

原

Linux ELF文件格式分析

2017年05月31日 23:47:21

谢健

阅读数：879

更多

版权声明：本文为博主原创文章，未经博主允许不得转载。 <https://blog.csdn.net/xj178926426/article/details/72825630>

Linux ELF文件格式分析

一、ELF文件格式

概述

ELF = Executable and Linkable Format，可执行连接格式，是UNIX系统实验室（USL）作为应用程序二进制接口（Application Binary Interface，ABI）而开放的。扩展名为elf。

其主要有三种主要类型：

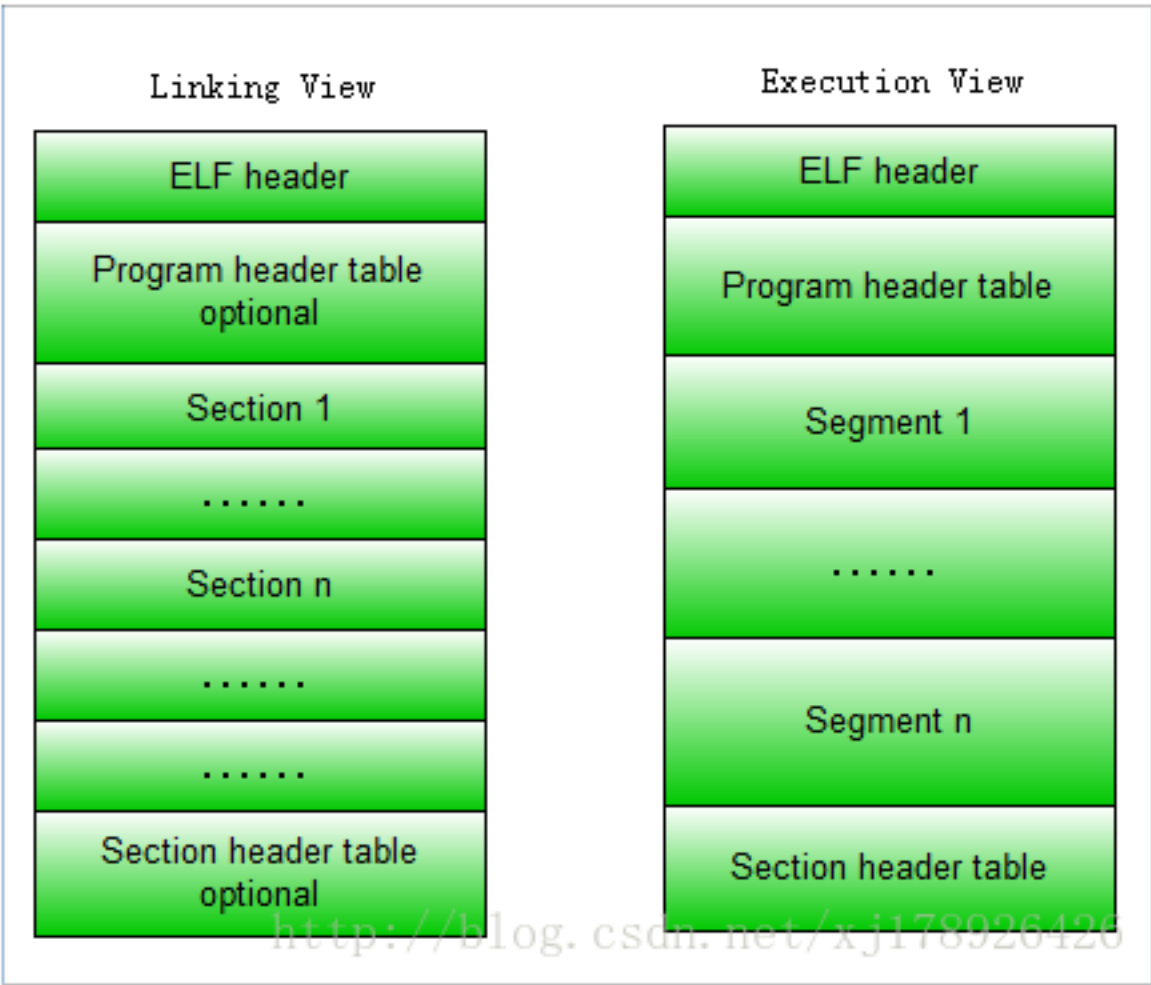
适于连接的可重定位文件(relocatable file)，可与其它目标文件一起创建可执行文件和共享目标文件。

适于执行的可执行文件(executable file)，用于提供程序的进程映像，加载的内存执行。

共享目标文件(shared object file)，连接器可将它与其它可重定位文件和共享目标文件连接成其它的目标文件，动态连接器又可将它与可执行文件和其它共享文件结合起来创建一个进程映像。

文件格式

为了方便和高效，ELF文件内容有两个平行的视角:一个是程序连接角度，另一个是程序运行角度，如图所示。



ELF header在文件开始处描述了整个文件的组织，Section提供了目标文件的各项信息（如指令、数据、符号表、重定位信息等），Program header table指向创建进程映像，含有每个program header的入口，section header table包含每一个section的入口，给出名字、大小等信息。其中Segment与section的关系后到。

要理解这个图，我们还要认识下ELF文件相关的几个重要的结构体：

1，ELF文件头

像bmp、exe等文件一样，ELF的文件头包含整个文件的控制结构。它的定义如下，可在/usr/include/elf.h中可以找到文件头结构定义：

```
typedef struct
{
    unsigned char e_ident[EI_NIDENT]; /* Magic number and other info */
    Elf64_Half e_type; /* Object file type */
    Elf64_Half e_machine; /* Architecture */
    Elf64_Word e_version; /* Object file version */
    Elf64_Addr e_entry; /* Entry point virtual address */
    Elf64_Off e_phoff; /* Program header table file offset */
    Elf64_Off e_shoff; /* Section header table file offset */
    Elf64_Word e_flags; /* Processor-specific flags */
    Elf64_Half e_ehsize; /* ELF header size in bytes */
    Elf64_Half e_phentsize; /* Program header table entry size */
    Elf64_Half e_phnum; /* Program header table entry count */
    Elf64_Half e_shentsize; /* Section header table entry size */
    Elf64_Half e_shnum; /* Section header table entry count */
    Elf64_Half e_shstrndx; /* Section header string table index */
} Elf64_Ehdr;
```

<http://blog.csdn.net/xj178926426>

其中e_ident的16个字节标识是个ELF文件（7F+’E’+’L’+’F’）。

e_type表示文件类型，2表示可执行文件。

e_machine说明机器类别，3表示386机器，8表示MIPS机器。

e_entry给出进程开始的虚地址，即系统将控制转移的位置。

e_phoff指出program header table的文件偏移。

e_phentsize表示一个program header表中的入口的长度（字节数表示）。

e_phnum给出program header表中的入口数目。类似的。

e_shoff, e_shentsize, e_shnum 分别表示section header表的文件偏移，表中每个入口的的字节数和入口数目。

e_flags给出与处理器相关的标志。

e_ehsize给出ELF文件头的长度（字节数表示）。

e_shstrndx表示section名表的位置，指出在section header表中的索引。

2,Program header

目标文件或者共享文件的program header table描述了系统执行一个程序所需要的段或者其它信息。目标文件的一个段（segment）包含一个或者多个section。Program header只对可执行文件和共享目标文件有意义，对于程序的链接没有任何意义。结构定义如下，可在/usr/include/elf.h中可以找到文件头结构定义：

```
typedef struct
{
    Elf64_Word p_type; /* Segment type */
    Elf64_Word p_flags; /* Segment flags */
    Elf64_Off p_offset; /* Segment file offset */
    Elf64_Addr p_vaddr; /* Segment virtual address */
    Elf64_Addr p_paddr; /* Segment physical address */
    Elf64_Xword p_filesz; /* Segment size in file */
    Elf64_Xword p_memsz; /* Segment size in memory */
    Elf64_Xword p_align; /* Segment alignment */
} Elf64_Phdr;
```

<http://blog.csdn.net/xj178926426>

其中p_type描述段的类型；

p_offset给出该段相对于文件开关的偏移量；

p_vaddr给出该段所在的虚拟地址；

p_paddr给出该段的物理地址；

p_filesz给出该段的大小，在字节为单元，可能为0；

p_memsz给出该段在内存中所占的大小，可能为0；

p_filesze与p_memsz的值可能会不相等。

Section Header

目标文件的section header table可以定位所有的section，它是一个Elf64_Shdr结构的数组，Section头表的索引是这个数组的下标。有些索引号是保留的，目不能使用这些特殊的索引。

Section包含目标文件除了ELF文件头、程序头表、section头表的所有信息，而且目标文件section满足几个条件：

目标文件中的每个section都只有一个section头项描述，可以存在不指示任何section的section头项。

每个section在文件中占据一块连续的空间。

Section之间不可重叠。

目标文件可以有非活动空间，各种headers和sections没有覆盖目标文件的每一个字节，这些非活动空间是没有定义的。

Section header结构定义如下，可在/usr/include/elf.h中可以找到文件头结构定义：


```
typedef struct
{
    Elf64_Word      sh_name;          /* Section name (string tbl index) */
    Elf64_Word      sh_type;          /* Section type */
    Elf64_Xword     sh_flags;         /* Section flags */
    Elf64_Addr      sh_addr;          /* Section virtual addr at execution */
    Elf64_Off       sh_offset;        /* Section file offset */
    Elf64_Xword     sh_size;          /* Section size in bytes */
    Elf64_Word      sh_link;          /* Link to another section */
    Elf64_Word      sh_info;          /* Additional section information */
    Elf64_Xword     sh_addralign;     /* Section alignment */
    Elf64_Xword     sh_entsize;       /* Entry size if section holds table */
} Elf64_Shdr;
http://blog.csdn.net/xjl78926426
```

其中sh_name指出section的名字， 它的值是后面将会讲到的section header string table中的偏移， 指出一个以null结尾的字符串。
sh_type是类别。
sh_flags指示该section在进程执行时的特性。
sh_addr指出若此section在进程的内存映像中出现， 则给出开始的虚地址。
sh_offset给出此section在文件中的偏移。其它字段的意义不太常用， 在此不细述。

文件的section含有程序和控制信息，系统使用一些特定的section， 并有其固定的类型和属性（由sh_type和sh_info指出）。下面介绍几个常用到的section：“.text”段含有占据程序内存映像的未初始化数据， 当程序开始运行时系统对这段数据初始为零， 但这个section并不占文件空间。“.data.”和“.data1”段包含占据内存映像的已初始化数据。“.rodata”和“.rodata1”段含程序映像中的只读数据。“.shstrtab”段含有每个section的名字， 由section入口结构中的sh_name索引值来获取。“.strtab”段含有符号表(symbol table)名字的字符串。“.symtab”段含有文件的符号表， 在后文专门介绍。“.text”段包含程序的可执行指令。

Symbol Table

目标文件的符号表包含定位或重定位程序符号定义和引用时所需要的信息。符号表入口结构定义如下， 可在usr/include/elf.h中可以找到文件头结构定义：

```
typedef struct
{
    Elf64_Word      st_name;          /* Symbol name (string tbl index) */
    unsigned char   st_info;          /* Symbol type and binding */
    unsigned char   st_other;         /* Symbol visibility */
    Elf64_Section    st_shndx;        /* Section index */
    Elf64_Addr      st_value;         /* Symbol value */
    Elf64_Xword     st_size;          /* Symbol size */
} Elf64_Sym;
http://blog.csdn.net/xjl78926426
```

其中st_name包含指向符号表字符串表(strtab)中的索引， 从而可以获得符号名。
st_value指出符号的值， 可能是一个绝对值、地址等。
st_size指出符号相关的内存大小， 比如一个数据结构包含的字节数等。
st_info规定了符号的类型和绑定属性， 指出这个符号是一个数据名、函数名、section名还是源文件名； 并且指出该符号的绑定属性是local、global还是weak。

二、结合实例分析

以一个最简单的helloworld程序为例：

```
1  #include <stdio.h>
2
3  int main()
4  {
5      printf("Hello World!\n");
6      return 0;
7  }
```

1. ELF文件头

使用工具查看ELF文件头： readelf -h obj

```
[james_xie@james-desk myCode]$ readelf -h helloworld
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:             ELF64
  Data:              2's complement, little endian
  Version:           1 (current)
  OS/ABI:            UNIX - System V
  ABI Version:       0
  Type:              EXEC (Executable file)
  Machine:           Advanced Micro Devices x86-64
  Version:           0x1
  Entry point address: 0x400440
  Start of program headers: 64 (bytes into file)
  Start of section headers: 6600 (bytes into file)
  Flags:             0x0
  Size of this header: 64 (bytes)
  Size of program headers: 56 (bytes)
  Number of program headers: 9
  Size of section headers: 64 (bytes)
  Number of section headers: 30
  Section header string table index: 27
```

大小总共为64字节，换算成十六进制为0x40。在十六进制代码中找到前0x40字节，即为文件头信息部分（阅读时注意反序问题）：

```
[james_xie@james-desk myCode]$ hexdump -C helloworld
00000000  7f 45 4c 46 02 01 01 00  00 00 00 00 00 00 00 00 | .ELF.....|
00000010  02 00 3e 00 01 00 00 00  40 04 40 00 00 00 00 00 | ..>.....@.@....|
00000020  40 00 00 00 00 00 00 00  c8 19 00 00 00 00 00 00 | @.....|
00000030  00 00 00 00 40 00 38 00  09 00 40 00 1e 00 1b 00 | .....@.8..@....|
00000040  06 00 00 00 05 00 00 00  40 00 00 00 00 00 00 00 | .....@.....|
```

对比上面结构体的定义，来解释下结构体各个字段的值：

e_ident：十六个字节，可通过这个字段对ELF文件进行识别，其中包括五个部分：

第一部分：占四个字节。**7f 45 4c 46**，对应ASCII码.ELF，表示这是一个ELF对象。

第二部分：占一个字节。**02**表示是一个64位对象。

第三部分：占一个字节。**01**表示是小端表示法。

第四部分：占一个字节。**01**表示文件头版本。

其余默认为0。

e_type：两个字节，**02 00**表示是一个可执行文件（ET_EXEC）。

e_machine：两个字节，**3e 00**表示是intel80386处理器体系结构。

e_version：四个字节，**01 00 00 00**表示是当前版本。

e_entry：八个字节，**40 04 40 00 00 00 00 00**表示当前程序入口点。

e_phoff：八个字节，**40 00 00 00 00 00 00 00**表示程序头表的偏移地址在 00 00 00 00 00 00 00 40处（这个地址是相对于本elf文件hellowrold来说,即程序表头在helloworld文件的0x40处，前面的0x40用来存放Elf64_Ehdr结构体信息，我们正在解这个结构体）。

e_shoff：八个字节，**c8 19 00 00 00 00 00 00**表示段表的偏移地址在00 00 00 00 00 00 19 c8处。

e_flags：四个字节，**00 00 00 00**表示未知处理器特定标志 **#define EF_SH_UNKNOWN 0x0。**

e_ehsize：两个字节，**40 00**表示elf文件头大小为00 40（64个字节）。

e_phentsize：两个字节，**38 00**表示重定位文件每个程序头表大小为00 38（56字节，从上面的e_phoff这个字段可以看出，程序头是在elf文件头的后面）。

e_phnum：两个字节，**09 00**表示重定位文件程序头表的个数为00 09（即9个程序表头，每个程序表头56字节）。

e_ehentsize：两个字节，**40 00** 表示段头大小为00 40（64字节）， section header table中每个header的大小。

e_shnum：两个字节，1e 00表示段表入口有30个，即段表有30段。

e_shstrndx：两个字节，1b 00 表示段表字符串在段表中的索引号，.shstrab段的段表索引号为00 1b，即27。

2,Program header

使用工具查看Program header： readelf -l obj

```
[james_xie@james-desk myCode]$ readelf -l helloworld

Elf file type is EXEC (Executable file)
Entry point 0x400440
There are 9 program headers, starting at offset 64

Program Headers:
  Type           Offset             VirtAddr           PhysAddr
                 FileSiz             MemSiz             Flags   Align
PHDR             0x0000000000000040 0x0000000000400040 0x0000000000400040
                 0x0000000000001f8 0x0000000000001f8   R E     8
INTERP           0x0000000000000238 0x0000000000400238 0x0000000000400238
                 0x00000000000001c 0x00000000000001c   R       1
  [Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
LOAD             0x0000000000000000 0x0000000000400000 0x0000000000400000
                 0x000000000000071c 0x000000000000071c   R E    200000
LOAD             0x0000000000000e10 0x0000000000600e10 0x0000000000600e10
                 0x0000000000000224 0x0000000000000228   RW     200000
DYNAMIC          0x0000000000000e28 0x0000000000600e28 0x0000000000600e28
                 0x00000000000001d0 0x00000000000001d0   RW     8
NOTE            0x0000000000000254 0x0000000000400254 0x0000000000400254
                 0x0000000000000044 0x0000000000000044   R       4
GNU_EH_FRAME    0x00000000000005f0 0x00000000004005f0 0x00000000004005f0
                 0x0000000000000034 0x0000000000000034   R       4
GNU_STACK       0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x0000000000000000 0x0000000000000000   RW     10
GNU_RELRO       0x0000000000000e10 0x0000000000600e10 0x0000000000600e10
                 0x00000000000001f0 0x00000000000001f0   R       1

Section to Segment mapping:
Segment Sections...
00
01      .interp
02      .interp .note.ABI-tag .note.gnu.build-id .gnu.hash .dynsym .dynstr .gnu.version .gnu.version_r .rela.dyn .rela.plt .init .plt .text .fini .rodata .eh_frame_hdr
03      .init_array .fini_array .jcr .dynamic .got .got.plt .data .bss
04      .dynamic
05      .note.ABI-tag .note.gnu.build-id
06      .eh_frame_hdr
07
08      .init_array .fini_array .jcr .dynamic .got
```

<http://blog.csdn.net/xj178926426>

从上图中可以知道，与我们上面对ELF文件头的分析完全对应的上，该目标文件一共有9个段，起始偏移地址是64。大小总共为56字节*9，换算成十六进制为0x1F8。在十六进制代码中偏移地址64字节开始找到前0x1F8字节，即为Program header信息部分（阅读时注意反序问题）：

```
00000040  06 00 00 00 05 00 00 00  40 00 00 00 00 00 00 00  | .....@.....|
00000050  40 00 40 00 00 00 00 00  40 00 40 00 00 00 00 00  | @.@.....@.@....|
00000060  f8 01 00 00 00 00 00 00  f8 01 00 00 00 00 00 00  | .....|
00000070  08 00 00 00 00 00 00 00  03 00 00 00 04 00 00 00  | .....|
00000080  38 02 00 00 00 00 00 00  38 02 40 00 00 00 00 00  | 8.....8.@.....|
00000090  38 02 40 00 00 00 00 00  1c 00 00 00 00 00 00 00  | 8.@.....|
000000a0  1c 00 00 00 00 00 00 00  01 00 00 00 00 00 00 00  | .....|
000000b0  01 00 00 00 05 00 00 00  00 00 00 00 00 00 00 00  | .....|
000000c0  00 00 40 00 00 00 00 00  00 00 40 00 00 00 00 00  | ..@.....@.....|
000000d0  1c 07 00 00 00 00 00 00  1c 07 00 00 00 00 00 00  | .....|
000000e0  00 00 20 00 00 00 00 00  01 00 00 00 06 00 00 00  | .. .....|
000000f0  10 0e 00 00 00 00 00 00  10 0e 60 00 00 00 00 00  | .....`.....|
00000100  10 0e 60 00 00 00 00 00  24 02 00 00 00 00 00 00  | ..`.....$.|
00000110  28 02 00 00 00 00 00 00  00 00 20 00 00 00 00 00  | ( .....|
00000120  02 00 00 00 06 00 00 00  28 0e 00 00 00 00 00 00  | .....( .....|
00000130  28 0e 60 00 00 00 00 00  28 0e 60 00 00 00 00 00  | (.`. ....(.`.....|
00000140  d0 01 00 00 00 00 00 00  d0 01 00 00 00 00 00 00  | .....|
00000150  08 00 00 00 00 00 00 00  04 00 00 00 04 00 00 00  | .....|
00000160  54 02 00 00 00 00 00 00  54 02 40 00 00 00 00 00  | T.....T.@.....|
00000170  54 02 40 00 00 00 00 00  44 00 00 00 00 00 00 00  | T.@.....D.....|
00000180  44 00 00 00 00 00 00 00  04 00 00 00 00 00 00 00  | D.....|
00000190  50 e5 74 64 04 00 00 00  f0 05 00 00 00 00 00 00  | P.td.....|
000001a0  f0 05 40 00 00 00 00 00  f0 05 40 00 00 00 00 00  | ..@.....@.....|
000001b0  34 00 00 00 00 00 00 00  34 00 00 00 00 00 00 00  | 4.....4.....|
000001c0  04 00 00 00 00 00 00 00  51 e5 74 64 06 00 00 00  | .....Q.td....|
000001d0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  | .....|
*
000001f0  00 00 00 00 00 00 00 00  10 00 00 00 00 00 00 00  | .....|
```

<http://blog.csdn.net/xj178926426>

这里我们要解释下上面的一个问题： Section和Segment的区别和联系
可执行文件中，一个program header描述的内容称为一个段（segment）。Segment包含一个或者多个section，我们以我们这个例子为例，看一下section与segment的映射关系：

```
Section to Segment mapping:
Segment Sections...
00
01      .interp
02      .interp .note.ABI-tag .note.gnu.build-id .gnu.hash .dynsym .dynstr .gnu.version .gnu.version_r .rela.dyn .rela.plt .init .plt .text .fini .rodata .eh_frame
03      .init_array .fini_array .jcr .dynamic .got .got.plt .data .bss
04      .dynamic
05      .note.ABI-tag .note.gnu.build-id
06      .eh_frame_hdr
07
08      .init_array .fini_array .jcr .dynamic .got
```

<http://blog.csdn.net/xj>

如上图映射关系可知，文本段并不仅仅包含.text节，数据段也不仅仅包含.data节，而是都包含了多个section。

3, Section Header

使用工具查看段表信息：readelf -S obj

```
[james_xie@james-desk myCode]$ readelf -S helloworld
There are 30 section headers, starting at offset 0x19c8:

Section Headers:
 [Nr] Name                               Type                               Address                               Offset
      Size                             EntSize                             Flags  Link  Info  Align
 [ 0]                                     NULL                                0000000000000000 00000000
      0000000000000000 0000000000000000 0 0 0
 [ 1] .interp                               PROGBITS                           0000000000400238 00000238
      0000000000000001c 0000000000000000 A 0 0 1
 [ 2] .note.ABI-tag                       NOTE                                0000000000400254 00000254
      00000000000000020 0000000000000000 A 0 0 4
 [ 3] .note.gnu.build-id                 NOTE                                0000000000400274 00000274
      00000000000000024 0000000000000000 A 0 0 4
 [ 4] .gnu.hash                           GNU_HASH                            0000000000400298 00000298
      0000000000000001c 0000000000000000 A 5 0 8
 [ 5] .dynsym                             DYNSYM                             00000000004002b8 000002b8
      00000000000000060 00000000000000018 A 6 1 8
 [ 6] .dynstr                             STRTAB                              0000000000400318 00000318
      0000000000000003d 0000000000000000 A 0 0 1
 [ 7] .gnu.version                       VERSYM                              0000000000400356 00000356
      00000000000000008 00000000000000002 A 5 0 2
 [ 8] .gnu.version_r                     VERNEED                             0000000000400360 00000360
      00000000000000020 0000000000000000 A 6 1 8
 [ 9] .rela.dyn                           RELA                                0000000000400380 00000380
      00000000000000018 00000000000000018 A 5 0 8
[10] .rela.plt                           RELA                                0000000000400398 00000398
      00000000000000048 00000000000000018 AI 5 12 8
[11] .init                               PROGBITS                           00000000004003e0 000003e0
      0000000000000001a 00000000000000000 AX 0 0 4
[12] .plt                                PROGBITS                           0000000000400400 00000400
```

<http://blog.csdn.net/xj>

	0000000000000040	0000000000000010	AX	0	0	16
[13]	.text	PROGBITS	0000000000400440	00000440		
	00000000000000184	0000000000000000	AX	0	0	16
[14]	.fini	PROGBITS	00000000004005c4	000005c4		
	00000000000000009	0000000000000000	AX	0	0	4
[15]	.rodata	PROGBITS	00000000004005d0	000005d0		
	0000000000000001d	0000000000000000	A	0	0	8
[16]	.eh_frame_hdr	PROGBITS	00000000004005f0	000005f0		
	00000000000000034	0000000000000000	A	0	0	4
[17]	.eh_frame	PROGBITS	0000000000400628	00000628		
	0000000000000000f4	0000000000000000	A	0	0	8
[18]	.init_array	INIT_ARRAY	0000000000600e10	00000e10		
	000000000000000008	0000000000000000	WA	0	0	8
[19]	.fini_array	FINI_ARRAY	0000000000600e18	00000e18		
	000000000000000008	0000000000000000	WA	0	0	8
[20]	.jcr	PROGBITS	0000000000600e20	00000e20		
	000000000000000008	0000000000000000	WA	0	0	8
[21]	.dynamic	DYNAMIC	0000000000600e28	00000e28		
	0000000000000001d0	00000000000000010	WA	6	0	8
[22]	.got	PROGBITS	0000000000600ff8	00000ff8		
	000000000000000008	00000000000000008	WA	0	0	8
[23]	.got.plt	PROGBITS	0000000000601000	00001000		
	000000000000000030	00000000000000008	WA	0	0	8
[24]	.data	PROGBITS	0000000000601030	00001030		
	000000000000000004	00000000000000000	WA	0	0	4
[25]	.bss	NOBITS	0000000000601034	00001034		
	000000000000000004	00000000000000000	WA	0	0	4
[26]	.comment	PROGBITS	0000000000000000	00001034		
	00000000000000002d	00000000000000001	MS	0	0	1
[27]	.shstrtab	STRTAB	0000000000000000	00001061		

	00000000000000108	0000000000000000	0	0	1
[28]	.symtab	SYMTAB	0000000000000000	00001170	
	000000000000000618	00000000000000018	29	45	8
[29]	.strtab	STRTAB	0000000000000000	00001788	
	00000000000000023c	00000000000000000	0	0	1

在文件头中e_shoff可以找到段表偏移地址00 00 00 00 00 00 19 c8，从这个地址去查找段表。

段表长度由e_ehentsize为00 40（64字节）。

段表个数由e_shnum可知有30个。

这么多section我就不一一全部详细的列出来，我选一个text section来进行分析，上图中可以看到，text section索引序号为13，我们的段表的起始偏移地址为0x19c8，每个段长度为0x40，其前面有13个段，所以我们text section的起始地址应该是0x19c8 + （0x40*0x0d） = 0x1d08，我们来通过

```
1 hexdump -C helloworld
```

命令来验证下文件中这个地址是不是text section。

00001d00	10 00 00 00 00 00 00 00 00	94 00 00 00 01 00 00 00@.....
00001d10	06 00 00 00 00 00 00 00 00	40 04 40 00 00 00 00 00@.
00001d20	40 04 00 00 00 00 00 00 00	84 01 00 00 00 00 00 00	@.....
00001d30	00 00 00 00 00 00 00 00 00	10 00 00 00 00 00 00 00
00001d40	00 00 00 00 00 00 00 00 00	9a 00 00 00 01 00 00 00

sh_name：四个字节，94 00 00 00表示该段名称在.shstrtab中偏移量，我们通过

```
1 readelf -x .shstrtab helloworld
```

来看下.shstrtab段里面偏移量为0x94处是不是.text：


```
[james_xie@james-desk myCode]$ readelf -x .shstrtab helloworld

Hex dump of section '.shstrtab':
0x00000000 002e7379 6d746162 002e7374 72746162  ..symtab..strtab
0x00000010 002e7368 73747274 6162002e 696e7465  ..shstrtab..inte
0x00000020 7270002e 6e6f7465 2e414249 2d746167  rp..note.ABI-tag
0x00000030 002e6e6f 74652e67 6e752e62 75696c64  ..note.gnu.build
0x00000040 2d696400 2e676e75 2e686173 68002e64  -id..gnu.hash..d
0x00000050 796e7379 6d002e64 796e7374 72002e67  ynsym..dynstr..g
0x00000060 6e752e76 65727369 6f6e002e 676e752e  nu.version..gnu.
0x00000070 76657273 696f6e5f 72002e72 656c612e  version_r..rela.
0x00000080 64796e00 2e72656c 72002e70 74002e69  dyn..rela.plt..i
0x00000090 6e697400 2e746578 74002e66 696e6900  nit..text..fini.
0x000000a0 2e726f64 61746100 2e65685f 6672616d  .rodata..eh_fram
0x000000b0 655f6864 72002e65 685f6672 616d6500  e_hdr..eh_frame.
0x000000c0 2e696e69 745f6172 72617900 2e66696e  .init_array..fin
0x000000d0 695f6172 72617900 2e6a6372 002e6479  i_array..jcr..dy
0x000000e0 6e616d69 63002e67 6f74002e 676f742e  namic..got..got.
0x000000f0 706c7400 2e646174 61002e62 7373002e  plt..data..bss..
0x00000100 636f6d6d 656e7400  http://blog.csdn.net/xj178926426
```

由上图可知，偏移量为0x94处确实就是.text，看来我们上面的推算都是正确的。

sh_type：四个字节，01 00 00 00表示这个段拥有程序所定义的信息，其格式和含义完全由该程序确定，这里表示PROGBITS。

sh_flags：八个字节，06 00 00 00 00 00 00 00表示alloc和execute。

sh_addr：八个字节，40 04 40 00 00 00 00 00表示是section在内存中的虚拟地址为0x400440。

sh_offset：八个字节，40 04 00 00 00 00 00 00表示是section与文件头之间的偏移为0x0440。

sh_size：八个字节，84 01 00 00 00 00 00 00表示文件里面section占用的大小为0x0184。

sh_link：四个字节，00 00 00 00表示没有链接信息。

sh_info：四个字节，00 00 00 00表示没有辅助信息。

sh_addralign：八个字节，10 00 00 00 00 00 00 00表示字节对齐长度。

sh_entsize：八个字节，00 00 00 00 00 00 00 00表示没有入口。

我们按照上面的sh_offset和sh_size字段通过hexdump命令来看看目标文件偏移0x440处，长度为0x0184的内容：

00000440	31	ed	49	89	d1	5e	48	89	e2	48	83	e4	f0	50	54	49	1.I..^H..H...PTI
00000450	c7	c0	c0	05	40	00	48	c7	c1	50	05	40	00	48	c7	c7@.H..P.@.H..
00000460	2d	05	40	00	e8	b7	ff	ff	ff	f4	66	0f	1f	44	00	00	-.@.....f..D..
00000470	b8	3f	10	60	00	55	48	2d	38	10	60	00	48	83	f8	0e	.?.`.UH-8.`.H...
00000480	48	89	e5	77	02	5d	c3	b8	00	00	00	00	48	85	c0	74	H..w.].....H..t
00000490	f4	5d	bf	38	10	60	00	ff	e0	0f	1f	80	00	00	00	00	.].8.`.....
000004a0	b8	38	10	60	00	55	48	2d	38	10	60	00	48	c1	f8	03	.8.`.UH-8.`.H...
000004b0	48	89	e5	48	89	c2	48	c1	ea	3f	48	01	d0	48	d1	f8	H..H..H..?H..H..
000004c0	75	02	5d	c3	ba	00	00	00	00	48	85	d2	74	f4	5d	48	u.].....H..t.]H
000004d0	89	c6	bf	38	10	60	00	ff	e2	0f	1f	80	00	00	00	00	...8.`.....
000004e0	80	3d	4d	0b	20	00	00	75	11	55	48	89	e5	e8	7e	ff	.=M. ...u.UH...~.
000004f0	ff	ff	5d	c6	05	3a	0b	20	00	01	f3	c3	0f	1f	40	00	..].... ..@.
00000500	48	83	3d	18	09	20	00	00	74	1e	b8	00	00	00	00	48	H.=.. ..t.....H
00000510	85	c0	74	14	55	bf	20	0e	60	00	48	89	e5	ff	d0	5d	..t.U. ...`..H....]
00000520	e9	7b	ff	ff	ff	0f	1f	00	e9	73	ff	ff	ff	55	48	89	.{.....s...UH.
00000530	e5	bf	e0	05	40	00	e8	d5	fe	ff	ff	b8	00	00	00	00@.....
00000540	5d	c3	66	2e	0f	1f	84	00	00	00	00	00	0f	1f	40	00].f.....@.
00000550	41	57	41	89	ff	41	56	49	89	f6	41	55	49	89	d5	41	AWA..AVI..AUI..A
00000560	54	4c	8d	25	a8	08	20	00	55	48	8d	2d	a8	08	20	00	TL.%.. ..UH.-... .
00000570	53	4c	29	e5	31	db	48	c1	fd	03	48	83	ec	08	e8	5d	SL).1.H...H....]
00000580	fe	ff	ff	48	85	ed	74	1e	0f	1f	84	00	00	00	00	00	...H..t.....
00000590	4c	89	ea	4c	89	f6	44	89	ff	41	ff	14	dc	48	83	c3	L..L..D..A...H..
000005a0	01	48	39	eb	75	ea	48	83	c4	08	5b	5d	41	5c	41	5d	.H9.u.H...[A\A]
000005b0	41	5e	41	5f	c3	90	66	2e	0f	1f	84	00	00	00	00	00	A^A...f..
000005c0	f3	c3	66	90	48	83	ec	08	48	83	c4	08	c3	00	00	00	..f.H...H.....

然后通过如下命令直接把text section的具体内容可以打印出来，

```
1 readelf -x .text helloworld
```

跟上图的数据进行对比：


```
[james_xie@james-desk myCode]$ readelf -x .text helloworld

Hex dump of section '.text':
 0x00400440 31ed4989 d15e4889 e24883e4 f0505449 1.I..^H..H...PTI
 0x00400450 c7c0c005 400048c7 c1500540 0048c7c7 ....@.H..P.@.H..
 0x00400460 2d054000 e8b7ffff fff4660f 1f440000 -.@.....f..D..
 0x00400470 b83f1060 0055482d 38106000 4883f80e .?.`.UH-8.`.H...
 0x00400480 4889e577 025dc3b8 00000000 4885c074 H..w.]......H..t
 0x00400490 f45dbf38 106000ff e00f1f80 00000000 .].8.`.....
 0x004004a0 b8381060 0055482d 38106000 48c1f803 .8.`.UH-8.`.H...
 0x004004b0 4889e548 89c248c1 ea3f4801 d048d1f8 H..H..H..?H..H..
 0x004004c0 75025dc3 ba000000 004885d2 74f45d48 u.]......H..t.]H
 0x004004d0 89c6bf38 106000ff e20f1f80 00000000 ...8.`.....
 0x004004e0 803d4d0b 20000075 11554889 e5e87eff .=M. ..u.UH...~.
 0x004004f0 ffff5dc6 053a0b20 0001f3c3 0f1f4000 ..].:. ....@.
 0x00400500 48833d18 09200000 741eb800 00000048 H.=. ..t.....H
 0x00400510 85c07414 55bf200e 60004889 e5ffd05d ..t.U.  ``.H....]
 0x00400520 e97bffff ff0f1f00 e973ffff ff554889 .{.....s...UH.
 0x00400530 e5bfe005 4000e8d5 fefffffb8 00000000 .....@.....
 0x00400540 5dc3662e 0f1f8400 00000000 0f1f4000 ].f.....@.
 0x00400550 41574189 ff415649 89f64155 4989d541 AWA..AVI..AUI..A
 0x00400560 544c8d25 a8082000 55488d2d a8082000 TL.%. ..UH.-... .
 0x00400570 534c29e5 31db48c1 fd034883 ec08e85d SL).1.H...H....]
 0x00400580 fefffff48 85ed741e 0f1f8400 00000000 ...H..t.....
 0x00400590 4c89ea4c 89f64489 ff41ff14 dc4883c3 L..L..D..A...H..
 0x004005a0 014839eb 75ea4883 c4085b5d 415c415d .H9.u.H...[A\A]
 0x004005b0 415e415f c390662e 0f1f8400 00000000 A^A_..f.....
 0x004005c0 f3c36690 .....f.....
```

<http://blog.csdn.net/xjl78926426>

很明显我前面的分析是对的，其他section都可以通过上面办法来验证下，至于text section里的具体内容是什么，程序执行具体的过程，我将在后续跟进，这只是简单的认识下ELF文件。
文中有什么错误之处或者表达不明白的位置欢迎大家拍砖指出！

linux逆向分析之ELF文件详解

前言 首先如果大家遇到ELF二进制文件的逆向首先考虑的可能就是通过IDA进行静态逆向分析算法，那么我们首先就要了解ELF(Executable ... 来自： [cavalier](#)

 想对作者说点什么

linux中的ELF文件有哪几类？（注意：静态库不是ELF文件） 5601

ELF = executable linkable format 可执行、链接格式 linux中的中ELF文件主要... 来自： [stpeace](#)的专栏

linux中ELF文件动态链接的加载、解析及实例分析（二）：函数解析与卸载 3516

相信读者已经看过了Intel平台下Linux中ELF文件动态链接的加载、解析及实例分析... 来自： [eros](#)的linux平台...

LinuxELF文件格式详解--Linux进程的管理与调度（十二） 6526

日期 内核版本 架构 作者 GitHub CSDN 2016-06-04 Linux-4.5 X86 & arm gatieme Li... 来自： [AderStep](#)

Python系统学习技能图谱免费领（程序员2019年薪资翻倍秘诀）

\\最近python很火啊，你看下我们能用爬取一下某网站数据，做些数据分析的工作吗?\\

Python系统学习技能图谱免费领（程序员2019年薪资翻倍秘诀）

\\最近python很火啊，你看下我们能用爬取一下某网站数据，做些数据分析的工作吗?\\

在 Linux 中修改一个现有的 elf 文件 506

github 项目地址： change-elf 在Linux中下修改一个现有的elf可执行程序 首先了解 el... 来自： [yin__ren](#)的博客

linux elf文件格式 297

一、ELF文件格式概述 1. ELF：是一种对象文件的格式，用于定义不同类型的对象... 来自： [swartz_lubel](#)的...

[Debug]linux elf文件格式 356

linux elf文件格式 浅谈Linux的可执行文件格式ELF 来自： [知了112](#)的专栏

Linux ELF文件 2200

版权声明：本文为博主原创文章，未经博主允许不得转载。 目录(?)[+] elf格式 ELF(... 来自： [xiaohuima_don](#)...

Linux中ELF格式文件介绍

7775

ELF(Executable and Linkable Format)即可执行连接文件格式，是一种比较复杂的文...

来自： 华的专栏

linux入门培训

开源网店系统有哪些

百度广告

广告

ELF文件的加载过程(load_elf_binary函数详解)--Linux进程的管理与调度...

1.2万

日期 内核版本 架构 作者 GitHub CSDN 2016-06-04 Linux-4.6 X86 & arm gatieme Li...

来自： AderStep

文章热词

LinuxLinux学习Linux视频教程Linux认证Linux教程

相关热词

c++读取bmp文件格式c++ 二进制保存文件格式bmp c++ png 文件格式c# 获取文件格式bmp图像文件格式c++

Linux系统ELF文件二进制格式分析(一)

1434

ELF是Executable andLinkable Format的缩写，它是Linux下可执行文件、目标文件...


来自： 日积月累

大空新一

35篇文章

关注

排名:千里之外

beyond702

113篇文章

关注

排名:千里之外

刘星石

201篇文章

关注

排名:千里之外

whatday

1067篇文章

关注

排名:247

ELF文件格式与动态链接/静态链接与动态库/静态库（Linux下 可执行文件...

834

ELF文件格式在Linux下，可执行文件/动态库文件/目标文件（可重定向文件）都...

来自： 我喜欢雨天的清...

Linux 查看 elf可执行文件格式的两个命令

3158

This article is from http://hi.baidu.com/widebright/blog/item/2acbf536ec3c12390b55...

来自： Harold Wang C...


目的檔格式 (ELF)

419

目的檔格式 (ELF) 目的檔ELF 格式(Executable and Linking Format) 是 UNIX/Linux ...

来自： kyokowl的专栏

linux编程学习

开源网店系统

百度广告

广告

下载 ELF 文件格式分析

05-04

嵌入式操作系统应用领域广,硬件环境复杂多样,降低开发成本、缩短开发周 期、提高产品质量是工业界和学术界共同关注的问题。借鉴软件复用的思想,采用基于构件的软件开发...

ELF文件-逆向工具

1169

转载地址： http://bdxnote.blog.163.com/blog/static/8444235201532911597959/ 1、...

来自： 爱自在的专栏

Linux系统误操作之-文件权限介绍和恢复分享

4346

每个Linux用户都有它所属的用户组, 用户或用户组构成了Linux文件或目录的权限访...

来自： 简单如阿甘

ELF文件格式头部

96

ELF格式文件格式头部

来自： qq_35467337的...

ELF文件解析和反汇编

1.2万

首先来看一段Unix/Linux下的汇编代码： #PURPOSE: This program finds the maxi...

来自： wuxinke_blog的...



国际认证RHCE Linux认证考试

百度广告

	• linux 下如何判断 elf 文件是32位还是64位？		👁 4233
	例如一个可执行文件messy，我想知道这个是32位还是64位的， 可以用readelf 这个... 来自： cloudusers的专栏		
	• <div>下载</div> 从程序员角度看ELF		11-17
	ELF文件 文件格式 linux ELF文件 文件格式 linux ELF文件 文件格式 linux		
	• Linux 系统 ELF 文件二进制格式 分析 (三)		👁 1364
	本文接着《Linux系统ELF文件二进制格式分析(二)》进行分析 四、符号表 符号表保... 来自： 日积月累		
	• <div>下载</div> 《ELF文件格式分析.pdf》与elf解析代码		03-22
	《ELF文件格式分析.pdf》文档，非常不错的elf格式参考文档，参考elf解析过程，能很快掌握elf文件格式		
	• <div>下载</div> 嵌入式linux小议： ELF 文件格式分析		07-19
	嵌入式linux小议： ELF 文件格式分析嵌入式linux小议： ELF 文件格式分析		
	<div></div> 千万不要再乱喝蜂蜜了!知情人士亲赴深山，发现惊人真相！		
	聚优 · 顶新		
	<div></div> 千万不要再乱喝蜂蜜了，美女亲赴深山，揭露背后惊人黑幕！		
	聚优 · 顶新		
	• <div>下载</div> ELF 文件格式分析-北京大学信息科学技术学院操作系统实验室		06-15
	ELF 文件格式分析-北京大学信息科学技术学院操作系统实验室		
	• Linux ELF 文件装入与执行概述		👁 7691
	ELF是linux中使用最广泛的一种应用程序格式，为了弄清楚Linux内核是如何讲ELF... 来自： 余璜的技术博客		
	• linux 下的 ELF 对象格式		👁 1284
	1.ELF概念 ELF是unix-like系统下的一种文件格式，它是一种对象文件的格式， ... 来自： @_囚徒-2018_...		
	• Linux 系统 ELF 文件二进制格式 分析 (二)		👁 2276
	本文接着《Linux系统ELF文件二进制格式分析(一)》继续分析ELF文件格式		来自： 日积月累
	• Linux 下的 ELF 可执行文件学习总结		👁 1964
	Linux下的ELF可执行文件的格式解析 http://blog.csdn.net/xuchao1229/article/details/... 来自： bcbobo21cn的...		
	<div></div> 白发千万不要染，饭后一件事，想要多黑就多黑		
	盛世艺灿 · 熾燚		
	<div></div> 紧急通知：吸这两种烟的人，快去医院检查！		
	华佰科技 · 顶新		
	• linux 下调试 elf 各命令		👁 773
	readelf -S： 查看段表信息 -s: 查看符号信息 objdump -d :将包含指令的段反汇编 -s :... 来自： liutianheng654...		
	• <div>下载</div> ELF 文件格式分析		11-25
	书名：ELF文件格式分析 作者：滕启明 北京大学信息科学技术学院操作系统实验室 2003年5月 文件格式pdf 非扫描版		
	• linux 下 elf 重定位理解		👁 981
	准备：可重定位文件（Relocatable file），可执行文件（Executable file），共享文... 来自： 大雄不爱吃肉		
	• bin 和 elf 文件格式的区别		👁 489
	嵌入式开发的时候，我们的编译一个*.S文件，并最终生成bin文件，编译命令大致如... 来自： 旭		
	• python 读取各种文件数据解析		👁 2377
	python读取.txt（.log）文件、.xml 文件、excel文件数据，并将数据类型转换为需要... 来自： yiweiyi329的博客		



白发千万不要染，饭后一件事，想要多黑就多黑

[盛世艺灿](#) · [熾燚](#)



白发千万不要染，饭后一件事，想要多黑就多黑

- 军事理论课答案（中国国防史）

中国国防史——秦至两晋南北朝已完成 成绩： 100.0分 1 【单选题】 中国哪个历史...

271870

来自： [ling_wang的博客](#)
- 100个小学生猜字谜大全及答案

100个小学生猜字谜大全及答案 1.字谜： 山上还有山。猜一字， 答案是:出 2.字谜： ...

208929

来自： [欢迎光临 包国...](#)
- tcp的java代码

服务器端 package com.car.client; import java.io.IOException; import java.io.Output...

15081

来自： [weixin_436941...](#)
- 学习Webpack（一）之 初识webpack

学习Webpack（一）之 初识webpack webpack简介 在官网中说，webpack是一个现...

52

来自： [幽幽小春](#)



谢健

关注

原创114

粉丝60

喜欢24

评论72

等级：

博客 5

访问：20万+

积分：3088

排名：1万+

勋章：

恒



加州的大学排名



联系我

 给我写信

最新文章

海力士内存条

在Centos7上编译synergy



【lua学习】学习一

【lua学习】Lua中pairs和ipairs区别

【lua学习】Lua 协同程序(coroutine)

个人分类

Linux 学习29篇

C/C++学习17篇

Qt 学习8篇

Php 学习5篇

Python Scrapy 学习6篇

展开



抑郁测试



linux红帽认证



台式机内存

■ 内存的分配

■ 上海linux 培训

■ 信号中断

■ 天魔传人

■ flags

■ 文件汇编

广告

联系我们



微信客服



QQ客服

👤 QQ客服 ✉ kefu@csdn.net
🗣 客服论坛 ☎ 400-660-0108
⌚ 工作时间 8:00-22:00

关于我们 | 招聘 | 广告服务 | 网站地图
🐾 百度提供站内搜索 京ICP证09002463号
©1999-2018 江苏乐知网络技术有限公司
江苏知之为计算机有限公司 北京创新乐知
信息技术有限公司版权所有

网络110报警服务 经营性网站备案信息
北京互联网违法和不良信息举报中心
中国互联网举报中心

联系我们

👤 QQ客服 ✉ kefu@csdn.net
🗣 客服论坛 ☎ 400-660-0108
⌚ 工作时间 8:00-22:00

关于我们 | 招聘 | 广告服务 | 网站地图
🐾 百度提供站内搜索 京ICP证09002463号
©1999-2018 江苏乐知网络技术有限公司
江苏知之为计算机有限公司 北京创新乐知
信息技术有限公司版权所有

网络110报警服务 经营性网站备案信息
北京互联网违法和不良信息举报中心
中国互联网举报中心