哈爾濱工業大學

实验报告

实 验（五）

题　　目　LinkLab

　　链接

专　　业　计算机类

学　　号　1190200523

班　　级　1903002

学　　生　石翔宇

指 导 教 师　郑贵滨

实 验 地 点　G709

实 验 日 期　2021.5.21

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、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

- Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz
- 16GB RAM
- 1TB HDD + 512G SSD

1.2.2 软件环境

- Windows 10 21H1
- Ubuntu 20.04 LTS

1.2.3 开发工具

- VSCode，CodeBlocks，gcc+gdb

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。
- 请按顺序写出 ELF 格式的可执行目标文件的各类信息。
- 请按照内存地址从低到高的顺序，写出 Linux 下 X64 内存映像。
- 请运行“LinkAddress -u 学号 姓名” 按地址顺序写出各符号的地址、空间。并按照 Linux 下 X64 内存映像结构，标出其所属各区。
  - gcc -m64 -o LinkAddress linkaddress.c

- 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)

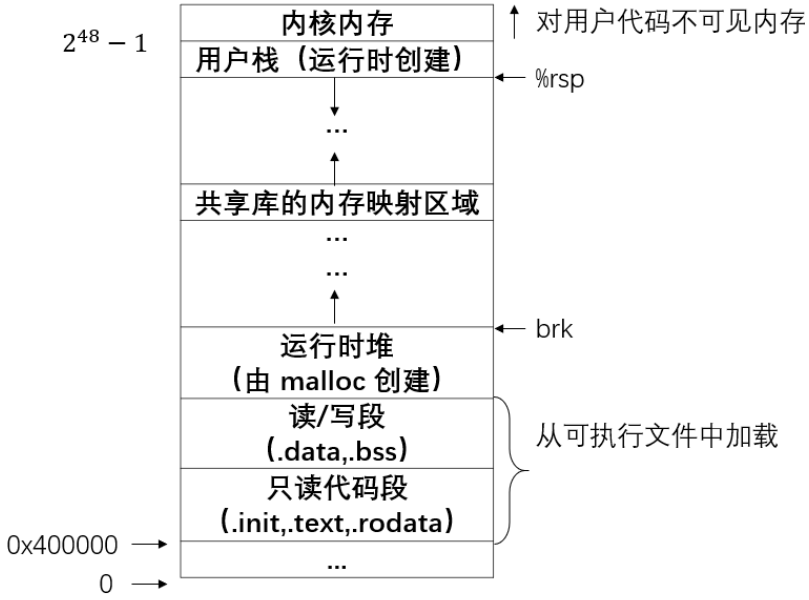
第 2 章 实验预习

2.1 ELF 文件格式解读

请按顺序写出 ELF 格式的可执行目标文件的各类信息（5 分）

2.2 程序的内存映像结构

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 程序中符号的位置分析

请运行“LinkAddress -u 学号 姓名” 按地址顺序写出各符号的地址，并按照 Linux 下 X64 内存映像标出其所属内存区段（5 分）

用户栈	env 0x7ffda06f26f0 140727295092464 env[0] *env 0x7ffda06f43c4 140727295099844 SHELL=/bin/bash env[1] *env 0x7ffda06f43d4 140727295099860 SESSION_MANAGER=local/ubuntu:@/tmp/.ICE-unix/1664,unix/ubuntu:/tmp/.ICE-unix/1664 env[2] *env 0x7ffda06f4426 140727295099942 QT_ACCESSIBILITY=1 env[3] *env 0x7ffda06f4439 140727295099961 COLORTERM=truecolor env[4] *env 0x7ffda06f444d 140727295099981 XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/etc/xdg env[5] *env 0x7ffda06f447a 140727295100026 XDG_MENU_PREFIX=gnome- env[6] *env 0x7ffda06f4491 140727295100049 GNOME_DESKTOP_SESSION_ID=this-is-deprecated env[7] *env 0x7ffda06f44bd 140727295100093 GNOME_SHELL_SESSION_MODE=ubuntu env[8] *env 0x7ffda06f44dd 140727295100125 SSH_AUTH_SOCK=/run/user/1000/keyring/ssh env[9] *env 0x7ffda06f4506 140727295100166 XMODIFIERS=@im=ibus env[10] *env 0x7ffda06f451a 140727295100186 DESKTOP_SESSION=ubuntu env[11] *env 0x7ffda06f4531 140727295100209 SSH_AGENT_PID=1628 env[12] *env 0x7ffda06f4544 140727295100228 GTK_MODULES=gail:atk-bridge env[13] *env 0x7ffda06f4560 140727295100256 DBUS_STARTER_BUS_TYPE=session env[14] *env 0x7ffda06f457e 140727295100286 PWD=/home/dedsec/CSAPP/Lab/Lab5 env[15] *env 0x7ffda06f459e 140727295100318 LOGNAME=dedsec env[16] *env 0x7ffda06f45ad 140727295100333 XDG_SESSION_DESKTOP=ubuntu env[17] *env 0x7ffda06f45c8 140727295100360 XDG_SESSION_TYPE=x11 env[18] *env 0x7ffda06f45dd 140727295100381 GPG_AGENT_INFO=/run/user/1000/gnupg/S.gpg-agent:0:1 env[19] *env 0x7ffda06f4611 140727295100433 XAUTHORITY=/run/user/1000/gdm/Xauthority env[20] *env 0x7ffda06f463a 140727295100474 WINDOWPATH=2 env[21] *env 0x7ffda06f4647 140727295100487 HOME=/home/dedsec env[22] *env 0x7ffda06f4659 140727295100505 USERNAME=dedsec env[23] *env 0x7ffda06f4669 140727295100521 IM_CONFIG_PHASE=1 env[24] *env 0x7ffda06f467b 140727295100539 LANG=en_US.UTF-8 env[25] *env 0x7ffda06f468c 140727295100556 LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;33;01:or=40;31;01:mi=00:su=37;41:sg=30;43:ca=30;41:tw=30;42:ow=34;42:st=37;44:ex=01;32:*.tar=01;31:*.tgz=01;31:*.arc=01;31:*.arj=01;31:*.taz=01;31:*.lha=01;31:*.lz4=01;31:*.lzh=01;31:*.lzma=01;31:*.tlz=01;31:*.txz=01;31:*.tzo=01;31:*.t7z=01;31:*.zip=01;31:*.z=01;31:*.dz=01;31:*.gz=01;31:*.lrz=01;31:*.lz=01;31:*.lzo=01;31:*.xz=01;31:*.zst=01;31:*.tzst=01;31:*.bz2=01;31:*.bz=01;31:*.tbz=01;31:*.tbz2=01;31:*.tz=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:*.war=01;31:*.ear=01;31:*.sar=01;31:*.rar=01;31:*.alz=01;31:*.ace=01;31:*.zoo=01;31:*.cpio=01;31:*.7z=01;31:*.rz=01;31:*.cab=01;31:*.wim=01;31:*.swm=01;31:*.dwm=01;31:*.esd=01;31:*.jpg=01;35:*.jpeg=01;35:*.mjpg=01;35:*.mjpeg=01;35:*.gif=01;35:*.bmp=01;35:*.pbm=01;35:*.pgm=01;35:*.ppm=01;35:*.tga=01;35:*.xbm=01;35:*.xpm=01;35:*.tif=01;35:*.tiff=01;35:*.png=01;35:*.svg=01;35:*.svgz=01;35:*.mng=01;35:*.pcx=01;35:*.mov=01;35:*.mpg=01;35:*.mpeg=01;35:*.m2v=01;35:*.mkv=01;35:*.webm=01;35:*.ogm=01;35:*.mp4=01;35:*.m4v=01;35:*.mp4v=01;35:*.vob=01;35:*.qt=01;35:*.nuv=01;35:*.wmv=01;35:*.asf=01;35:*.rm=01;35:*.rmvb=01;35:*.flc=01;35:*.avi=01;35:*.fli=01;35:*.flv=01;35:*.gl=01;35:*.dl=01;35:*.xcf=01;35:*.xwd=01;35:*.yuv=01;35:*.cgm=01;35:*.emf=01;35:*.ogv=01;35:*.ogx=01;35:*.aac=00;36:*.au=00;36:*.flac=00;36:*.m4a=00;36:*.mid=00;36:*.midi=00;36:*.mka=00;36:*.mp3=00;36:*.mpc=00;36:*.ogg=00;36:*.ra=00;36:*.wav=00;36:*.oga=00;36:*.opus=00;36:*.spx=00;36:*.xspf=00;36: env[26] *env 0x7ffda06f4c6e 140727295102062 XDG_CURRENT_DESKTOP=ubuntu:GNOME

	env[27] *env 0x7ffda06f4c8f 140727295102095 VTE_VERSION=6003 env[28] *env 0x7ffda06f4ca0 140727295102112 GNOME_TERMINAL_SCREEN=/org/gnome/Terminal/screen/2b10d8c9_37b1_4f96_918e_d0020abaeb61 env[29] *env 0x7ffda06f4cf6 140727295102198 INVOCATION_ID=83bcbcac7e0943bbb5049b2199ace651 env[30] *env 0x7ffda06f4d25 140727295102245 MANAGERPID=1454 env[31] *env 0x7ffda06f4d35 140727295102261 LESSCLOSE=/usr/bin/lesspipe %s %s env[32] *env 0x7ffda06f4d57 140727295102295 XDG_SESSION_CLASS=user env[33] *env 0x7ffda06f4d6e 140727295102318 TERM=xterm-256color env[34] *env 0x7ffda06f4d82 140727295102338 LESSOPEN=| /usr/bin/lesspipe %s env[35] *env 0x7ffda06f4da2 140727295102370 USER=dedsec env[36] *env 0x7ffda06f4dae 140727295102382 GNOME_TERMINAL_SERVICE=:1.183 env[37] *env 0x7ffda06f4dcc 140727295102412 DISPLAY=:0 env[38] *env 0x7ffda06f4dd7 140727295102423 SHLVL=1 env[39] *env 0x7ffda06f4ddf 140727295102431 QT_IM_MODULE=ibus env[40] *env 0x7ffda06f4df1 140727295102449 DBUS_STARTER_ADDRESS=unix:path=/run/user/1000/bus,guid=ffd78ad7b63b39dfef2dd062607fcd38 env[41] *env 0x7ffda06f4e49 140727295102537 XDG_RUNTIME_DIR=/run/user/1000 env[42] *env 0x7ffda06f4e68 140727295102568 JOURNAL_STREAM=8:49163 env[43] *env 0x7ffda06f4e7f 140727295102591 XDG_DATA_DIRS=/usr/share/ubuntu:/usr/local/share/:/usr/share/:/var/lib/snapd/desktop env[44] *env 0x7ffda06f4ed4 140727295102676 PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin env[45] *env 0x7ffda06f4f3c 140727295102780 GDMSESSION=ubuntu env[46] *env 0x7ffda06f4f4e 140727295102798 DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus,guid=ffd78ad7b63b39dfef2dd062607fcd38 env[47] *env 0x7ffda06f4faa 140727295102890 _=./LinkAddr env[48] *env 0x7ffda06f4fb7 140727295102903 OLDPWD=/home/dedsec/CSAPP/Lab/Lab5/linklab-1190200523 argc 0x7ffda06f258c 140727295092108 argv 0x7ffda06f26c8 140727295092424 argv[0] 7ffda06f43a1 argv[1] 7ffda06f43ac argv[2] 7ffda06f43af argv[3] 7ffda06f43ba argv[0] 0x7ffda06f43a1 140727295099809 ./LinkAddr argv[1] 0x7ffda06f43ac 140727295099820 -u argv[2] 0x7ffda06f43af 140727295099823 1190200523 argv[3] 0x7ffda06f43ba 140727295099834 石翔宇 local 0x7ffda06f2590 140727295092112
共享库的内存映射区域	exit 0x7f9624964bc0 140282835651520 printf 0x7f962497fe10 140282835762704 malloc 0x7f96249b8260 140282835993184 free 0x7f96249b8850 140282835994704
运行时堆	p1 0x7f961491a010 140282566909968 p3 0x7f96148f9010 140282566774800 p4 0x7f95d48f8010 140281493028880 p5 0x7f95548f7010 140279345541136
读/写段	big array 0x55f05ff94040 94490890682432 huge array 0x55f01ff94040 94489816940608 global 0x55f01ff9402c 94489816940588 p2 0x55f061b726b0 94490919904944

只读代码段	show_pointer 0x55f01fd9281a 94489814837274 useless 0x55f01fd9284d 94489814837325 main 0x55f01fd92858 94489814837336

2.4 程序运行过程分析

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(使用 gcc 与 objdump/GDB/EDB)（5 分）

从开始执行到 main 前：	_start __libc_start_main __GI___cxa_atexit __internal_atexit __lll_cas_lock __new_exitfn __libc_csu_init _init frame_dummy register_tm_clones __libc_csu_init _setjmp __sigsetjmp __sigjmp_save
main 后执行：	__GI_exit __run_exit_handlers __GI___call_tls_dtors __lll_cas_lock _IO_cleanup _IO_flush_all_lockp IO_validate_vtable _IO_unbuffer_all _IO_new_file_setbuf _IO_default_setbuf _IO_new_file_sync __GI__IO_setb

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：

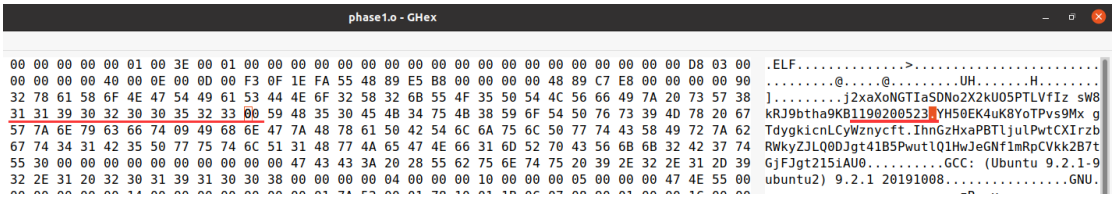
```
dedsec@ubuntu:~/CSAPP/Lab/Lab5$ gcc -o linkbomb1 main.o phase1.o -no-pie
dedsec@ubuntu:~/CSAPP/Lab/Lab5$ ./linkbomb1
1190200523
```

分析与设计的过程：

1. 将原文件编译运行得到结果：

```
dedsec@ubuntu:~/CSAPP/Lab/Lab5$ ./linkbomb1
5ZmttRyULxfYH50EK4uK8YoTPvs9Mx gTdygkicnLCyWznycft      IhnGzHxaPBTljulPwtCXIrzb
RWkyZJLQ0DJgt41B5PwutlQ1HwJeGNf1mRpCVkk2B7tGjFJgt215iAU0
```

2. 用 Linux 下的二进制文件编辑器 GHex 打开 phase1.o 文件，查找“5Zmtt……”字符串，将其修改为学号对应的 ASCII 的 16 进制，并将末尾改为 00



3. 修改成功，保存并重新编译后得到预期结果。

3.2 阶段 2 的分析

程序运行结果截图：

```
dedsec@ubuntu:~/CSAPP/Lab/Lab5$ gcc -o linkbomb2 main.o phase2.o -no-pie
dedsec@ubuntu:~/CSAPP/Lab/Lab5$ ./linkbomb2
1190200523
```

分析与设计的过程：

1. 将 phase2.o 反编译得知，我们需要在 do_phase 中替换 nop 代码，使得能够跳转到 jyrTGbdI，并且将学号作为参数传入。

```
0000000000000000 <jyrTGbdI>:
   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:   48 89 7d f8             mov    %rdi,-0x8(%rbp)
  10:   48 8b 45 f8             mov    -0x8(%rbp),%rax
  14:   be 00 00 00 00          mov    $0x0,%esi
  19:   48 89 c7                mov    %rax,%rdi
  1c:   e8 00 00 00 00          callq  21 <jyrTGbdI+0x21>
  21:   85 c0                   test   %eax,%eax
  23:   75 0e                   jne    33 <jyrTGbdI+0x33>
  25:   48 8b 45 f8             mov    -0x8(%rbp),%rax
  29:   48 89 c7                mov    %rax,%rdi
  2c:   e8 00 00 00 00          callq  31 <jyrTGbdI+0x31>
  31:   eb 01                   jmp    34 <jyrTGbdI+0x34>
  33:   90                      nop
  34:   c9                      leaveq
  35:   c3                      retq

0000000000000036 <do_phase>:
  36:   f3 0f 1e fa             endbr64
  3a:   55                      push   %rbp
  3b:   48 89 e5                mov    %rsp,%rbp
  3e:   90                      nop
  3f:   90                      nop
  40:   90                      nop
```

2. 在.rodata 段找到了学号，则只需将对应地址的内容当作参数传入函数即可

```
0000000000000000 <.rodata>:
   0:   31 31                   xor    %esi,(%rcx)
   2:   39 30                   cmp    %esi,(%rax)
   4:   32 30                   xor    (%rax),%dh
   6:   30                      .byte 0x30
   7:   35                      .byte 0x35
   8:   32 33                   xor    (%rbx),%dh
        ...
```

3. 将 linkbomb2 反编译得到学号的地址，为 0x40207c

```
  402079:       65 2e 00 31             gs add %dh,%cs:(%rcx)
  40207d:       31 39                   xor    %edi,(%rcx)
  40207f:       30 32                   xor    %dh,(%rdx)
  402081:       30 30                   xor    %dh,(%rax)
  402083:       35                      .byte 0x35
  402084:       32 33                   xor    (%rbx),%dh
```

4. 考虑代码 mov $0x40207c,%rdi，将其编译并反汇编得到对应的汇编指令 48 c7 c7 7c 20 40 00

5. 查找 callq PC 相对寻址的汇编指令代码为 e8，计算 0x4011db 到 0x401196 的 PC 相对地址差，为 ff ff ff b6，将其拼装为 e8 b6 ff ff ff。

6. 利用 GHex 将 nop 代码修改为上述代码，并保存。

```
00000056  00 00 00 48 89 C7 E8 00 00 00 00 85 C0 75 0E 48 8B 45 F8 48 89 C7 E8 00 00 00 00 EB 01 90 C9 C3 F3 0F 1E FA 55 48 89 E5 48 C7 C7
00000081  7C 20 40 00 E8 B6 FF FF FF 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 5D C3 00 00 00 00 00 00 00 00 00 00 00 00
```

7. 修改成功，保存并重新编译后得到预期结果。

3.3 阶段 3 的分析

程序运行结果截图：

```
dedsec@ubuntu:~/CSAPP/Lab/Lab5$ gcc -c -o phase3_patch.o phase3_patch.c
dedsec@ubuntu:~/CSAPP/Lab/Lab5$ gcc -o linkbomb3 main.o phase3.o phase3_patch.o -no-pie
dedsec@ubuntu:~/CSAPP/Lab/Lab5$ ./linkbomb3
1190200523
```

分析与设计的过程：

1. 将 phase3.o 与 main.o 链接编译并反汇编可得 linkbomb3 的反汇编代码。

2. 分析 do_phase 的反汇编代码可知：do_phase 会输出 10 个字符。这些字符的起始位置是 0x404060，10 个字符的位置偏移分别是：6d 6f 67 6b 63 78 76 69 73 61

```
408   4011d1:   48 b8 6d 6f 67 6b 63   movabs $0x697678636b676f6d,%rax
409   4011d8:   78 76 69
410   4011db:   48 89 45 ed            mov    %rax,-0x13(%rbp)
411   4011df:   66 c7 45 f5 73 61      movw   $0x6173,-0xb(%rbp)
412   4011e5:   c6 45 f7 00            movb   $0x0,-0x9(%rbp)
413   4011e9:   c7 45 e8 00 00 00 00   movl   $0x0,-0x18(%rbp)
414   4011f0:   eb 24                  jmp    401216 <do_phase+0x60>
415   4011f2:   8b 45 e8               mov    -0x18(%rbp),%eax
416   4011f5:   48 98                  cltq
417   4011f7:   0f b6 44 05 ed         movzbl -0x13(%rbp,%rax,1),%eax
418   4011fc:   0f b6 c0               movzbl %al,%eax
419   4011ff:   48 98                  cltq
420   401201:   0f b6 80 60 40 40 00   movzbl 0x404060(%rax),%eax
421   401208:   0f be c0               movsbl %al,%eax
422   40120b:   89 c7                  mov    %eax,%edi
423   40120d:   e8 4e fe ff ff         callq  401060 <putchar@plt>
```

3. 由 readelf linkbomb3 -a 可知，一个名为 kkXgvmmBoA 的长度为 256 个字节的数组起始地址就是 0x404060

```
51: 0000000000404160     0 NOTYPE  GLOBAL DEFAULT   25 _edata
52: 00000000004012b8     0 FUNC    GLOBAL HIDDEN    16 _fini
53: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND __stack_chk_fail@@GLIBC_2
54: 0000000000404060   256 OBJECT  GLOBAL DEFAULT   25 kkXgvmmBoA
55: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND __libc_start_main@@GLIBC_
56: 0000000000404040     0 NOTYPE  GLOBAL DEFAULT   25 __data_start
57: 00000000004011b6   137 FUNC    GLOBAL DEFAULT   15 do_phase
58: 0000000000000000     0 NOTYPE  WEAK   DEFAULT  UND __gmon_start__
59: 0000000000404048     0 OBJECT  GLOBAL HIDDEN    25 __dso_handle
60: 0000000000402000     4 OBJECT  GLOBAL DEFAULT   17 _IO_stdin_used
61: 0000000000401240   101 FUNC    GLOBAL DEFAULT   15 __libc_csu_init
```

4. 那么我们可以编写 C 代码，对 kkXgvmmBoA 数组赋初值。由于我们编写的代码中 kkXgvmmBoA 数组是强符号，链接后原代码中访问到的 0x404060 地址中的内容就是我们赋值的了。

5. 我们构造 kkXgvmmBoA 数组的内容，其中第 109、111、103、107、99、120、118、105、115、97 个字符（下标从 0 开始）分别为’1’、’1’、’9’、’0’、’2’、’0’、’0’、’5’、’2’、’3’。

6. 得到 phase3_patch.c 的代码：

```
char kkXgvmmBoA[256] = {'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a',
'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a',
'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a',
'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a',
'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a',

'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a',
'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a',
'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a',
'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a',
'a', 'a', 'a', 'a', 'a', 'a', 'a', '3', 'a', '2',

'a', 'a', 'a', '9', 'a', '5', 'a', '0', 'a', '1',
'a', '1', 'a', 'a', 'a', '2', 'a', 'a', '0', 'a',
'0', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a'};
/*
6d 6f 67 6b 63 78 76 69 73 61
109 111 103 107 99 120 118 105 115 97
*/
```

7. 将 phase3_patch.c 编译得到 phase3_patch.o，再与 phase3.o、main.o 编译链接得到新的 linkbomb3，运行得到预期结果。

3.4 阶段 4 的分析

程序运行结果截图：

```
dedsec@ubuntu:~/CSAPP/Lab/Lab5$ gcc -o linkbomb4 main.o phase4.o -no-pie
dedsec@ubuntu:~/CSAPP/Lab/Lab5$ ./linkbomb4
1190200523
```

分析与设计的过程：

1. 将 phase3.o 反汇编。

2. 分析 do_phase 的反汇编代码可知：do_phase 首先会根据地址表来跳转到相应的代码位置。地址表共有 10 个地址，这些地址的位置偏移分别是：0x6、0x18、0x2、0x14、0x19、0x4、0x10、0x16、0x12、0xf（原值减去 0x41）。转成十进制则为 6、24、2、20、25、4、16、22、18、15。

```
14  15:  48 89 45 f8           mov    %rax,-0x8(%rbp)
15  19:  31 c0                 xor    %eax,%eax
16  1b:  48 b8 47 59 43 55 5a  movabs $0x5751455a55435947,%rax
17  22:  45 51 57
18  25:  48 89 45 ed           mov    %rax,-0x13(%rbp)
19  29:  66 c7 45 f5 53 50     movw   $0x5053,-0xb(%rbp)
20  2f:  c6 45 f7 00           movb   $0x0,-0x9(%rbp)
21  33:  c7 45 e8 00 00 00 00  movl   $0x0,-0x18(%rbp)
22  3a:  e9 e0 00 00 00        jmpq   11f <do_phase+0x11f>
23  3f:  8b 45 e8              mov    -0x18(%rbp),%eax
24  42:  48 98                 cltq
25  44:  0f b6 44 05 ed        movzbl -0x13(%rbp,%rax,1),%eax
26  49:  88 45 e7              mov    %al,-0x19(%rbp)
27  4c:  0f be 45 e7           movsbl -0x19(%rbp),%eax
28  50:  83 e8 41              sub    $0x41,%eax
29  53:  83 f8 19              cmp    $0x19,%eax
30  56:  0f 87 b4 00 00 00     ja     110 <do_phase+0x110>
31  5c:  89 c0                 mov    %eax,%eax
32  5e:  48 8b 04 c5 00 00 00  mov    0x0(,%rax,8),%rax
33  65:  00
34  66:  3e ff e0              notrack jmpq *%rax
```

3. 由 phase3.o 的 ELF 表可知每个偏移量对应的跳转代码位置，也就知道了实际上要输出的内容。

```
Relocation section '.rela.rodata' at offset 0x4e8 contains 26 entries:
  Offset          Info           Type           Sym. Value    Sym. Name + Addend
000000000000  000200000001 R_X86_64_64       0000000000000000 .text + 69
000000000008  000200000001 R_X86_64_64       0000000000000000 .text + 72
000000000010  000200000001 R_X86_64_64       0000000000000000 .text + 7b
000000000018  000200000001 R_X86_64_64       0000000000000000 .text + 84
000000000020  000200000001 R_X86_64_64       0000000000000000 .text + 8d
000000000028  000200000001 R_X86_64_64       0000000000000000 .text + 93
000000000030  000200000001 R_X86_64_64       0000000000000000 .text + 99
000000000038  000200000001 R_X86_64_64       0000000000000000 .text + 9f
000000000040  000200000001 R_X86_64_64       0000000000000000 .text + a5
000000000048  000200000001 R_X86_64_64       0000000000000000 .text + ab
000000000050  000200000001 R_X86_64_64       0000000000000000 .text + b1
000000000058  000200000001 R_X86_64_64       0000000000000000 .text + b7
000000000060  000200000001 R_X86_64_64       0000000000000000 .text + bd
000000000068  000200000001 R_X86_64_64       0000000000000000 .text + c3
000000000070  000200000001 R_X86_64_64       0000000000000000 .text + c9
000000000078  000200000001 R_X86_64_64       0000000000000000 .text + cf
000000000080  000200000001 R_X86_64_64       0000000000000000 .text + d5
000000000088  000200000001 R_X86_64_64       0000000000000000 .text + db
000000000090  000200000001 R_X86_64_64       0000000000000000 .text + e1
000000000098  000200000001 R_X86_64_64       0000000000000000 .text + e7
0000000000a0  000200000001 R_X86_64_64       0000000000000000 .text + ed
0000000000a8  000200000001 R_X86_64_64       0000000000000000 .text + f3
0000000000b0  000200000001 R_X86_64_64       0000000000000000 .text + f9
0000000000b8  000200000001 R_X86_64_64       0000000000000000 .text + ff
0000000000c0  000200000001 R_X86_64_64       0000000000000000 .text + 105
0000000000c8  000200000001 R_X86_64_64       0000000000000000 .text + 10b
```

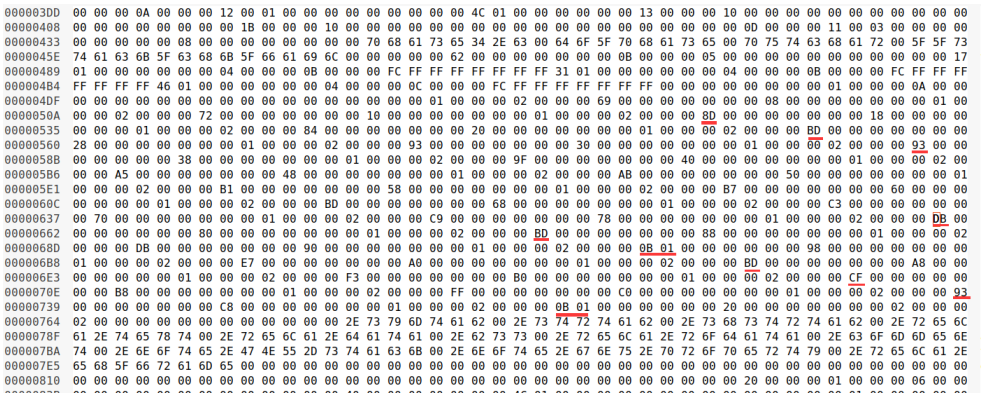
4. 我们在 do_phase 中找到拼成学号所需代码的位置，分别为 0x93、0x93、0x8d、0xbd、0x10b、0xbd、0xbd、0xcf、0x10b、0xdb。

```
43   8d:  c6 45 e7 39        movb   $0x39,-0x19(%rbp)     9
44   91:  eb 7d              jmp    110 <do_phase+0x110>
45   93:  c6 45 e7 31        movb   $0x31,-0x19(%rbp)     1
46   97:  eb 77              jmp    110 <do_phase+0x110>
47   99:  c6 45 e7 54        movb   $0x54,-0x19(%rbp)
48   9d:  eb 71              jmp    110 <do_phase+0x110>
49   9f:  c6 45 e7 54        movb   $0x54,-0x19(%rbp)
50   a3:  eb 6b              jmp    110 <do_phase+0x110>
51   a5:  c6 45 e7 47        movb   $0x47,-0x19(%rbp)
52   a9:  eb 65              jmp    110 <do_phase+0x110>
53   ab:  c6 45 e7 5e        movb   $0x5e,-0x19(%rbp)
54   af:  eb 5f              jmp    110 <do_phase+0x110>
55   b1:  c6 45 e7 37        movb   $0x37,-0x19(%rbp)
56   b5:  eb 59              jmp    110 <do_phase+0x110>
57   b7:  c6 45 e7 52        movb   $0x52,-0x19(%rbp)
58   bb:  eb 53              jmp    110 <do_phase+0x110>
59   bd:  c6 45 e7 30        movb   $0x30,-0x19(%rbp)     0
60   c1:  eb 4d              jmp    110 <do_phase+0x110>
61   c3:  c6 45 e7 73        movb   $0x73,-0x19(%rbp)
62   c7:  eb 47              jmp    110 <do_phase+0x110>
63   c9:  c6 45 e7 50        movb   $0x50,-0x19(%rbp)
64   cd:  eb 41              jmp    110 <do_phase+0x110>
65   cf:  c6 45 e7 35        movb   $0x35,-0x19(%rbp)     5
66   d3:  eb 3b              jmp    110 <do_phase+0x110>
67   d5:  c6 45 e7 75        movb   $0x75,-0x19(%rbp)
68   d9:  eb 35              jmp    110 <do_phase+0x110>
69   db:  c6 45 e7 33        movb   $0x33,-0x19(%rbp)     3
70   df:  eb 2f              jmp    110 <do_phase+0x110>
71   e1:  c6 45 e7 64        movb   $0x64,-0x19(%rbp)
72   e5:  eb 29              jmp    110 <do_phase+0x110>
73   e7:  c6 45 e7 6b        movb   $0x6b,-0x19(%rbp)
74   eb:  eb 23              jmp    110 <do_phase+0x110>
75   ed:  c6 45 e7 66        movb   $0x66,-0x19(%rbp)
76   f1:  eb 1d              jmp    110 <do_phase+0x110>
77   f3:  c6 45 e7 4e        movb   $0x4e,-0x19(%rbp)
78   f7:  eb 17              jmp    110 <do_phase+0x110>
79   f9:  c6 45 e7 59        movb   $0x59,-0x19(%rbp)
80   fd:  eb 11              jmp    110 <do_phase+0x110>
81   ff:  c6 45 e7 74        movb   $0x74,-0x19(%rbp)
82  103:  eb 0b              jmp    110 <do_phase+0x110>
83  105:  c6 45 e7 34        movb   $0x34,-0x19(%rbp)
84  109:  eb 05              jmp    110 <do_phase+0x110>
85  10b:  c6 45 e7 32        movb   $0x32,-0x19(%rbp)     2
```

5. 即将 ELF 表中第 6、24、2、20、25、4、16、22、18、15 个地址（注意下

标从 0 开始）的值分别修改为 0x93、0x93、0x8d、0xbd、0x10b、0xbd、0xbd、0xcf、0x10b、0xdb。

6. 利用二进制编辑器 GHex，将地址表中的内容修改为上述内容。



7. 修改成功，保存并重新编译后得到预期结果。

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

1. 更加熟悉了 ELF 结构；
2. 对程序的运行周期有了更深的理解；
3. 对 readelf、objdump、gdb 等工具的运用更加熟练。

4.2 请给出对本次实验内容的建议

1. 希望实验指引能够更加具体。

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.