PE101 Windows可执行文件详解

Ange Albertini
corkami.com

CC BY

PE文件剖析

SHA-1 b7af4cb51ce38e43e030656eb2698fab408cf9cb
下载 @ pe101.corkami.com

simple

simple.exe

头部
可执行文件的技术细节

段表
定义文件如何加载到内存中

段
可执行文件的内容

代码段
被执行体

导入表
可执行文件和Windows库之间的连接

数据段
代码使用的信息

可选头部
可执行文件信息

数据目录
指向附加结构（导出表，导入表，...）的指针

DOS头部
显示文件为二进制可执行文件

PE头部
显示文件为新的二进制可执行文件

16进制转储	ASCII字符转储	域	值	解释
4D 5A 00 00-00 00 00 00-00 00 00 00-00 00 00 00	MZ.....			
偏移:0x30 00 00 00 00-00 00 00 00-00 00 00 00-40 00 00 00	.....@...	e_magic e_lfanew	'MZ' 0x40	固定签名 PE头部偏移 ❶
偏移:0x40 50 45 00 00-4C 01 03 00-00 00 00 00-00 00 00 00	PE..L.....	Signature Machine NumberOfSections	'PE', 0, 0 0x14c [intel 386] 3	固定签名 处理器：ARM/MIPS/Intel 段数目 ❷
00 00 00 00-E0 00 02 01...	....a...	SizeOfOptionalHeader Characteristics	0xe0 0x102 [32b EXE]	段表的相对偏移 ❷ EXE/DLL/...
偏移:0x58 ...0B 01 00 00-00 00 00 00	.....	Magic	0x10b [32b]	32位/64位
00 00 00 00-00 00 00 00-00 10 00 00-00 00 00 00	.....	AddressOfEntryPoint	0x1000	程序开始执行的入口地址 ❺
00 00 00 00-00 00 40 00-00 10 00 00-00 02 00 00	.....@.....	ImageBase	0x400000	文件映射到内存中的基地址
00 00 00 00-00 00 00 00-04 00 00 00-00 00 00 00	.....	SectionAlignment	0x1000	内存中段对齐地址 ❷
00 40 00 00-00 02 00 00-00 00 00 00-02 00 00 00	.@.....	FileAlignment	0x200	文件中段对齐地址 ❷
00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00	.....	MajorSubsystemVersion	4 [NT 4 或后续版本]	所需Windows系统的版本
00 00 00 00-10 00 00 00...	.....	SizeOfImage SizeOfHeaders Subsystem NumberOfRvaAndSizes	0x4000 0x200 2 [GUI] 16	所需内存总大小 头部的总大小 ❸ 驱动/GUI/命令行... 数据目录个数 ❹
...00 00 00 00-00 00 00 00	.....			
00 20 00 00-00 00 00 00-00 00 00 00-00 00 00 00	.....	ImportsVA	0x2000	导入表的相对虚拟地址* ❹
00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00	.....			

段表

Name	VirtualSize (RVA*)	VirtualAddress (RVA*)	SizeOfRawData (实际大小)	PointerToRawData (实际偏移)	Characteristics
.text	0x1000	0x1000	0x200	0x200	CODE EXECUTE READ
.rdata	0x1000	0x2000	0x200	0x400	INITIALIZED READ
.data	0x1000	0x3000	0x200	0x600	DATA READ WRITE

对于每一个段，在文件偏移PointerToRawData处读取大小为SizeofRawData的块。
这块数据会被加载到地址为ImageBase + VirtualAddress，大小为VirtualSize的内存块中，并标记特定的Characteristics.

偏移:0x138		
2E 74 65 78-74 00 00 00	.text...	
00 10 00 00-00 10 00 00-00 02 00 00-00 02 00 00	.....	
00 00 00 00-00 00 00 00-00 00 00 00-20 00 00 60	...`...	
2E 72 64 61-74 61 00 00-00 10 00 00-00 20 00 00	.rdata.....	
00 02 00 00-00 04 00 00-00 00 00 00-00 00 00 00	.....	
00 00 00 00-40 00 00 40-2E 64 61 74-61 00 00 00	....@..@.data...	
00 10 00 00-00 30 00 00-00 02 00 00-00 06 00 00	.....0.....	
00 00 00 00-00 00 00 00-00 00 00 00-40 00 00 C0	.....@..+	

偏移:0x200/RVA:0x401000		x86汇编	等价的C语言代码
6A 00 68 00-30 40 00 68-17 30 40 00-6A 00 FF 15	j.h.0@.h.0@.j. .	push 0	
70 20 40 00-6A 00 FF 15-68 20 40 00	p.@.j. .h.@.	push 0x403000	
		push 0x403017	
		push 0	
		call [0x402070]	MessageBox(0, "Hello World!","a simple PE executable", 0);
		push 0	
		call [0x402068]	ExitProcess(0);

偏移:0x400/RVA:0x402000		导入表结构	结果
3C 20 00 00-00 00 00 00-00 00 00 00-78 20 00 00	<.....x...	描述符	
68 20 00 00-44 20 00 00-00 00 00 00-00 00 00 00	h...D.....	0x203c → 0x204c, 0 (INT*)	
85 20 00 00-70 20 00 00-00 00 00 00-00 00 00 00	à...p.....	0x2078 → kernel32.dll → 0,ExitProcess (HINT,函数名)	加载完成后， 0x402068内存处将指向kernel32.dll中ExitProcess函数 0x402070内存处将指向user32.dll中MessageBoxA函数
00 00 00 00-00 00 00 00-00 00 00 00-4C 20 00 00	.....L...	0x2068 → 0x204c, 0 (IAT*)	
00 00 00 00-5A 20 00 00-00 00 00 00-00 00 45 78	....Z.....Ex	0x2044 → 0x205a, 0 (INT*)	
69 74 50 72-6F 63 65 73-73 00 00 00-4D 65 73 73	itProcess...Mess	0x2085 → user32.dll → 0,MessageBoxA (HINT,函数名)	
61 67 65 42-6F 78 41 00-4C 20 00 00-00 00 00 00	ageBoxA.L.....	0x2070 → 0x205a, 0 (IAT*)	
5A 20 00 00-00 00 00 00-6B 65 72 6E-65 6C 33 32	Z.....kernel32	0 0 0 0 0	
2E 64 6C 6C-00 75 73 65-72 33 32 2E-64 6C 6C 00	.dll.user32.dll.		

*所有的地址都是相对虚拟地址。

偏移:0x600/RVA:0x403000		字符串
61 20 73 69-6D 70 6C 65-20 50 45 20-65 78 65 63	a.simple.PE.exec	a simple PE executable\0
75 74 61 62-6C 65 00 48-65 6C 6C 6F-20 77 6F 72	utable.Hello.wor	Hello world!\0
6C 64 21 00	ld!.	

这是整个文件的结构，但是大部分PE文件包含更多的元素。为了简洁，解释写得比较简单。

版本1，2013年6月28日

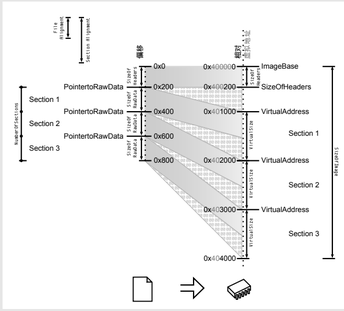
译者 童进

加载过程

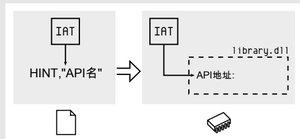
❶ 头部
解析DOS头部
解析PE头部
（它的偏移值是DOS头部中的e_lfanew）
解析可选头部
（它在PE头部后面）

❷ 段表
解析段表
（它的位置：可选头部的偏移 + SizeOfOptionalHeader）
它包含NumberOfSections个元素
利用对齐进行合法性检查：
FileAlignments 和 SectionAlignments

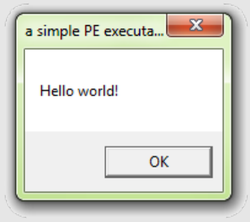
❸ 映射
文件被映射到内存中，依据：
ImageBase
SizeOfHeaders
Sections table



❹ 导入表
解析数据目录
它们在可选头部后面
它们的数目是NumOfRVAAndSizes
导入表总是第二个
解析导入表
每个描述符会指定一个DLL名
相应的DLL被加载进内存
IAT和INT被同步解析
对INT中的每一个API
它的地址被写入IAT条目中



❺ 执行
代码在EntryPoint处被调用
通过IAT找到API以实现代码中的调用



注释

MZ头部也称为DOS头部
以"MZ"开始（MS-DOS开发者Mark Zbikowski的缩写）
PE头部也称为IMAGE_FILE_HEADERS或COFF文件头
以"PE"开始 (Portable Executable)
可选头部也称为IMAGE_OPTIONAL_HEADER
对于非标准PE文件是可选的，但对于可执行文件是必须的

RVA 相对虚拟地址
相对ImageBase的地址（在ImageBase处，RVA = 0）
头部中差不多所有地址都是相对虚拟地址
在代码段中，地址不是相对的。

INT 导入名称表
以Null结束的指针链表，指向包含HINT和函数名的结构体
IAT 导入地址表
以Null结束的指针链表
在文件中，它是INT的拷贝
加载完成后，它指向导入的API
HINT
导入函数在DLL导出表中的索引
不是必须的，但通过减少查找提供了加速