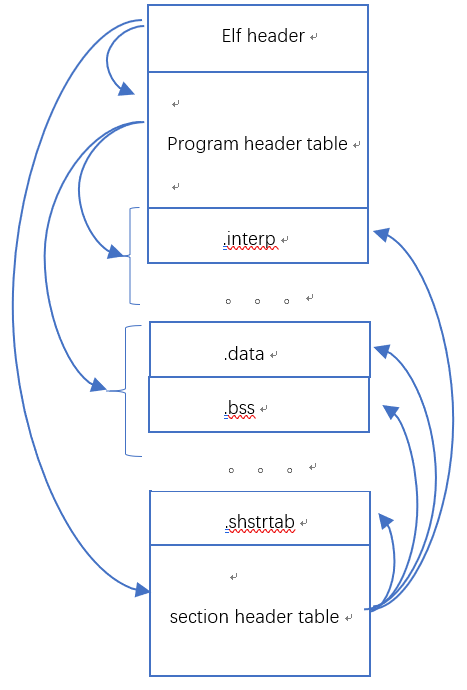
**elf文件介绍**

1. **elf文件结构**



1. **elf头部介绍**

elf头部存放位置从文件开头第一个字节开始，对应数据结构是Elf64\_Ehdr:

typedef struc

{

unsigned char e\_ident[EI\_NIDENT]; /\* 共16个字节，包含magic，64位还是32位、大小端等信息 \*/

Elf64\_Half e\_type; /\* 类别，如ELF64或者ELF32\*/

Elf64\_Half e\_machine; /\* 机器架构，如x86\_64 \*/

Elf64\_Word e\_version; /\* 版本\*/

Elf64\_Addr e\_entry; /\*代码入口地址，.so或者.a库该值为0\*/

Elf64\_Off e\_phoff; /\* 段头部表在文件中的偏移\*/

Elf64\_Off e\_shoff; /\*节头部表在文件中的偏移\*/

Elf64\_Word e\_flags; /\* Processor-specific flags \*/

Elf64\_Half e\_ehsize; /\* ELF头部大小 \*/

Elf64\_Half e\_phentsize; /\* 段头部表元素（段头部）大小\*/

Elf64\_Half e\_phnum; /\* 段头部表元素（段头部数量）\*/

Elf64\_Half e\_shentsize; /\* 节头部表元素（节头部）大小\*/

Elf64\_Half e\_shnum; /\* 节头部表元素（节头部）数量 \*/

Elf64\_Half e\_shstrndx; /\* .shstrable节在section header table中的索引\*/

} Elf64\_Ehdr;

1. **elf段头部表介绍**

elf段头表存放的是elf所有段的头部信息，存放起始位置由Elf64\_Ehdr的e\_phoff指定，一般都紧接着Elf64\_Ehdr存放，表元素数量为e\_phnum，元素大小为e\_phentsize，对应的结构体：

Elf64\_Phdr：

typedef struct

{

Elf64\_Word p\_type; /\* Segment 类型，比如PHDR、DYNAMIC、NOTE等\*/

Elf64\_Word p\_flags; /\* Segment flags \*/

Elf64\_Off p\_offset; /\* Segment 在文件中存放的偏移值\*/

Elf64\_Addr p\_vaddr; /\* Segment virtual address \*/

Elf64\_Addr p\_paddr; /\* Segment physical address \*/

Elf64\_Xword p\_filesz; /\* Segment在文件中的大小\*/

Elf64\_Xword p\_memsz; /\* Segment 在内存中的小 \*/

Elf64\_Xword p\_align; /\* Segment alignment \*/

} Elf64\_Phdr;

1. **elf节头部表介绍**

elf节表头部表存放的是elf文件所有节的头部信息，存放起始位置由Elf64\_Ehdr的e\_shoff指定，表元素数量为e\_shnum，元素大小为e\_shentsize，对应的结构体为：

typedef struct

{

Elf64\_Word sh\_name; /\* section名称在shstrable中存放的偏移值\*/

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在文件中存放的偏移值 \*/

Elf64\_Xword sh\_size; /\* section 大小 \*/

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;

1. **查看elf文件相关的工具**
2. readelf

-V : 查看elf文件版本信息

-h 查看elf文件头部信息

-l： 查看程序段头信息，包括段和节之间的映射关系

-S: 查看节头信息

-s: 查看代码符号表

-e：查看全部头部信息，等价于：-h -l -S

2、file

查看文件类型、版本、大小端、cpu架构等信息

* 1. strings

输出elf文件中所有的字符串，包括所有的段名称、节名称、符号名等。

* 1. size

指出目标文件段大小

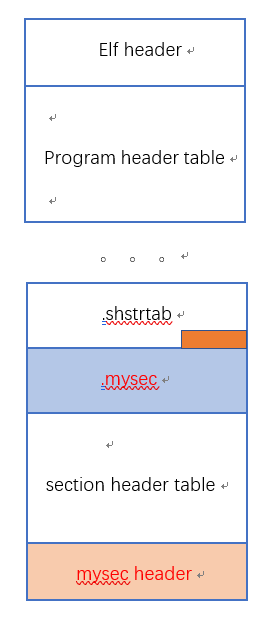
* 1. objdump

能查看很多二进制信息

1. **修改elf文件思路**
2. 向elf文件末尾插入一个新的节

原则是不能破坏elf文件原有的数据存放结构，同时涉及修改的内容尽可能少。

增加一个名为.mysec的新section后文件结构如下：



代码流程：

判断elf文件是否是64位？，

并且大小端是否和本机一致？

调整相关头部表字段：

p\_ehdr->e\_shnum（节数量）加1；

p\_ehdr->e\_shoff（节头表偏移）指向调整后的位置；

shstrtable的size加上新增节名字长度

把构造好新增节头部上面一步的后面（及原来sction header tables的末尾）

把buff内容写入到新增节后面

在shstrtable后面写入新增节的内容

读取elf文件头部信息

将shstrtable以后的内容(实际上就是section header table)读取出来存放到一个临时buff中

在shstrtable末尾写入新增节的名字

1. 从elf文件中读出并删除末尾的一个节

代码流程：

用零时文件去覆盖目标文件

调整零时文件相关表头部信息：

p\_ehdr->e\_shnum(节头部表元素)减一

p\_ehdr->e\_shoff（节头部表偏移）指向当前存放起始位置

shstrable size 减去目标section name 的长度。

拷贝整个section header table(除最后一个外)追加到临时文件

拷贝目标上section name以前的内容到临时文件

拷贝存放目标section name以前的内容到临时文件

读取目标section的内容

根据给定的section name，在elf文件中，查找是否有这个section

判断elf文件最后一个section是否是要删除的section？

判断elf文件是否是64位？，

并且大小端是否和本机一致？