哈爾濱工業大學

实验报告

实 验（五）

题　　目 LinkLab

链接

专　　业 计算机科学与技术

学　　号 1183200123

班　　级 1803003

学　　生 祁 天

指 导 教 师 史先俊

实 验 地 点 G712

实 验 日 期 2019 年 11 月 19 日

计算机科学与技术学院

目 录

第 1 章 实验基本信息

1.1 实验目的

理解链接的作用与工作步骤

掌握 ELF 结构与符号解析与重定位的工作过程

熟练使用 Linux 工具完成 ELF 分析与修改

1.2 实验环境与工具

1.2.1 硬件环境

X64 CPU；2GHz；2G RAM；256GHD Disk 以上

1.2.2 软件环境

Windows7 64 位以上；VirtualBox/Vmware 11 以上；Ubuntu 16.04 LTS 64 位/优麒麟 64 位；

1.2.3 开发工具

Visual Studio 2010 64 位以上；GDB/OBJDUMP；DDD/EDB 等

1.3 实验预习

上实验课前，必须认真预习实验指导书（PPT 或 PDF）

了解实验的目的、实验环境与软硬件工具、实验操作步骤，复习与实验有关的理论知识。

第 2 章 实验预习

2.1 请按顺序写出 ELF 格式的可执行目标文件的各类信息（5 分）

ELF 头
段头部表
.init
.text
.rodata
.data
.bss
.symtab
.debug
.line
.strtab
节头部表

ELF 头：描述文件的总体格式。还包括程序的入口点，及程序要运行时所要执行的第一条指令的地址。

段头部表：用于将连续的文件节映射到运行时的内存段

.init：定义了一个叫做_init 的小程序，在程序初始化代码会用到。

.text：已编译程序的机器代码

.rodata：只读数据。

.data：已初始化的全局和静态 C 变量。

.bss：未初始化的全局和静态 C 变量。

.symtab：符号表，存放程序中定义和引用的函数和全局变量的信息。

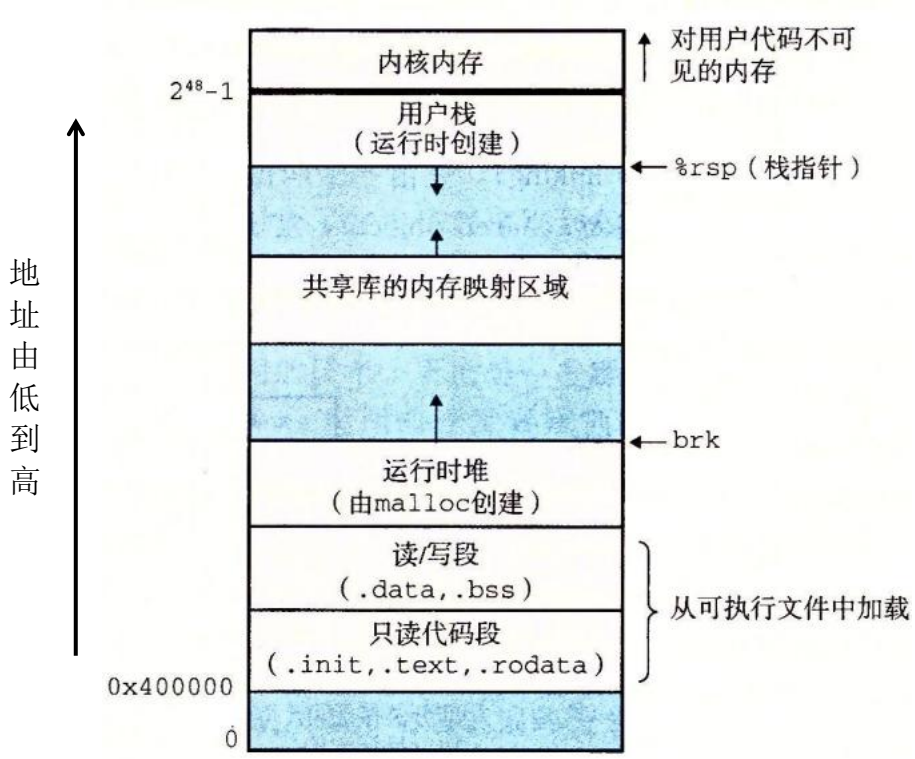
.debug：调试符号表，只有在以-g 选项编译时才会有，其条目是程序中定义的局部变量和类型定义。

.line:原始 c 源程序中的行号和.text 机器指令之间的映射，只有在以-g 选项编译时才会有。

.strtab:字符串表.

节头部表:描述目标文件的节

2.2 请按照内存地址从低到高的顺序，写出 Linux 下 X64 内存映像。（5 分）



2.3 请运行“LinkAddress -u 学号 姓名” 按地址循序写出各符号的地址、空间。并按照 Linux 下 X64 内存映像标出其所属各区。

（5 分）

1. 内存虚拟内存区：
   0x800000000000 - 0xffffffffffff

2. envstring 环境变量字符串：
   env[0] *env 0x7ffc9d25926a 140722944971370
   SHELL=/bin/bash
   .
   .
   .
   env[63] *env 0x7ffc9d259fc9 140722944974793
   LC_NUMERIC=en_HK.UTF-8
   env[64] *env 0x7ffc9d259fe0 140722944974816
   _=./linkaddr

3. argv string 命令行字符串：
   argv[0] 0x7ffc9d25924a 140722944971338
   ./linkaddr

argv[1] 0x7ffc9d259255 140722944971349
-u
argv[2] 0x7ffc9d259258 140722944971352
1183200123
argv[3] 0x7ffc9d259263 140722944971363
祁天

4. env pointers 环境变量指针表：
   env 0x7ffc9d258d30 140722944970032

5. 命令行参数指针表：
   argv 0x7ffc9d258d08 140722944969992

6. 命令行参数个数
   argc 0x7ffc9d2587cc 140722944968652

7. main 函数的栈帧（因为 local int 为 main 函数里的第一个局部变量）：
   local int 0 0x7ffc9d2587dc 140722944968668
   local int 1 0x7ffc9d2587e0 140722944968672
   local static int 0 0x5607346de0d094588944376016
   local static int 1 0x5607346de0a4 94588944375972
   local astr 0x7ffc9d258820 140722944968736
   local pstr 0x5607346dc0d094588944367824

8. 共享函数映射区：
   exit 0x7f53aece73c0 139997391778752
   printf 0x7f53aed02830 139997391890480
   malloc 0x7f53aed38a40 139997392112192
   free 0x7f53aed391d0 139997392114128
   strcpy 0x7f53aee275c0 139997393089984

9. 共享内存分配区：（mmap > 128k）
   p1 0x7f539ec9f010 139997123047440
   p3 0x7f539ec7e010 139997122912272
   p4 0x7f535ec7d010 139996049166352
   p5 0x7f52dec7c010 139993901678608

10. heap 运行时堆区：
    p2 0x560775baf670 94590039946864

11. .bss 未初始化全局变量区：
    big array 0x5607746de0e0 94590018117856
    huge array 0x5607346de0e0 94588944376032

12. .data 初始化的全局变量区：

global 0x5607346de02094588944375840
gint0 0x5607346de0cc 94588944376012
glong 0x5607346de02894588944375848

13. .rodata 只读数据区：
gc 0x5607346dc04c 94588944367692
cc 0x5607346dc060 94588944367712

14. .text 只读代码段：
show_pointer 0x5607346db160 94588944363872
useless 0x5607346db155 94588944363861
main 0x5607346db193 94588944363923

15. init 初始化代码段：
init
start

2.4 请按顺序写出 LinkAddress 从开始执行到 main 前/后执行的子程序的名字。（gcc 与 objdump/GDB/EDB）（5 分）

（main 之前）

ld_linux.so id.so.conf

call _dl_start
call dl init(dl-init.c); call init
jmp start

_start:
call __libc_ start main(libc-start.c) halt

__libc_ start main:
call _GI__cxa_atexit (cxa atexitc)
call _libc_csu_init： call init (41B)
call setjmp (bsd-_setjmp.S)
call main

(main 之后)

call _GI_exit(exit.c)

__GI_exit(exit.c):
call __run_exit_handlers(exitc)

run exit handlers(exitc) :
call _dl_fini
call _IO_cleanup
call __exit: syscall

第 3 章 各阶段的原理与方法

每阶段 40 分，phasex.o 20 分，分析 20 分，总分不超过 80 分

3.1 阶段 1 的分析

程序运行结果截图：

```
qt1183200123@ubuntu:~/linklab$ gcc -m32 -o linkbomb main.o phase1.o
qt1183200123@ubuntu:~/linklab$ ./linkbomb 1183200123
1183200123
```

分析与设计的过程：

在未进行修改处理前，运行得到以下结果：

```
qt1183200123@ubuntu:~/linklab$ ./linkbomb 1183200123
8Jdy    augmOPU971MATK3KNxW8Dg45LxG0XvXlQbEHNby01ulLKxa2kJZvTXOp 3brcGaReHpEZGb1T5WX1gHDRM9AmZEiaUCLR
xrJJZ49a aEfNhcP5wrZmDDUxK2tU2uZl3dnYrbt6jyCX8gU5l7tY9Fsu2       DOSRQ
```

输出的字符串位于.data 节，使用 readelf 工具，先查看 phase1.o 的节头表，找到.data 节的偏移：

qt1183200123@ubuntu: ~/linklab

[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		13	13	4
[2]	.text	PROGBITS	00000000	00003c	00002b	00	AX	0	0	1
[3]	.rel.text	REL	00000000	000330	000020	08	I	13	2	4
[4]	.data	PROGBITS	00000000	000080	0000cd	00	WA	0	0	32
[5]	.bss	NOBITS	00000000	00014d	000000	00	WA	0	0	1
[6]	.data.rel.local	PROGBITS	00000000	000150	000004	00	WA	0	0	4
[7]	.rel.data.rel.loc	REL	00000000	000350	000008	08	I	13	6	4
[8]	.text.__x86.get_p	PROGBITS	00000000	000154	000004	00	AXG	0	0	1
[9]	.comment	PROGBITS	00000000	000158	000025	01	MS	0	0	1
[10]	.note.GNU-stack	PROGBITS	00000000	00017d	000000	00		0	0	1
[11]	.eh_frame	PROGBITS	00000000	000180	000050	00	A	0	0	4
[12]	.rel.eh_frame	REL	00000000	000358	000010	08	I	13	11	4
[13]	.symtab	SYMTAB	00000000	0001d0	000110	10		14	12	4
[14]	.strtab	STRTAB	00000000	0002e0	00004d	00		0	0	1
[15]	.shstrtab	STRTAB	00000000	000368	00008e	00		0	0	1

.data 节的偏移为 0x80.

于是，使用 hexedit 工具查看 phase1.o 的内容：

```
00000064   FC C9 C3 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
00000078   00 00 00 00  00 00 00 00  57 6F 35 77  38 52 62 34  ........Wo5w8Rb4
0000008C   67 7A 44 62  79 46 7A 30  35 42 6C 6C  56 5A 6C 42  gzDbyFz05BllVZlB
000000A0   79 47 61 4D  56 39 70 33  33 38 4A 64  79 09 61 75  yGaMV9p338Jdy.au
000000B4   55 39 37 31  4D 41 54 4B  33 4B 4E 78  57 38 44 67  U971MATK3KNxW8Dg
000000C8   47 30 58 76  58 6C 51 62  45 48 4E 62  79 30 31 75  G0XvXlQbEHNby01u
000000DC   61 32 6B 4A  5A 76 54 58  4F 70 20 33  62 72 63 47  a2kJZvTXOp 3brcG
000000F0   70 45 5A 47  62 31 54 35  57 58 31 67  48 44 52 4D  pEZGb1T5WX1gHDRM
00000104   45 69 61 55  43 4C 52 78  72 4A 4A 5A  34 39 61 20  EiaUCLRxrJJZ49a
00000118   68 63 50 35  77 72 5A 6D  44 44 55 78  4B 32 74 55  hcP5wrZmDDUxK2tU
0000012C   33 64 6E 59  72 62 74 36  6A 79 43 58  38 67 55 35  3dnYrbt6jyCX8gU5
00000140   39 46 73 75  32 09 20 44  4F 53 52 51  00 00 00 00  9Fsu2. DOSRQ....
00000154   8B 04 24 C3  00 47 43 43  3A 20 28 55  62 75 6E 74  ..$..GCC: (Ubunt
00000168   33 2E 30 2D  31 36 75 62  75 6E 74 75  33 29 20 37  3.0-16ubuntu3) 7
```

结合我们最初得到的输出内容，确定输出字符串的起始位置：

```
qt1183200123@ubuntu:~/linklab$ ./linkbomb 1183200123
8Jdy    augmOPU971MATK3KNxW8Dg45LxG0XvXlQbEHNby01ulLKxa2kJZvTXOp 3brcGaReHpEZGb1T5WX1gHDRM9AmZEiaUCLR
xrJJZ49a aEfNhcP5wrZmDDUxK2tU2uZl3dnYrbt6jyCX8gU5l7tY9Fsu2       DOSRQ
```

修改其对应的 ASCII 码：

```
00000078   00 00 00 00  00 00 00 00  57 6F 35 77  38 52 62 34  ........Wo5w8Rb4
0000008C   67 7A 44 62  79 46 7A 30  35 42 6C 6C  56 5A 6C 42  gzDbyFz05BllVZlB
000000A0   79 47 61 4D  56 39 70 33  33 31 31 38  33 32 30 30  yGaMV9p331183200
000000B4   55 39 37 31  4D 41 54 4B  33 4B 4E 78  57 38 44 67  U971MATK3KNxW8Dg
000000C8   47 30 58 76  58 6C 51 62  45 48 4E 62  79 30 31 75  G0XvXlQbEHNby01u
000000DC   61 32 6B 4A  5A 76 54 58  4F 70 20 33  62 72 63 47  a2kJZvTXOp 3brcG
000000F0   70 45 5A 47  62 31 54 35  57 58 31 67  48 44 52 4D  pEZGb1T5WX1gHDRM
00000104   45 69 61 55  43 4C 52 78  72 4A 4A 5A  34 39 61 20  EiaUCLRxrJJZ49a
00000118   68 63 50 35  77 72 5A 6D  44 44 55 78  4B 32 74 55  hcP5wrZmDDUxK2tU
```

重新链接编译运行，得到目标结果：

```
qt1183200123@ubuntu:~/linklab$ gcc -m32 -o linkbomb main.o phase1.o
qt1183200123@ubuntu:~/linklab$ ./linkbomb 1183200123
1183200123
```

3.2 阶段 2 的分析

程序运行结果截图：

```
qt1183200123@ubuntu:~/linklab$ gcc -m32 -o linkbomb main.o phase2.o
qt1183200123@ubuntu:~/linklab$ ./linkbomb 1183200123
1183200123
```

分析与设计的过程：

使用 objdump 查看 phase2.o 的反汇编码：

```
qt1183200123@ubuntu: ~/linklab
Disassembly of section .text:

00000000 <qfluAEFn>:
   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 <qfluAEFn+0x8>
   c:   81 c3 02 00 00 00       add    $0x2,%ebx
  12:   83 ec 08                sub    $0x8,%esp
  15:   8d 83 00 00 00 00       lea    0x0(%ebx),%eax
  1b:   50                      push   %eax
  1c:   ff 75 08                pushl  0x8(%ebp)
  1f:   e8 fc ff ff ff          call   20 <qfluAEFn+0x20>
  24:   83 c4 10                add    $0x10,%esp
  27:   85 c0                   test   %eax,%eax
  29:   75 10                   jne    3b <qfluAEFn+0x3b>
  2b:   83 ec 0c                sub    $0xc,%esp
  2e:   ff 75 08                pushl  0x8(%ebp)
  31:   e8 fc ff ff ff          call   32 <qfluAEFn+0x32>
  36:   83 c4 10                add    $0x10,%esp
  39:   eb 01                   jmp    3c <qfluAEFn+0x3c>
  3b:   90                      nop
  3c:   8b 5d fc                mov    -0x4(%ebp),%ebx
  3f:   c9                      leave
  40:   c3                      ret
```

根据 Slides 上的提示，我们需要将学号做为 do_phase 的参数：

phase2.c程序框架

```
static void OUTPUT_FUNC_NAME( const char *id )   // 该函数名对每名学生均不同
{
    if( strcmp(id,MYID) != 0 ) return;
    printf("%s\n", id);
}
```

我们查看 do_phase 的反汇编代码：

```
00000041 <do_phase>:
  41:   55                      push   %ebp
  42:   89 e5                   mov    %esp,%ebp
  44:   e8 fc ff ff ff          call   45 <do_phase+0x4>
  49:   05 01 00 00 00          add    $0x1,%eax
  4e:   90                      nop
  4f:   90                      nop
  50:   90                      nop
  51:   90                      nop
  52:   90                      nop
  53:   90                      nop
```

红框部分与 qfluAEFn 函数访问.rodata 处有相似之处，所以猜想可模仿其操作。但是其中有一部分为重定位部分，无法得知其真实值。

我们可以先链接出可执行程序，在反汇编出相应汇编代码，从而获得其重定位之后的值。

```
00001201 <qfluAEFn>:
    1201:       55                      push   %ebp
    1202:       89 e5                   mov    %esp,%ebp
    1204:       53                      push   %ebx
    1205:       83 ec 04                sub    $0x4,%esp
    1208:       e8 a3 fe ff ff          call   10b0 <__x86.get_pc_thunk.bx>
    120d:       81 c3 c7 2d 00 00       add    $0x2dc7,%ebx
    1213:       83 ec 08                sub    $0x8,%esp
    1216:       8d 83 a8 e0 ff ff       lea    -0x1f58(%ebx),%eax
    121c:       50                      push   %eax
    121d:       ff 75 08                pushl  0x8(%ebp)
    1220:       e8 0b fe ff ff          call   1030 <strcmp@plt>
    1225:       83 c4 10                add    $0x10,%esp
    1228:       85 c0                   test   %eax,%eax
    122a:       75 10                   jne    123c <qfluAEFn+0x3b>
    122c:       83 ec 0c                sub    $0xc,%esp
    122f:       ff 75 08                pushl  0x8(%ebp)
    1232:       e8 09 fe ff ff          call   1040 <puts@plt>
    1237:       83 c4 10                add    $0x10,%esp
```

此时，重定位的值已经出现。我们可以模仿其编写汇编代码了：

```
lea     -0x18b0(%eax),%eax      .rodata的值给%eax
push    %eax                    将其作为第一个参数压入栈中
call    0x8                     占位符，具体机器代码见后续分析
mov     %ebp,%esp               还原%rsp
```

反汇编得到如下机器代码：

```
00000000 <.text>:
   0:   8d 83 a8 e0 ff ff       lea    -0x1f58(%ebx),%eax
   6:   50                      push   %eax
   7:   e8 04 00 00 00          call   0x10
   c:   89 ec                   mov    %ebp,%esp
qt1183200123@ubuntu:~/linklab$
```

我们打算把代码插入的位置是：

```
qt1183200123@ubuntu: ~/linklab
  40:   c3                      ret

00000041 <do_phase>:
  41:   55                      push   %ebp
  42:   89 e5                   mov    %esp,%ebp
  44:   e8 fc ff ff ff          call   45 <do_phase+0x4>
  49:   05 01 00 00 00          add    $0x1,%eax
  4e:   90                      nop
  4f:   90                      nop
  50:   90                      nop
  51:   90                      nop
```

我们编写的汇编代码如下并将其转化为机器代码：

```
8d 98 a8 e0 ff ff           lea    -0x1f58(%eax),%ebx
53                          push   %ebx
e8 a6 ff ff ff              call   0 <qfluAEFn>
c9                          leave
c3                          ret
```

然后使用 readelf 查看 phase2.o：

```
qt1183200123@ubuntu: ~/linklab

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     16  20  4
  [ 2] .group            GROUP           00000000 00003c 000008 04     16  15  4
  [ 3] .text             PROGBITS        00000000 000044 000071 00  AX  0   0  1
  [ 4] .rel.text         REL             00000000 000344 000038 08   I 16   3  4
  [ 5] .data             PROGBITS        00000000 0000b5 000000 00  WA  0   0  1
  [ 6] .bss              NOBITS          00000000 0000b5 000000 00  WA  0   0  1
  [ 7] .rodata           PROGBITS        00000000 0000b5 00000b 00   A  0   0  1
  [ 8] .data.rel.local   PROGBITS        00000000 0000c0 000004 00  WA  0   0  4
  [ 9] .rel.data.rel.loc REL             00000000 00037c 000008 08   I 16   8  4
  [10] .text.__x86.get_p PROGBITS        00000000 0000c4 000004 00 AXG  0   0  1
  [11] .text.__x86.get_p PROGBITS        00000000 0000c8 000004 00 AXG  0   0  1
  [12] .comment          PROGBITS        00000000 0000cc 000025 01  MS  0   0  1
  [13] .note.GNU-stack   PROGBITS        00000000 0000f1 000000 00      0   0  1
  [14] .eh_frame         PROGBITS        00000000 0000f4 000084 00   A  0   0  4
  [15] .rel.eh_frame     REL             00000000 000384 000020 08   I 16  14  4
  [16] .symtab           SYMTAB          00000000 000178 000160 10     17  15  4
  [17] .strtab           STRTAB          00000000 0002d8 00006a 00      0   0  1
  [18] .shstrtab         STRTAB          00000000 0003a4 0000b2 00      0   0  1
```

确定代码段偏移为 0x44，于是使用 hexedit 编辑：

qt1183200123@ubuntu: ~/linklab

```
00000000   7F 45 4C 46   01 01 01 00   00 00 00 00   00 00 00 00   .ELF............
00000010   01 00 03 00   01 00 00 00   00 00 00 00   00 00 00 00   ................
00000020   58 04 00 00   00 00 00 00   34 00 00 00   00 00 28 00   X.......4.....(.
00000030   13 00 12 00   01 00 00 00   0A 00 00 00   01 00 00 00   ................
00000040   0B 00 00 00   55 89 E5 53   83 EC 04 E8   FC FF FF FF   ....U..S........
00000050   81 C3 02 00   00 00 83 EC   08 8D 83 00   00 00 00 50   ...............P
00000060   FF 75 08 E8   FC FF FF FF   83 C4 10 85   C0 75 10 83   .u...........u..
00000070   EC 0C FF 75   08 E8 FC FF   FF FF 83 C4   10 EB 01 90   ...u............
00000080   8B 5D FC C9   C3 55 89 E5   E8 FC FF FF   FF 05 01 00   .]...U..........
00000090   00 00 8D 98   A8 E0 FF FF   53 E8 A6 FF   FF FF C9 C3   ........S.......
000000A0   90 90 90 90   90 90 90 90   90 90 90 90   90 90 90 90   ................
000000B0   90 90 90 5D   C3 31 31 38   33 32 30 30   31 32 33 00   ...].1183200123.
000000C0   00 00 00 00   8B 04 24 C3   8B 1C 24 C3   00 47 43 43   ......$...$..GCC
000000D0   3A 20 28 55   62 75 6E 74   75 20 37 2E   33 2E 30 2D   : (Ubuntu 7.3.0-
000000E0   31 36 75 62   75 6E 74 75   33 29 20 37   2E 33 2E 30   16ubuntu3) 7.3.0
000000F0   00 00 00 00   14 00 00 00   00 00 00 00   01 7A 52 00   .............zR.
00000100   01 7C 08 01   1B 0C 04 04   88 01 00 00   20 00 00 00   .|.......... ...
00000110   1C 00 00 00   00 00 00 00   41 00 00 00   00 41 0E 08   ........A....A..
00000120   85 02 42 0D   05 44 83 03   79 C5 C3 0C   04 04 00 00   ..B..D..y.......
00000130   1C 00 00 00   40 00 00 00   41 00 00 00   30 00 00 00   ....@...A...0...
00000140   00 41 0E 08   85 02 42 0D   05 6C C5 0C   04 04 00 00   .A....B..l......
00000150   10 00 00 00   60 00 00 00   00 00 00 00   04 00 00 00   ....`...........
00000160   00 00 00 00   10 00 00 00   74 00 00 00   00 00 00 00   ........t.......
---  phase2.o        --0x0/0x750-----------------------------------------------
```

重新链接编译运行，得到目标结果：

```
qt1183200123@ubuntu:~/linklab$ hexedit phase2.o
qt1183200123@ubuntu:~/linklab$ gcc -m32 -o linkbomb main.o phase2.o
qt1183200123@ubuntu:~/linklab$ ./linkbomb 1183200123
1183200123
qt1183200123@ubuntu:~/linklab$
```

3.3 阶段 3 的分析

程序运行结果截图：

```
qt1183200123@ubuntu:~/linklab$ gcc -m32 -o linkbomb main.o phase3.o phase3_p
atch.o
qt1183200123@ubuntu:~/linklab$ ./linkbomb 1183200123
1183200123
```

分析与设计的过程：

结合 Slides 中内容的解析：

phase3.c程序框架

```
char PHASE3_CODEBOOK[256];
void do_phase(){
    const char char cookie[] = PHASE3_COOKIE;
    for( int i=0; i<sizeof(cookie)-1; i++ )
        printf( "%c", PHASE3_CODEBOOK[ (unsigned char)(cookie[i]) ] );
    printf( "\n" );
}
```

首先要查看 cookie 数组的内容：

使用 objdump 工具反汇编 phase_3.o：

```
qt1183200123@ubuntu: ~/linklab
Disassembly of section .text:

00000000 <do_phase>:
   0:   55                      push   %ebp
   1:   89 e5                   mov    %esp,%ebp
   3:   53                      push   %ebx
   4:   83 ec 24                sub    $0x24,%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:   65 a1 14 00 00 00       mov    %gs:0x14,%eax
  18:   89 45 f4                mov    %eax,-0xc(%ebp)
  1b:   31 c0                   xor    %eax,%eax
  1d:   c7 45 e9 77 70 62 6d    movl   $0x6d627077,-0x17(%ebp)
  24:   c7 45 ed 6f 79 69 71    movl   $0x7169796f,-0x13(%ebp)
  2b:   66 c7 45 f1 75 7a       movw   $0x7a75,-0xf(%ebp)
  31:   c6 45 f3 00             movb   $0x0,-0xd(%ebp)
  35:   c7 45 e4 00 00 00 00    movl   $0x0,-0x1c(%ebp)
  3c:   eb 2b                   jmp    69 <do_phase+0x69>
  3e:   8d 55 e9                lea    -0x17(%ebp),%edx
  41:   8b 45 e4                mov    -0x1c(%ebp),%eax
  44:   01 d0                   add    %edx,%eax
  46:   0f b6 00                movzbl (%eax),%eax
  49:   0f b6 c0                movzbl %al,%eax
  4c:   8b 93 00 00 00 00       mov    0x0(%ebx),%edx
                        4e: R_386_GOT32X        SDFyFKcKmx
  52:   0f b6 04 02             movzbl (%edx,%eax,1),%eax
  56:   0f be c0                movsbl %al,%eax
```

为了查看运行时-0x17(%ebp)内存里的值，使用 gdb 运行可执行目标文件，运行到对应位置，使用(gdb) x/10ub　$ebp – 0x17 命令查看 cookie 数组的值：

```
(gdb) x/10ub $ebp-0x17
0xffffcf41:     119     112     98      109     111     121     105     113
0xffffcf49:     117     122
```

因此，我们要使得，全局变量数组的相应下标的元素，为我的学号中的数字：

PHASE3_CODEBOOK[119] = ‘1’

PHASE3_CODEBOOK[112] = ‘1’

PHASE3_CODEBOOK[98] = ‘8’

PHASE3_CODEBOOK[109] = ‘3’

PHASE3_CODEBOOK[111] = ‘2’

PHASE3_CODEBOOK[121] = ‘0’

PHASE3_CODEBOOK[105] = ‘0’

PHASE3_CODEBOOK[113] = ‘1’

PHASE3_CODEBOOK[117] = ‘2’

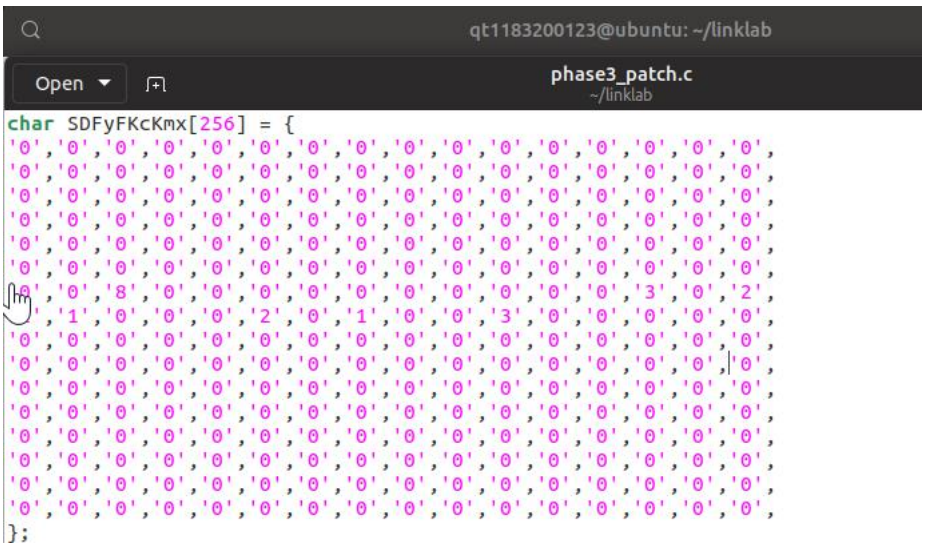
PHASE3_CODEBOOK[122] = ‘3’

由于在 phase3.o 中，PHASE3_CODEBOOK 数组是未初始化的全局变量，在符号解析中属于弱符号，所以，我们在 phase3_patch.o 中，只需定义同名的全局变量，并初始化为我们想要的数值即可。由于是强符号，符号解析时会优先选择它。

从上面的反汇编代码中，可知该全局数组的符号名为：SDFyFKcKmx

```
46:   0f b6 00                movzbl (%eax),%eax
49:   0f b6 c0                movzbl %al,%eax
4c:   8b 93 00 00 00 00       mov    0x0(%ebx),%edx
                      4e: R_386_GOT32X        SDFyFKcKmx
52:   0f b6 04 02             movzbl (%edx,%eax,1),%eax
56:   0f be c0                movsbl %al,%eax
59:   83 ec 0c                sub    $0xc,%esp
```

于是，编写 phase3_patch.c 文件如下：



编译为.o 文件，再与 phase3.o main.o 链接得到可执行文件，运行，成功：

```
qt1183200123@ubuntu:~/linklab$ gedit phase3_patch.c
qt1183200123@ubuntu:~/linklab$ gcc -m32 -c phase3_patch.c
qt1183200123@ubuntu:~/linklab$ gcc -m32 -o linkbomb main.o phase3.o phase3_p
atch.o
qt1183200123@ubuntu:~/linklab$ ./linkbomb 1183200123
1183200123
```

3.4 阶段 4 的分析

程序运行结果截图：

分析与设计的过程：

3.5 阶段 5 的分析

程序运行结果截图：

分析与设计的过程：

第 4 章 总结

4.1 请总结本次实验的收获

4.2 请给出对本次实验内容的建议

注：本章为酌情加分项。

参考文献

为完成本次实验你翻阅的书籍与网站等

[1] 林来兴. 空间控制技术[M]. 北京：中国宇航出版社，1992：25-42.

[2] 辛希孟. 信息技术与信息服务国际研讨会论文集：A 集[C]. 北京：中国科学出版社，1999.

[3] 赵耀东. 新时代的工业工程师[M/OL]. 台北：天下文化出版社，1998 [1998-09-26]. http://www.ie.nthu.edu.tw/info/ie.newie.htm（Big5）.

[4] 谌颖. 空间交会控制理论与方法研究[D]. 哈尔滨：哈尔滨工业大学，1992：8-13.

[5] KANAMORI H. Shaking Without Quaking[J]. Science，1998，279（5359）：2063-2064.

[6] CHRISTINE M. Plant Physiology: Plant Biology in the Genome Era[J/OL]. Science，1998，281：331-332[1998-09-23]. http://www.sciencemag.org/cgi/collection/anatmorp.