实验 4：LINK 实验

一、实验目的

通过对可重定位目标文件的分析和修改并将其链接成为可正确运行的程序，加深对理论课程中关于（ELF）目标文件的基本结构组成、程序链接过程（符号解析与重定位）等基本概念的理解，并掌握常用目标文件分析工具的使用。

二、实验内容

各阶段 phase[n].c 中包含一个 do_phase 函数完成相应阶段的具体功能，逐步修改构成目标程序“linkbomb”的各阶段二进制模块（phase[n].o 文件），使其在和主程序模块 main.o 链接后运行时实现实验指定的行为，即显示自己的学号。要求修改目标包括：二进制可重定位目标文件中的数据、符号表和重定位记录等。

（1）phase1：静态数据与 ELF 数据节

修改二进制可重定位目标文件“phase1.o”的数据节内容（不允许修改其他节），使其与 main.o 链接后运行时输出自己的学号：

$ gcc -no-pie -o linkbomb main.o phase1.o

$ ./linkbomb

学号

【实验提示】将子程序 do_phase 中调用的 printf（实为 puts）函数参数所指向的字符串内容替换为学号的 ASCII 表示。

（2）phase2：链接过程中的符号解析

修改 phase2.o 模块中的符号表节内容，并创建生成一个名为“phase2_patch.o”的二进制可重定位目标文件，使其与 main.o、phase2.o 链接后能够运行和输出（且仅输出）自己的学号：

$ gcc -c phase2_patch.c

$ gcc -no-pie -o linkbomb main.o phase2.o phase2_patch.o

$ ./linkbomb

学号

【实验提示】phase2 中定义的字符串数组未赋初值 char PHASE3_CODEBOOK[256]（注：这里数组名仅表示示意），通过反汇编阅读 phase2 程序，设计 phase2_patch.c 中同名全局字符串数组变量的初始值，并修改 phase2.o 模块中的符号表节中内容，将该符号名的类型从 COM 未初始化改为引用 UND（0）。

（3）phase3：选择控制结构中的重定位（选做）

修改二进制可重定位目标文件“phase3.o”中相应节中的重定位节内容（不允

许修改.text 节的内容），使其与 main.o 链接后能够运行输出（且仅输出）自己的学号：

$ gcc -no-pie -o linkbomb main.o phase3.o

$ ./linkbomb

学号

【实验提示】修改 .rela.rodata 节中的内容，根据每一 COOKIE 字符所对应 switch 的 case 项，将跳转表中的相应表项中的偏移量更改为指向：能够输出目标字符的 case 指令块的（在.text 节中的）起始偏移量。

三、实验过程

1. 请各位同学在 ubuntu 的浏览器输入 http://10.160.106.190/linklab.tar，下载该压缩包。

2. 使用：tar -xvf linklab.tar 解压刚下载的压缩包，检查是否有名为 linklab 目录。

3. 进入目录，检查是否存在以下文件：

main.o phase1.o phase2.o phase3.o（总共 4 个文件）。

4. 安装二进制编辑工具用于修改目标文件。

sudo apt update

sudo apt install hexedit

5. 按上面实验要求进行链接实验。

四、实验工具

（1）readelf

读取 ELF 格式二进制目标文件中的各方面信息并打印输出，如节（节名、偏移量及其中数据等）、符号表、字符串表、重定位记录等。

命令行格式：readelf <options> elf-file(s)

Options（部分）：

-a –all 等同于同时使用：-h -l -S -s -r -d -V -A -I

-h --file-header 显示 ELF 文件头

-l --program-headers 显示程序头

-S --section-headers 显示节头

-t --section-details 显示节详细信息

-s --syms 显示符号表

-r --relocs 显示重定位信息

-x --hex-dump =<number|name> 字节形式显示输出指定节的内容

例如：readelf -x .data phase1.o

-p --string-dump=<number|name> 以字符串形式显示输出指定节的内容

-R --relocated-dump =<number|name> 以重定位后的字节形式显示输出指定节内容

（2）objdump

反汇编二进制目标文件中包含的机器指令，获得对应的汇编指令供分析

objdump -d prog 输出 prog 程序的反汇编结果

objdump -d prog > prog.s 获得 prog 程序的反汇编 结果并保存于文本文件 prog.s 中

objdump -t prog 打印 prog 程序的符号表，其中包含 prog 中所有函数、全局变量的名称和存储地址。

（3）hexedit：二进制文件编辑工具

键盘输入字符，方向键左右上下移动。

Ctrl+X	保存并退出(save and exit)
Ctrl+C	不保存退出(exit without save)
Tab	十六进制/ASCII 码切换(toggle hex/ascii)
Backspace	撤销前一个字符(undo previous character)
Ctrl+U	撤销全部操作(undo all)
Ctrl+S	向下查找(search forward)
Ctrl+R	向上查找(search forward)

五、实验程序提交

1.在个人目录下创建子目录 lab4。

>>>>> 在服务器上 >>>>>

cd /home/username

mkdir lab4

4.将修改过的：phase1.o phase2.o phase3.o 和实验报告复制到子目录 lab4 中。

实验报告命名格式：学号姓名 lab4.docx

六、通知

1.截止日期：2024.6.30

2.当实验室提交日期到期时，我们将从子目录中取走符合命名格式的作业，延迟

提交或不符合命名格式的作业将不会被取走。

3.这个运行于 10.160.106.190 的实验室系统在我们的校园内。如果你在我们的外面校园，请使用南邮 VPN 客户端程序 ENWAgent。

七、附录

在linux-4.10/include/uapi/linux/elf.h

```
7  /* 32-bit ELF base types. */
8  typedef __u32    Elf32_Addr;
9  typedef __u16    Elf32_Half;
10 typedef __u32    Elf32_Off;
11 typedef __s32    Elf32_Sword;
12 typedef __u32    Elf32_Word;
13
14 /* 64-bit ELF base types. */
15 typedef __u64    Elf64_Addr;
16 typedef __u16    Elf64_Half;
17 typedef __s16    Elf64_SHalf;
18 typedef __u64    Elf64_Off;
19 typedef __s32    Elf64_Sword;
20 typedef __u32    Elf64_Word;
21 typedef __u64    Elf64_Xword;
22 typedef __s64    Elf64_Sxword;
```

.symtab 节记录符号表信息，是一个结构数组

- 符号表（symtab）中每个条目的结构如下：

函数名在text节中
变量名在data节或bss节中

```
ypedef  struct {
    int    name;   /*符号对应字符串在strtab节中的偏移量*/
   char  type: 4, /*符号对应目标的类型：数据、函数、源文件、节*/
          binding: 4; /*符号类别：全局符号、局部符号*/
    char  reserved;
    short  section;  /*符号对应目标所在的节，或其他情况*/
    long   value;   /*在对应节中的偏移量，可执行文件中是虚拟地址*/
    long   size;    /*符号对应目标所占字节数*/
Elf64_Symbol;
```

函数大小或变量长度

其他情况：ABS表示不该被重定位；UND(0)表示未定义；COM表示未初始化数据（.bss），此时，value表示对齐要求，size给出最小大小

ELF重定位条目的结构如下：

```
typedef struct
{ long offset; // 需要被修改的节偏移
  int type; // 重定位类型
  int symbol; // 被修改引用指向的符号
  long addend; // 重定位使用的调整偏移值
} Elf64_Rela;
```