统的可移植执行文件及其架构,对其格式中比较重要的部分进行了较为详细的分析。

关键词　PE　EXE　文件格式

Abstract: This paper describes portable executable file of the operation system and its structure. It also analyses, in detail, the more important part of the file format.

Key words: Portable Executable(PE) ; execute ; file format

0　引　言

PE文件格式是Windows NT、Windows95/98、Win32s下使用的二进制可执行文件格式。EXE文件、32位DLL文件、OBJ文件、NT的设备驱动程序都采用这种格式。它首先由Microsoft提出,在1993年又由TIS(Tool Interface Standard)组织标准化。PE文件格式继承了VAX/VMS及UNIX上可执行文件格式COFF(Common Object File Format)的思想。这种文件格式在所有的Win32平台上通用,不论其CPU是否用Intel的芯片。由于可执行文件的格式是操作系统本身执行机制的反映,所以研究PE文件格式能使我们对Windows本身有更深入的了解,编写出更好的代码。

1　PE的框架结构

我们用"Module"这一术语来表示已装入内存的可执行文件或DLL的代码、数据及资源。除程序中直接用到的代码和数据外,一个Module也指Windows中用于判断代码及数据在内存中位置的支撑数据结构。在16位Windows中,支撑数据结构在Module Database中(HMODULE指向这个段)。而在32位Windows中,这些数据结构存放在PE部首中。

对于PE文件,磁盘中的执行文件和被Windows装载程序装入后的Module看起来非常相似。Windows装载程序使用文件内存映象机制将磁盘文件映射到虚拟地址空间。一旦Module被装入,在Windows中就同其它已经装入的文件一样。图1是一个PE文件的整体架构。

1.1　PE文件首部

象其他可执行文件一样,PE文件也是一个首部信息的集合,描述了文件的基本内容。这个首部包含了诸如代码和数据的地址、长度及该文件适合何种操作系统、堆栈初始大小及其它一些重要信息。PE文件首部并不是在文件的开始部分,开始的几百个字节是MS－DOS stub(WINSTUB),这个极小的DOS程序显示"This program cannot be run in MS－DOS mode"之类的

```
Next n
Next m
End Sub
```

点击s3中的"分析"按钮,便可得到该项目敏感性分析的结果,如图3所示。

参考文献

1　赵艳霞,卢正明. Excel2000基础与应用. 北京:高等教育出版社,2001:78～91

2　陈明. Visual Basic程序设计. 北京:中央广播电视大学出版社,2001:35～39

3　张树兵,戴红. Visual Basic 6.0入门与提高. 北京:清华大学出版社,1999:65～67

4　聂让,许金良. 公路施工测量手册. 北京:清华大学出版社,2000:121～139

5　交通部. 公路工程造价编制与项目经济评价. 北京:交通部公路工程定额站,1999:35～37

收稿日期:2003-08-25

信息,提示用户应在 Win32 的环境中运行。

PE 首部的起始地址隐含在 MS—DOS 首部中。在 Winnt.h 头文件中包含了 DOS stub 的结构定义。E_lfanew 字段就是真正 PE 首部的相对偏移(或 RVA),要得到内存中的 PE 首部指针,只需要用基址加上 e_lfanew 即可。

1.2 PE 文件中的常见节

1).text 节。.text 节是在编译或汇编结束时产生的一种节,它的内容全是指令代码。

2).data 节。如同.text 是默认的代码节一样,.data 是初始化的数据节。这些数据包括编译时被初始化的 globle 和 static 变量,也包括字符串。

3).idata 节。.idata 包含其他外来 DLL 的函数和数据信息。其功能与 NE 文件的模块引用表类似,关键差异在于 PE 文件中的每个输入函数都明确的列于该节中,要在 NE 格式中找到相同的信息,必须从各个段的重定位数据中查找。

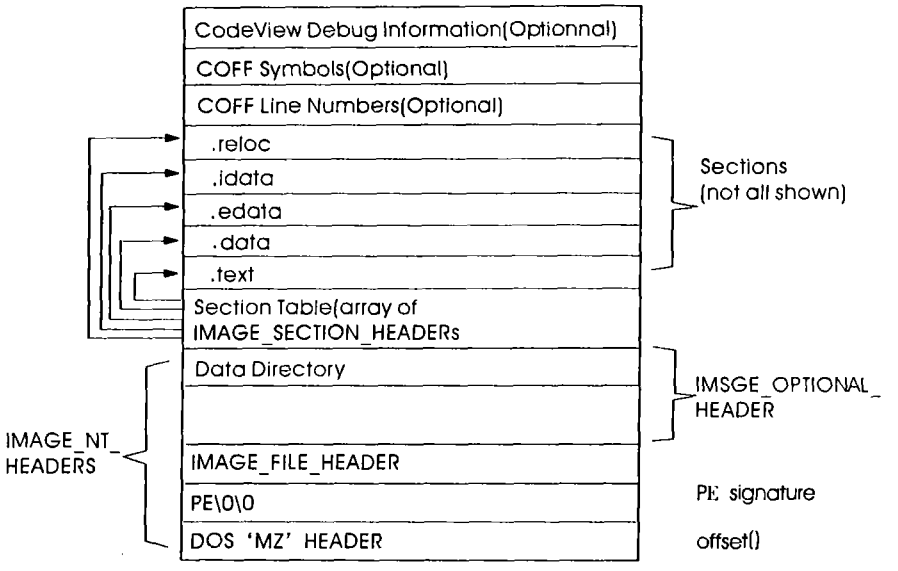

图1 PE 文件总体框图

4).rsrc 节。.rsrc 包含模块的全部资源。如图标、菜单、位图等等。

5).reloc 节。.reloc 保存基地址重定位表。

6).edata 节。.edata 是该 PE 文件输出函数和数据的列表,以供其他模块引用。

7).tls 节。TLS 的名称是因为 Thread Local Storage 而来,和 Win32 API 函数家族有密切关系。

8).rdata 节。.rdata 节通常是在.data 或.bss 中间,但程序中很少用到该节中的数据。至少有两种情况下要用到.rdata。一是在 Microsoft 的连结器产生的 EXE 文件中,用于存放调试目录。二是用于存放说明字符串。如果程序的 DEF 文件中指定了 DESCRIPTION,则字符串就会出现在.rdata 中。

9)其他常见的节 。包括:.debug$s、.debug$t、.drective、.crt、.bss 等等。他们在程序中很少用到。

1.3 PE 文件的 IMPORT

在调用外部 DLL 时,CALL 指令实际上是被转化成 EXE 文件.text 块中的 Jmp dword ptr [xxxxxxxx]指令(如果使用的是 Borland C++,则在.icode 块中)。Jmp 指令要跳转到的地方才是真正的目的地址。装载程序判定目标函数的地址并将该函数插补到执行文件的映像中,所需要的信息都放在 PE 文件的.idata 中,也就是 Import 节。Idata 节以一个 IMAGE_IMPORT_DESCRIPTOR 数组开始。每一个被 PE 文件隐式连结进来的 DLL 都有一个 IMAGE_IMPORT_DESCRIPTOR。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,可以由此计算出该数组的项数。

引入函数是指 PE 文件在执行时需要调用的其它 EXE 文件或 DLL 文件中的函数。因此 PE 文件本身应该包含这些函数的一些信息如 DLLs 文件名、函数名、或序号。PE 文件将这些信息存放在 IMAGE_IMPORT_DESCRIPTOR 数据结构的数组中,这个数组被称为引入函数表。前面已经说过在 PE 文件的 PE 首部的可选头部 OptionalHeader 中有一项 DataDirectory,这一项是

一个 IMAGE_DATA_DIRECTORY 结构的数组，共有 16 项。它保存了 PE 文件中一些极其重要数据结构的偏移地址和大小。

1.4 PE 的 EXPORT

当一个 EXE/DLL 引出一个函数给其他 EXE/DLL 使用时有两种方式。一是以函数名引出，另一是以序号引出。如 DLL 中有一个名为 GetSysConfig 的函数，其他 EXE/DLL 调用这个函数时就可以指定 GetSysConfig 这个名字。若 GetSysConfig 函数在 DLL 中的序号是 16，则其他 EXE/DLL 也可以通过序号 16 来调用这个函数，这就是以序号引出的方式。不能以序号引出的方式，因为它会带来维护上的困难，如果 DLL 文件经过升级之后导致函数序号发生变化，则其它通过序号调用该 DLL 中函数的可执行文件将可能崩溃。与引入函数表一样引出函数表的 RVA 可以在 DataDirectory 的第一项找到。引出函数表是一个 IMAGE_EXPORT_DIRECTORY 结构，它包括 11 项。

引出表的设计是为了方便 PE 装载器工作。首先，模块必须保存所有引出函数的地址以供 PE 装载器查询。模块将这些信息保存在 AddressOfFunctions 域指向的数组中，而数组元素数目存放在 NumberOfFunctions 域中。因此，如果模块引出 40 个函数，则 AddressOfFunctions 指向的数组必定有 40 个元素，而 NumberOfFunctions 值为 40。现在如果有一些函数是通过名字引出的，那么模块必定也在文件中保留了这些信息。这些名字的 RVAs 存放在一数组中以供 PE 装载器查询。该数组由 AddressOfNames 指向，NumberOfNames 包含名字数目。考虑一下 PE 装载器的工作机制，它知道函数名，并想以此获取这些函数的地址。迄今为止，PE 已有两个模块：名字数组和地址数组，但两者之间还没有联系的纽带。因此我们还需要一些联系函数名及其地址的东西。PE 使用地址数组的索引作为联接，因此 PE 装载器在名字数组中找到匹配名字的同时，也获取了指向地址表中对应元素的索引。这些索引保存在由 AddressOfNameOrdinals 域指向的另一个数组（最后一个）中。由于该数组起到了联系名字和地址的作用，所以其元素数目必定和名字数组相同，如每个名字有且仅有一个相关地址。反过来的情况却不一定，每个地址可以对应几个名字。因此需要给同一个地址取“别名”。为了起到连接作用，名字数组和索引数组必须并行地成对使用，譬如，索引数组的第一个元素必定含有第一个名字的索引，以此类推。

当加载程序查找 PE 文件中引入函数的地址时，即查找目标 DLL 文件中相应引出函数的地址时，若是通过名字调用，它就先查找函数名字表。如在第 18 项，找到后将对应名字序号表中第 18 项的序号取出，假设为 25，然后再查找函数地址表中的第 25 号，得到调用函数地址。若是通过序号调用，则直接查找函数地址表中的第 25 号得到调用函数地址。

可以设想在编写 DLL 库时，有时为了一些特殊原因给一个函数赋以序号 200，但这并不意味着前面就一定有了 200 个函数。如果没有 Base，在函数地址表中前面 200 个地址都被空置浪费了。我们可将 Base 设为 200，意味着前 200 个函数为空，这样就不用在函数地址表中空置 200 个地址。先将函数序号减去 Base 后再去查找函数地址表，既节约了空间又节约了时间。

2 PE 中的资源

资源是指对话框、菜单、图标、字符串等数据。在 PE 文件中常为它单独分作一块（一节）。并将该块（节）的属性设为 IMAGE_SCN_INITIALIZED_DATA 和 IMAGE_SCN_MEM_READ，表示已初始化数据与可读。它的首地址放在 DataDirectory 的第 3 项 PE 文件中，其资源是按着树形结构存放的，如图 2 所示。

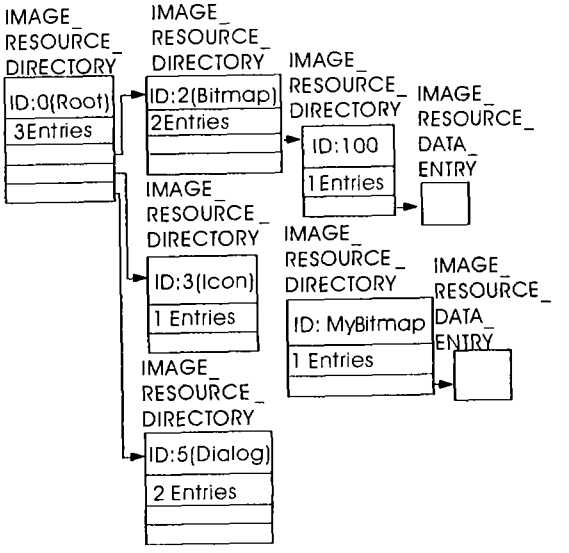

图 2　资源关系图

其中，每一个节点都是一个 IMAGE_RE-

SOURCE_DIRECTORY 形式的结构，而每一个叶子是 IMAGE_RESOURCE_DIRECTORY_ENTRY 形式的结构。

资源支持多种语言版本。系统会根据本机的语言选择相对应的语言版本。若没有，则采用相近的语言版本代替。用宏 PRIMARYLANGID() 与 SUBLANGID()可以分离出语言 ID 的主 bit0～9 次 bit10～15 标识，从而区分各个语系。若还没有，则会选择 ID 数值最小的语言版本，但这种选择仅在 NT 下才有效，在 Win95 下，系统只会选择第 1 种语言版本的资源。ID 既可以是数值也可以是字符形式的名称，当 ID 最高位是 0 时，表示 ID 是数值。其中系统默认的数值含义如下，其它数值都认为是用户定义的。

1 Cursor; 2 Bitmap; 3 Icon; 4 Menu; 5 Dialog; 6 String table; 7 Font directory; 8 Font; 9 Accelerators; 10 Unformatted resource data; 11 Message table; 12 Group cursor; 14 Group icon; 16 Version information

当 ID 最高位是 1 时，表示 ID 是字符形式的名称。它的低 31 位表示名称字符串到 Rawdata 起始处的偏移量，注意这里是 Unicode 字符串，并且不以 0 结尾(资源中的任何字符串都是 Unicode 字符，用户定义的资源除外)。

同样 Offset 的最高位也有特殊的含义：当它为 1 时表示低 31 位代表结点数据的偏移量，即它所指向的是 IMAGE_RESOURCE_DIRECTORY 结构。当它为 0 时，表示低 31 位代表叶子数据的偏移量，即指向 IMAGE_RESOURCE_DATA_ENTRY 结构。具体到 Rawdata 的格式与资源的类型有关具体说明可参见 MS SDK 文档。

3 重定位

另一个需要提到的是基址重定位目录，它的首地址放在 DataDirectory 的第五项。在 PE 文件中，它往往单独分为一节，用. reloc 表示。并将该块设为以下 3 种属性：IMAGE_SCN_CNT_INITIALIZED_DATA 、IMAGE_SCN_MEM_DISCARDABLE、IMAGE_SCN_MEM_READ，即表示已初始化数据可放弃可读。

当 PE 文件由于某些原因不能被加载到 PE 可选头中指定的地址时，加载器需要利用这里的数据进行重定位操作，以修改文件中的静态变量字符串等的绝对地址，以保证程序正常运行。

重定位目录是由许多段串接成的。每一段存放着 4K 大小 PE 映像的重定位信息。每一段都以 IMAGE_BASE_RELOCATION 结构开始，后跟一串两字节的重定位数据。

由此，重定位数据的个数可以由下式得到：

(SizeOfBlock - sizeof(IMAGE_BASE_RELOCATION)) /2

重定位目录的结束标志是以一个特殊的段作为结尾，该段的 VirtualAddress 值为 0。前面已说到一个重定位数据有两个字节共 16 位，它又分为高 4 位与低 12 位。高 4 位代表重定位类型，低 12 位是重定位地址。它与 IMAGE_BASE_RELOCATION 中的 VirtualAddress 相加，即是指向 PE 映像中需要修改的地址数据的指针。

执行 PE 文件前，加载程序在进行重定位时，会将 PE 文件在内存中实际的映像地址 actual_base 减去 PE 文件所要求的映像地址 prefered_base，得到一个差值符号数，再将这一差值根据重定位类型的不同添加到地址数据。

4 结束语

从多种意义上讲，操作系统的可执行文件的格式是操作系统本身执行机制的反映。虽然研究可执行文件格式并非一个程序员的首要任务，但这种工作能积累大量的知识，使我们对系统的最底层工作原理有更深刻的认识。

Winnt. h 提供了 PE 文件中要使用的原始数据结构，但有关结构和标记含义的注释很有限，要深入理解 PE 格式，这些还远远不够。Microsoft 在推出的 Visual Studio 6. 0 中的配套光盘 MSDN(Microsoft Developer Network)为我们提供了 Microsoft 的大量资料，其中包含了较为详尽的关于 PE 可执行文件的信息。

参考文献

1 Namir Clement Shammas 著. Windows 程序员使用指南(二). 张新宇，杨明译. 北京：清华大学出版社，1994：17～45

2 Tom Armstrong Ron Patton 著. ATL 开发指南. 董梁，丁杰，李长业等译. 北京：电子工业出版社，2000：33～35

3 Jeffrey Richter 著. Microsoft . NET Framework 程序设计. 商丽媛译. 北京：清华大学出版社，2002：56～81