1. **如何修改动态库符号表**

<http://dev.firnow.com/course/6_system/linux/linuxjq/2007211/18010.html>

**ELF 文件中代码、连接信息和注释是以节(section)为单位存放的，并存有一个节头表(section header)**。对每一节，在节头表中都有一个表项(节头表项)与之对应，表项记录了该节的一些信息，例如该节在文件中的位置信息和该节的字节长度信息。

**一、ELF 文件和有关术语**

Unix 系统的可执行文件和动态库文件是以 ELF 格式存放的。为使下面的叙述清晰而没有伎义，先简要介绍一下 ELF 文件格式，并约定一些术语。关于ELF文件格式的详细情况请参看有关文献。

ELF 文件中代码、连接信息和注释是以节(section)为单位存放的，并存有一个节头表(section header)。对每一节，在节头表中都有一个表项(节头表项)与之对应，**表项记录了该节的一些信息，例如该节在文件中的位置信息和该节的字节长度信息。**

程序运行读入内存时，是以程序段(program segment)为单位读入的。在 ELF文件中有一个程序头表(program header table)，每个程序段在程序头表中有一个表项(程序头表项)与之对应，表项记录了该程序段的有关信息，例如该程序段在文件中的位置信息和该程序段的字节长度信息。程序段的内容由若干节组成，节的内容组合在一起连成一片构成程序段的内容。

在所有这些节中，**有一节的内容由字符串构成，这些字符串是各节的名称，叫节名**。**下面称这一节为节名表**。另有一节，**节名为".dynsym"**，它的内容为符号表，符号表的每一表项记录了一个符号的有关信息，例如**该符号对应的代码的地址值**。还有一节，**节名为".dynstr"**，它的内容由字符串构成。大多数符号在该节中有一个字符串与之对应，这个字符串是该符号的符号名。**而每一函数对应一个符号，函数名即为符号名**。下面称被某一函数对应的符号为函数符号。

ELF 文件开始处的一段叫 ELF 文件头。**它记录了程序头表在文件中的偏移**、程序头表的表项数目、程序头表每一表项的字节长度、节头表在文件中的偏移、节头表的表项数目、节头表每个表项的字节长度。它还记录了节名表所在的节的索引序号。

**二、动态库符号表修改方法**

修改动态库符号表的方法和步骤如下：

**第一步：**

读取 ELF 文件头，取出

(1) 程序头表在文件中的偏移，获取程序头表在文件中的位置；

(2) 程序头表的表项数目和程序头表每一表项的字节长度；

(3) 节头表在文件中的偏移，获取节头表在文件中的位置；

(4) 节头表的表项数目和节头表每个表项的字节长度；

(5) 节名表所在的节的索引序号。

ELF 文件头在文件中的偏移为零，即起始于 ELF 文件开头的第一字节，它的数据结构为：

#define EI\_NIDENT (16)

typedef uint16\_t Elf32\_Half;  
typedef uint32\_t Elf32\_Word;  
typedef uint32\_t Elf32\_Addr;  
typedef uint32\_t Elf32\_Off;  
typedef uint16\_t Elf32\_Section;

typedef struct  
{  
unsigned char e\_ident; /\* Magic number及其他信息 \*/  
Elf32\_Half e\_type; /\* ELF 文件类型 \*/  
Elf32\_Half e\_machine; /\* 机器型号 \*/  
Elf32\_Word e\_version; /\* 版本 \*/  
Elf32\_Addr e\_entry; /\* 程序入口虚地址 \*/  
Elf32\_Off e\_phoff; /\* 程序头表在文件中的偏移 \*/  
Elf32\_Off e\_shoff; /\* 节头表在文件中的偏移 \*/  
Elf32\_Word e\_flags; /\* 处理器标志 \*/  
Elf32\_Half e\_ehsize; /\* ELF 文件头长度 \*/  
Elf32\_Half e\_phentsize; /\* 程序头表每个表项长度 \*/  
Elf32\_Half e\_phnum; /\* 程序头表的表项总数 \*/  
Elf32\_Half e\_shentsize; /\* 节头表每个表项长度 \*/  
Elf32\_Half e\_shnum; /\* 节头表的表项总数 \*/  
Elf32\_Half e\_shstrndx; /\* 节名表所在的节的表项索引号 \*/  
} Elf32\_Ehdr;

**第二步：**

依据节头表在文件中的偏移和节名表所在的节的索引序号，**定出节名表在节头表中对应的表项的文件偏移，即定出该表项在文件中的位置。**读取该表项，取出节名表在文件中的偏移和该节在文件中的字节长度。节头表由若干表项组成，每个表项的内容按下面的数据结构来组织：

typedef struct  
{  
Elf32\_Word sh\_name; /\* 节名索引号 \*/  
Elf32\_Word sh\_type; /\* 节类型 \*/  
Elf32\_Word sh\_flags; /\* 节标志 \*/  
Elf32\_Addr sh\_addr; /\* 执行时该节虚地址 \*/  
Elf32\_Off sh\_offset; /\* 在文件中的偏移 \*/  
Elf32\_Word sh\_size; /\* 节长度 \*/  
Elf32\_Word sh\_link; /\* 到其他节的连接 \*/  
Elf32\_Word sh\_info; /\* 其他信息 \*/  
Elf32\_Word sh\_addralign; /\* alignment \*/  
Elf32\_Word sh\_entsize; /\* 如内容为表，每个表项的长度 \*/  
} Elf32\_Shdr;

**第三步：**

按节名表在文件中的偏移和该节的长度读取节名表，并缓存在一个缓冲区中。

**第四步：**

依据节头表在文件中的偏移、节头表的表项总数、以及节头表每个表项长度搜索节头表。对每个节头表项，读出节名索引号，由节名索引号从上面缓存在缓冲区中节名表得出该节头表项对应的节的名字。如果名字为".dynsym"，记录该节的在文件中的偏移和字节长度。名字为 ".dynsym" 的节的内容是符号表，除了记录它在文件中的偏移和字节长度外，还要记下它的每个表项的长度。每个表项即是对一个符号所记录的信息，表项的数据结构为：

typedef struct  
{  
Elf32\_Word st\_name; /\* 符号名索引号 \*/  
Elf32\_Addr st\_value; /\* 符号地址值 \*/  
Elf32\_Word st\_size; /\* 符号对应的代码长度 \*/  
unsigned char st\_info; /\* 符号类型和梆定信息 \*/  
unsigned char st\_other; /\* 未用，值为 0 \*/  
Elf32\_Section st\_shndx; /\* 所在节的节索引号 \*/  
} Elf32\_Sym;

在搜索节头表时，除了找 ".dynsym" 节外，还要找到名字为 ".dynstr" 的节，记下它的在文件中的偏移和字节长度。由该结在文件中的偏移和字节长度读取它的内容，并缓存在一个缓冲区中。

**第五步：**

按第四步中得到的.dynsym节的字节长度和符号表的表项的长度算出符号表表项数目，也即符号的数目。然后依据第四步中得到的符号表(即.dynsym节)在文件中的偏移把文件指针打到符号表所在的位置，检索符号表找到要修改的符号。方法是从符号表表项中读出符号名索引号(st\_name)的值，这个值即是该表项记录其信息的符号的符号名字符串在.dynstr节中的偏移，由这个值在第四步中缓存 .dynstr 节的缓冲区中取出符号名，把符号名和要找的符号的符号名进行比较。

**第六步：**

第五步中找到了要修改的符号，现在可以进行修改了。所谓修改符号就是修改该符号在符号表（.dynsym 节）中表项，因为表项的内容是对该符号的有关信息的记录。hook 时需要关注的是符号地址值(st\_value)和符号对应的代码长度(st\_size)。可以把符号地址值(st\_value)和符号对应的代码长度(st\_size)修改为动态库中另一符号的相应的st\_value和st\_size值。通常修改的是函数符号。如果是函数符号，那么修改后当调用该函数时，实际调用的是上面修改时取其 st\_value 和 st\_size 值的另一符号所对应的函数。也可以向ELF文件中加入的一段几十字节的 shellcode 或其他代码，修改符号表时把所修改的符号在符号表中的表项的 st\_value 值指向这段 shellcode 或其他代码，st\_size 的值置为这段代码的字节长度。那么，程序调用被修改的符号所对应的函数时，实际调用的是这段 shellcode 或其他代码。

**三、示例程序**

为对上面所说的进行解释，下面的给出一个示例程序。它打印出 ELF 文件的有关信息，然后通过修改符号表把动态库的函数1 hook 到函数2。这样做以后，如果某程序编译时与该动态库连接了，它调用函数1时，程序运行时实际调用的是函数2。函数2可以是动态库本身就有的函数，也可以是你向 ELF 文件中偷偷加入的一段几十字节的 shellcode。这是向系统置入后门的一个方法，特别是 hook 经常被调用的动态库。

测试时做了一个小动态库，它仅有两个函数 haha() 和 huhu():  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
/\* haha.c \*/

#include

void haha(void)  
{  
printf(" --- haha\n");  
return;  
}  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
/\* huhu.c \*/

#include

void huhu(void)  
{  
printf(" --- huhu\n");  
return;  
}  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

$ gcc -c -fPIC -O3 haha.c huhu.c  
$ gcc -shared haha.o huhu.o -o libtst.so.1.0  
$ ln -s libtst.so.1.0 libtst.so

程序 m.c 调用 huhu() 和 haha():  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
/\* m.c \*/  
int main()  
{  
haha();  
huhu();  
return 0;  
}  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

$ gcc m.c -L. -ltst -o ttt  
$ gcc -O3 hook\_elf.c -o elf\_hook  
$ ./ttt  
--- haha  
--- huhu  
$ ./elf\_hook libtst.so huhu haha  
.  
.  
.  
$ ./ttt  
--- haha  
--- haha  
$

下面是 hook\_elf.c 程序：  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
/\*  
\* C Program File: hook\_elf.c ---  
\*  
\* Description: This program read and print relevant information of ELF  
\* File, then hook function fun1 to fun2. After hooking, when  
\* some program call function fun1, actually it is fun2 being  
\* called.  
\* Usage:  
\* hook\_elf   
\* (Note: when dst\_sym == src\_sym, ELF file is not changed.)  
\*  
\* Author: wangdb (wangdb@nsfocus.com)  
\*/

#include   
#include   
#include   
#include   
#include   
#include   
#include

#define EI\_NIDENT (16)

typedef uint16\_t Elf32\_Half;  
typedef uint32\_t Elf32\_Word;  
typedef uint32\_t Elf32\_Addr;  
typedef uint32\_t Elf32\_Off;  
typedef uint16\_t Elf32\_Section;

/\* 下面的数据结构定义取自 elf.h 头文件 \*/

/\* The ELF file header. This appears at the start of every ELF file. \*/

typedef struct  
{  
unsigned char e\_ident; /\* Magic number and other info \*/  
Elf32\_Half e\_type; /\* Object file type \*/  
Elf32\_Half e\_machine; /\* Architecture \*/  
Elf32\_Word e\_version; /\* Object file version \*/  
Elf32\_Addr e\_entry; /\* Entry point virtual address \*/  
Elf32\_Off e\_phoff; /\* Program header table file offset \*/  
Elf32\_Off e\_shoff; /\* Section header table file offset \*/  
Elf32\_Word e\_flags; /\* Processor-specific flags \*/  
Elf32\_Half e\_ehsize; /\* ELF header size in bytes \*/  
Elf32\_Half e\_phentsize; /\* Program header table entry size \*/  
Elf32\_Half e\_phnum; /\* Program header table entry count \*/  
Elf32\_Half e\_shentsize; /\* Section header table entry size \*/  
Elf32\_Half e\_shnum; /\* Section header table entry count \*/  
Elf32\_Half e\_shstrndx; /\* Section header string table index \*/  
} myElf32\_Ehdr;

/\* Program segment header. \*/

typedef struct  
{  
Elf32\_Word p\_type; /\* Segment type \*/  
Elf32\_Off p\_offset; /\* Segment file offset \*/  
Elf32\_Addr p\_vaddr; /\* Segment virtual address \*/  
Elf32\_Addr p\_paddr; /\* Segment physical address \*/  
Elf32\_Word p\_filesz; /\* Segment size in file \*/  
Elf32\_Word p\_memsz; /\* Segment size in memory \*/  
Elf32\_Word p\_flags; /\* Segment flags \*/  
Elf32\_Word p\_align; /\* Segment alignment \*/  
} myElf32\_Phdr;

/\* Section header. \*/

typedef struct  
{  
Elf32\_Word sh\_name; /\* Section name (string tbl index) \*/  
Elf32\_Word sh\_type; /\* Section type \*/  
Elf32\_Word sh\_flags; /\* Section flags \*/  
Elf32\_Addr sh\_addr; /\* Section virtual addr at execution \*/  
Elf32\_Off sh\_offset; /\* Section file offset \*/  
Elf32\_Word sh\_size; /\* Section size in bytes \*/  
Elf32\_Word sh\_link; /\* Link to another section \*/  
Elf32\_Word sh\_info; /\* Additional section information \*/  
Elf32\_Word sh\_addralign; /\* Section alignment \*/  
Elf32\_Word sh\_entsize; /\* Entry size if section holds table \*/  
} myElf32\_Shdr;

/\* Symbol table entry. \*/

typedef struct  
{  
Elf32\_Word st\_name; /\* Symbol name (string tbl index) \*/  
Elf32\_Addr st\_value; /\* Symbol value \*/  
Elf32\_Word st\_size; /\* Symbol size \*/  
unsigned char st\_info; /\* Symbol type and binding \*/  
unsigned char st\_other; /\* No defined meaning, 0 \*/  
Elf32\_Section st\_shndx; /\* Section index \*/  
} myElf32\_Sym;

/\* The syminfo section if available contains additional information about  
every dynamic symbol. \*/

typedef struct  
{  
Elf32\_Half si\_boundto; /\* Direct bindings, symbol bound to \*/  
Elf32\_Half si\_flags; /\* Per symbol flags \*/  
} myElf32\_Syminfo;

/\* Main routine \*/

int main(int argc, char \*argv)  
{  
myElf32\_Ehdr \*e\_hdr\_ptr;  
myElf32\_Phdr \*p\_hdr\_ptr;  
myElf32\_Shdr \*s\_hdr\_ptr;

myElf32\_Sym \*symptr;  
myElf32\_Syminfo \*HashSymPtr;

int fd, i;  
unsigned char buf;

unsigned int ProHdrFileOffset;  
unsigned int SecHdrFileOffset;  
unsigned int NamStrSecTblIndex;  
unsigned int ProHdrTblEntrNum;  
unsigned int SecHdrTblEntrNum;  
unsigned int ProHdrTblEntrSize;  
unsigned int SecHdrTblEntrSize;

unsigned int SecNamStrTblFileOffset = 0;  
char SecNameStrTable;  
unsigned int SecNameIndex = 0;

unsigned char SymTblEntry;

unsigned int DebugInfoFileOffset = 0;  
int DebugInfoSymTblNum = 0;  
unsigned int DebugInfoStrTblFileOffset = 0;  
char DebugInfoStrTable;  
unsigned int DebugInfoStrTblSize = 0;

unsigned int SymTblFileOffset = 0;  
int SymTblNum = 0;  
unsigned int SymNamStrTblFileOffset = 0;  
char SymNamStrTable;  
unsigned int SymNamStrTblSize = 0;

unsigned int HashOffset = 0;  
int HashTblNum = 0;

unsigned char src\_sym, dst\_sym;  
unsigned char tmp\_sym\_addr;  
unsigned char tmp\_sym\_size;  
unsigned int src\_sym\_tbl = 0, dst\_sym\_tbl = 0;

if (argc < 4) {  
fprintf(stderr, "Usage: %s \n", argv);  
exit(1);  
}

if ( (fd = open(argv, O\_RDONLY)) == -1 ) {  
fprintf(stderr, "Can't open file \"%s\".\n", argv);  
exit(1);  
}

fprintf(stdout, "Dump content of the ELF file '%s'\n", argv);

fprintf(stdout, "Part I: ELF File Header...\n");

/\* 读取 ELF 文件头 \*/  
if ( read(fd, buf, 52) != 52 ) {  
fprintf(stderr, "read error\n");  
close(fd); exit(1);  
}

e\_hdr\_ptr = (myElf32\_Ehdr \*)buf;

fprintf(stdout, "(Magic number and other info)e\_ident: %s\n",  
e\_hdr\_ptr->e\_ident);  
fprintf(stdout, "(Object file type)e\_type: 0xX\n",  
e\_hdr\_ptr->e\_type);  
fprintf(stdout, "(Architecture)e\_machine: 0xX\n",  
e\_hdr\_ptr->e\_machine);  
fprintf(stdout, "(Object file version)e\_version: 0xX\n",  
e\_hdr\_ptr->e\_version);  
fprintf(stdout, "(Entry point virtual address)e\_entry: 0xX\n",  
e\_hdr\_ptr->e\_entry);  
fprintf(stdout, "(Program header table file offset)e\_phoff: 0xX\n",  
e\_hdr\_ptr->e\_phoff);  
fprintf(stdout, "(Section header table file offset)e\_shoff: 0xX\n",  
e\_hdr\_ptr->e\_shoff);  
fprintf(stdout, "(Processor-specific flags)e\_flags: 0xX\n",  
e\_hdr\_ptr->e\_flags);  
fprintf(stdout, "(ELF header size in bytes)e\_ehsize: 0xX\n",  
e\_hdr\_ptr->e\_ehsize);  
fprintf(stdout, "(Program header table entry size)e\_phentsize: 0xX\n",  
e\_hdr\_ptr->e\_phentsize);  
fprintf(stdout, "(Program header table entry count)e\_phnum: 0xX\n",  
e\_hdr\_ptr->e\_phnum);  
fprintf(stdout, "(Section header table entry size)e\_shentsize: 0xX\n",  
e\_hdr\_ptr->e\_shentsize);  
fprintf(stdout, "(Section header table entry count)e\_shnum: 0xX\n",  
e\_hdr\_ptr->e\_shnum);  
fprintf(stdout, "(Section header string table index)e\_shstrndx: 0x%X\n",  
e\_hdr\_ptr->e\_shstrndx);

/\* 记下程序头表在文件中的偏移、节头表在文件中的偏移、  
节名表所在的节的索引序号、程序头表表项字节长度、程序头表表项数目、  
节头表表项字节长度、节头表表项数目。\*/  
ProHdrFileOffset = (unsigned int)e\_hdr\_ptr->e\_phoff;  
SecHdrFileOffset = (unsigned int)e\_hdr\_ptr->e\_shoff;  
**NamStrSecTblIndex = (unsigned int)e\_hdr\_ptr->e\_shstrndx;**  
ProHdrTblEntrNum = (unsigned int)e\_hdr\_ptr->e\_phnum;  
SecHdrTblEntrNum = (unsigned int)e\_hdr\_ptr->e\_shnum;  
ProHdrTblEntrSize = (unsigned int)e\_hdr\_ptr->e\_phentsize;  
SecHdrTblEntrSize = (unsigned int)e\_hdr\_ptr->e\_shentsize;

fprintf(stdout, "Part II: Program Header Table...\n");

if ( lseek(fd, (off\_t)ProHdrFileOffset, SEEK\_SET) != ProHdrFileOffset ) {  
fprintf(stderr, "lseek to program header error.\n");  
close(fd); exit(1);  
}

for (i = 0; i < (int)ProHdrTblEntrNum; i )

{

if ( read(fd, buf, (size\_t)ProHdrTblEntrSize) !=

(ssize\_t)ProHdrTblEntrSize )

{

fprintf(stderr, "read error\n");

close(fd);

exit(1);

}

fprintf(stdout, "Program Header Entry for Segment %d:\n", i 1);

p\_hdr\_ptr = (myElf32\_Phdr\*)buf;

fprintf(stdout, "(Segment type)p\_type: 0xX\n",

p\_hdr\_ptr->p\_type);

fprintf(stdout, "(Segment flags)p\_flags: 0xX\n",

p\_hdr\_ptr->p\_flags);

fprintf(stdout, "(Segment file offset)p\_offset: 0xX\n",

p\_hdr\_ptr->p\_offset);

fprintf(stdout, "(Segment virtual address)p\_vaddr: 0xX\n",

p\_hdr\_ptr->p\_vaddr);

fprintf(stdout, "(Segment physical address)p\_paddr: 0xX\n",

p\_hdr\_ptr->p\_paddr);

fprintf(stdout, "(Segment size in file)p\_filesz: 0xX\n",

p\_hdr\_ptr->p\_filesz);

fprintf(stdout, "(Segment size in memory)p\_memsz: 0xX\n",

p\_hdr\_ptr->p\_memsz);

fprintf(stdout, "(Segment alignment)p\_align: 0xX\n",

p\_hdr\_ptr->p\_align);

}

fprintf(stdout, "Part III: Section Header Table...\n");

/\* 定出节名表所在的节在节头表中对应的表项的文件偏移。\*/  
SecNamStrTblFileOffset = SecHdrFileOffset + NamStrSecTblIndex \* 40;  
if ( lseek(fd, (off\_t)SecNamStrTblFileOffset, SEEK\_SET) !=  
SecNamStrTblFileOffset || SecNamStrTblFileOffset == 0 ) {  
fprintf(stderr,  
"lseek to Section Table Entry for Section Name String Table error.\n");  
close(fd); exit(1);  
}  
if ( read(fd, buf, (size\_t)SecHdrTblEntrSize) != (ssize\_t)SecHdrTblEntrSize ) {  
fprintf(stderr, "read error\n");  
close(fd); exit(1);  
}  
s\_hdr\_ptr = (myElf32\_Shdr \*)buf;  
SecNamStrTblFileOffset = (unsigned int)s\_hdr\_ptr->sh\_offset;

/\* 读取节名表，并缓存在一个缓冲区中。\*/  
if ( lseek(fd, (off\_t)SecNamStrTblFileOffset, SEEK\_SET) !=  
SecNamStrTblFileOffset || SecNamStrTblFileOffset == 0 ) {  
fprintf(stderr, "lseek to Section Name String Table error.\n");  
close(fd); exit(1);  
}  
if ( read(fd, SecNameStrTable, (size\_t)s\_hdr\_ptr->sh\_size) !=  
(ssize\_t)s\_hdr\_ptr->sh\_size ) {  
fprintf(stderr, "read error\n");  
close(fd); exit(1);  
}

if ( lseek(fd, (off\_t)SecHdrFileOffset, SEEK\_SET) != SecHdrFileOffset ||  
SecHdrFileOffset == 0 ) {  
fprintf(stderr, "lseek to section header error.\n");  
close(fd); exit(1);  
}

/\* 记录符号表（即.dynsym节）在文件中的偏移，由它的字节长度和每个表项的  
长度算出符号表的表项数目。同时记下.dynstr节在文件中的偏移和字节长度。\*/  
for (i = 0; i < (int)SecHdrTblEntrNum; i )

{

if ( read(fd, buf, (size\_t)SecHdrTblEntrSize) !=

(ssize\_t)SecHdrTblEntrSize )

{

fprintf(stderr, "read error\n");

close(fd); exit(1);

}

s\_hdr\_ptr = (myElf32\_Shdr \*)buf;

if ( strcmp(SecNameStrTable s\_hdr\_ptr->sh\_name, ".symtab") == 0 )

{

DebugInfoFileOffset = (unsigned int)s\_hdr\_ptr->sh\_offset;

DebugInfoSymTblNum = (int)((s\_hdr\_ptr->sh\_size)/(s\_hdr\_ptr->sh\_entsize));

}

if ( strcmp(SecNameStrTable s\_hdr\_ptr->sh\_name, ".strtab") == 0 )

{

DebugInfoStrTblFileOffset = (unsigned int)s\_hdr\_ptr->sh\_offset;

DebugInfoStrTblSize = (unsigned int)s\_hdr\_ptr->sh\_size;

}

if ( strcmp(SecNameStrTable s\_hdr\_ptr->sh\_name, ".dynsym") == 0 )

{

SymTblFileOffset = (unsigned int)s\_hdr\_ptr->sh\_offset;

SymTblNum = (int)((s\_hdr\_ptr->sh\_size)/(s\_hdr\_ptr->sh\_entsize));

}

if ( strcmp(SecNameStrTable s\_hdr\_ptr->sh\_name, ".dynstr") == 0 )

{

SymNamStrTblFileOffset = (unsigned int)s\_hdr\_ptr->sh\_offset;

SymNamStrTblSize = (unsigned int)s\_hdr\_ptr->sh\_size;

}

if ( strcmp(SecNameStrTable s\_hdr\_ptr->sh\_name, ".hash") == 0 )

{

HashOffset = (unsigned int)s\_hdr\_ptr->sh\_offset;

HashTblNum = (int)((s\_hdr\_ptr->sh\_size)/(s\_hdr\_ptr->sh\_entsize));

}

}

if ( lseek(fd, (off\_t)SecHdrFileOffset, SEEK\_SET) != SecHdrFileOffset ) {  
fprintf(stderr, "lseek to section header error.\n");  
close(fd); exit(1);  
}

for (i = 0; i < (int)SecHdrTblEntrNum; i ) {  
if ( read(fd, buf, (size\_t)SecHdrTblEntrSize) !=  
(ssize\_t)SecHdrTblEntrSize ) {  
fprintf(stderr, "read error\n");  
close(fd); exit(1);  
}  
s\_hdr\_ptr = (myElf32\_Shdr \*)buf;  
fprintf(stdout, "Section %d:\n", i);  
SecNameIndex = (unsigned int)s\_hdr\_ptr->sh\_name;  
fprintf(stdout, "(Section name (string tbl index))sh\_name: 0xX -> %s\n",  
s\_hdr\_ptr->sh\_name, SecNameStrTable + SecNameIndex);  
fprintf(stdout, "(Section type)sh\_type: 0xX\n",  
s\_hdr\_ptr->sh\_type);  
fprintf(stdout, "(Section flags)sh\_flags: 0xX\n",  
s\_hdr\_ptr->sh\_flags);  
fprintf(stdout, "(Section virtual addr at execution)sh\_addr: 0xX\n",  
s\_hdr\_ptr->sh\_addr);  
fprintf(stdout, "(Section file offset)sh\_offset: 0xX\n",  
s\_hdr\_ptr->sh\_offset);  
fprintf(stdout, "(Section size in bytes)sh\_size: 0xX\n",  
s\_hdr\_ptr->sh\_size);  
fprintf(stdout, "(Link to another section)sh\_link: 0xX\n",  
s\_hdr\_ptr->sh\_link);  
fprintf(stdout, "(Additional section information)sh\_info: 0xX\n",  
s\_hdr\_ptr->sh\_info);  
fprintf(stdout, "(Section alignment)sh\_addralign: 0xX\n",  
s\_hdr\_ptr->sh\_addralign);  
fprintf(stdout, "(Entry size if section holds table)sh\_entsize: 0xX\n",  
s\_hdr\_ptr->sh\_entsize);  
}

fprintf(stdout,  
"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

if ( lseek(fd, (off\_t)DebugInfoStrTblFileOffset, SEEK\_SET) !=  
DebugInfoStrTblFileOffset || DebugInfoStrTblFileOffset == 0 ) {  
fprintf(stderr, "lseek to Debug Info String Table error.\n");  
goto next;  
}  
read(fd, DebugInfoStrTable, (size\_t)(DebugInfoStrTblSize 1));

if ( lseek(fd, (off\_t)DebugInfoFileOffset, SEEK\_SET) !=  
DebugInfoFileOffset || DebugInfoFileOffset == 0 ) {  
fprintf(stderr, "lseek to Debug Info Symbol Table error.\n");  
goto next;  
}  
for (i = 0; i < DebugInfoSymTblNum; i )

{

read(fd, SymTblEntry, (size\_t)16);

symptr = (myElf32\_Sym \*)SymTblEntry;

fprintf(stdout, "Debug Infomation -> Symbol ID: %d\n", i);

fprintf(stdout, "Symbol\_index\_and\_name: 0xX -> %s\n",

symptr->st\_name, DebugInfoStrTable symptr->st\_name);

fprintf(stdout, "Symbol\_value: 0xX\n", symptr->st\_value);

fprintf(stdout, "Symbol\_size: 0xX\n", symptr->st\_size);

fprintf(stdout, "Symbol\_type\_and\_binding: 0xX\n", symptr->st\_info);

fprintf(stdout, "Section\_index: 0xX\n", symptr->st\_shndx);

fprintf(stdout,

"--------------------------------------------------------\n");

}

fprintf(stdout,"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

next:

/\* 读取 .dynstr 节的内容，并缓存在一个缓冲区中。\*/

if ( lseek(fd, (off\_t)SymNamStrTblFileOffset, SEEK\_SET) !=SymNamStrTblFileOffset ||

SymNamStrTblFileOffset == 0 )

{

fprintf(stderr, "lseek to Dynamical symbol name string error.\n");

close(fd); exit(1);

}

read(fd, SymNamStrTable, (size\_t)(SymNamStrTblSize + 1));

if ( lseek(fd, (off\_t)SymTblFileOffset, SEEK\_SET) != SymTblFileOffset ||

SymTblFileOffset == 0 ) {  
fprintf(stderr, "lseek to Dynamical symbol Table error.\n");  
close(fd); exit(1);  
}

for (i = 0; i < SymTblNum; i ++)

{

read(fd, SymTblEntry, (size\_t)16);

symptr = (myElf32\_Sym \*)SymTblEntry;

fprintf(stdout, "Symbol ID: %d\n", i);

fprintf(stdout, "Symbol\_index\_and\_name: 0xX -> %s\n",

symptr->st\_name, SymNamStrTable symptr->st\_name);

fprintf(stdout, "Symbol\_value: 0xX\n", symptr->st\_value);

fprintf(stdout, "Symbol\_size: 0xX\n", symptr->st\_size);

fprintf(stdout, "Symbol\_type\_and\_binding: 0xX\n", symptr->st\_info);

fprintf(stdout, "Section\_index: 0xX\n", symptr->st\_shndx);

fprintf(stdout,"--------------------------------------------------------\n");

}

fprintf(stdout,

"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

if ( lseek(fd, (off\_t)HashOffset, SEEK\_SET) != HashOffset ||  
HashOffset == 0 ) {  
fprintf(stderr, "lseek to hash table error.\n");  
close(fd); exit(-1);  
}  
for (i = 0; i < HashTblNum; i ) {  
fprintf(stdout, "Hash Table ID: %d\n", i);  
read(fd, SymTblEntry, (size\_t)4);  
HashSymPtr = (myElf32\_Syminfo \*)SymTblEntry;  
fprintf(stdout, "Direct\_bindings, symbol\_bound\_to: 0xX\n",  
HashSymPtr->si\_boundto);  
fprintf(stdout, "Per\_symbol\_flags: 0xX\n", HashSymPtr->si\_flags);  
fprintf(stdout,  
"--------------------------------------------------------\n");  
}

close(fd); /\* End of Printing \*/

/\* Change symbol value, hook it \*/

if ( (fd = open(argv, O\_RDWR)) < 0 ) {  
fprintf(stderr, "Can't open object file '%s'.\n", argv);  
exit(-1);  
}

/\* 检索符号表搜索要修改的符号。\*/  
if ( lseek(fd, (off\_t)SymTblFileOffset, SEEK\_SET) != SymTblFileOffset )

{

fprintf(stderr, "lseek error.\n");

close(fd); exit(-1);

}

for (i = 0; i < SymTblNum; i ++)

{

read(fd, SymTblEntry, (size\_t)16);

symptr = (myElf32\_Sym \*)SymTblEntry;

if ( strcmp(argv, SymNamStrTable + symptr->st\_name) == 0 )

{

memcpy(dst\_sym, SymTblEntry, 16);

dst\_sym\_tbl = (unsigned int)lseek(fd, (off\_t)0, SEEK\_CUR) - 16;

}

if ( strcmp(argv, SymNamStrTable + symptr->st\_name) == 0 )

{

memcpy(src\_sym, SymTblEntry, 16);

src\_sym\_tbl = (unsigned int)lseek(fd, (off\_t)0, SEEK\_CUR) - 16;

}

}

/\* 修改符号表中要修改的符号所对应的表项的 st\_value 和 st\_size 值。\*/  
symptr = (myElf32\_Sym \*)src\_sym;  
memcpy(tmp\_sym\_addr, &symptr->st\_value, 4);  
memcpy(tmp\_sym\_size, &symptr->st\_size, 4);  
symptr = (myElf32\_Sym \*)dst\_sym;  
memcpy(&symptr->st\_value, tmp\_sym\_addr, 4);  
memcpy(&symptr->st\_size, tmp\_sym\_size, 4);

if ( dst\_sym\_tbl == 0 || src\_sym\_tbl == 0 ||  
lseek(fd, (off\_t)dst\_sym\_tbl, SEEK\_SET) != dst\_sym\_tbl ) {  
fprintf(stderr, "lseek error.\n");  
close(fd); exit(-1);  
}  
if ( write(fd, dst\_sym, (size\_t)16) != (ssize\_t)16 ) {  
fprintf(stderr, "write error\n");  
close(fd); exit(-1);  
}

close(fd);  
return 0;  
}

/\* EOF \*/