華中科技大學

课 程 实 验 报 告

课程名称： 计算机系统基础

实验名称： ELF 文件与程序链接

院　　系：计算机科学与技术

专业班级：计算机 202201

学　　号：U202215357

姓　　名：王文涛

指导教师：朱 虹

2024 年 5 月 12 日

一、实验目的与要求

通过修改给定的可重定位的目标文件（链接炸弹），加深对可重定位目标文件格式、目标文件的生成、以及链接的理论知识的理解。

实验环境：Ubuntu

工具：GCC、GDB、readelf、hexdump、hexedit、od 等。

二、实验内容

任务　　链接炸弹的拆除

在二进制层面，逐步修改构成目标程序“linkbomb”的多个二进制模块（“.o 文件”），然后链接生成可执行程序，要求可执行程序运行能得到指定的效果。修改目标包括可重定位目标文件中的数据、机器指令、重定位记录等。

1、第 1 关 数据节的修改

修改二进制可重定位目标文件 phase1.o 的数据节中的内容（不允许修改其他节），使其与 main.o 链接后，生成的执行程序，可以输出自己的学号。

2、第 2 关 简单的机器指令修改

修改二进制可重定位目标文件 phase2.o 的代码节中的内容（不允许修改其他节），使其与 main.o 链接后，生成的执行程序。在 phase_2.c 中，有一个静态函数 static void myfunc( ) ，要求在 do_phase 函数中调用 myfunc( )，显示信息 myfunc is called. Good!。

3、第 3 关 有参数的函数调用的机器指令修改

修改二进制可重定位目标文件 phase3.o 的代码节中的内容（不允许修改其他节），使其与 main.o 链接后，生成的执行程序。在 phase_3.c 中，有一个静态函数 static void myfunc(int offset) ，要求在 do_phase 函数中调用 myfunc(pos)，将 do_phase 的参数 pos 直接传递 myfunc，显示相应的信息。

4、第 4 关 有局部变量的机器指令修改

修改二进制可重定位目标文件 phase4.o 的代码节中的内容（不允许修改其他节），使其与 main.o 链接后，生成的执行程序。在 phase_4.c 中，有一个静态函数 static void myfunc(char *s) ，要求在 do_phase 函数中调用 myfunc(s)，显示出自己的学号。

5、第 5 关 重定位表的修改

修改二进制可重定位目标文件 phase5.o 的重定位节中的内容（不允许修改代码节和数据节），使其与 main.o 链接后，生成的执行程序运行时，显示 Class Name : Computer Foundation. Teacher Name : Zhu Hong。

6、第 6 关 强弱符号

不准修改 main.c 和 phase6.o，通过增补一个文件，使得程序链接后，能够输出自己的学号。

```
#gcc  -no-pie  -o  linkbomb6   main.o  phase6.o phase6_patch.o
```

7、第 7 关 只读数据节的修改

修改 phase7.o 中只读数据节（不准修改代码节），使其与 main.o 链接后，能够输出自己的学号。

三、实验记录及问题回答

1. 第一关

首先将 main.c 中的 void (*phase)(int　i)改成 extern void (*phase)(int　i)（否则链接时会报错），接着使用 gcc -c -g -m32 main.c　-o main.o 生成 32 位的 main.o，接着使用 gcc -no-pie -m32 -o　linkbomb1 main.o　phase1.o 链接生成可执行程序 linkbomb1。运行程序，得到一段字符串。

```
f@f-virtual-machine:~/Desktop/linkbomb$ ./linkbomb1
please input you stuid : U202215357
your ID is : mnopqrstuvwxyz0123456789
Bye Bye !
```

使用 readelf -x .data phase1.o 查看数据段，找到字符串所在地点

```
f@f-virtual-machine:~/Desktop/linkbomb$ readelf -x .data phase1.o

Hex dump of section '.data':
  0x00000000 61626364 65666768 696a6b6c 6d6e6f70 abcdefghijklmnop
  0x00000010 71727374 75767778 797a3031 32333435 qrstuvwxyz012345
  0x00000020 36373839 00000000                   6789....
```

使用 hexedit 将数据段中对应的字符串改为学号。

```
.ELF.............................h.......4.....(.............
U..S.............M............R......R............]........
........abcdefghijklU202215357.xyz0123456789........your ID
```

再次运行程序，结果正确。

```
f@f-virtual-machine:~/Desktop/linkbomb$ ./linkbomb1
please input you stuid : U202215357
your ID is : U202215357
Bye Bye !
```

2. 第二关

首先查看 section headers，text 偏移量为 3c。

```
Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .group            GROUP           00000000 000034 000008 04     23  11  4
  [ 2] .text             PROGBITS        00000000 00003c 00004d 00  AX  0   0  1
  [ 3] .rel.text         REL             00000000 0005e8 000030 08   I 23   2  4
  [ 4] .data             PROGBITS        00000000 000089 000000 00  WA  0   0  1
```

查看 symbol table，myfunc 在开头，所以 myfunc 在 phase2.o 中的位置是 3c。

```
Symbol table '.symtab' contains 16 entries:
   Num:    Value  Size Type    Bind   Vis      Ndx Name
     0: 00000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 00000000     0 FILE    LOCAL  DEFAULT  ABS phase2.c
     2: 00000000     0 SECTION LOCAL  DEFAULT    2 .text
     3: 00000000     0 SECTION LOCAL  DEFAULT    6 .rodata
     4: 00000000    43 FUNC    LOCAL  DEFAULT    2 myfunc
     5: 00000000     0 SECTION LOCAL  DEFAULT    9 .text.__x86.get_[...]
     6: 00000000     0 SECTION LOCAL  DEFAULT   10 .debug_info
     7: 00000000     0 SECTION LOCAL  DEFAULT   12 .debug_abbrev
     8: 00000000     0 SECTION LOCAL  DEFAULT   15 .debug_line
     9: 00000000     0 SECTION LOCAL  DEFAULT   17 .debug_str
    10: 00000000     0 SECTION LOCAL  DEFAULT   18 .debug_line_str
    11: 00000000     0 FUNC    GLOBAL HIDDEN     9 __x86.get_pc_thunk.ax
    12: 00000000     0 NOTYPE  GLOBAL DEFAULT  UND _GLOBAL_OFFSET_TABLE_
    13: 00000000     0 NOTYPE  GLOBAL DEFAULT  UND puts
    14: 00000000     4 OBJECT  GLOBAL DEFAULT    7 phase
    15: 0000002b    34 FUNC    GLOBAL DEFAULT    2 do_phase
```

反汇编 linkbomb2，得到 myfunc 地址为 0x084939d，call 语句的下一句地址为 0x08493da

```
0804939d <myfunc>:
```

```
080493c8 <do_phase>:
 80493c8:       55                      push   %ebp
 80493c9:       89 e5                   mov    %esp,%ebp
 80493cb:       e8 1a 00 00 00          call   80493ea <__x86.get_pc_thunk.ax>
 80493d0:       05 30 2c 00 00          add    $0x2c30,%eax
 80493d5:       e8 c3 ff ff ff          call   804939d <myfunc>
 80493da:       90                      nop
```

call 指令地址为 0x804939D-0x80493DA=0xFFFF FFC3

使用 hexedit 打开 phase2.o，在 do_phase 最后加上 e8 c3 ff ff ff。

```
0000003C   55 89 E5 53  83 EC 04 E8  FC FF FF FF
00000048   05 01 00 00  00 83 EC 0C  8D 90 00 00
00000054   00 00 52 89  C3 E8 FC FF  FF FF 83 C4
00000060   10 90 8B 5D  FC C9 C3 55  89 E5 E8 FC
0000006C   FF FF FF 05  01 00 00 00  E8 C3 FF FF
00000078   FF 90 90 90  90 90 90 90  90 90 90 90
```

编译运行，结果正确。

```
f@f-virtual-machine:~/Desktop/linkbomb$ ./linkbomb2
please input you stuid : U202215357
myfunc is called. Good!
Bye Bye !
```

3. 第三关

首先直接将原始的 phase3.o 与 main.o 链接，反汇编生成的 linkbomb3，发现函数将一个立即数存入了 eax 中，可知 eax 为参数 offset。

```
0804939d <myfunc>:
 804939d:       55                      push   %ebp
 804939e:       89 e5                   mov    %esp,%ebp
 80493a0:       53                      push   %ebx
 80493a1:       83 ec 04                sub    $0x4,%esp
 80493a4:       e8 47 00 00 00          call   80493f0 <__x86.get_pc_thunk.ax>
 80493a9:       05 57 2c 00 00          add    $0x2c57,%eax
 80493ae:       83 ec 08                sub    $0x8,%esp
 80493b1:       ff 75 08                push   0x8(%ebp)
 80493b4:       8d 90 13 e1 ff ff       lea    -0x1eed(%eax),%edx
 80493ba:       52                      push   %edx
 80493bb:       89 c3                   mov    %eax,%ebx
 80493bd:       e8 8e fc ff ff          call   8049050 <printf@plt>
 80493c2:       83 c4 10                add    $0x10,%esp
 80493c5:       90                      nop
 80493c6:       8b 5d fc                mov    -0x4(%ebp),%ebx
 80493c9:       c9                      leave
 80493ca:       c3                      ret

080493cb <do_phase>:
 80493cb:       55                      push   %ebp
 80493cc:       89 e5                   mov    %esp,%ebp
 80493ce:       e8 1d 00 00 00          call   80493f0 <__x86.get_pc_thunk.ax>
 80493d3:       05 2d 2c 00 00          add    $0x2c2d,%eax
 80493d8:       90                      nop
 80493d9:       90                      nop
```

使用 hexedit 将代码段后加上 push %eax 和 call myfunc

```
0000002e <do_phase>:
  2e:   55                      push   %ebp
  2f:   89 e5                   mov    %esp,%ebp
  31:   e8 fc ff ff ff          call   32 <do_phase+0x4>
  36:   05 01 00 00 00          add    $0x1,%eax
  3b:   50                      push   %eax
  3c:   e8 bf ff ff ff          call   0 <myfunc>
```

根据 call 语句地址以及 myfunc 地址，得到 call 指令偏移量为

0x804939d-0x80493dd=0xffffffc4

运行程序，结果正确。

```
f@f-virtual-machine:~/Desktop/linkbomb$ ./linkbomb3
please input you stuid : U202215357
gate 3: offset is : 134529024!
```

4. 第四关

首先对 do_phase 函数反汇编，发现函数将字符串 U202212345 存在看-0x17(%ebp)处。

所以，首先修改字符串为自己的学号 U202215357，接着在后面加上 lea -0x17(%ebp), %eax 和 push %eax 以及 call myfunc 的机器码。

```
0000002e <do_phase>:
  2e:   55                      push   %ebp
  2f:   89 e5                   mov    %esp,%ebp
  31:   83 ec 18                sub    $0x18,%esp
  34:   e8 fc ff ff ff          call   35 <do_phase+0x7>
  39:   05 01 00 00 00          add    $0x1,%eax
  3e:   65 a1 14 00 00 00       mov    %gs:0x14,%eax
  44:   89 45 f4                mov    %eax,-0xc(%ebp)
  47:   31 c0                   xor    %eax,%eax
  49:   c7 45 e9 55 32 30 32    movl   $0x32303255,-0x17(%ebp)
  50:   c7 45 ed 32 31 32 33    movl   $0x33323132,-0x13(%ebp)
  57:   66 c7 45 f1 34 35       movw   $0x3534,-0xf(%ebp)
  5d:   c6 45 f3 00             movb   $0x0,-0xd(%ebp)
  61:   90                      nop
  62:   90                      nop
  63:   90                      nop
```

算出 call 指令偏移量为 0x804939d-0x8049407=0xfffffff96

通过 hexedit 更改代码段，最后得到的 do_phase 代码如下。

```
0000002e <do_phase>:
  2e:   55                      push   %ebp
  2f:   89 e5                   mov    %esp,%ebp
  31:   83 ec 18                sub    $0x18,%esp
  34:   e8 fc ff ff ff          call   35 <do_phase+0x7>
  39:   05 01 00 00 00          add    $0x1,%eax
  3e:   65 a1 14 00 00 00       mov    %gs:0x14,%eax
  44:   89 45 f4                mov    %eax,-0xc(%ebp)
  47:   31 c0                   xor    %eax,%eax
  49:   c7 45 e9 55 32 30 32    movl   $0x32303255,-0x17(%ebp)
  50:   c7 45 ed 32 31 35 33    movl   $0x33353132,-0x13(%ebp)
  57:   66 c7 45 f1 35 37       movw   $0x3735,-0xf(%ebp)
  5d:   c6 45 f3 00             movb   $0x0,-0xd(%ebp)
  61:   8d 45 e9                lea    -0x17(%ebp),%eax
  64:   50                      push   %eax
  65:   e8 96 ff ff ff          call   0 <myfunc>
  6a:   90                      nop
```

编译运行，结果正确。

```
f@f-virtual-machine:~/Desktop/linkbomb$ ./linkbomb4
please input you stuid : U202215357
gate 4: your ID is : U202215357!
Bye Bye !
```

5. 第五关

首先使用 objdump -dr phase5.o 查看带重定位信息的 phase5.o 反汇编代码。发现函数使用了全局变量 originalclass 和 orginalteacher。

```
00000000 <do_phase>:
   0:   55                      push   %ebp
   1:   89 e5                   mov    %esp,%ebp
   3:   53                      push   %ebx
   4:   83 ec 04                sub    $0x4,%esp
   7:   e8 fc ff ff ff          call   8 <do_phase+0x8>
                        8: R_386_PC32    __x86.get_pc_thunk.bx
   c:   81 c3 02 00 00 00       add    $0x2,%ebx
                        e: R_386_GOTPC   _GLOBAL_OFFSET_TABLE_
  12:   83 ec 08                sub    $0x8,%esp
  15:   8d 83 00 00 00 00       lea    0x0(%ebx),%eax
                        17: R_386_GOTOFF        originalclass
  1b:   50                      push   %eax
  1c:   8d 83 00 00 00 00       lea    0x0(%ebx),%eax
                        1e: R_386_GOTOFF        .rodata
  22:   50                      push   %eax
  23:   e8 fc ff ff ff          call   24 <do_phase+0x24>
                        24: R_386_PLT32 printf
  28:   83 c4 10                add    $0x10,%esp
  2b:   83 ec 08                sub    $0x8,%esp
  2e:   8d 83 00 00 00 00       lea    0x0(%ebx),%eax
                        30: R_386_GOTOFF        originalteacher
  34:   50                      push   %eax
  35:   8d 83 0f 00 00 00       lea    0xf(%ebx),%eax
                        37: R_386_GOTOFF        .rodata
  3b:   50                      push   %eax
  3c:   e8 fc ff ff ff          call   3d <do_phase+0x3d>
                        3d: R_386_PLT32 printf
  41:   83 c4 10                add    $0x10,%esp
```

查看 symbol table，发现 originalclass 和 orginalteacher 分别为 12，13 号符号。而我们要使用的为 10，11 号符号。

```
Symbol table '.symtab' contains 19 entries:
   Num:    Value  Size Type    Bind   Vis      Ndx Name
     0: 00000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 00000000     0 FILE    LOCAL  DEFAULT  ABS phase5.c
     2: 00000000     0 SECTION LOCAL  DEFAULT    2 .text
     3: 00000000     0 SECTION LOCAL  DEFAULT    8 .rodata
     4: 00000000     0 SECTION LOCAL  DEFAULT    9 .text.__x86.get_[...]
     5: 00000000     0 SECTION LOCAL  DEFAULT   10 .debug_info
     6: 00000000     0 SECTION LOCAL  DEFAULT   12 .debug_abbrev
     7: 00000000     0 SECTION LOCAL  DEFAULT   15 .debug_line
     8: 00000000     0 SECTION LOCAL  DEFAULT   17 .debug_str
     9: 00000000     0 SECTION LOCAL  DEFAULT   18 .debug_line_str
    10: 00000000    20 OBJECT  GLOBAL DEFAULT    4 classname
    11: 00000014    20 OBJECT  GLOBAL DEFAULT    4 teachername
    12: 00000028    20 OBJECT  GLOBAL DEFAULT    4 originalclass
    13: 0000003c    20 OBJECT  GLOBAL DEFAULT    4 originalteacher
    14: 00000000     4 OBJECT  GLOBAL DEFAULT    6 phase
    15: 00000000    95 FUNC    GLOBAL DEFAULT    2 do_phase
    16: 00000000     0 FUNC    GLOBAL HIDDEN     9 __x86.get_pc_thunk.bx
    17: 00000000     0 NOTYPE  GLOBAL DEFAULT  UND _GLOBAL_OFFSET_TABLE_
    18: 00000000     0 NOTYPE  GLOBAL DEFAULT  UND printf
```

查看重定位节，偏移量为 0x17 处需要重定位，Info 为 0x0c09，其中 09 代表寻址方式为 R_386_GOTOFF，0c 代表用 13 号符号即 originalclass 来进行重定位。所以我们将 info 中的 0c 改为 0a 就能让原来是 originalclass 的地方变成 classname。同理，将偏移量为 0x30，Info 0x0d09 中的 0d 改为 0b 就能让 orginalteacher 变成 teachername。

```
Relocation section '.rel.text' at offset 0x764 contains 8 entries:
 Offset     Info    Type            Sym.Value  Sym. Name
00000008  00001002 R_386_PC32        00000000   __x86.get_pc_thunk.bx
0000000e  0000110a R_386_GOTPC       00000000   _GLOBAL_OFFSET_TABLE_
00000017  00000c09 R_386_GOTOFF      00000028   originalclass
0000001e  00000309 R_386_GOTOFF      00000000   .rodata
00000024  00001204 R_386_PLT32       00000000   printf
00000030  00000d09 R_386_GOTOFF      0000003c   originalteacher
00000037  00000309 R_386_GOTOFF      00000000   .rodata
0000003d  00001204 R_386_PLT32       00000000   printf
```

查看节头表，.rel.text 偏移量为 0x764。在 hexedit 中找到对应位置，修改重定位节。

```
Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .group            GROUP           00000000 000034 000008 04     23  16  4
  [ 2] .text             PROGBITS        00000000 00003c 00005f 00  AX  0   0  1
  [ 3] .rel.text         REL             00000000 000764 000040 08   I 23   2  4
```

```
00000760   66 00 00 00  08 00 00 00  02 10 00 00  0E 00 00 00
00000770   0A 11 00 00  17 00 00 00  09 0A 00 00  1E 00 00 00
00000780   09 03 00 00  24 00 00 00  04 12 00 00  30 00 00 00
00000790   09 0B 00 00  37 00 00 00  09 03 00 00  3D 00 00 00
```

重新使用 objdump -dr phase5.o 查看带重定位信息的 phase5.o 反汇编代码。发现函数重定位目标变成了 classname 和 teachername。

```
00000000 <do_phase>:
   0:   55                      push   %ebp
   1:   89 e5                   mov    %esp,%ebp
   3:   53                      push   %ebx
   4:   83 ec 04                sub    $0x4,%esp
   7:   e8 fc ff ff ff          call   8 <do_phase+0x8>
                        8: R_386_PC32    __x86.get_pc_thunk.bx
   c:   81 c3 02 00 00 00       add    $0x2,%ebx
                        e: R_386_GOTPC   _GLOBAL_OFFSET_TABLE_
  12:   83 ec 08                sub    $0x8,%esp
  15:   8d 83 00 00 00 00       lea    0x0(%ebx),%eax
                        17: R_386_GOTOFF        classname
  1b:   50                      push   %eax
  1c:   8d 83 00 00 00 00       lea    0x0(%ebx),%eax
                        1e: R_386_GOTOFF        .rodata
  22:   50                      push   %eax
  23:   e8 fc ff ff ff          call   24 <do_phase+0x24>
                        24: R_386_PLT32 printf
  28:   83 c4 10                add    $0x10,%esp
  2b:   83 ec 08                sub    $0x8,%esp
  2e:   8d 83 00 00 00 00       lea    0x0(%ebx),%eax
                        30: R_386_GOTOFF        teachername
  34:   50                      push   %eax
  35:   8d 83 0f 00 00 00       lea    0xf(%ebx),%eax
                        37: R_386_GOTOFF        .rodata
  3b:   50                      push   %eax
  3c:   e8 fc ff ff ff          call   3d <do_phase+0x3d>
                        3d: R_386_PLT32 printf
  41:   83 c4 10                add    $0x10,%esp
```

编译运行 linkbomb5，结果正确。

```
f@f-virtual-machine:~/Desktop/linkbomb$ ./linkbomb5
please input you stuid : U202215357
Class Name Computer Foundation
Teacher Name Zhu Hong
Bye Bye !
```

6. 第六关

首先将 main 函数重新编译成 64 位.o 文件，接着将 main.o 和 phase6.o 链接成 linkbomb6，使用 objdump 查看 linkbomb6 的反汇代码。

函数 do_phase 将变量 myprint 存入寄存器 rdx 中，接着 call *%rdx，这意味着 myprint 储存着一个函数的首地址。在 do_phase 中调用这个函数来打印学号。

```
0000000000401385 <do_phase>:
  401385:       f3 0f 1e fa             endbr64
  401389:       55                      push   %rbp
  40138a:       48 89 e5                mov    %rsp,%rbp
  40138d:       48 83 ec 10             sub    $0x10,%rsp
  401391:       89 7d fc                mov    %edi,-0x4(%rbp)
  401394:       48 8b 05 c5 2c 00 00    mov    0x2cc5(%rip),%rax        # 404060 <myprint>
  40139b:       48 85 c0                test   %rax,%rax
  40139e:       74 10                   je     4013b0 <do_phase+0x2b>
  4013a0:       48 8b 15 b9 2c 00 00    mov    0x2cb9(%rip),%rdx        # 404060 <myprint>
  4013a7:       b8 00 00 00 00          mov    $0x0,%eax
  4013ac:       ff d2                   call   *%rdx
  4013ae:       eb 0c                   jmp    4013bc <do_phase+0x37>
  4013b0:       48 8d 3d 60 0d 00 00    lea    0xd60(%rip),%rdi        # 402117 <_IO_stdin_used+0x117>
  4013b7:       e8 d4 fc ff ff          call   401090 <puts@plt>
  4013bc:       90                      nop
  4013bd:       c9                      leave
  4013be:       c3                      ret
```

创建 c 文件 phase6_patch.c 如下图所示，由于函数 print 在链接后会位于 do_phase 函数之后，所以将 myprint 的值设置为 0x4013bf。

```
#include<stdio.h>
long long myprint=0x4013bf;
void print()
{
        printf("U202215357\n");
}
```

将 phase6_patch.c 编译位 phase6_patch.o。接着将 main.o phase6.o phase6_patch.c 链接成 linkbomb6 并运行，结果正确。

```
f@f-virtual-machine:~/Desktop/linkbomb$ ./linkbomb6
please input you stuid : U202215357
U202215357
Bye Bye !
```

7. 第七关

首先查看 do_pahse 的带重定位信息的反汇编代码。代码使用了只读的变量.rodata

```
00000000 <do_phase>:
   0:   55                      push   %ebp
   1:   89 e5                   mov    %esp,%ebp
   3:   53                      push   %ebx
   4:   83 ec 04                sub    $0x4,%esp
   7:   e8 fc ff ff ff          call   8 <do_phase+0x8>
                        8: R_386_PC32   __x86.get_pc_thunk.ax
   c:   05 01 00 00 00          add    $0x1,%eax
                        d: R_386_GOTPC  _GLOBAL_OFFSET_TABLE_
  11:   83 ec 0c                sub    $0xc,%esp
  14:   8d 90 00 00 00 00       lea    0x0(%eax),%edx
                        16: R_386_GOTOFF        .rodata
  1a:   52                      push   %edx
  1b:   89 c3                   mov    %eax,%ebx
  1d:   e8 fc ff ff ff          call   1e <do_phase+0x1e>
                        1e: R_386_PLT32 puts
  22:   83 c4 10                add    $0x10,%esp
  25:   90                      nop
  26:   8b 5d fc                mov    -0x4(%ebp),%ebx
  29:   c9                      leave
  2a:   c3                      ret
```

在节头表中查看.rodata 的位置。发现.rodata 偏移量为 6c

```
Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .group            GROUP           00000000 000034 000008 04     23  12  4
  [ 2] .text             PROGBITS        00000000 00003c 00002b 00  AX  0   0  1
  [ 3] .rel.text         REL             00000000 00053c 000020 08   I 23   2  4
  [ 4] .data             PROGBITS        00000000 000067 000000 00  WA  0   0  1
  [ 5] .bss              NOBITS          00000000 000067 000000 00  WA  0   0  1
  [ 6] .data.rel.local   PROGBITS        00000000 000068 000004 00  WA  0   0  4
  [ 7] .rel.data.re[...] REL             00000000 00055c 000008 08   I 23   6  4
  [ 8] .rodata           PROGBITS        00000000 00006c 000013 00   A  0   0  1
```

使用 hexedit 找到对应位置，将字符串改为自己的学号。

```
0000006C   47 61 74 65  20 37 3A 20  55 32 30 32  Gate 7: U202
00000078   32 31 35 33  35 37 00 8B  04 24 C3 B5  215357...$..
```

编译运行程序，结果正确。

```
f@f-virtual-machine:~/Desktop/linkbomb$ ./linkbomb7
please input you stuid : U202215357
Gate 7: U202215357
Bye Bye !
```

四、体会

经过这次实验，我学会了如何使用 hexedit 编辑二进制文件，以及如何看懂二进制文件。通过使用 readelf 查看重定位目标文件，我对之前学习的文件链接有了更深刻的理解。并且将 elf 格式的信息与二进制文件进行对比，我也理解了节头表，数据段，代码段，重定位段等等在二进制文件中是如何存放的。并且知道了各种重定位方式，能够解读重定位信息，并且能够根据重定位信息找到需要修改的符号位置。完成实验后，我对链接的原理理解更加深刻了。