比尔丁子

爱吃爱睡爱踢球，爱贫爱闹爱旅游

个人资料

qiming_zhang

访问：114329次

积分：1461

等级：BLOG 4

PE结构详解

标签：windows dll table microsoft exception dos

2012-03-01 15:42 15609人阅读 评论(4) 收藏 举报

分类：

C/C++（14）

目录(?) [+]

1 基本概念

下表描述了贯穿于本文中的一些概念：

名称	描述
地址	是“虚拟地址”而不是“物理地址”。为什么不是“物理地址”呢？因为数据在内存的位置经常在变，这样可以节省内存开支、避开错误的内存位置等的优势。同时用户并不需要知道具体的“真实地址”，因为

排名：千里之外

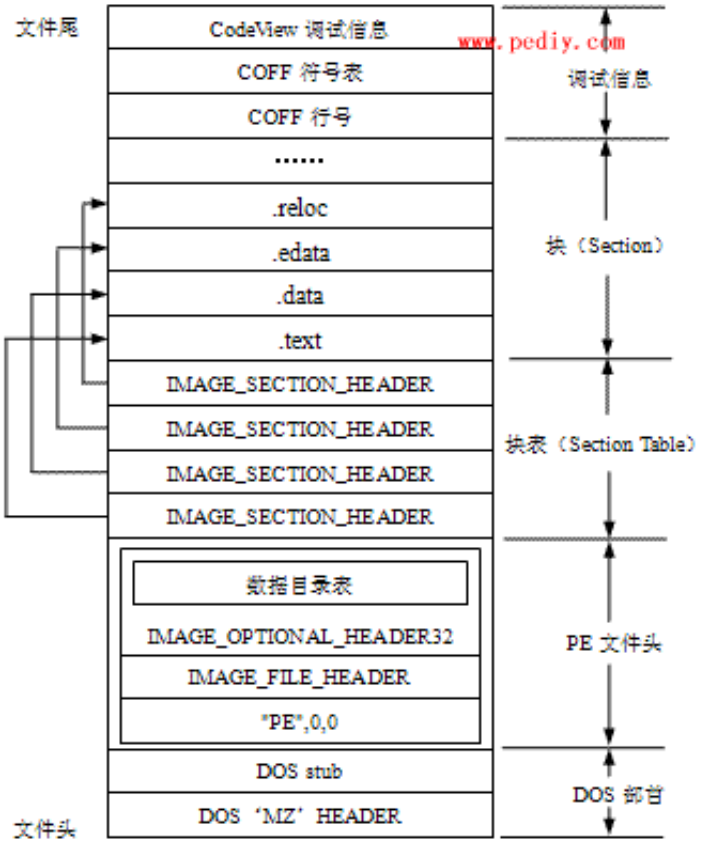
原创：29篇 转载：17篇
译文：0篇 评论：8条

	系统自己会为程序准备好内存空间的(只要内存足够大)
镜像文件	包含以EXE文件为代表的“可执行文件”、以DLL文件为代表的“动态链接库”。为什么用“镜像”？这是因为他们常常被直接“复制”到内存，有“镜像”的某种意思。看来西方人挺有想象力的哦^0^
RVA	英文全称Relatively Virtual Address。偏移(又称“相对虚拟地址”)。相对镜像基址的偏移。
节	节是PE文件中代码或数据的基本单元。原则上讲，节只分为“代码节”和“数据节”。
VA	英文全称Virtual Address。基址

2 概览

x86都是32位的，IA-64都是64位的。64位Windows需要做的只是修改PE格式的少数几个域。这种新的格式被称为PE32+。它并没有增加任何新域，仅从PE格式中删除了一个域。其余的改变就是简单地把某些域从32位扩展到64位。在大部分情况下，你都能写出同时适用于32位和64位PE文件的代码。

EXE文件与DLL文件的区别完全是语义上的。它们使用的是相同的PE格式。惟一的不同在于一个位，这个位用来指示文件应该作为EXE还是DLL。甚至DLL文件的扩展名也完全也是人为的。你可以给DLL一个完全不同的扩展名，例如.OCX控件和控制面板小程序(.CPL)都是DLL。



PE 文件框架结构

图1 解释了Microsoft PE可执行文件格式：

PE文件总体上分为“头”和“节”。“头”是“节”的描述、简化、说明，“节”是“头”的具体化。

3 文件头

PE文件的头分为DOS头、NT头、节头。注意，这是本人的分法，在此之前并没有这种分法。这样分法会更加合理，更易理解。因为这三个部分正好构成SizeOfHeaders所指的范围，所以将它们合为“头”。这里的3个头与别的文章的头的定义会有所区别。

节头紧跟在NT头后面。

3.1 DOS头（PE文件签名的偏移地址就是大小）

用记事本打开任何一个镜像文件，其头2个字节必为字符串“MZ”，这是Mark Zbikowski的姓名缩写，他是最初的MS-DOS设计者之一。然后是一些在MS-DOS下的一些参数，这些参数是在MS-DOS下运行该程序时要用到的。在这些参数的末尾也就是文件的偏移0x3C(第60字节)处是是一个4字节的PE文件签名的偏移地址。该地址有一个专用名称叫做“E_lfanew”。这个签名是“PE00”(字母“P”和“E”后跟着两个空字节)。紧跟着E_lfanew的是一个MS-DOS程序。那是一个运行于MS-DOS下的合法应用程序。当可执行文件(一般指exe、com文件)运行于MS-DOS下时，这个程序显示“This program cannot be run in DOS mode(此程序不能在DOS模式下运行)”这条消息。用户也可以自己更改该程序，有些还原软件就是这么干的。同时，有些程序既能运行于DOS又能运行于Windows下就是这个原因。Notepad.exe整个DOS头大小为224个字节，大部分不能在DOS下运行的Win32文件都是这个值。MS-DOS程序是可有可无的，如果你想使文件大小尽可能的小可以省掉MS-DOS程序，同时把前面的参数都清0。

3.2 NT头(244或260个字节)

紧跟着PE文件签名之后，是NT头。NT头分成3个部分，因为第2部分在32与64位系统里有区别，第3部分虽然也是头，但实际很不像“头”。

第1部分(20个字节)

偏移	大小	英文名	中文名	描述
0	2	Machine	机器数	标识CPU的数字。参考3.2.1节“机器类型”。
2	2	NumberOfSections	节数	节的数目。Windows加载器限制节的最大数目为96。
4	4	TimeDateStamp	时间/日期标记	UTC时间1970年1月1日00:00起的总秒数的低32位，它指出
8	8	已经废除		
16	2	SizeOfOptionalHeader	可选头大小	第2部分+第3部分的总大小。这个大小在32位和64位文件于64位文件来说，它是240。
18	2	FillCharacteristics	文件特征值	指示文件属性的标志。参考3.2.2节“特征”。

第2部分(96或112个字节)

了

偏移	大小	英文名	中文名	描述
0	2	Magic	魔数	这个无符号整数指出了镜像文件的状态。 0x10B表明这是一个32位镜像文件。 0x107表明这是一个ROM镜像。 0x20B表明这是一个64位镜像文件。
2	1	MajorLinkerVersion	链接器的主版本号	链接器的主版本号。
3	1	MinorLinkerVersion	链接器的次版本号	链接器的次版本号。
4	4	SizeOfCode	代码节大小	一般放在“.text”节里。如果有多个代码节的话，它是所有代码节的和。必须是FileAlignment的整数倍，是在文件里的大小。
8	4	SizeOfInitializedData	已初始化数大小	一般放在“.data”节里。如果有多个这样的节话，它是所有这些节的和。必须是FileAlignment的整数倍，是在文件里的大小。
12	4	SizeOfUninitializedData	未初始化数大小	一般放在“.bss”节里。如果有多个这样的节话，它是所有这些节的和。必须是FileAlignment的整数倍，是在文件里的大小。
16	4	AddressOfEntryPoint	入口点	当可执行文件被加载进内存时其入口点RVA。对于一般程序镜像来说，它就是启动地址。为0则从ImageBase开始执行。对于dll文件是可选的。

20	4	BaseOfCode	代码基址	当镜像被加载进内存时代码节的开头RVA。必须是SectionAlignment的整数倍。
24	4	BaseOfData	数据基址	当镜像被加载进内存时数据节的开头RVA。(在64位文件中此处被并入紧随其后的ImageBase中。)必须是SectionAlignment的整数倍。
28/24	4/8	ImageBase	镜像基址	当加载进内存时镜像的第1个字节的首选地址。它必须是64K的倍数。DLL默认是10000000H。Windows CE 的EXE默认是00010000H。Windows 系列的EXE默认是00400000H。
32	4	SectionAlignment	内存对齐	当加载进内存时节的对齐值(以字节计)。它必须≥FileAlignment。默认是相应系统的页面大小。
36	4	FileAlignment	文件对齐	用来对齐镜像文件的节中的原始数据的对齐因子(以字节计)。它应该是界于512和64K之间的2的幂(包括这两个边界值)。默认是512。如果SectionAlignment小于相应系统的页面大小，那么FileAlignment必须与SectionAlignment相等。
40	2	MajorOperatingSystemVersion	主系统的主版本号	操作系统的版本号可以从“我的电脑”→“帮助”里面看到，Windows XP是

				5.1。5是主版本号，1是次版本号
42	2	MinorOperatingSystemVersion	主系统的次版本号	
44	2	MajorImageVersion	镜像的主版本号	
46	2	MinorImageVersion	镜像的次版本号	
48	2	MajorSubsystemVersion	子系统的主版本号	
50	2	MinorSubsystemVersion	子系统的次版本号	
52	2	Win32VersionValue	保留，必须为0	
56	4	SizeOfImage	镜像大小	当镜像被加载进内存时的大小，包括所有的文件头。向上舍入为SectionAlignment的倍数。
60	4	SizeOfHeaders	头大小	所有头的总大小，向上舍入为FileAlignment的倍数。可以以此值作为PE文件第一节的文件偏移量。
64	4	CheckSum	校验和	镜像文件的校验和。计算校验和的算法被合并到了Imagehlp.DLL 中。以下程序在加载时被校验以确定其是否合法：所有的驱动程序、任何在引导时被加载的DLL以及加载进关键Windows进程中的DLL。
68	2	Subsystem	子系统类型	运行此镜像所需的子系统。参考后面的“Windows子系统”部分。
70	2	DllCharacteristics	DLL标识	参考后面的“DLL特征”部分。
72	4/8	SizeOfStackReserve	堆栈保留大小	最大栈大小。CPU的堆栈。默认是

				1MB。
76/80	4/8	SizeOfStackCommit	堆栈提交大小	初始提交的堆栈大小。默认是4KB。
80/88	4/8	SizeOfHeapReserve	堆保留大小	最大堆大小。编译器分配的。默认是1MB。
84/96	4/8	SizeOfHeapCommit	堆栈交大小	初始提交的局部堆空间大小。默认是4KB。
88/104	4	LoaderFlags	保留，必须为0	
92/108	4	NumberOfRvaAndSizes	目录项数目	数据目录项的个数。由于以前发行的Windows NT的原因，它只能为16。

第3部分数据目录(128个字节)

偏移(PE32/PE32+)	大小	英文名	描述
96/112	8	Export Table	导出表的地址和大小。参考5.1节“.edata”
104/120	8	Import Table	导入目录表的地址和大小。参考5.2.1节“.idata”
112/128	8	Resource Table	资源表的地址和大小。参考5.6节“.rsrc”
120/136	8	Exception Table	异常表的地址和大小。参考5.3节“.pdata”
128/144	8	Certificate Table	属性证书表的地址和大小。参考6节“属性证书表”
136/152	8	Base Relocation Table	基址重定位表的地址和大小。参考5.4节“.reloc”
144/160	8	Debug	调试数据起始地址和大小。
152/168	8	Architecture	保留，必须为0
			将被存储在全局指针寄存器中的一个值的RVA。这个结构的Size域必

160/176	8	Global Ptr	须为0
168/184	8	TLS Table	线程局部存储(TLS)表的地址和大小。
176/192	8	Load Config Table	加载配置表的地址和大小。参考5.5节“加载配置结构”
184/200	8	Bound Import	绑定导入查找表的地址和大小。参考5.2.2节“导入查找表”
192/208	8	IAT	导入地址表的地址和大小。参考5.2.4节“导入地址表”
200/216	8	Delay Import Descriptor	延迟导入描述符的地址和大小。
208/224	8	CLR Runtime Header	CLR运行时头部的地址和大小。(已废除)
216/232	8	保留，必须为0	

3.2.1 机器类型

Machine域可以取以下各值中的一个来指定CPU类型。镜像文件仅能运行于指定处理器或者能够模拟指定处理器的系统上。

值	描述
0x0	适用于任何类型处理器
0x1d3	Matsushita AM33处理器
0x8664	x64处理器
0x1c0	ARM小尾处理器
0xebc	EFI字节码处理器
0x14c	Intel 386或后继处理器及其兼容处理器
0x200	Intel Itanium处理器
0x9041	Mitsubishi M32R小尾处理器

0x266	MIPS16处理器
0x366	带FPU的MIPS处理器
0x466	带FPU的MIPS16处理器
0x1f0	PowerPC小尾处理器
0x1f1	带符点运算支持的PowerPC处理器
0x166	MIPS小尾处理器
0x1a2	Hitachi SH3处理器
0x1a3	Hitachi SH3 DSP处理器
0x1a6	Hitachi SH4处理器
0x1a6	Hitachi SH5处理器
0x1c2	Thumb处理器
0x169	MIPS小尾WCE v2处理器

3.2.2 特征

Characteristics域包含镜像文件属性的标志。以下加粗的是常用的属性。当前定义了以下值(由低位往高位):

位置	描述
0	它表明此文件不包含基址重定位信息，因此必须被加载到其首选基地址上。如果基地址不可用，加载器会报错。
1	它表明此镜像文件是合法的。看起来有点多此一举，但又不能少。
2	保留，必须为0。
3	

4	
5	应用程序可以处理大于2GB的地址。
6 7	保留，必须为0。
8	机器类型基于32位体系结构。
9	调试信息已经从此镜像文件中移除。
10	如果此镜像文件在可移动介质上，完全加载它并把它复制到交换文件中。几乎不用
11	如果此镜像文件在网络介质上，完全加载它并把它复制到交换文件中。几乎不用
12	此镜像文件是系统文件，而不是用户程序。
13	此镜像文件是动态链接库(DLL)。
14	此文件只能运行于单处理器机器上。
15	保留，必须为0。

Windows子系统

为NT头第2部分的Subsystem域定义了以下值以确定运行镜像所需的Windows子系统(如果存在)：

值	描述
0	未知子系统
1	设备驱动程序和Native Windows进程
2	Windows图形用户界面(GUI)子系统(一般程序)
3	Windows字符模式(CUI)子系统(从命令提示符启动的)
7	Posix字符模式子系统
9	Windows CE
10	可扩展固件接口(EFI)应用程序

11	带引导服务的EFI驱动程序
12	带运行时服务的EFI驱动程序
13	EFI ROM镜像
14	XBOX

DLL特征

为NT头的DllCharacteristics域定义了以下值:

位置	描述
1	保留，必须为0。
2	
3	
4	
5	官方文档缺失
6	官方文档缺失
7	DLL可以在加载时被重定位。
8	强制进行代码完整性校验。
9	镜像兼容于NX。
10	可以隔离，但并不隔离此镜像。
11	不使用结构化异常(SE)处理。
12	不绑定镜像。
13	保留，必须为0。
14	WDM驱动程序。
15	官方文档缺失

16	可以用于终端服务器。

每个数据目录给出了Windows使用的表或字符串的地址和大小。这些数据目录项全部被被加载进内存以备系统运行时使用。数据目录是按照如下格式定义的一个8字节结构：

```
typedef struct
  DWORD VirtualAddress;  //数据的RVA
  DWORD Size;            //数据的大小
typedef ENDS
```

第1个域——VirtualAddress，实际上是表的RVA。相对镜像基址偏移地址。NT头第2部分的ImageBase
第2个域给出了表的大小(以字节计)。数据目录组成了NT头的最后一部分。
Certificate Table域指向属性证书表。它的第一个域是一个文件指针，而不是通常的RVA。

3.3 节头

在镜像文件中，每个节的RVA值必须由链接器决定。这样能够保证这些节位置相邻且按升序排列，并且这些RVA值必须是NT头中SectionAlignment域的倍数。
每个节头(节表项)格式如下，共40个字节：

偏移	大小	英文名	描述
0	8	Name	这是一个8字节ASCII编码的字符串，不足8字节时用NULL填充，必须使其达到8字节。如果它正好是8字节，那就没有最后的NULL字符。可执行镜像不支持长度超过8字节的节名。
8	4	VirtualSize	当加载进内存时这个节的总大小。如果此值比SizeOfRawData大，那么多出的部分用0填充。这是节的数据在没有进行对齐处理前的实际大小，不需要内存对齐。
12	4	VirtualAddress	内存中节相对于镜像基址的偏移。必须是SectionAlignment的整数倍。
16	4	SizeOfRawData	磁盘文件中已初始化数据的大小。它必须是NT头中FileAlignment域的倍数。当节中仅包含未初始化的数据时，这个域应该为0。

20	4	PointerToRawData	节中数据起始的文件偏移。它必须是NT头中FileAlignment域的倍数。当节中仅包含未初始化的数据时，这个域应该为0。
24	4	PointerToRelocations	重定位项开头的文件指针。对于可执行文件或没有重定位项的文件来说，此值应该为0。
28	4	已经废除。	
32	2	NumberOfRelocations	节中重定位项的个数。对于可执行文件或没有重定位项的文件来说，此值应该为0。
34	2	已经废除。	
36	4	Characteristics	描述节特征的标志。参考“节标志”。

3.3.1 节标志

节头中的Characteristics标志指出了节的属性。（以下加粗的是常用的属性值）

位置	描述
1	已经废除
2	
3	
4	
5	
6	此节包含可执行代码。代码段才用“.text”
7	此节包含已初始化的数据。“.data”
8	此节包含未初始化的数据。“.bss”
9	
10	

11	已经废除
12	
13	
14	
15	
16	此节包含通过全局指针(GP)来引用的数据。
17	已经废除
18	
19	
20	
21	
22	
23	
24	
25	此节包含扩展的重定位信息。
26	此节可以在需要时被丢弃。
27	此节不能被缓存。
28	此节不能被交换到页面文件中。
29	此节可以在内存中共享。
30	此节可以作为代码执行。
31	此节可读。(几乎都设置此节)

32	此节可写。

第25标志表明节中重定位项的个数超出了节头中为每个节保留的16位所能表示的范围(也就是65535个函数)。如果设置了此标志并且节头中的NumberOfRelocations域的值是0xffff，那么实际的重定位项个数被保存在第一个重定位项的VirtualAddress域(32位)中。如果设置了第25标志但节中的重定位项的个数少于0xffff，则表示出现了错误。

4 一些注意信息

1.PE头是怎么计算的？

SizeOfHeaders所指的头是从文件的第1个字节开始算起的，而不是从PE标记开始算起的。快速的计算方法是从文件的偏移0x3C(第59字节)处获得一个4字节的PE文件签名的偏移地址，这个偏移地址就是本文所定义的DOS头的大小。NT头在32位系统是244字节，在64位系统是260字节。节头的大小由NT头的第1部分的NumberOfSections(节的数量)*40字节(每个节头是40字节)得出。如此，DOS头、NT头、节头3个头的大小加起来并向上舍入为FileAlignment(文件对齐)的正整数倍的最小值就是SizeOfHeaders(头大小)值。

2.节数量的问题

Windows读取NumberOfSections的值然后检查节表里的每个结构，如果找到一个全0结构就结束搜索，否则一直处理完NumberOfSections指定数目的结构。没有规定节头必须以全0结构结束。所以加载器使用了双重标准——全0、达到NumberOfSections数量就不再搜索了。

3.未初始化问题

①未初始化数据在文件中是不占空间的，但在内存里还是会占空间的，它们依然依据指定的大小存在内存里。所以说未初始化数据只在文件大小上有优势，在内存里与已初始化数据是一样的。

②未初始化数据的方法有2种:1是通过节头的VirtualSize>SizeOfRawData。未初始化数据的大小就是VirtualSize-SizeOfRawData的值。2是节特征的标志置为“此节包含未初始化的数据”，这时SizeOfUninitializedData才会非0。现在 都使用第1种，把它们集成到.data里面可以加快速度。

4.已初始化问题

数据目录里面所对应的块中除了属性证书表、调试信息和几个废除的目录项外，全都属于

SizeOfInitializedData(已初始化数据大小)范围。当然，已初始化数据不只这些，还可以是常见的代码段等等。

5.节对齐的问题

如果NT头的SectionAlignment域的值小于相应操作系统(有些资料说是根据CPU来的，这不一定。因为CPU本身就允许改分页大小，只是大部分时候操作系统是用CPU默认值的。x86平台默认页面大小是4K。IA-64平台默认页面大小是8K。MIPS平台默认页面大小是4K。Itanium平台默认页面大小是8K。)平台的页面大小，那么镜像文件有一些附加的限制。对于这种文件，当镜像被加载到内存中时，节中数据在文件中的位置必须与它在内存中的位置相等，因此节中数据的物理偏移与RVA相同。

6.镜像大小

SizeOfImage所代表的内存镜像大小没有包含属性证书表和调试信息，这是因为加载器并不将属性证书和调试信息映射进内存。同时加载器规定，属性证书和调试信息必须被放在镜像文件的最后，并且属性证书表在调试信息节之前。

7.数据的组织

CPU的段主要分为4个:代码段、数据段、堆栈段、附加段。而操作系统给程序员留下只有代码段和数据段，堆栈段和附加段就由系统自行处理了，我们不用管。PE文件的数据组织方式是以BaseOfCode、BaseOfData为基准，以节为主体，以数据目录为辅助。

①BaseOfCode、BaseOfData是与后面相应的代码节、数据节的VirtualAddress一致。(这里的数据节是狭义的数据节，是特指代码段、数据目录所指定的数据除外的那一部分，也就是我们编程时定义的常量、变量、未初始化数据等)

②所有的代码、数据都必须在节里面，否则就算是代码基址、数据基址、数据目录都有指定，而节头里没有指定，加载器也会报错，不能运行

③导入函数、导出函数、资源、重定位表等是为了辅助程序主体的，这些都由系统负责处理

5 特殊的节

下表描述了保留的节以及它们的属性，后面是对出现在可执行文件中的节的详细描述。这些节是微软的编译产品所定义的不是系统定义的，实际可以不拘泥于此。

节名	内容
.bss	未初始化的数据
.data	代码节
.edata	导出表
.idata	导入表
.idlsym	包含已注册的SEH，它们用以支持IDL属性
.pdata	异常信息
.rdata	只读的已初始化数据(用于常量)
.reloc	重定位信息
.rsrc	资源目录
.sbss	与GP相关的未初始化数据
.sdata	与GP相关的已初始化数据
.srdata	与GP相关的只读数据
.text	默认代码节

5.1 .edata节

文件A的函数K被文件B调用时，函数K就称为导出函数。导出函数通常出现在DLL中，也可以是exe文件。

下表描述了导出节的一般结构。

表名	描述
导出	

目录表	它给出了其它各种导出表的位置和大小。
导出地址表	一个由导出函数的RVA组成的数组。它们是导出的函数和数据在代码节和数据节内的实际地址。其它镜像文件可以通过使用这个表的索引(序数)来调用函数。
导出名称指针表	一个由指向导出函数名称的指针组成的数组，按升序排列。大小写敏感。
导出序数表	一个由对应于导出名称指针表中各个成员的序数组成的数组。它们的对应是通过位置来体现的，因此导出名称指针表与导出序数表成员数目必须相同。
导出名称表	一系列以NULL结尾的ASCII码字符串。导出名称指针表中的成员都指向这个区域。它们都是公用名称，函数导入与导出就是通过它们。

当其它镜像文件通过名称导入函数时，Win32加载器通过导出名称指针表来搜索匹配的字符串。如果找到，它就查找导出序数表中相应的成员(也就是说，将找到的导出名称指针表的索引作为导出序数表的索引来使用)来获取与导入函数相关联的序数。获取的这个序数是导出地址表的索引，这个索引对应的元素给出了所需函数的实际位置。每个导出函数都可以通过序数进行访问。

当其它镜像文件通过序数导入函数时，就不再需要通过导出名称指针表来搜索匹配的字符串。因此直接使用序数效率会更高。但是导出名称容易记忆，它不需要用户记住各个符号在表中的索引。

5.1.1 导出目录表

导出目录表是导出函数信息的开始部分，它描述了导出函数信息中其余部分的内容。

偏移	大小	英文名	描述
0	4	Export Flags	保留，必须为0。
4	4	Time/Date StampMajor Version	导出函数被创建的日期和时间。这个值与NT头的第一部分TimeDateStamp相同。
8	2	Major Version	主版本号。
10	2	Minor Version	次版本号。
12	4	Name RVA	包含这个DLL全名的ASCII码字符串RVA。以一个NULL字节结尾。
16	4	Ordinal Base	导出函数的起始序数值。它通常被设置为1。
20	4	NumberOfFunctions	导出函数中所有元素的数目。
24	4	NumberOfNames	导出名称指针表中元素的数目。它同时也是导出序数表中元素的数目。
28	4	AddressOfFunctions	导出地址表RVA。
32	4	AddressOfNames	导出名称指针表RVA。
36	4	AddressOfNameOrdinals	导出序数表RVA。

5.1.2 导出地址表(Export Address Table, EAT)

导出地址表的格式为下表所述的两种格式之一。如果指定的地址不是位于导出节(其地址和长度由

NT头给出)中，那么这个域就是一个Export RVA;否则这个域是一个Forwarder RVA，它给出了一个位于其它DLL中的符号的名称。

偏移	大小	域	描述
0	4	Export RVA	当加载进内存时，导出函数RVA。
0	4	Forwarder RVA	这是指向导出节中一个以NULL结尾的ASCII码字符串的指针。这个字符串必须位于Export Table(导出表)数据目录项给出的范围之内。这个字符串给出了导出函数所在DLL的名称以及导出函数的名称(例如“MYDLL.expfunc”)，或者DLL的名称以及导出函数的序数值(例如“MYDLL.#27”)。

Forwarder RVA导出了其它镜像中定义的函数，使它看起来好像是当前镜像导出的一样。因此对于当前镜像来说，这个符号同时既是导入函数又是导出函数。

例如对于Windows XP系统中的Kernel32.dll文件来说，它导出的“HeapAlloc”被转发到“NTDLL.RtlAllocateHeap”。这样就允许应用程序使用Windows XP系统中的Ntdll.dll模块而不需要实际包含任何相关的导入信息。应用程序的导入表只与Kernel32.dll有关。

导出地址表的的值有时为0，此时表明这里没有导出函数。这是为了能与以前版本兼容，省去修改的麻烦。

5.1.3 导出名称指针表

导出名称指针表是由导出名称表中的字符串的地址(RVA)组成的数组。二进制进行排序的，以便于搜索。

只有当导出名称指针表中包含指向某个导出名称的指针时，这个导出名称才算被定义。换句话说，导出名称指针表的值有可能为0，这是为了能与前面版本兼容。

5.1.4 导出序数表

导出序数表是由导出地址表的索引组成的一个数组，每个序数长16位。必须从序数值中减去Ordinal Base域的值得到的才是导出地址表真正的索引。注意，导出地址表真正的索引真正的索引是从0开始

的。由此可见，微软弄出Ordinal Base是找麻烦的。导出序数表的值和导出地址表的索引的值都是无符号数。

导出名称指针表和导出名称序数表是两个并列的数组，将它们分开是为了使它们可以分别按照各自的边界(前者是4个字节，后者是2个字节)对齐。在进行操作时，由导出名称指针这一列给出导出函数的名称，而由导出序数这一列给出这个导出函数对应的序数。导出名称指针表的成员和导出序数表的成员通过同一个索引相关联。

5.1.5 导出名称表(Export Name Table，ENT)

导出名称表的结构就是长度可变的一系列以NULL结尾的ASCII码字符串。导出名称表包含的是导出名称指针表实际指向的字符串。这个表的RVA是由导出名称指针表的第1个值来确定的。这个表中的字符串都是函数名称，其它文件可以通过它们调用函。

5.1.6 举例

①用序数调用

当可执行文件用序数调用函数时，该序数就是导出函数地址表的真实索引。如果索引是错误的就有可能出现不可预知的错误。最著名的例子就是Windows XP在升级Server 2补丁之后，有很多程序都不能运行就是这个原因。微软用序数这种方法被大多数危险程序(病毒、木马)所引用，同样的微软自己也用这种方法来使用一些隐含的函数。最后受害者还是广大的用户，因为使用序数方法的绝大部分程序是有着不可告人的目的的。

②用函数名调用

当可执行文件用函数名调用时，加载器会通过AddressOfNames以2进制的方法找到第一个相同的函数名。假如找到的是第X个函数名，则在AddressOfNameOrdinals中取出第X个值，该值再减去Ordinal Base则为函数地址的真实索引。

5.2.idata节

首先，您得了解什么是导入函数。一个导入函数是被某模块调用的但又不在调用者模块中的函数，因而命名为“import(导入)”。导入函数实际位于一个或者更多的DLL里。调用者模块里只保留一些函数信息，包括函数名及其驻留的DLL名。现在，我们怎样才能找到PE文件中保存的信息呢? 转到 data

directory 寻求答案吧。

文件中导入信息的典型布局如下:

典型的导入节布局

5.2.1 导入目录表

导入目录表是由导入目录项组成的数组，每个导入目录项对应着一个导入的DLL。最后一个导入目录项是空的(全部域的值都为NULL)，用来指明目录表的结尾。

每个导入目录项的格式如下:

偏移	大小	域	描述
0	4	Import Lookup Table RVA	导入查找表的RVA。这个表包含了每一个导入函数的名称或序数。
4	4	Time/Date Stamp	当镜像与相应的DLL绑定之后，这个域被设置为这个DLL的日期/时间戳。
8	4	Forwarder Chain	第一个转发项的索引。
12	4	Name RVA	包含DLL名称的ASCII码字符串RVA。

16	4	Import Address RVA	导入地址表的RVA。这个表的内容与导入查找表的内容完全一样。

5.2.2 导入查找表

导入查找表是由长度为32位(PE32)或64位(PE32+)的数字组成的数组。其中的每一个元素都是位域，其格式如下表所示。在这种格式中，位31(PE32)或位63(PE32+)是最高位。这些项描述了从给定的DLL导入的所有函数。最后一个项被设置为0(NULL)，用来指明表的结尾。

偏移	大小	位域	描述
31/63	1	Ordinal/Name Flag	如果这个位为1，说明是通过序数导入的。否则是通过名称导入的。测试这个位的掩码为0x80000000(PE32)或0x8000000000000000(PE32+)。
15-0	16	Ordinal Number	序数值(16位长)。只有当Ordinal/Name Flag域为1(即通过序数导入)时才使用这个域。位30-15(PE32)或62-15(PE32+)必须为0。
30-0	31	Hint/Name Table RVA	提示/名称表项的RVA(31位长)。只有当Ordinal/Name Flag域为0(即通过名称导入)时才使用这个域。对于PE32+来说，位62-31必须为0。

5.2.3 提示/名称表

提示/名称表中的每一个元素结构如下：

偏移	大小	域	描述
0	2	Hint	指出名称指针表的索引。当搜索匹配字符串时首选使用这个值。如果匹配失败，再在DLL的导出名称指针表中进行2进制搜索。
2	可变	Name	包含导入函数名称的ASCII码字符串。这个字符串必须与DLL导出的函数名称匹配。同时这个字符串区分大小写并且以NULL结尾。
*	0或1	Pad	为了让提示/名称表的下一个元素出现在偶数地址，这里可能需要填充0个或1个NULL字节。

5.2.4 导入地址表

导入地址表的结构和内容与导入查找表完全一样，直到文件被绑定。在绑定过程中，用导入函数的

32位(PE32)或64位(PE32+)地址覆盖导入地址表中的相应项。这些地址是导入函数的实际内存地址，尽管技术上仍把它们称为“虚拟地址”。加载器通常会处理绑定。

5.3 .pdata节（可有可无，谁也不希望自己的函数出问题的吧！）

.pdata节是由用于异常处理的函数表项组成的数组。NT头中的Exception Table（异常表）域指向它。在将它们放进最终的镜像文件之前，这些项必须按函数地址（下列每个结构的第一个域）排序。下面描述了函数表项的3种格式，使用哪一种取决于目标平台。

对于32位的MIPS镜像来说，其函数表项格式如下：

偏移	大小	域	描述
0	4	Begin Address	相应函数的VA
4	4	End Address	函数结尾的VA
8	4	Exception Handler	指向要执行的异常处理程序的指针
12	4	Handler Data	指向要传递给异常处理程序的附加数据的指针
16	4	Prolog End Address	函数prolog代码结尾的VA

对于ARM、PowerPC、SH3和SH4 Windows CE平台来说，其函数表项格式如下：

偏移	大小	域	描述
0	4	Begin Address	相应函数的VA
4	8位	Prolog Length	函数prolog代码包含的指令数
4	22位	Function Length	函数代码包含的指令数
4	1位	32-bit Flag	如果此位为1，表明函数由32位指令组成。否则，函数由16位指令组成。
4	1位	Exception Flag	如果此位为1，表明存在用于此函数的异常处理程序；否则，不存在异常处理程序。

对于x64和Itanium平台来说，其函数表项格式如下：

偏移	大小	域	描述
0	4	Begin Address	相应函数的RVA

4	4	End Address	函数结尾的RVA
8	4	Unwind Information	用于异常处理的展开(Unwind)信息的RVA

5.4 .reloc节

基址重定位表包含了镜像中所有需要重定位的内容。NT头中的数据目录中的Base Relocation Table(基址重定位表)域给出了基址重定位表所占的字节数。基址重定位表被划分成许多块，每一块表示一个4K页面范围内的基址重定位信息，它必须从32位边界开始。

5.4.1 基址重定位块

每个基址重定位块的开头都是如下结构：

偏移	大小	域	描述
0	4	Page RVA	将镜像基址与这个域(页面RVA)的和加到每个偏移地址处最终形成一个VA，这个VA就是要进行基址重定位的地方。
4	4	Block Size	基址重定位块所占的总字节数，其中包括Page RVA域和Block Size域以及跟在它们后面的Type/Offset域。

Block Size域后面跟着数目不定的Type/Offset位域。它们中的每一个都是一个WORD(2字节)，其结构如下：

偏移	大小	域	描述
0	4位	Type	它占这个WORD的最高4位，这个值指出需要应用的基址重定位类型。参考5.4.2节“基址重定位类型”。
0	12位	Offset	它占这个WORD的其余12位，这个值是从基址重定位块的Page RVA域指定的地址处开始的偏移。这个偏移指出需要进行基址重定位的位置。

为了进行基址重定位，需要计算镜像的首选基地址与实际被加载到的基地址之差。如果镜像本身就被加载到了其首选基地址，那么这个差为零，因此也就不需要进行基址重定位了。

5.4.2 基址重定位类型

值	描述
0	基址重定位被忽略。这种类型可以用来对其它块进行填充。
1	基址重定位时将差值的高16位加到指定偏移处的一个16位域上。这个16位域是一个32位字的高半部分。
2	基址重定位时将差值的低16位加到指定偏移处的一个16位域上。这个16位域是一个32位字的低半部分。
3	基址重定位时将所有的32位差值加到指定偏移处的一个32位域上。
4	进行基址重定位时将差值的高16位加到指定偏移处的一个16位域上。这个16位域是一个32位字的高半部分，而这个个Type/Offset位域后面的一个16位字中。也就是说，这一个基址重定位项占了两个Type/Offset位域的位置。
5	对MIPS平台的跳转指令进行基址重定位。
6	保留，必须为0
7	保留，必须为0
9	对MIPS16平台的跳转指令进行基址重定位。
10	进行基址重定位时将差值加到指定偏移处的一。

5.5 加载配置结构(不清楚，大概又是多余的吧)

加载配置结构最初用于Windows NT操作系统自身几种非常有限的场合——在镜像文件头或NT头中描述各种特性太困难或这些信息尺寸太大。当前版本的Microsoft链接器和Windows XP以及后续版本的Windows使用的是这个结构的新版本，将之用于包含保留的SEH技术的基于x86的32位系统上。它提供了一个安全的结构化异常处理程序列表，操作系统在进行异常派送时要用到这些异常处理程序。如果异常处理程序的地址在镜像的VA范围之内，并且镜像被标记为支持保留的SEH，那么这个异常处理程序必须在镜像的已知安全异常处理程序列表中，否则操作系统将终止这个应用程序。这是为了防止利用“x86异常处理程序劫持”来控制操作系统，它在以前已经被利用过。

Microsoft的链接器自动提供一个默认的加载配置结构来包含保留的SEH数据。如果用户的代码已经提供了一个加载配置结构，那么它必须包含新添加的保留的SEH域。否则，链接器将不能包含保留的

SEH数据，这样镜像文件就不能被标记为包含保留的SEH。

5.5.1 加载配置目录

对应于预保留的SEH加载配置结构的数据目录项必须为加载配置结构指定一个特别的大小，因为操作系统加载器总是希望它为这样一个特定值。事实上，这个大小只是用于检查这个结构的版本。为了与Windows XP以及以前版本的Windows兼容，x86镜像文件中这个结构的大小必须为64。

5.5.2 加载配置结构布局

用于32位和64位PE文件的加载配置结构布局如下：

偏移	大小	域	描述
0	4	Characteristics	指示文件属性的标志，当前未用。
4	4	TimeDateStamp	日期/时间戳。这个值表示从UTC时间1970年1月1日午夜(00:00:00)以来经过的总秒数，它是根据系统时钟算出的。可以用C运行时函数time来获取这个时间戳。
8	2	MajorVersion	主版本号
10	2	MinorVersion	次版本号
12	4	GlobalFlagsClear	当加载器启动进程时，需要被清除的全局加载器标志。
16	4	GlobalFlagsSet	当加载器启动进程时，需要被设置的全局加载器标志。
20	4	CriticalSectionDefaultTimeout	用于这个进程处于无约束状态的临界区的默认超时值。
24	8	DeCommitFreeBlockThreshold	返回到系统之前必须释放的内存数量(以字节计)。
32	8	DeCommitTotalFreeThreshold	空闲内存总量(以字节计)。
40	8	LockPrefixTable	[仅适用于x86平台]这是一个地址列表的VA。这个地址列表中保存的是使用LOCK前缀的指令的地址，这样便于在单处理器机器上将这些LOCK前缀替换为NOP指令。
48	8	MaximumAllocationSize	最大的分配粒度(以字节计)。

56	8	VirtualMemoryThreshold	最大的虚拟内存大小(以字节计)。
64	8	ProcessAffinityMask	将这个域设置为非零值等效于在进程启动时将这个设定的值作为参数去调用SetProcessAffinityMask函数(仅适用于.exe文件)。
72	4	ProcessHeapFlags	进程堆的标志,相当于函数的第一个参数。这些标志用于在进程启动过程中创建的堆。
76	2	CSDVersion	Service Pack版本标识。
78	2	Reserved	必须为0
80	8	EditList	保留,供系统使用。
60/88	4/8	SecurityCookie	指向cookie的指针。cookie由Visual C++编译器的GS实现所使用。
64/96	4/8	SEHandlerTable	[仅适用于x86平台]这是一个地址列表的VA。这个地址列表中保存的是镜像中每个合法的、独一无二的SE处理程序的RVA,并且它们已经按RVA排序。
68/104	4/8	SEHandlerCount	[仅适用于x86平台]表中独一无二的SE处理程序的数目。

5.6 .rsrc节

资源节可以看成是一个磁盘的分区,盘符是资源目录表,下面有3层目录(资源目录项),最后是文件(资源数据)。

①资源目录表是一个16字节组成的结构。其第一个字节又称为“根节点”。其前的12字节虽然有定义,但加载器并不理会,所以任何值都可以。

②第1层目录(资源目录项)是资源类型,微软已经定义了21种。其结构是一个16字节的数组。资源目录项分为名称项和ID项,这取决于资源目录表。资源目录表指出跟着它的名称项和ID项各有多少个(表中所有的名称项在所有的ID项前面)。表中的所有项按升序排列:名称项是按不区分大小写的字符串,而ID项则是按数值。第0-3字节表示资源类型的名称字符串的地址或是32位整数,第4-7字节表示第二层目录(资源目录项)相对于根节点的偏移。

一系列资源目录表按如下方式与各层相联系:每个目录表后面跟着一系列目录项,它们给出那个层

(类型、名称或语言)的名称或标识(ID)及其数据描述或另一个目录表的地址。如果这个地址指向一个数据描述，那么那个数据就是这棵树的叶子。如果这个地址指向另一个目录表，那么那个目录表列出了下一层的目录项。

一个叶子的类型、名称和语言ID由从目录表到这个叶子的路径决定。第1个表决定类型ID，第2个表(由第一个表中的目录项指向)决定名称ID，第3个表决定语言ID。

.rsrc节的一般结构如下：

数据	描述
资源目录表	所有的顶层(类型)结点都被列于第1个表中。这个表中的项指向第2层表。每个第2层树的类型ID相同但是名称ID不同。第3层树的类型ID和名称ID都相同但语言ID不同。每个单个的表后面紧跟着目录项，每一项都有一个名称或数字标识和一个指向数据描述或下一层表的指针。
资源目录项	
资源目录字符串	按2字节边界对齐的Unicode字符串，它是作为由资源目录项指向的字符串数据来使用的。

资源数据描述	一个由记录组成的数组，由表指向它，描述了资源数据的实际大小和位置。这些记录是资源描述树中的叶子。
资源数据	资源节的原始数据。资源据描述域中的大小和位置信息将资源数据分成单个的区域。

资源目录表

偏移	大小	域	描述
0	4	Characteristics	资源标志。保留供将来使用。当前它被设置为0。
4	4	Time/Date Stamp	资源数据被资源编译器创建的时间。
8	2	Major Version	主版本号，由用户设定。
10	2	Minor Version	次版本号，由用户设定。
12	2	Number of Name Entries	紧跟着这个表头的目录项的个数，这些目录项使用名称字符串来标识类型、名称或语言项。
14	2	Number of ID Entries	紧跟着这个表头的目录项的个数，这些目录项使用数字来标识类型、名称或语言项。

资源目录项

具体的情况是资源目录表后面紧跟着以名称项和ID项所组成的数组。资源目录表与资源目录项之间不能有空隙。名称项组成的数组在ID项组成的数组前面，且两个数组不能有空隙。

偏移	大小	域	描述
0	4	Name RVA	表示类型、名称或语言ID项的名称字符串的地址。
0	4	Integer ID	表示类型、名称或语言ID项的32位整数。
4	4	Data Entry RVA	最高位为0。低31位是资源数据项的地址。
4	4	Subdirectory RVA	最高位为1。低31位是另一个资源目录表(下一层)的地址。

资源目录字符串

资源目录字符串区由按字边界对齐的Unicode字符串组成。这些字符串被存储在最后一个资源目录项之后、第一个资源数据项之前。这样能够使这些长度可变的字符串对长度固定的目录项的对齐情况影响最小。每个资源目录字符串格式如下:

偏移	大小	域	描述
0	2	Length	字符串的长度, 不包括Length域本身。
2	可变	Unicode String	可变 Unicode String 按字边界对齐的可变长度的Unicode字符串。

资源数据项

每个资源数据项描述了资源数据区中一个实际单元的原始数据。资源数据项格式如下:

偏移	大小	域	描述
0	4	Data RVA	资源数据区中一个单元的资源数据的地址。
4	4	Size	由Data RVA域指向的资源数据的大小(以字节计)。
8	4	Codepage	用于解码资源数据中的代码点值的代码页。通常这个代码页应该是Unicode代码页。
12	4	保留, 必须为0	保留, 必须为0

6 属性证书表

可以给镜像文件添加属性证书表使它与属性证书相关联。有多种不同类型的属性证书, 最常用的是Authenticode签名。

属性证书表包含一个或多个长度固定的表项, 可以通过NT头中的数据目录中的Certificate Table(证

书表)域找到它们。这个表的每个表项给出了相应证书的开始位置和长度。存储在这个节中的每个证书都有一个相应的证书表项。证书表项的数目可以通过将证书表的大小除以证书表中每一项的大小(8)得到。注意证书表的大小仅包括它的表项，并不包括这些表项实际指向的证书。

每个表项格式如下：

偏移	大小	域	描述
0	4	Certificate Data	指向证书实际数据的文件指针。它指向的地址总是按8字节倍数边界(即最低3个位都是0)对齐。
0	4	Size of Certificate	这是一个无符号整数，它指出证书的大小(以字节计)。

注意证书总是从8进制字(从任意字节边界开始的16个连续字节)边界开始。如果一个证书的长度不是8进制字长度的偶数倍，那么就一直用0填充到下一个八进制字边界。但是证书长度并不包括这些填充的0。因此任何处理证书的软件必须向上舍入到下一个8进制字才能找到另一个证书。

证书的起始位置和长度由证书表中相应的表项给出。每个证书都有惟一一个与它对应的表项。

来源：看雪论坛http://bbs.pediy.com/showthread.php?t=145912

上一篇 IOS 一个ping的例子 simplePing

下一篇 脱壳 OEP 程序的入口点

我的同类文章

C/C++(14)

- windows内核安全编程——... 2012-12-20 阅读 1029
- DLL 2012-05-21 阅读 438
- 浮点数的存储方式 float 2011-11-13 阅读 743
- C/C++ 疑难点(不断更新中... 2011-10-23 阅读 363
- 9X、2000、XP、2003注册表... 2011-05-09 阅读 386
- 模块基地址重定位和绑定 2012-05-22 阅读 604
- 【大数据处理】bitmap思想... 2012-04-17 阅读 751
- 回调的写法 2011-10-23 阅读 380
- 解析文件，要验证文件完整性 2011-07-04 阅读 297
- Google发布C++编程规范 2011-03-08 阅读 483

更多文章

猜你在找

windows server 2008 r2服务器操作系统
Windows Server 2012 R2 远程桌面管理
Windows Server 2012 Hyper-v 管理
Windows Server 2012 组策略管理
Windows系统内核-保护模式
PE文件详解之IMAGE_NT_HEADER结构
PE文件结构详解六重定位
解密系列系统篇_PE结构详解笔记1
PE结构详解
PE结构详解

查看评论

3楼 funte 2014-04-27 20:25发表

请问 AddressOfNames 数组中RVA指向的是ansi还是unicode,好像两者都见过耶？怎么区别呢

2楼 aizimoon 2013-07-29 00:52发表

5.4.2 基址重定位类型 的offset 8去哪了= =

1楼 aizimoon 2013-07-28 20:23发表

Win32VersionValue写错了，应该占用4个字节吧？下面的SizeOfImage起始是56

核心技术类目

全部主题 Hadoop AWS 移动游戏 Java Android iOS Swift 智能硬件 Docker OpenStack
VPN Spark ERP IE10 Eclipse CRM JavaScript 数据库 Ubuntu NFC WAP jQuery
BI HTML5 Spring Apache .NET API HTML SDK IIS Fedora XML LBS Unity
Splashtop UML components Windows Mobile Rails QEMU KDE Cassandra CloudStack
FTC coremail OPhone CouchBase 云计算 iOS6 Rackspace Web App SpringSide
Maemo Compuware 大数据 aptech Perl Tornado Ruby Hibernate ThinkPHP HBase
Pure Solr Angular Cloud Foundry Redis Scala Django Bootstrap

公司简介 | 招贤纳士 | 广告服务 | 银行汇款帐号 | 联系方式 | 版权声明 | 法律顾问 | 问题报告 | 合作伙伴 | 论坛反馈

网站客服 杂志客服 微博客服 webmaster@csdn.net 400-600-2320 | 北京创新乐知信息技术有限公司 版权所有 | 江苏乐知网络技术有限公司 提供商务支持