哈爾濱工業大學

实验报告

实 验（四）

题　　目 LinkLab

链接

专　　业 计算学部

学　　号 _____

班　　级 _____

学　　生 _____

指 导 教 师 _____

实 验 地 点 _____

实 验 日 期 _____

计算机科学与技术学院

目 录

第 1 章 实验基本信息 - 3 -

1.1 实验目的 - 3 -
1.2 实验环境与工具 - 3 -
1.2.1 硬件环境 - 3 -
1.2.2 软件环境 - 3 -
1.2.3 开发工具 - 3 -
1.3 实验预习 - 3 -

第 2 章 实验预习 - 4 -

2.1 ELF 文件格式解读 - 4 -
2.2 程序的内存映像结构 - 4 -
2.3 程序中符号的位置分析 - 5 -
2.4 程序运行过程分析 - 7 -

第 3 章 各阶段的原理与方法 - 8 -

3.1 阶段 1 的分析 - 8 -
3.2 阶段 2 的分析 - 9 -
3.3 阶段 3 的分析 - 11 -
3.4 阶段 4 的分析 - 13 -
3.5 阶段 5 的分析 - 13 -

第 4 章 总结 - 14 -

4.1 请总结本次实验的收获 - 14 -
4.2 请给出对本次实验内容的建议 - 14 -

参考文献 - 15 -

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

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。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 内存映像标出其所属各区。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）

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

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）

内核内存
用户栈 （运行时 创建）
（栈-向下） ... （映射区域-向上）
共享库的内存映射区域
... （堆-向上）
运行时堆 （由 malloc 创建）
读/写段 （.data,.bss）

只读代码段 （.init,.text,.rodata）
……

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）

所属区	各符号的地址、空间（地址从小到大）
只读代码段 （.init , .text , .rodata ）	show_pointer 0x5596072bb1b3 94102853759411 useless 0x5596072bb1a9 94102853759401 main 0x5596072bb1df 94102853759455 exit 0x7f3ea6e84be0 139907064941536 printf 0x7f3ea6e9fd90 139907065052560 malloc 0x7f3ea6ed7560 139907065279840 free 0x7f3ea6ed7b70 139907065281392
读写段 （.data .bss）	big array 0x5596472be060 94103927513184 huge array 0x5596072be060 94102853771360 local 0x7ffdc8217cd4 140727961091284 global 0x5596072be040 94102853771328
运行时堆	p1 0x7f3e96e3d010 139906796212240 p2 0x5596499e66b0 94103968573104 p3 0x7f3e96e1c010 139906796077072 p4 0x7f3e56e1b010 139905722331152 p5 0x7f3dd6e1a010 139903574843408
用户栈（运行时创建）	argc 0x7ffdc8217ccc140727961091276 argv 0x7ffdc8217e08 140727961091592 argv[0] 7ffdc821934c argv[1] 7ffdc8219354 argv[2] 7ffdc8219357 argv[3] 7ffdc8219362 argv[0] 0x7ffdc821934c 140727961097036 ./a.out argv[1] 0x7ffdc8219354 140727961097044 -u argv[2] 0x7ffdc8219357 140727961097047 1190300321 argv[3] 0x7ffdc8219362 140727961097058 郑晟赫 env 0x7ffdc8217e30 140727961091632 env[0] *env 0x7ffdc821936c 140727961097068 SHELL=/bin/bash env[1] *env 0x7ffdc821937c 140727961097084 SESSION_MANAGER=local/zsh-virtual-machine:@/tmp/.ICE-unix/3605,unix/zsh-virtual-machine:/tmp/.ICE-unix/3605 env[2] *env 0x7ffdc82193e8 140727961097192 QT_ACCESSIBILITY=1 env[3] *env 0x7ffdc82193fb140727961097211 COLORTERM=truecolor env[4] *env 0x7ffdc821940f140727961097231 XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/etc/xdg env[5] *env 0x7ffdc821943c 140727961097276 XDG_MENU_PREFIX=gnome- env[6] *env 0x7ffdc8219453 140727961097299 GNOME_DESKTOP_SESSION_ID=this-is-deprecated env[7] *env 0x7ffdc821947f140727961097343 LANGUAGE=zh_CN:zh:en_US:en env[8] *env 0x7ffdc821949a 140727961097370 LC_ADDRESS=zh_CN.UTF-8 env[9] *env 0x7ffdc82194b1 140727961097393 GNOME_SHELL_SESSION_MODE=ubuntu

env[10] *env 0x7ffdc82194d1 140727961097425
LC_NAME=zh_CN.UTF-8
env[11] *env 0x7ffdc82194e5 140727961097445
SSH_AUTH_SOCK=/run/user/1000/keyring/ssh
env[12] *env 0x7ffdc821950e 140727961097486
XMODIFIERS=@im=ibus
env[13] *env 0x7ffdc8219522 140727961097506
DESKTOP_SESSION=ubuntu
env[14] *env 0x7ffdc8219539 140727961097529
LC_MONETARY=zh_CN.UTF-8
env[15] *env 0x7ffdc8219551 140727961097553
SSH_AGENT_PID=2945
env[16] *env 0x7ffdc8219564 140727961097572
GTK_MODULES=gail:atk-bridge
env[17] *env 0x7ffdc8219580 140727961097600
PWD=/home/zsh/code/c/lab5
env[18] *env 0x7ffdc821959a 140727961097626
LOGNAME=zsh
env[19] *env 0x7ffdc82195a6 140727961097638
XDG_SESSION_DESKTOP=ubuntu
env[20] *env 0x7ffdc82195c1 140727961097665
XDG_SESSION_TYPE=x11
env[21] *env 0x7ffdc82195d6 140727961097686
GPG_AGENT_INFO=/run/user/1000/gnupg/S.gpg-agent:0:1
env[22] *env 0x7ffdc821960a 140727961097738
XAUTHORITY=/run/user/1000/gdm/Xauthority
env[23] *env 0x7ffdc8219633 140727961097779
WINDOWPATH=2
env[24] *env 0x7ffdc8219640 140727961097792
HOME=/home/zsh
env[25] *env 0x7ffdc821964f140727961097807
USERNAME=zsh
env[26] *env 0x7ffdc821965c 140727961097820
IM_CONFIG_PHASE=1
env[27] *env 0x7ffdc821966e 140727961097838
LC_PAPER=zh_CN.UTF-8
env[28] *env 0x7ffdc8219683 140727961097859
LANG=zh_CN.UTF-8
env[29] *env 0x7ffdc8219694 140727961097876
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
.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:*.ca
35:*.webp=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:*.
env[30] *env 0x7ffdc8219c83 140727961099395
XDG_CURRENT_DESKTOP=ubuntu:GNOME
env[31] *env 0x7ffdc8219ca4 140727961099428
VTE_VERSION=6200
env[32] *env 0x7ffdc8219cb5 140727961099445
G_ENABLE_DIAGNOSTIC=0
env[33] *env 0x7ffdc8219ccb 140727961099467
GNOME_TERMINAL_SCREEN=/org/gnome/Terminal/screen/67a9e97e_8855_45b9_a944_66f2ed86868d
env[34] *env 0x7ffdc8219d21 140727961099553
INVOCATION_ID=27d777da66614f0c8b1cff836b19d691
env[35] *env 0x7ffdc8219d50 140727961099600
MANAGERPID=2506
env[36] *env 0x7ffdc8219d60 140727961099616
LESSCLOSE=/usr/bin/lesspipe %s %s
env[37] *env 0x7ffdc8219d82 140727961099650
XDG_SESSION_CLASS=user
env[38] *env 0x7ffdc8219d99 140727961099673
TERM=xterm-256color
env[39] *env 0x7ffdc8219dad 140727961099693
LC_IDENTIFICATION=zh_CN.UTF-8
env[40] *env 0x7ffdc8219dcb 140727961099723
LESSOPEN=| /usr/bin/lesspipe %s
env[41] *env 0x7ffdc8219deb 140727961099755
USER=zsh
env[42] *env 0x7ffdc8219df4140727961099764
GNOME_TERMINAL_SERVICE=:1.80
env[43] *env 0x7ffdc8219e11 140727961099793

	DISPLAY=:0 env[44] *env 0x7ffdc8219e1c 140727961099804 SHLVL=1 env[45] *env 0x7ffdc8219e24 140727961099812 LC_TELEPHONE=zh_CN.UTF-8 env[46] *env 0x7ffdc8219e3d 140727961099837 QT_IM_MODULE=ibus env[47] *env 0x7ffdc8219e4f140727961099855 LC_MEASUREMENT=zh_CN.UTF-8 env[48] *env 0x7ffdc8219e6a 140727961099882 XDG_RUNTIME_DIR=/run/user/1000 env[49] *env 0x7ffdc8219e89 140727961099913 LC_TIME=zh_CN.UTF-8 env[50] *env 0x7ffdc8219e9d 140727961099933 JOURNAL_STREAM=8:103597 env[51] *env 0x7ffdc8219eb5 140727961099957 XDG_DATA_DIRS=/usr/share/ubuntu:/usr/local/share/:/usr/share/:/var/lib/snapd/desktop env[52] *env 0x7ffdc8219f0a140727961100042 PATH=/home/zsh/.local/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin env[53] *env 0x7ffdc8219f87140727961100167 GDMSESSION=ubuntu env[54] *env 0x7ffdc8219f99140727961100185 DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus env[55] *env 0x7ffdc8219fcf 140727961100239 LC_NUMERIC=zh_CN.UTF-8 env[56] *env 0x7ffdc8219fe6140727961100262 _=./a.out

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）

时间段	程序
Main 函数执行前	Ld-2.27.so!_dl_start Ld-2.27.so!_dl_init Libc-2.27.so!_cxa_atexit Linkaddress!_init Linkaddress!_register_tm_clones Libc-2.27.so!_setjmp Libc2.27.so!__sigsetjmp Libc2.27.so!__sigjmpsave
Main 函数执行之后	Linkaddress!puts@plt Linkaddress!useless@plt Linkaddress!showpointer@plt malloc Linkaddress!.plt Libc-2.27.so!exit

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：

```
zsh@zsh-virtual-machine:~/code/c/lab5$ gcc -m32 -o linkbomb main.o phase1.o
/usr/bin/ld: main.o: warning: relocation in read-only section `.text'
/usr/bin/ld: warning: creating DT_TEXTREL in a PIE
zsh@zsh-virtual-machine:~/code/c/lab5$ ./linkbomb
1190300321
```

分析与设计的过程：

首先我们将没有修改过的 phase1.o 和 main.o 链接，并输出结果，可以发现是一段乱码，我们猜想如果把这一段乱码改成学号就可以输出我们需要的结果。

```
zsh@zsh-virtual-machine:~/code/c/lab5$ gcc -m32 -o linkbomb main.o phase1.o
/usr/bin/ld: main.o: warning: relocation in read-only section `.text'
/usr/bin/ld: warning: creating DT_TEXTREL in a PIE
zsh@zsh-virtual-machine:~/code/c/lab5$ ./linkbomb
hsWeHJoTK5mFBS1UpDHir   mvUbVZjxOzRIMBdHoFOtO5INreJ     NrQlz4drS 0YouNTFvIxoE4h
9li90 ArggaKP3v96tp6CGxgJLkSLT
```

因此我们可以使用 hexedit 读取 phase.o 的信息，可以发现一段字符串如下：

```
0x060  3170 334E 714E 554A 5453 7938 7309 7771  1p3NqNUJTSy8s.wq
0x070  5120 426B 4854 4375 5746 454C 7248 356D  Q BkHTCuWFELrH5m
0x080  536E 7459 746F 7043 4375 6E54 3553 5245  SntYtopCCunT5SRE
0x090  5A61 7930 3773 7649 5353 3355 564B 5756  Zay07svISS3UVKWV
0x0A0  366E 676F 5A4A 5254 6B4A 6743 6873 5748  6ngoZJRTkJgChsWH
0x0B0  4A6F 544B 356D 4646 4253 3155 7044 4869  JoTK5mFFBS1UpDHi
0x0C0  7209 6D76 5562 565A 6A78 4F7A 5249 4D42  r.mvUbVZjxOzRIMB
0x0D0  6448 6F46 4F74 4F35 494E 7265 4A09 4E72  dHoFOtO5INreJ.Nr
0x0E0  516C 7A34 6472 5320 3059 6F75 4E54 4676  Qlz4drS 0YouNTFv
0x0F0  4978 6F45 3468 396C 6939 3020 4172 6767  IxoE4h9li90 Argg
0x100  614B 5033 7639 3674 7036 4347 7867 4A4C  aKP3v96tp6CGxgJL
0x110  6B53 4C54 0000 0000 0000 0000 0047 4343  kSLT.........GCC
```

我们发现这里有一段字符串和输出的字符串是一致的，那么就可以猜测将这一段字符串的前面一部分改成学号即可，而学号最后需要一个 0x00 标志字符串输出结束，那么我们可以得到如下的修改结果，

```
3170 334E 714E 554A 5453 7938 7309 7771  1p3NqNUJTSy8s.wq
5120 426B 4854 4375 5746 454C 7248 356D  Q BkHTCuWFELrH5m
536E 7459 746F 7043 4375 6E54 3553 5245  SntYtopCCunT5SRE
5A61 7930 3773 7649 5353 3355 564B 5756  Zay07svISS3UVKWV
366E 676F 5A4A 5254 6B4A 6743 3131 3930  6ngoZJRTkJgC1190
3330 3033 3231 0046 4253 3155 7044 4869  300321.FBS1UpDHi
7209 6D76 5562 565A 6A78 4F7A 5249 4D42  r.mvUbVZjxOzRIMB
6448 6F46 4F74 4F35 494E 7265 4A09 4E72  dHoFOtO5INreJ.Nr
516C 7A34 6472 5320 3059 6F75 4E54 4676  Qlz4drS 0YouNTFv
4978 6F45 3468 396C 6939 3020 4172 6767  IxoE4h9li90 Argg
614B 5033 7639 3674 7036 4347 7867 4A4C  aKP3v96tp6CGxgJL
6B53 4C54 0000 0000 0000 0000 0047 4343  kSLT.........GCC
```

将修改过的 phase1.o 和 main.o 链接即可得到结果。

3.2 阶段 2 的分析

程序运行结果截图：

```
zsh@zsh-virtual-machine:~/code/c/lab5$ gcc -m32 -o linkbomb2 main.o phase2.o -no
-pie
zsh@zsh-virtual-machine:~/code/c/lab5$ ./linkbomb2
1190300321
```

分析与设计的过程：

首先，根据 ppt 提示这一问我们需要修改.text 的内容，那么首先我们先查看 phase2.o 的反汇编如下，我们可以发现我们需要调用的函数的逻辑是根据一个输入参数和一个事先定义好的变量进行比较，如果相等的话就输出，如果不相等的话就不输出并退出函数。这也和 PPT 上的提示相一致。

```
00000000 <lpqWzYWG>:
   0:   f3 0f 1e fb             endbr32
   4:   55                      push   %ebp
   5:   89 e5                   mov    %esp,%ebp
   7:   83 ec 08                sub    $0x8,%esp
   a:   83 ec 08                sub    $0x8,%esp
   d:   68 00 00 00 00          push   $0x0
  12:   ff 75 08                pushl  0x8(%ebp)
  15:   e8 fc ff ff ff          call   16 <lpqWzYWG+0x16>
  1a:   83 c4 10                add    $0x10,%esp
  1d:   85 c0                   test   %eax,%eax
  1f:   75 10                   jne    31 <lpqWzYWG+0x31>
  21:   83 ec 0c                sub    $0xc,%esp
  24:   ff 75 08                pushl  0x8(%ebp)
  27:   e8 fc ff ff ff          call   28 <lpqWzYWG+0x28>
  2c:   83 c4 10                add    $0x10,%esp
  2f:   eb 01                   jmp    32 <lpqWzYWG+0x32>
  31:   90                      nop
  32:   c9                      leave
  33:   c3                      ret
```

由于我们需要输出的是学号，因此我们可以大胆猜测已经定义好的变量存储的就是我们的学号，我们可以进行验证。首先将 phase2.o 和 main.o 进行链接之后查看其中的一些变量的值。

```
080491c6 <lpqWzYWG>:
 80491c6:       f3 0f 1e fb             endbr32
 80491ca:       55                      push   %ebp
 80491cb:       89 e5                   mov    %esp,%ebp
 80491cd:       83 ec 08                sub    $0x8,%esp
 80491d0:       83 ec 08                sub    $0x8,%esp
 80491d3:       68 7c a0 04 08          push   $0x804a07c
 80491d8:       ff 75 08                pushl  0x8(%ebp)
 80491db:       e8 60 fe ff ff          call   8049040 <strcmp@plt>
 80491e0:       83 c4 10                add    $0x10,%esp
 80491e3:       85 c0                   test   %eax,%eax
 80491e5:       75 10                   jne    80491f7 <lpqWzYWG+0x31>
 80491e7:       83 ec 0c                sub    $0xc,%esp
 80491ea:       ff 75 08                pushl  0x8(%ebp)
 80491ed:       e8 5e fe ff ff          call   8049050 <puts@plt>
 80491f2:       83 c4 10                add    $0x10,%esp
 80491f5:       eb 01                   jmp    80491f8 <lpqWzYWG+0x32>
 80491f7:       90                      nop
 80491f8:       c9                      leave
 80491f9:       c3                      ret
```

可以发现我们需要调用的函数在进行链接之后的反汇编如上图所示，我们发现它向栈中压入了两个值，一个是 0x804a07c，另外一个是 0x8(%ebp)，那么显然前一个是已经定义好值的变量，后一个使我们需要输入的变量，我们可以使用 gdb 查看一下前一个变量存储的值，如下：

```
(gdb) x/s 0x804a07c
0x804a07c:      "1190300321"
```

我们发现这个值就是我们需要输出的信息，因此下一步我们只需要将 0x8(%ebp)也修改成这个值就可以了，因此下一步我们可以开始修改 do_phase 函数的内容。这个函数有两部分功能：首先是将学号存储的内存地址存储在一个寄存器之后压栈，第二步是调用 lpqWzYWG 函数，因此我们可以写出我们需要加入的汇编代码如下：

```
1 movl $0x804a07c,%eax
2 push %eax
3 call 0xffffffbe
4 pop %eax
```

主要的思路就是直接将%eax 指向存储了我们学号的地址，并将其压栈，这样的话调用我们的函数的时候 0x8(%ebp)指向的地址就是存储了我们学号的地址，这样比较的话就可以成功。第三行的思路是首先找到 do_phase 函数在 phase2.o 中的地址是 0x34，而 lpqWzYWG 的地址是 0x00，二者的差值是 0xffffffbe，因此我们用这个指令可以直接调用 lpqWzYWG。到此我们的汇编代码编写完成，以下就是修改 do_phase 部分。

上一步生成的机器码如下：

```
00000000 <.text>:
   0:   b8 7c a0 04 08          mov    $0x804a07c,%eax
   5:   50                      push   %eax
   6:   e8 ba ff ff ff          call   0xffffffc5
   b:   58                      pop    %eax
```

我们只需要将生成的机器码填写到 phase2.o 中的 nop 部分即可，我们可以得到相关部分修改后的机器码如下：

```
0x069 0F1E FB55 89E5 B87C A004 0850 E8BA FFFF FF58 9090 9090 9090 9090 9090 9090 9090 9090 90
```

这时候我们的 phase2.o 也就修改完成了，只需要将它与 main.o 链接之后生成 linkbomb2 即可，如下图所示：

```
zsh@zsh-virtual-machine:~/code/c/lab5$ gcc -m32 -o linkbomb2 main.o phase2.o -no
-pie
zsh@zsh-virtual-machine:~/code/c/lab5$ ./linkbomb2
1190300321
```

3.3 阶段 3 的分析

程序运行结果截图：

```
zsh@zsh-virtual-machine:~/code/c/lab5$ gcc -m32 -o linkbomb3 main.o phase3.o pha
se3_patch.o -no-pie
zsh@zsh-virtual-machine:~/code/c/lab5$ ./linkbomb3
1190300321
```

分析与设计的过程：

首先我们得知我们需要做的就是符号解析，我们可以先看一下 do_phase 的反汇编代码，如下：

```
(gdb) x/20i do_phase
   0x80491d6 <do_phase>:      endbr32
   0x80491da <do_phase+4>:    push   %ebp
   0x80491db <do_phase+5>:    mov    %esp,%ebp
   0x80491dd <do_phase+7>:    sub    $0x28,%esp
   0x80491e0 <do_phase+10>:   mov    %gs:0x14,%eax
   0x80491e6 <do_phase+16>:   mov    %eax,-0xc(%ebp)
   0x80491e9 <do_phase+19>:   xor    %eax,%eax
   0x80491eb <do_phase+21>:   movl   $0x717a6564,-0x17(%ebp)
   0x80491f2 <do_phase+28>:   movl   $0x6a776672,-0x13(%ebp)
   0x80491f9 <do_phase+35>:   movw   $0x7074,-0xf(%ebp)
   0x80491ff <do_phase+41>:   movb   $0x0,-0xd(%ebp)
   0x8049203 <do_phase+45>:   movl   $0x0,-0x1c(%ebp)
   0x804920a <do_phase+52>:   jmp    0x8049234 <do_phase+94>
   0x804920c <do_phase+54>:   lea    -0x17(%ebp),%edx
   0x804920f <do_phase+57>:   mov    -0x1c(%ebp),%eax
   0x8049212 <do_phase+60>:   add    %edx,%eax
   0x8049214 <do_phase+62>:   movzbl (%eax),%eax
   0x8049217 <do_phase+65>:   movzbl %al,%eax
   0x804921a <do_phase+68>:   movzbl 0x804c060(%eax),%eax
   0x8049221 <do_phase+75>:   movsbl %al,%eax
(gdb)
```

里面出现了一个地址，我们直接查看%ebp-0x17 地址的内容，我们发现了一个字符串，这个就是我们的 COOKIE 字符串，我们可以先把它转换成 ascii 码如下：100 101 122 113 114 102 119 106 116 112，这个也就是我们需要输出的学号在全局变量字符串中的索引。

```
(gdb) x/s $ebp-0x17
0xffffd071:     "dezqrfwjtp"
```

我们可以查看全局变量数组的名字，我们发现它的名字为 WzDTGpmQDa：

```
 49: 080492d5     0 FUNC    GLOBAL HIDDEN    13 __x86.get_pc_thunk.bp
 50: 0804c028     0 NOTYPE  GLOBAL DEFAULT   23 _edata
 51: 0804c060   256 OBJECT  GLOBAL DEFAULT   24 WzDTGpmQDa
 52: 080492dc     0 FUNC    GLOBAL HIDDEN    14 _fini
```

因此我们需要做的就是把 100 101 122 113 114 102 119 106 116 112 改为学号，因此可以写出如下 c 语言代码：

```
char WzDTGpmQDa[256]=
"aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa110aaa3aaaaa103a2aa0aa9aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

将这个代码编译之后与 phase3.o 和 main.o 链接就可以得到我们需要的结果。

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

对于链接有了更加深刻的了解，对于链接的一些基本形式也都进行了实际操作，加深了印象

4.2 请给出对本次实验内容的建议

PPT 里可以多一些展示的环节，同时 PPT 中可以更多一些教学的内容

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.