

## 第0x6章：二元分类

### 📝 笔记:

这本书正在进行中。

我们鼓励您直接在这些页面上发表评论 ...建议编辑、更正和/或其他内容！

要发表评论，只需突出显示任何内容，然后单击  
就在文档的边界上）。

✚ 出现在屏幕上的图标

由我们的 [Friends of Objective-See](#):



[Airo](#)



[SmugMug](#)



[守护防火墙](#)



[SecureMac](#)



[iVerify](#)



[光环隐私](#)

苹果公司指出，Mach-O（Mach对象文件格式的缩写）“是OS X中二进制文件的本机可执行格式，是传送代码的首选格式。” [1]

由于大多数Mac恶意软件都是以Mach-O二进制文件的形式编译和分发的，因此对这种文件格式有深入的了解非常重要。

```
$ file Final_演示文稿.app/Contents/MacOS/usrnode 最后的演讲.app/Contents/MacOS/usrnode:  
Mach-O 64位可执行文件x86_64
```

64位Mach-O可执行文件（  
OSX.WindTail）

不幸的是，由于Mach-O是一种二进制文件格式，分析和理解此类文件需要特定的分析工具。通常以拆卸器达到顶点的工具。

#### 📝 笔记:

有关Mach-O二进制文件的权威指南，请参阅苹果文档：

[“OS X ABI Mach-O文件格式参考” \[1\]](#)

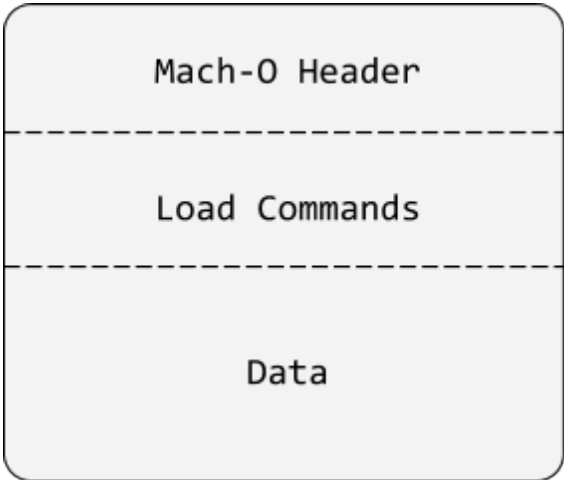
可执行二进制文件格式相当复杂，Mach-O文件格式也不例外。好消息是，出于恶意软件分析的目的，只需要对Mach-O文件格式和几个相关概念有一个初步的了解。

#### 📝 笔记:

对于感兴趣的读者来说，可以在这里找到一个关于Mach-O文件格式的深入的、坦率地说相当优秀的书写：

“解析Mach-O文件” [2]

基本上，Mach-O文件由三个连续的部分或区域组成：头、加载命令和数据。



马赫-O割台

Mach-O文件以Mach-O头开始：

“每个Mach-O文件的开头都有一个头结构，将文件标识为Mach-O文件。头还包含其他基本文件类型信息，指示目标体系结构，并包含指定影响文件其余部分解释的选项的标志。” [1]

mach-o header是mach-o/loader中定义的mach\_header\_64（或32位mach\_header）类型的结构。h：

01 结构mach\_头\_64 {

02     uint32\_t cpu\_类    魔术cpu\_type     /\* 马赫幻数标识符\*/ cpu说明符\*/

03     \_t cpu\_子类型     ;                 /\* 机器说明符\*/文件类型

04     \_t uint32\_t        cpu\_subtype     /\* \*/

05     uint32\_t           ; 文件类型;     /\* 加载命令的数量\*/

06                        ncmds;           /\*

```
07      uint32_t      sizeofcmds    /*所有加载命令的大小*/
08      uint32_t      ; 旗帜；含蓄    /*旗帜*/
09      uint32_t      ;              /*含蓄的*/
};
```

*mach\_收割台\_64结构 (mach-  
o/loader.h)*

苹果在加载器中的评论。h文件应提供每个成员（在mach\_header\_64结构中）的充分（尽管简洁）描述。

需要特别注意的是filetype成员，它描述了文件的类型。几个可能的值包括（来自马赫数-o/loader.h）：

- MH\_EXECUTE (0x2)  
标准Mach-O可执行文件
- MH\_DYLIB (0x6)  
Mach-O动态链接库（即dylib）
- MH\_BUNDLE (0x8)  
马赫-O束（即束）

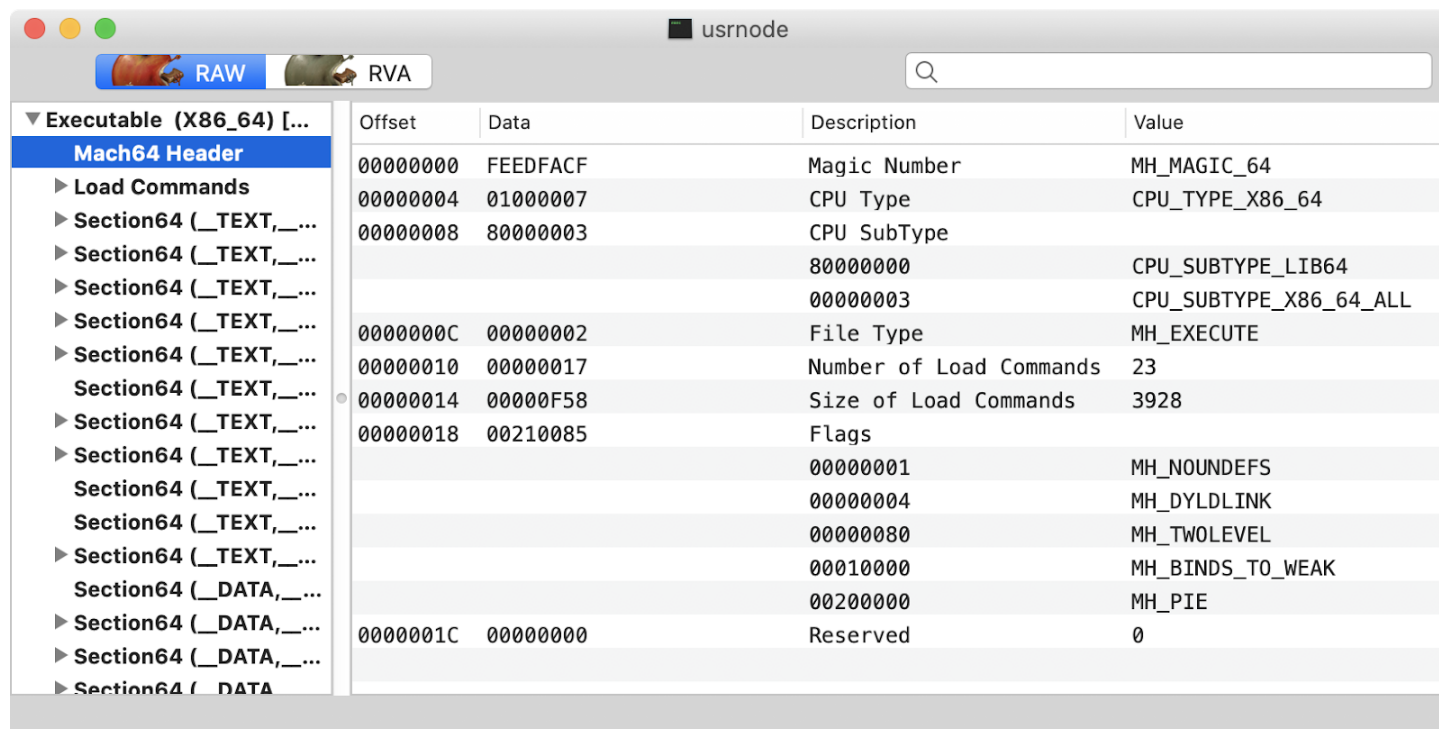
要转储或解析Mach-O文件的内容，可以使用/usr/bin/otool实用程序。例如，要转储Mach-O标头，请使用-hv标志执行otool：

```
$ otool -hv Final_Presentation.app/Contents/MacOS/usrnode
```

马赫收割台 魔术	cputype	cpusubtype	文件类型	ncmds	sizeofcmds
MH_MAGIC_64	X86_64	ALL	执行	23	3928

*卸下OSX。WindTail的Mach-O收割台（通过  
otool）*

或者，如果你喜欢UI，MachOView [3]是一个可爱的工具！



The screenshot shows a tool window titled 'usrnode' with tabs for 'RAW' and 'RVA'. The left sidebar lists 'Executable (X86\_64) [...]' and 'Mach64 Header'. The main table displays the header fields:

Offset	Data	Description	Value
00000000	FEEDFACF	Magic Number	MH_MAGIC_64
00000004	01000007	CPU Type	CPU_TYPE_X86_64
00000008	80000003	CPU SubType	80000000
			CPU_SUBTYPE_LIB64
			00000003
			CPU_SUBTYPE_X86_64_ALL
0000000C	00000002	File Type	MH_EXECUTE
00000010	00000017	Number of Load Commands	23
00000014	00000F58	Size of Load Commands	3928
00000018	00210085	Flags	
			00000001
			MH_NOUNDEFS
			00000004
			MH_DYLDLINK
			00000080
			MH_TWOLEVEL
			00010000
			MH_BINDS_TO_WEAK
			00200000
			MH_PIE
0000001C	00000000	Reserved	0

卸下Mach-O收割台（通过  
MachOView）

## 笔记:

苹果指出，“Mach-O文件包含一个架构的代码和数据。” [1]

为了创建可以在具有不同体系结构（即32位、64位等）的系统上执行的单个二进制文件，可以将多个Mach-O二进制文件包装在通用（或“fat”）二进制文件中。

这样的二进制文件从一个头（类型：fat\_头）开始，然后是特定于体系结构的头 Mach-O二进制文件连接在一起。

你可以通过：`otool -fv`转储fat\_头

## 马赫-O载荷指令

Mach-O头之后是二进制文件的加载命令，它指示（“命令”）动态加载程序（dyld）如何在内存中加载（和布局）二进制文件。

“标题后面是一系列可变大小的加载命令，用于指定文件的布局和链接特征。除其他信息外，加载命令还可以指定：

- 文件在虚拟内存中的初始布局
- 符号表的位置（用于动态链接）
- 程序主线程的初始执行状态
- 包含主可执行文件导入符号定义的共享库的名称“[1]

Mach-O二进制文件的加载命令可以使用-l标志通过otool查看：

```
$ otool -l Final_Presentation.app/Contents/MacOS/usrnode
```

```
...
```

加载命令0

```
cmd LC_段64
```

```
cmdsize 72
```

```
segname__PAGEZERO
```

```
vmaddr 0x0000000000000000
```

```
vmsize 0x0000000100000000
```

```
关闭0
```

```
文件大小0
```

```
maxprot 0x00000000
```

```
初始保护0x00000000
```

```
nsects 0标
```

```
志0x0
```

加载命令1

```
cmd LC_段64
```

```
cmdsize 952
```

```
segname文本
```

```
vmaddr 0x0000000100000000
```

```
vmsize 0x00000000000013000
```

```
关闭0
```

```
文件大小77824
```

```
maxprot 0x00000007
```

```
initprot 0x00000005
```

```
nsects 11
```

```
个标志0x0
```

```
...
```

卸下OSX。WindTail的加载命令（通过otool）

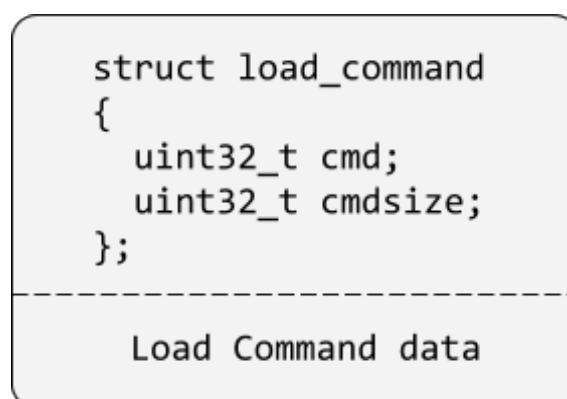
为了进行恶意软件分析，我们的目标是对Mach-O文件格式有一个基本的了解，所以我们不会涵盖所有受支持的加载命令。然而，有几个是非常相关的。

加载命令都以mach-o/loader中定义的load\_命令结构开始。h:

```
01 结构加载命令{
02      uint32_t cmd;          /*加载命令的类型*/
03      uint32_t cmd尺寸      /*命令的总大小（字节）*/
04      ;
};
```

*Load\_命令结构（马赫数  
/Loader.h）*

这里，加载命令。cmd描述加载命令的类型，而加载命令的大小在load\_命令中指定。尺寸。请注意，load命令的数据紧跟在load\_命令结构之后，并且这些数据特定于load命令的类型：



加载命令的一种常见类型是LC\_段/LC\_段\_64，它描述一个段。苹果用以下方式定义了一个细分市场：

“一个段定义了Mach-O文件中的一系列字节，以及动态链接器加载应用程序时这些字节映射到虚拟内存的地址和内存保护属性。” [1]

如下图所示，LC\_SEGMENT/LC\_SEGMENT\_64 load命令包含动态加载程序（dyld）将段映射到内存（并设置其内存权限）的所有相关信息：

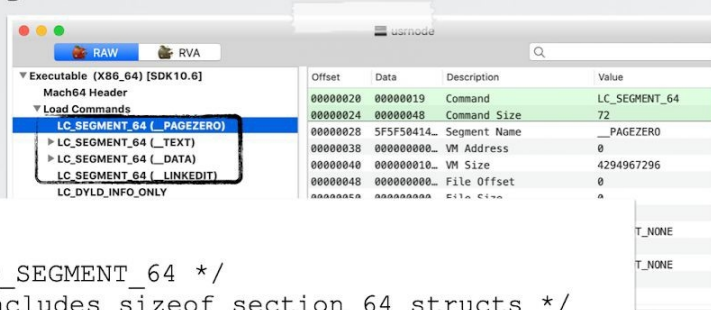
## File Type

Mach-O binary: load\_cmds (segments)

```
01 struct segment_command_64 {
02     uint32_t cmd;           /* LC_SEGMENT_64 */
03     uint32_t cmdsize;       /* includes sizeof section_64 structs */
04     char segname[16];       /* segment name */
05     uint64_t vmaddr;        /* memory address of this segment */
06     uint64_t vmsize;        /* memory size of this segment */
07     uint64_t fileoff;       /* file offset of this segment */
08     uint64_t filesize;      /* amount to map from the file */
09     vm_prot_t maxprot;      /* maximum VM protection */
10     vm_prot_t initprot;     /* initial VM protection */
11     uint32_t nsects;        /* number of sections in segment */
12     uint32_t flags;         /* flags */
13 };
```

struct 'segment\_command\_64'

LC\_段/LC\_段64加载命令



Offset	Data	Description	Value
00000020	00000019	Command	LC_SEGMENT_64
00000024	00000048	Command Size	72
00000028	5F5F50414...	Segment Name	__PAGEZERO
00000038	000000000...	VM Address	0
00000040	000000010...	VM Size	4294967296
00000048	000000000...	File Offset	0
00000050	000000000...	File Size	0

在分析Mach-O二进制文件时，可能会遇到以下几个部分：

- **\_\_文本段**  
包含只读的可执行代码和数据
- **\_\_数据段**  
包含可写的数据
- **\_\_LINKEDIT段**  
包含链接器（dyld）的信息，如“符号、字符串和重定位表条目” [1]

如果二进制文件是用objective-C编写的，它可能有一个 **\_\_包含** Objective-C运行时使用的信息。虽然这些信息也可能被找到在 **\_\_数据段**，在不同的 **\_\_objc\_\***部分。

### 📝 笔记:

段可以包含多个段（每个段包含相同类型的代码或数据）。更多关于以下章节的信息...



一旦二进制文件被加载到内存中（由动态链接器/加载程序dyld），执行就从二进制文件的入口点开始。dyld如何定位上述入口点？通过LC\_主加载命令！

该加载命令（累积）是entry\_point\_command类型的结构：

```
01 结构入口点命令{
02      uint32_t  cmd;          /* LC_MAIN仅用于MH_执行文件类型*/
03      uint32_t  规模;        /* 24 */
04      uint64_t  entryoff; /*main（）的文件（文本）偏移量*/
05      uint64_t  堆栈大小; /*如果不是零，则为初始堆栈大小*/
06  };
```

*LC\_MAIN的入口点命令结构 (mach-o/loader.h)*

LC\_MAIN load命令最重要的成员是entryoff,它包含二进制文件入口点的偏移量。加载时，dyld只需将该值添加到

（在内存中）二进制代码的基，然后跳到此指令开始执行二进制代码。

“LC\_MAIN给出入口点（MAIN（））的地址，[loader]dyld直接跳到该地址...” [4]

#### 笔记:

LC\_MAIN load命令取代了不推荐使用的LC\_UNIXTHREAD load命令。

如果您在分析较旧的Mach-O二进制文件，可能仍然会遇到LC\_UNIXTHREAD,它包含初始线程的整个上下文（读取：寄存器值）。在此上下文中，EIP/RIP寄存器包含二进制文件初始入口点的地址。

#### 笔记:

Mach-O二进制文件可以包含一个或多个构造函数，这些构造函数将在LC\_MAIN中指定的地址之前执行。

任何构造函数的偏移量都保存在  
\_\_数据常量段。

稍后将详细介绍此主题，但在分析Mac恶意软件时请注意，在二进制文件的主入口点（LC\_MAIN）之前，可能会在此类构造函数中开始执行。

在分析Mac恶意软件时，另一个相关的加载命令类型是LC\_load\_DYLIB。简而言之，LC\_load\_DYLIB load命令描述了一个动态库依赖项，它指示加载程序（dyld）加载并链接所述库。Mach-O二进制文件需要的每个库都有一个LC\_load\_DYLIB load命令（即依赖于）。

这个load命令（累积）是dylib\_command类型的结构（它包含一个struct dylib,描述实际的依赖动态库）：

```

01 struct dylib_命令{
02     uint32_t      cmd;           /* LC_LOAD_{,WEAK_}DYLIB */
03     uint32_t      规模;         /*包括路径名字符串*/
04     结构动态库    dylib;        /*图书馆标识*/
05 };
06
07 结构动态库{
08     工会会员姓名;               /*库的路径名*/
09     uint32_t时间戳;             /*图书馆的建设时间戳*/
10     uint32_t current_version;   /*库的当前版本号*/
11     uint32_t compatibility_version; /*图书馆号*/
12 };

```

LC\_LOAD\_dylib的dylib\_命令和DYLIB结构（mach-o/loader.h）

要解析Mach-O二进制文件的LC\_load\_DYLIB LOAD命令以查看二进制文件的依赖关系，请使用带有-L标志的otool实用程序。或者，MachOView [3]也可以。