文章编号: 1007- 2985(2003)01- 0077- 03

32 位 Windows 系统下的 PE 文件结构及其应用*

何迎生, 段明秀
(吉首大学数学与计算机科学系, 湖南 吉首 416000)

摘 要: PE 文件结构是 Win32 环境所带的可执行文件格式, 该结构由 Dos ' MZ' Header, Dos Stub, PE Header, Section Table, Section 等 5 大部分组成, 各部分之间通过指针相互联系形成一个整体, 借助这些指针即可实现对任何一个可执行文件的内部结构和所用到的资源进行分析的操纵.

关键词: PE 结构; 相对虚拟地址; 动态连接库; Windows Api; 句柄
中图分类号: TP311.56 文献标识码: A

无论是早期的还是现在的病毒, 大部分都是通过感染可执行文件来传播的. 因此, 熟练掌握可执行文件的文件结构, 对编制防病毒软件和病毒预防都有重要意义. 迄今为止, 已有许多对 MS- DOS 下的可执行文件结构的介绍, 但现在经常使用的 32 位 Windows 系统下可执行文件的文件结构却很少见报道. 笔者对 32 位 Windows 系统下的可执行文件结构进行了分析.

1 基本概念

PE: Portable Executable(可移植的执行体)[1]. 它是 Win32 环境自身所带的执行体文件格式, 它的一些特性继承自 Unix 的 Coff (common object file format) 文件格式. 可移植的执行体意味着此文件格式是跨 Win32 平台的. 即使 Windows 运行在非 Intel 的 CPU 上, 任何 Win32 平台的 PE 装载器都能识别和使用该文件格式.

RVA: Relative Virtual Address(相对虚拟地址)[1]. 一个 RVA 是某一个数据项的偏移地址(从文件被映射到内存的首地址算起). 例如, 有个文件被 Windows 装载器映射到虚拟地址空间的 0x400000 处, 如果文件中有一表格开始于 RVA 地址 0x2410 处, 那么表格在虚拟地址空间的首地址是 0x402410, 即虚拟空间地址 0x402410= 基底地址 0x400000+ RVA 0x2410.

Section: PE 文件的内容被划分成块, 称之为 Sections(节)[1], 每节是一块拥有共同属性的数据. 例如, 程序的可执行代码存放在. text Section 中, 它类似于汇编语言编程时的代码段, 专用于存储程序的代码部分.

2 PE 的基本结构

从整体上看, PE 的基本结构如图 1[1] 所示.

(1) DOS ' MZ' Header. 所有 PE 文件(甚至 32 位的 DLLs)都必须以一个简单的 DOS ' MZ' Header 开始. 它是一个带有 19 个属性域的结构体, 一般情况下, 有用的属性域只有 2 个. 一个是 E _ magic 属性域, 它总是等于"MZ"的 ASCII 码值, 当程序在 DOS 下执行时, DOS 通过该值就能识别出这是有效的执行体, 然后运行紧随 MZ header 之后的 DOS stub. 另一个是 E _ lfanew 属性域, 它是一个 RVA 地址, 用于存储 PE Header 在文件中的偏移地址, 通过这个地址才有可能找到文件中的其它有用信息.

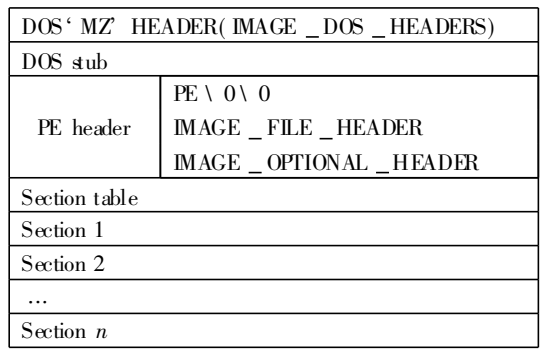

图 1 PE 的基本结构

(2) DOS Stub. 它实际上是个有效的 EXE 文件, 在不支持 PE 文件格式的操作系统中, 它将简单显示一个错误提示. 例如, 在 MS- DOS 下, 它简单调用中断 21H 服务 9 来显示字符串"This program cannot run in DOS mode". 大多数情况下, 它是由

* 收稿日期: 2002- 10- 08
作者简介: 何迎生(1974-), 男, 湖南省浏阳市人, 吉首大学数学与计算机科学系助教, 主要从事计算机网络、面向对象程序设计研究.

编译器自动生成. 当然程序员可根据自己的意图实现完整的 DOS 代码, 这样就可得到一种“两栖”型的程序, 即该程序既可以在 DOS 下也可以在 Windows 下运行.

(3) PE Header. 它是一个结构体, 在 VC+ + 中对应的结构叫作 IMAGE _NT _HEADERS, 事实上它才是 Windows 下可执行文件的头部. 该结构由 3 个属性域组成. 第 1 个属性域叫 Signature , 它的值总是等于字符串“PE\0\0”的 ASCII 码值, 借助这一属性, Windows 装载器可以判断一个文件是否是可执行文件. 第 2 个属性域叫 IMAGE _FILE _HEADER, 第 3 个叫 IMAGE _OPTIONAL _HEADER, 它们又分别是一个结构体. IMAGE0 _FILE _HEADER 结构包含了关于 PE 文件物理分布的一般信息, 包含 7 个属性域, 但有用的只有 Number Of Sections. 该属性域指定文件的节数目, 确定了后面的 Section table 结构数组元素的个数, 如果要在文件中增加或删除 1 个节就需要修改这个值. IMAGE _OPTIONAL _HEADER 结构域包含了关于 PE 文件逻辑分布的信息, 它是 PE header 中最后、最大也是最重要的成员. 该结构共有 31 个域, 这里只介绍有用的域. Address Of Entry Point Windows 的 PE 装载器准备运行的 PE 文件的第 1 个指令的 RVA, 即整个可执行代码的入口地址, 若想改变整个执行的流程, 可以将该值指定到新的 RVA, 这样, 新 RVA 处的指令首先被执行. Section Alignment 内存中节对齐的单位. 例如, 如果该值是 4 096 (1000H) , 那么每节的起始地址必须是 4 096 的倍数. 若第 1 节从 401000H 开始且大小是 10 个字节, 则下一节必定从 402000H 开始, 即使 401000H 和 402000H 之间还有很多空间没被使用. File Alignment 文件中节对齐的单位. 例如, 如果该值是 200H, 那么每节的起始地址必须是 512 的倍数. 若第 1 节从文件偏移量 200H 开始且大小是 10 个字节, 则下一节必定位于偏移量 400H, 即使偏移量 512 和 1 024 之间还有很多空间没被使用. 众所周知的 CIH 病毒正是借助这些空间把病毒代码插入可执行文件而不会增加文件大小的. Size Of Image 整个 PE 映像体在内存中的尺寸, 它是所有头和节经过节对齐处理后的大小. DataDirectory 这一结构数组, 每个结构给出一个重要数据结构的 RVA, 比如引入地址表(即本文件所调用到的动态连接库 DLL 中的函数地址表)、指向资源的 RVA、引出地址表(一般对 DLL 文件有效, 指出本动态连接库所能提供的函数及各函数地址表)等. 与该数组相关的结构还有很多, 限于篇幅不再加以说明.

(4) Section table. 节表其实就是紧挨着 PE header 的一结构数组. 该数组成员的数目由 IMAGE _FILE _HEADER 结构中的 NumberOfSections 属性域的域值来决定, 有多少个成员就说明紧跟节表后有多少个节. 数组的每个成员是一个命名为 IMAGE _SECTION _HEADER 的结构, 该结构共有 10 个属性域. 同样, 不是所有成员都很有用, 笔者只介绍较重要的.

NAME: 这是一个 8 个字节长的 ANSI 字符串, 表示节的名称. 大多数节名称都以点号开始(如“. text”或“. code”代表代码节,“. rsrc”代表资源数据节等) , 但这不是必要条件, 也可以自己命名甚至空着. Virtual Size: 前面文件中节对齐的单位 File Alignment: 指明了任何一节都必须是 File Alignment 的整数倍, 而 Virtual Size 记载的是本节的实际长度, 即没有经过节对齐的字节数. SizeOfRawData: 经过文件对齐处理后的节尺寸, PE 装载器通过本域值来确定将节映射入内存时所需的字节数. 假设某一个文件的文件对齐尺寸是 0x200, 如果前面的 Virtual Size 域指示本节长度是 0x388 字节, 则本域值为 0x400, 表示本节是 0x400 字节长. Pointer To RawData: 这是节基于文件的偏移量. 如果以文件映射的方式把一个可执行文件装入内存, 而不是通过 Windows 的 PE 装载器装入, 则这个属性域是非常重要的, 因为这时是完全以线性的方式映射文件, 而不是通过 Virtual Address 中的 RVA 值装入, 所以必须通过本域值找到节数据在文件中的位置, 以此操纵各种数据.

(5) Section. 紧跟节表 Section Table 之后的就是节, 它们才是具体存放 PE 数据的地方. 所有的代码、初始化数据、未初始化数据、资源、引入函数、引出函数、调试信息等各类数据, 最终都按各自的特点存放在各个不同的节中. Windows 的 PE 装载器通过前面介绍的各种结构及 RVA 在节中找到的各种数据的位置和大小并装入内存, 最终保证了各 PE 文件的正确运行.

3 PE 结构的应用

熟练地掌握了 PE 结构之后, 以前很难做到的事现在都可轻松做到. 例如, 想使用他人程序中漂亮的图标, 可以通过 PE 结构找到图标资源在原文件中的位置, 并读出来存储为一个独立的文件供使用; 也可以做 1 个工具, 对所有不想让他人随便执行的程序加把锁, 且不论这些程序是不是自己编制的; 还可以将 2 个 EXE 文件捆绑成 1 个文件, 使得某一个文件执行时另一个也执行, 例如将自己做的程序与 Windows 系统程序捆绑在一起, 当系统启动时就运行自己的程序, 这比将自己的程序添加到 Windows 启动菜单和注册表中都可靠得多.

下面, 以从 EXE 文件中找出文件所调用的动态连接库和用到了每个库中的哪些函数为例, 介绍 PE 结构的具体操作过程.

(1) 利用 Windows Api 函数 CreateFile, CreateFileMapping 和 MapViewOfFile[2] 将所分析的 EXE 文件映射入内存, 并获得该文件的句柄.

(2) 从文件的句柄开始取结构 DOS _HEADER, 并判断该结构中的 E _magic 属性域是否等于“MZ”. 如果等于则读出结构中 E _lfanew 属性域的值, 并由此定位 PE _HEADER 结构; 否则说明该文件不是可执行文件.

(3) 判断 PE _HEADER 中的 Signature 属性域是否等于“PE\0\0”, 如果等于则继续, 否则说明该文件不是 PE 结构.

(4) 定位 PE _HEADER 中 IMAGE _OPTIONAL _HEADER 结构下的 DataDirectory 数组, 转至第 2 个数组成员并提取其 VirtualAddress 值, 利用该值定位第 1 个 IMAGE _IMPORT _DESCRIPTOR 结构.

(5) 检查 IMAGE _ IMPORT _ DESCRIPTOR 结构中的OriginalFirstThunk 值, 若不为0 则顺着 OriginalFirstThunk 里的 RVA 值转入 RVA 数组, 若为 0 就改用 FirstThunk 值.

(6) 对于每个数组元素, 比较元素值是否等于80000000H. 如果该元素值的最高二进位为 1, 那么函数是由序数引入的, 可以从该值的低字节提取序数. 如果元素值的最高二进位为 0, 就将该值作为 RVA 转入 IMAGE _ IMPORT _ BY _ NAME 数组, 跳过 Hint 就是函数名字了.

(7) 再跳至下一个数组元素提取函数名一直到数组底部(它以 null 结尾). 至此已遍历完一个 DLL 的引入函数, 接着处理下一个 DLL, 即跳转到下一个 IMAGE _ IMPORT _ DESCRIPTOR 并进行处理. 如此循环直到数组见底.

4　结语

由于相关技术资料的缺乏, PE 结构直到现在还不为大家所熟知. 但 PE 结构与操作系统的联系是如此紧密, 一旦熟练地掌握了它的结构和运行机制, 就有机会深入地了解 Windows 的内部结构.

参考文献:

[1]　MATT PIETREK. Windows95 系统程式设计大奥秘[M]. 台湾: 旗标出版有限公司, 1997.

[2]　季雪岗, 王晓辉, 张宏林. Delphi 编程疑难详解[M]. 北京: 人民邮电出版社, 2000.

PE Structure Under the 32- Bit Windows System and Its Application

HE Ying-sheng, DUAN Ming-xiu

(Deparment of Mathematics and Computer Science, Jishou University, Jishou 416000, Hunan China)

Abstract: PE Structure is a file- structure of executable of file under the Win32's system, this structure is made up of 5 parts, Dos 'MZ' Header, Dos Stub, PE Header, Section Table & Section. All the parts form a unit, communication with each other by the pointer. By the pointer, we can analyse & operate the resources used and the innards of any executable file.

Key words: PE structure; relative virtual address; dynamic link library; Windows Api; handle