华中科技大学

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：本硕博 202301

学　　号：U202315752

姓　　名：陈宇航

指导教师：李海波

2024 年 10 月 24 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 : Xu Xiangyang。

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）实验结果及操作过程记录

第一节 数据节的修改

我们先编译 main.c 和 phase1.o 链接一下，看看什么情况：

```
francis@francis-VMware-Virtual-Platform:~/桌面/链接炸弹_学生端$ gcc -no-pie -o l
inkbomb1 main.o phase1.o
francis@francis-VMware-Virtual-Platform:~/桌面/链接炸弹_学生端$ ./linkbomb1
please input you stuid : U202315752
your ID is : hijklmnopqrstuvwxyz0123456789
Bye Bye !
```

图 3.1.1 第一关链接结果

我们的直觉告诉我们，应该是要修改 phase1.o 里面的 data 段的内容，让他输出的内容和我们的学号一样，我们下面用命令“readelf -x .data phase1.o”看看 data 段什么情况：

```
francis@francis-VMware-Virtual-Platform:~/桌面/链接炸弹_学生端$ readelf -x .data
 phase1.o

“.data”节的十六进制输出：
  0x00000000 61626364 65666768 696a6b6c 6d6e6f70 abcdefghijklmnop
  0x00000010 71727374 75767778 797a3031 32333435 qrstuvwxyz012345
  0x00000020 36373839 00000000                   6789....
```

图 3.1.2 第一关查看信息

我们发现，这里出现了和我们的输出一摸一样的内容，所以我们下面应该是要研究怎么修改这段内容。因此，我们还需要知道这段文字的具体位置。使用“readelf -S phase1.o”命令查看位置：

```
节头：
  [号] 名称              类型             地址              偏移量
       大小              全体大小          旗标    链接    信息    对齐
  [ 0]                   NULL             0000000000000000  00000000
       0000000000000000  0000000000000000           0       0     0
  [ 1] .text             PROGBITS         0000000000000000  00000040
       0000000000000035  0000000000000000  AX       0       0     1
  [ 2] .rela.text        RELA             0000000000000000  00000300
       0000000000000048  0000000000000018   I      13      1     8
  [ 3] .data             PROGBITS         0000000000000000  00000080
       0000000000000028  0000000000000000  WA       0       0     32
```

图 3.1.3 第一关 data 段偏移位置

可以看到.data 节在 phase1.o 中偏移量为 0x80，但是我们学号不一定就要放在偏移为 0x80 的位置，我们需要根据我们第一次试探的输出 ijklmn 等字符的输出判断我们实际上需要修改的数据的位置，提前计算得到学号对应的十六进制表示 55 32 30 32 33 31 35 37 35 32，将指定位置的的数据替换为我们的学号，别忘记了末尾用 00 填充。

我们输入“hexedit phase1.o”进行修改：

```
00 00  00 00 40 00  10 00 0F 00  F3 0F 1E FA  55 48 89 E5  48 83 EC 10  89 7D FC 8B  ..........@.....@.........UH.
48 89  C6 48 8D 3D  00 00 00 00  B8 00 00 00  00 E8 00 00  00 00 90 C9  C3 00 00 00  E.H.H.....H..H..H.=..........
67 55  32 30 32 33  31 35 37 35  32 00 73 74  75 76 77 78  79 7A 30 31  32 33 34 35  ........abcdefgU202315752.stuvw
00 00  79 6F 75 72  20 49 44 20  69 73 20 3A  20 25 73 0A  00 00 47 43  43 3A 20 28  6789...........your ID is : %s
75 62  75 6E 74 75  31 7E 32 30  2E 30 34 2E  32 29 20 39  2E 34 2E 30  00 00 00 00  Ubuntu 9.4.0-1ubuntu1~20.04.2)
55 00  02 00 00 C0  04 00 00 00  03 00 00 00  00 00 00 00  14 00 00 00  00 00 00 00  ............GNU.............
00 00  1C 00 00 00  1C 00 00 00  00 00 00 00  35 00 00 00  00 45 0E 10  86 02 43 0D  .zR..x.....................5..
00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  01 00 00 00  04 00 F1 FF  .l............................
00 00  00 00 00 00  03 00 01 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ............................
00 00  00 00 00 00  00 00 00 00  00 00 00 00  03 00 04 00  00 00 00 00  00 00 00 00  ............................
```

图 3.1.4 第一关修改

修改完之后用“ctrl+x”保存退出，然后删除原 linkbomb1.o 文件，重新链接，得到结果：

```
francis@francis-VMware-Virtual-Platform:~/桌面/链接炸弹_学生端$ ./linkbomb1
please input you stuid : U202315752
your ID is : U202315752
Bye Bye !
francis@francis-VMware-Virtual-Platform:~/桌面/链接炸弹_学生端$
```

图 3.1.5 第一关修改结果

第二节 简单的机器指令修改

这一阶段的目标是修改 phase2.o 的.text 节的内容，我们先试着去链接一下 main.o 和 phase2.o。

```
francis@francis-VMware-Virtual-Platform:~/桌面/链接炸弹_学生端$ gcc -no-pie -o linkbomb2 main.o phase2.o
francis@francis-VMware-Virtual-Platform:~/桌面/链接炸弹_学生端$ ./linkbomb2
please input you stuid : U202315752
Bye Bye !
francis@francis-VMware-Virtual-Platform:~/桌面/链接炸弹_学生端$
```

图 3.1.6 第二关尝试

发现啥也没输出。我们直接反汇编一下 phase2.o，发现 do_phase 函数里面并没有什么具体的实现，而且 myfun 函数的偏移是 0：

```
Disassembly of section .text:

0000000000000000 <myfunc>:
   0:   f3 0f 1e fa             endbr64
   4:   55                      push   %rbp
   5:   48 89 e5                mov    %rsp,%rbp
   8:   48 8d 3d 00 00 00 00    lea    0x0(%rip),%rdi        # f <myfunc+0xf>
   f:   e8 00 00 00 00          call   14 <myfunc+0x14>
  14:   90                      nop
  15:   5d                      pop    %rbp
  16:   c3                      ret

0000000000000017 <do_phase>:
  17:   f3 0f 1e fa             endbr64
  1b:   55                      push   %rbp
  1c:   48 89 e5                mov    %rsp,%rbp
  1f:   89 7d fc                mov    %edi,-0x4(%rbp)
  22:   90                      nop
  23:   90                      nop
  24:   90                      nop
  25:   90                      nop
```

图 3.1.7 第二关反汇编情况

由于我们采用的是 PC 相对寻址的方式，我们计算新的重定位内容并在 mov 指令后加一条 call 指令跳转到我们的 myfunc 函数，转移目标地址=PC+偏移地址，PC 为 0x22，目标地址为 0，所以重定位的值为 0-22=-22,转换为十六进制为 0xffffffd9。所以我们要去对应的.txt 节中修改这个重定位的值。

经过查询得到我们.txt 节的偏移为 0x3c。我们实际需要修改的位置是 0x3c+0x3d 处的值。

```
0000000000000017 <do_phase>:
  17:   f3 0f 1e fa             endbr64
  1b:   55                      push   %rbp
  1c:   48 89 e5                mov    %rsp,%rbp
  1f:   89 7d fc                mov    %edi,-0x4(%rbp)
  22:   e8 d9 ff ff ff          call   0 <myfunc>
```

图 3.1.8 第二关修改

修改完我们再次反汇编一下 phase2.o 代码。

```
francis@francis-VMware-Virtual-Platform:~/桌面/链接炸弹_学生端$ gcc -no-pie -o l
inkbomb2 main.o phase2.o
francis@francis-VMware-Virtual-Platform:~/桌面/链接炸弹_学生端$ ./linkbomb2
please input you stuid : U202315752
myfunc is called. Good!
Bye Bye !
francis@francis-VMware-Virtual-Platform:~/桌面/链接炸弹_学生端$
```

图 3.1.9 第二关修改结果

第三节 带参数的机器指令修改

我们先查看一下 phase3.o 里面.txt 节的位置还是在 0x22 的位置，这一关我们不仅需要在 do_phase 中调用 myfunc 函数还需要传参，经过查询，得知 do_phase 中的参数保存-0x4（%rbp）中，且在 myfunc 中也是调用-0x4（%rbp）来调用参数。

```
0000000000000000 <myfunc>:
   0:   f3 0f 1e fa             endbr64
   4:   55                      push   %rbp
   5:   48 89 e5                mov    %rsp,%rbp
   8:   48 83 ec 10             sub    $0x10,%rsp
   c:   89 7d fc                mov    %edi,-0x4(%rbp)
   f:   8b 45 fc                mov    -0x4(%rbp),%eax
  12:   89 c6                   mov    %eax,%esi
  14:   48 8d 3d 00 00 00 00    lea    0x0(%rip),%rdi        # 1b <myfunc+0x1b>
  1b:   b8 00 00 00 00          mov    $0x0,%eax
  20:   e8 00 00 00 00          call   25 <myfunc+0x25>
  25:   90                      nop
  26:   c9                      leave
  27:   c3                      ret

0000000000000028 <do_phase>:
  28:   f3 0f 1e fa             endbr64
  2c:   55                      push   %rbp
  2d:   48 89 e5                mov    %rsp,%rbp
  30:   89 7d fc                mov    %edi,-0x4(%rbp)
```

图 3.1.10 第三关反汇编

因此，我们这么补充代码：

```
0000000000000000 <myfunc>:
   0:   f3 0f 1e fa             endbr64
   4:   55                      push   %rbp
   5:   48 89 e5                mov    %rsp,%rbp
   8:   48 83 ec 10             sub    $0x10,%rsp
   c:   89 7d fc                mov    %edi,-0x4(%rbp)
   f:   8b 45 fc                mov    -0x4(%rbp),%eax
  12:   89 c6                   mov    %eax,%esi
  14:   48 8d 3d 00 00 00 00    lea    0x0(%rip),%rdi        # 1b <myfunc+0x1b>
  1b:   b8 00 00 00 00          mov    $0x0,%eax
  20:   e8 00 00 00 00          call   25 <myfunc+0x25>
  25:   90                      nop
  26:   c9                      leave
  27:   c3                      ret

0000000000000028 <do_phase>:
  28:   f3 0f 1e fa             endbr64
  2c:   55                      push   %rbp
  2d:   48 89 e5                mov    %rsp,%rbp
  30:   89 7d fc                mov    %edi,-0x4(%rbp)
  33:   8b 7d fc                mov    -0x4(%rbp),%edi
  36:   e8 d9 ff ff ff          call   14 <myfunc+0x14>
```

图 3.1.11 第三关修改

修改成功：

```
  4a:   c3                      ret
francis@francis-VMware-Virtual-Platform:~/桌面/链接炸弹_学生端$ gcc -no-pie -o linkbomb3 main.o phase3.o
francis@francis-VMware-Virtual-Platform:~/桌面/链接炸弹_学生端$ ./linkbomb3
please input you stuid : U202315752
gate 3: offset is : 2!
Bye Bye !
```

图 3.1.12 第三关修改结果

第四节 带局部变量的机器指令修改

我们先查看一下 phase4.o 是什么情况：

```
2b:   f3 0f 1e fa             endbr64
2f:   55                      push   %rbp
30:   48 89 e5                mov    %rsp,%rbp
33:   48 83 ec 30             sub    $0x30,%rsp
37:   89 7d dc                mov    %edi,-0x24(%rbp)
3a:   64 48 8b 04 25 28 00    mov    %fs:0x28,%rax
41:   00 00
43:   48 89 45 f8             mov    %rax,-0x8(%rbp)
47:   31 c0                   xor    %eax,%eax
49:   48 b8 55 32 30 32 32    movabs $0x3332313232303255,%rax
50:   31 32 33
53:   48 89 45 ed             mov    %rax,-0x13(%rbp)
57:   66 c7 45 f5 34 35       movw   $0x3534,-0xb(%rbp)
5d:   c6 45 f7 00             movb   $0x0,-0x9(%rbp)
61:   90                      nop
```

图 3.1.13 第四关反汇编

这一关我们需要修改局部变量并把这个局部变量传入 myfunc 函数，查看 do_phase 函数，发现了局部变量存储的是一个字符串，里面是我们的学号，我们需要修改为自己的学号并调用函数输出。

那么，我们的学号存在哪里呢？他又会怎么传过去呢？我们可以很清晰，很明显的可以看出来，我们的学号在 rax 里面，又传给 rbp 的-0x13 的位置，所以 rbp-0x13 的位置是我们字符串学号的首地址。而在 myfun 函数里面，我们注意到，会把 rdi 的值给到 rax 里面，然后有给到 rsi 进行输出，所以，我们的学号给了 rdi 进行储存，因此，我们的汇编代码可以这么写：

```
61:   48 89 e8              mov    %rbp,%rax
64:   48 83 e8 13           sub    $0x13,%rax
68:   48 89 c7              mov    %rax,%rdi
6b:   e8 90 ff ff ff        call   0 <myfunc>
70:   90                    nop
```

图 3.1.14 第四关代码撰写

分析一下为什么这么写，第一步是把 rbp 里面的内容，也就是一个地址传给 rax，因为我们前面知道 rbp 里面的地址减去 0x13 就是我们输入字符串的首地址，再把这个值传给 rdi，完成了我们的道路。下面看看结果：

```
francis@francis-VMware-Virtual-Platform:~/桌面/链接炸弹_学生端$ gcc -no-pie -o l
inkbomb4 main.o phase4.o
francis@francis-VMware-Virtual-Platform:~/桌面/链接炸弹_学生端$ ./linkbomb4
please input you stuid : U202315752
gate 4: your ID is : U202315752!
Bye Bye !
francis@francis-VMware-Virtual-Platform:~/桌面/链接炸弹_学生端$
```

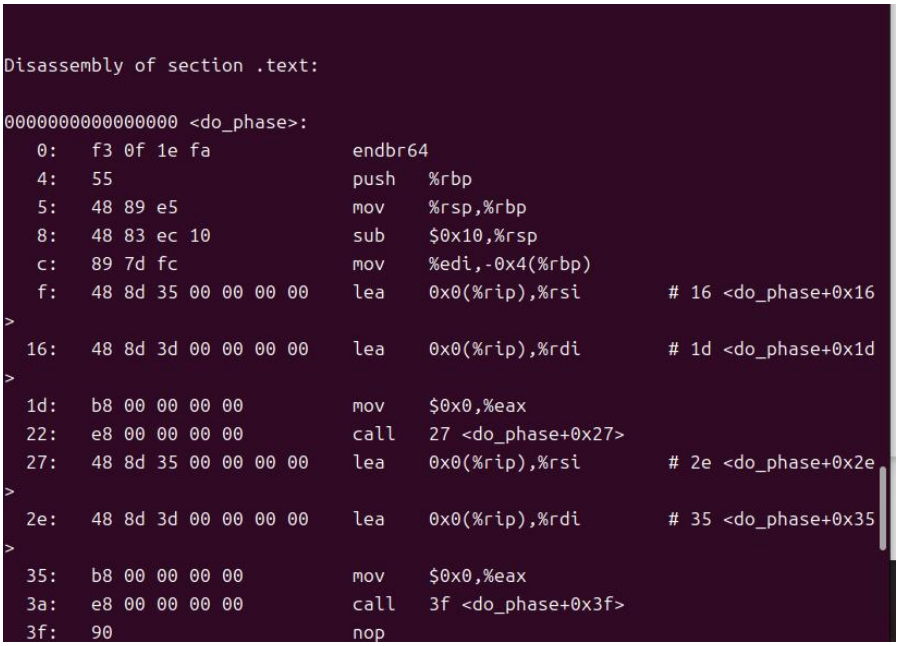
图 3.1.15 第四关结果

第五关 重定位表的修改

老样子，先试试运行一次，反汇编一次看看什么情况：

```
francis@francis-VMware-Virtual-Platform:~/桌面/链接炸弹_学生端$ gcc -no-pie -o l
inkbomb5 main.o phase5.o
francis@francis-VMware-Virtual-Platform:~/桌面/链接炸弹_学生端$ ./linkbomb5
please input you stuid : U202315752
Class Name c programming
Teacher Name ma
Bye Bye !
francis@francis-VMware-Virtual-Platform:~/桌面/链接炸弹_学生端$
```

图 3.1.16 第五关尝试



```
Disassembly of section .text:

0000000000000000 <do_phase>:
   0:   f3 0f 1e fa             endbr64
   4:   55                      push   %rbp
   5:   48 89 e5                mov    %rsp,%rbp
   8:   48 83 ec 10             sub    $0x10,%rsp
   c:   89 7d fc                mov    %edi,-0x4(%rbp)
   f:   48 8d 35 00 00 00 00    lea    0x0(%rip),%rsi        # 16 <do_phase+0x16
>
  16:   48 8d 3d 00 00 00 00    lea    0x0(%rip),%rdi        # 1d <do_phase+0x1d
>
  1d:   b8 00 00 00 00          mov    $0x0,%eax
  22:   e8 00 00 00 00          call   27 <do_phase+0x27>
  27:   48 8d 35 00 00 00 00    lea    0x0(%rip),%rsi        # 2e <do_phase+0x2e
>
  2e:   48 8d 3d 00 00 00 00    lea    0x0(%rip),%rdi        # 35 <do_phase+0x35
>
  35:   b8 00 00 00 00          mov    $0x0,%eax
  3a:   e8 00 00 00 00          call   3f <do_phase+0x3f>
  3f:   90                      nop
```

图 3.1.17 第五关反汇编

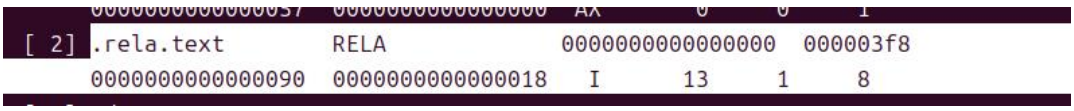
看不懂，没头绪，看看重定位节什么情况：

```
重定位节 '.rela.text' at offset 0x3f8 contains 6 entries:
  偏移量          信息           类型           符号值        符号名称 + 加数
000000000012  000d00000002 R_X86_64_PC32     0000000000000040 originalclass - 4
000000000019  000600000002 R_X86_64_PC32     0000000000000000 .rodata - 4
000000000023  001200000004 R_X86_64_PLT32    0000000000000000 printf - 4
00000000002a  000e00000002 R_X86_64_PC32     0000000000000060 originalteacher - 4
000000000031  000600000002 R_X86_64_PC32     0000000000000000 .rodata + b
00000000003b  001200000004 R_X86_64_PLT32    0000000000000000 printf - 4

重定位节 '.rela.data.rel.local' at offset 0x488 contains 1 entry:
  偏移量          信息           类型           符号值        符号名称 + 加数
000000000000  001000000001 R_X86_64_64       0000000000000000 do_phase + 0

重定位节 '.rela.eh_frame' at offset 0x4a0 contains 1 entry:
  偏移量          信息           类型           符号值        符号名称 + 加数
000000000020  000200000002 R_X86_64_PC32     0000000000000000 .text + 0
francis@francis-VMware-Virtual-Platform:~/桌面/链接炸弹_学生端$
```

图 3.1.18 第五关重定位节



```
0000000000000037  0000000000000000  AX       0     0     1
[ 2] .rela.text        RELA            0000000000000000  000003f8
     0000000000000090  0000000000000018   I      13     1     8
```

图 3.1.18 第五关偏移

但是这里并没有我们希望出现的有关名字和班级的符号，通过查阅知道，这些东西的查看要用符号表头，用“readelf -S --syms phase5.o”查看：

```
Symbol table '.symtab' contains 19 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
     0: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS phase5.c
     2: 0000000000000000     0 SECTION LOCAL  DEFAULT    1 .text
     3: 0000000000000000     0 SECTION LOCAL  DEFAULT    3 .data
     4: 0000000000000000     0 SECTION LOCAL  DEFAULT    4 .bss
     5: 0000000000000000     0 SECTION LOCAL  DEFAULT    5 .data.rel.local
     6: 0000000000000000     0 SECTION LOCAL  DEFAULT    7 .rodata
     7: 0000000000000000     0 SECTION LOCAL  DEFAULT    9 .note.GNU-stack
     8: 0000000000000000     0 SECTION LOCAL  DEFAULT   10 .note.gnu.property
     9: 0000000000000000     0 SECTION LOCAL  DEFAULT   11 .eh_frame
    10: 0000000000000000     0 SECTION LOCAL  DEFAULT    8 .comment
    11: 0000000000000000    20 OBJECT  GLOBAL DEFAULT    3 classname
    12: 0000000000000020    20 OBJECT  GLOBAL DEFAULT    3 teachername
    13: 0000000000000040    20 OBJECT  GLOBAL DEFAULT    3 originalclass
    14: 0000000000000060    20 OBJECT  GLOBAL DEFAULT    3 originalteacher
    15: 0000000000000000     8 OBJECT  GLOBAL DEFAULT    5 phase
    16: 0000000000000000    87 FUNC    GLOBAL DEFAULT    1 do_phase
    17: 0000000000000000     0 NOTYPE  GLOBAL DEFAULT  UND _GLOBAL_OFFSET_TABLE_
    18: 0000000000000000     0 NOTYPE  GLOBAL DEFAULT  UND printf
```

图 3.1.19 第五关符号节点表

根据上图信息知，originalclass 的地址减 0x40 即为 classname 的地址，originalteacher 的地址减 0x40 即为 teachername 的地址。我们修改.rela.text 节，该节从 0x3f8 开始。我们只需将 originalclass 和 originalteacher 的辅助偏移量 Addend 从-0x4 改为-0x44，即将 FFFF FFFF FFFF FFFC 改为 FFFF FFFF FFFF FFBC，那么重定位的时候就会将地址指向 classname 和 teachername：

```
000003F0   6E 74 66 00   00 00 00 00   12 00 00 00   00 00 00 00   n
00000400   02 00 00 00   0D 00 00 00   BC FF FF FF   FF FF FF FF   .
00000410   19 00 00 00   00 00 00 00   02 00 00 00   06 00 00 00   .
00000420   FC FF FF FF   FF FF FF FF   23 00 00 00   00 00 00 00   .
00000430   04 00 00 00   12 00 00 00   FC FF FF FF   FF FF FF FF   .
00000440   2A 00 00 00   00 00 00 00   02 00 00 00   0E 00 00 00   *
00000450   BC FF FF FF   FF FF FF FF   31 00 00 00   00 00 00 00   .
00000460   02 00 00 00   06 00 00 00   0B 00 00 00   00 00 00 00   .
-**  phase5.o        --0x450/0x948--46%----------------------------
```

图 3.1.20 第五关符号节点修改

修改结果如下：

```
francis@francis-VMware-Virtual-Platform:~/桌面/链接炸弹_学生端$ gcc -no-pie -o l
inkbomb5 main.o phase5.o
francis@francis-VMware-Virtual-Platform:~/桌面/链接炸弹_学生端$ ./linkbomb5
please input you stuid : u202315752
Class Name Computer Foundation
Teacher Name Xu Xiangyang
Bye Bye !
```

图 3.1.21 第五关结果

第六关 强弱符号

这一关我们需要自己写一个程序，这一关我注意到标题是强弱符号，那么什么是强弱符号，“函数名和已初始化的全局变量名是强符号，未初始化的全局变量名是弱符号”。我们分别反汇编 main.o 和 phase6.o，

查看 phase 变量和 myprint 变量。

老规矩，直接先看看什么情况：

```
francis@francis-VMware-Virtual-Platform:~/桌面/链接炸弹_学生端$ gcc -no-pie -o l
inkbomb6 main.o phase6.o
francis@francis-VMware-Virtual-Platform:~/桌面/链接炸弹_学生端$ ./linkbomb6
please input you stuid : U202315752
please write phase6_patch.c
Bye Bye !
```

图 3.1.22 第六关尝试

在 phase6. 我们发现有一个 myprint 变量是未初始化全局变量（COM），且其大小为 8,我们猜测是指针，

```
Symbol table '.symtab' contains 16 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
     0: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS phase6.c
     2: 0000000000000000     0 SECTION LOCAL  DEFAULT    1 .text
     3: 0000000000000000     0 SECTION LOCAL  DEFAULT    3 .data
     4: 0000000000000000     0 SECTION LOCAL  DEFAULT    4 .bss
     5: 0000000000000000     0 SECTION LOCAL  DEFAULT    5 .data.rel.local
     6: 0000000000000000     0 SECTION LOCAL  DEFAULT    7 .rodata
     7: 0000000000000000     0 SECTION LOCAL  DEFAULT    9 .note.GNU-stack
     8: 0000000000000000     0 SECTION LOCAL  DEFAULT   10 .note.gnu.property
     9: 0000000000000000     0 SECTION LOCAL  DEFAULT   11 .eh_frame
    10: 0000000000000000     0 SECTION LOCAL  DEFAULT    8 .comment
    11: 0000000000000000     8 OBJECT  GLOBAL DEFAULT    5 phase
    12: 0000000000000000    58 FUNC    GLOBAL DEFAULT    1 do_phase
    13: 0000000000000008     8 OBJECT  GLOBAL DEFAULT  COM myprint
    14: 0000000000000000     0 NOTYPE  GLOBAL DEFAULT  UND _GLOBAL_OFFSET_TABLE_
    15: 0000000000000000     0 NOTYPE  GLOBAL DEFAULT  UND puts
```

图 3.1.23 第六关符号节点表

通过.rela.text 表得知重定位在.text 节中发生的位置。

```
重定位节 '.rela.text' at offset 0x2e8 contains 4 entries:
  偏移量          信息           类型           符号值        符号名称 + 加数
000000000012  000d00000002 R_X86_64_PC32     0000000000000008 myprint - 4
00000000001e  000d00000002 R_X86_64_PC32     0000000000000008 myprint - 4
00000000002e  000600000002 R_X86_64_PC32     0000000000000000 .rodata - 4
000000000033  000f00000004 R_X86_64_PLT32    0000000000000000 puts - 4
```

图 3.1.24 第六关偏移地址查询

f、16、19 行是判断 myprint 是否赋值，若未初始化则值默认为 0，跳转到 2b 行开始打印只读字符串，即上文的 please write~。若 myprint 不为 NULL，注意第 27 行，call *%rdx，说明 myprint 是一个函数指针，按照题意这个函数要能打印我们的学号。故创建一个新的文件，在其中定义，myprint 函数指针，并指向 func1 函数，func1 函数仅用来打印学号。随后进行编译，重新链接

```
0000000000000000 <do_phase>:
   0:   f3 0f 1e fa             endbr64
   4:   55                      push   %rbp
   5:   48 89 e5                mov    %rsp,%rbp
   8:   48 83 ec 10             sub    $0x10,%rsp
   c:   89 7d fc                mov    %edi,-0x4(%rbp)
   f:   48 8b 05 00 00 00 00    mov    0x0(%rip),%rax        # 16 <do_phase+0x16
>
  16:   48 85 c0                test   %rax,%rax
  19:   74 10                   je     2b <do_phase+0x2b>
  1b:   48 8b 15 00 00 00 00    mov    0x0(%rip),%rdx        # 22 <do_phase+0x22
>
  22:   b8 00 00 00 00          mov    $0x0,%eax
  27:   ff d2                   call   *%rdx
  29:   eb 0c                   jmp    37 <do_phase+0x37>
  2b:   48 8d 3d 00 00 00 00    lea    0x0(%rip),%rdi        # 32 <do_phase+0x32
>
  32:   e8 00 00 00 00          call   37 <do_phase+0x37>
  37:   90                      nop
  38:   c9                      leave
  39:   c3                      ret
```

图 3.1.25 第六关反汇编情况

我们撰写的 c 语言代码如下：

```c
#include<stdio.h>
void f();
extern void (*myprint)();
void (*myprint)()=f;
void f(){
printf("U202315752\n");
}
```

图 3.1.26 第六关代码撰写

注意代码撰写细节，由于我们只定义了指针 myprint,也就是说，我们自己撰写的函数（图中为 f 函数）无法直接链接，因为这个标识符无人知晓，所以我们定义的 f 函数要让 myprinf 函数指针指向他，运行结果如下，链接指令如图：

```
francis@francis-VMware-Virtual-Platform:~/桌面/链接炸弹_学生端$ gcc -no-pie -o l
inkbomb6 main.o phase6.o phase6_patch.o
francis@francis-VMware-Virtual-Platform:~/桌面/链接炸弹_学生端$ ./linkbomb6
please input you stuid : U202315752
U202315752
Bye Bye !
francis@francis-VMware-Virtual-Platform:~/桌面/链接炸弹_学生端$
```

图 3.1.27 第六关代码结果

7、第 7 关 只读数据节的修改

有了前几关的基础，第七关就比较简单了。我们先尝试直接输出，发现最后输出的最后五位与我们的学号不同。

```
francis@francis-VMware-Virtual-Platform:~/桌面/链接炸弹_学生端$ gcc -no-pie -o l
inkbomb7 main.o phase7.o
francis@francis-VMware-Virtual-Platform:~/桌面/链接炸弹_学生端$ ./linkbomb7
please input you stuid : U202315752
Gate 7: U202212345
Bye Bye !
francis@francis-VMware-Virtual-Platform:~/桌面/链接炸弹_学生端$
```

图 3.1.28 第七关尝试

然后用 readelf 指令查看我们 phase7.o 里面各个节对应的位置，找到我们的.rodata 只读数据节（因为这一关的任务是修改只读数据节）对应的偏移，可以观察到是偏移量 0x68，大小是 0x13：

```
       0000000000000018  0000000000000018   1       13     5     8
  [ 7] .rodata           PROGBITS         0000000000000000  00000068
       0000000000000013  0000000000000000   A       0      0     1
```

图 3.1.29 第七关偏移地址

再利用 hexedit 指令在该数据节中依据右侧对应的值找到我们最终输出答案的存放位置，将错误的位改成我们的学号即可。

```
00000050   8D 3D 00 00  00 00 E8 00  00 00 00 90  C9 C3 00 00  .=..............
00000060   00 00 00 00  00 00 00 00  47 61 74 65  20 37 3A 20  ........Gate 7:
00000070   55 32 30 32  33 31 35 37  35 32 00 00  47 43 43 3A  U202315752..GCC:
00000080   20 28 55 62  75 6E 74 75  20 39 2E 34  2E 30 2D 31   (Ubuntu 9.4.0-1
```

图 3.1.30 第七关未定义变量节点修改

修改结果如下：

```
francis@francis-VMware-Virtual-Platform:~/桌面/链接炸弹_学生端$ ./linkbomb7
bash: ./linkbomb7: 没有那个文件或目录
francis@francis-VMware-Virtual-Platform:~/桌面/链接炸弹_学生端$ gcc -no-pie -o l
inkbomb7 main.o phase7.o
francis@francis-VMware-Virtual-Platform:~/桌面/链接炸弹_学生端$ ./linkbomb7
please input you stuid : U202315752
Gate 7: U202315752
Bye Bye !
francis@francis-VMware-Virtual-Platform:~/桌面/链接炸弹_学生端$
```

图 3.1.31 第七关结果

（2） 描述修改各个文件的基本思想

1、第 1 关 数据节的修改

查询“.rela 节”的偏移地址后，用 hexedit 指令修改二进制可重定位目标文件 phase1.o 的数据节中的内容，使其与 main.o 链接后，生成的执行程序，可以输出自己的学号，需要注意的是我们的学号在实际的汇编代码中不是连续传递，会先传递前八个值，再传后面两个值。

2、第 2 关 简单的机器指令修改

先用反汇编查看 myfun 函数的地址，然后通过写简单的 call 函数的汇编指令，反汇编得到二进制指令后插入修改二进制可重定位目标文件 phase2.o 的代码节中的内容，使其与 main.o 链接后，生成的执行程序。在 phase_2.c 中，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，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) ，在 do_phase 函数中调用 myfunc(pos)，将 do_phase 的参数 pos 直接传递 myfunc，其中，找到参数 pos 储存的位置，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，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) ，其中，s 为一个指针，通过间接传址找到，然后 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，找到符号表节后，通过找到他们的偏移地址后进行修改，把储存正确信息的符号的地址传到现在的位置，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 : Xu Xiangyang。

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。其中，我们需要找到没有定义的函数指针，让他在后续我们写的代码里面有空间指向我们写的函数

7、第 7 关 只读数据节的修改

修改 phase7.o 中只读数据节，其与 main.o 链接后，能够输出自己的学号。

四、体会

首先，我深入了解了 ELF 文件的结构和原理，这使我对程序的编译和链接过程有了全新的认识。ELF 文件作为可执行文件的一种标准格式，其内部复杂的组织结构，包括文件头、节表和段表等，为我揭示了程序如何被计算机所识别和执行。通过亲手操作和分析 ELF 文件，我能够更加清晰地理解程序的构成和运行机制。

其次，在实验中，我掌握了多种强大的工具来解析和编辑 ELF 文件，如 objdump、readelf 和 hexedit 等。这些工具不仅帮助我直观地查看了 ELF 文件的汇编代码、头部信息和节表内容，还让我能够亲手尝试修改文件内容，从而更深入地了解程序的内部结构。特别是将源代码编译成可重定位文件（.o 文件），并进一步链接成可执行文件的过程，让我对 C 语言程序的编译链接流程有了更加全面的认识。

同时，我也深刻体会到了修改 ELF 文件可能带来的风险。在实验过程中，我尝试修改了节表的某些属性或大小，结果导致程序无法正常运行。这让我意识到，在修改 ELF 文件时需要格外小心，必须确保对修改的后果有充分的认识和准备。这也让我更加明白，在编程和软件开发中，每一个细节都可能影响到程序的稳定性和正确性。

此外，与同学们的交流讨论让我受益匪浅。通过互相学习和分享经验，我们共同解决了实验中遇到的难题，也从彼此的观点中获得了新的启示。这种团队合作和互助精神不仅让我感受到了学习的乐趣，也让我更加珍惜与同学们共度的时光。

总的来说，这次 ELF 文件与程序链接的实验不仅加深了我对计算机系统基础知识的理解，还提高了我的实践能力和问题解决能力。我相信这些宝贵的经验和收获将对我未来的学习和工作产生深远的影响。