哈爾濱工業大學

实验报告

实 验（四）

题　　目 LinkLab

链接

专　　业 计算机类

学　　号 1190200122

班　　级 1903001

学　　生 袁野

指 导 教 师 郑贵滨

实 验 地 点 G709

实 验 日 期 2020-5-21

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

Legion Y7000P 2019 PG0
CPU:Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz (12 CPUs), ~2.6GHz
RAM: 16384MB

1.2.2 软件环境

Windows 10 家庭中文版 64-bit
Ubuntu 20.04.2 LTS
VMware® Workstation 16 Player 16.1.0 build-17198959

1.2.3 开发工具

Microsoft Visual Studio Community 2019 版本 16.9.2
Microsoft Visual 1.54.3
GCC 9.3.0

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。

■ 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：字段 e_entry 给出执行程序时第一条指令的地址

程序头表：是一个结构数组，将连续的文件映射到运行时的内存段

. init ： 定义_init 函数，该函数用来执行可执行目标文件开始执行时的初始化工作

. text ： 已编译程序的机器代码

. rodata ： 只读数据，比如 printf 语句中的格式串和开关语句的跳转表

. 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 ：一个符号表，它存放在程序中定义和引用的函数和全局变量的信息

. debug ： 一个调试符号表，其条目时程序中定义的全局变量和类型定义，程序 中定义和引用的全局变量，以及原始的 C 源文件。

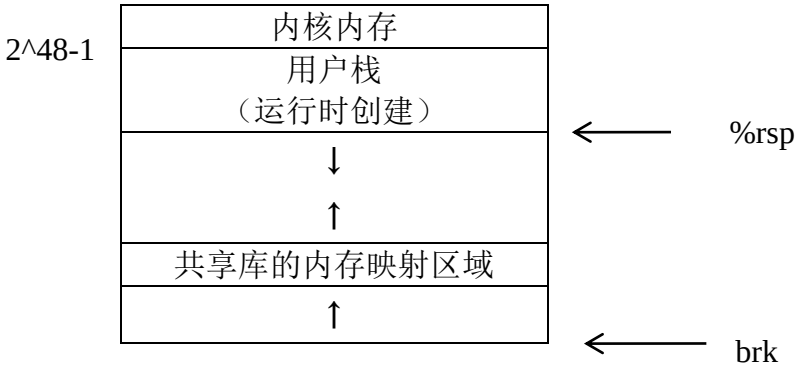
. line ： 原始 C 源程序的行号和.text 节中机器指令之间的映射

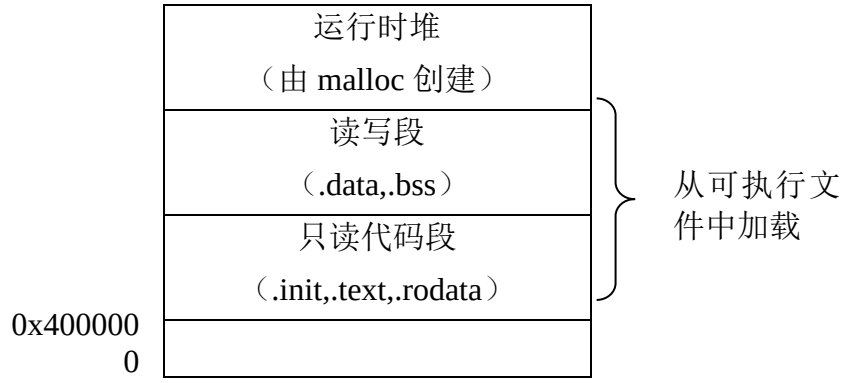
. strtab ： 一个字符串表，其内容包括 .symtab 和 .debug 节中的符号表，以及节 头部中的节名字。

节头部表：描述目标文件的节。

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）

所属区	各符号的地址、空间
只读代码段（.init , .text , .rodata）	show_pointer 0x559cc57b1199 94131816436121 useless 0x559cc57b11d0 94131816436176 main 0x559cc57b11df 94131816436191
读写段（.data .bss）	global 0x559cc57b402c 94131816448044 huge array 0x559cc57b4040 94131816448064 big array 0x559d057b4040 94132890189888 p2 0x559d087526b0 94132940121776
运行时堆（由 malloc 创建）	p1 0x7f2056e6e010 139776873652240 p3 0x7f2056e4d010 139776873517072 p4 0x7f2016e4c010 139775799771152 p5 0x7f1f96e4b010 139773652283408
共享库的内存映射区域	exit 0x7f2066eb8bc0 139777142393792 printf 0x7f2066ed3e10 139777142504976 malloc 0x7f2066f0c260 139777142735456

	free 0x7f2066f0c850 139777142736976
用户栈（运行时创建）	local 0x7fff31eb4f80 140734030892928 argc0x7fff31eb4f7c 140734030892924 argv 0x7fff31eb50b8 140734030893240 argv[0] 7fff31eb6296 argv[1] 7fff31eb62a4 argv[2] 7fff31eb62a7 argv[3] 7fff31eb62b2 argv[0] 0x7fff31eb6296 140734030897814 ./LinkAddress argv[1] 0x7fff31eb62a4 140734030897828 -u argv[2] 0x7fff31eb62a7 140734030897831 1190200122 argv[3] 0x7fff31eb62b2 140734030897842 袁野 env 0x7fff31eb50e0 140734030893280 env[0] *env 0x7fff31eb62b9 140734030897849 SHELL=/bin/bash env[1] *env 0x7fff31eb62c9 140734030897865 SESSION_MANAGER=local/1190200122-yuanye:@/tmp/.ICE-unix/1810,unix/1190200122-yuanye:/tmp/.ICE-unix/1810 env[2] *env 0x7fff31eb6331 140734030897969 QT_ACCESSIBILITY=1

	env[3] *env 0x7fff31eb6344 140734030897988 COLORTERM=truecolor env[4] *env 0x7fff31eb6358 140734030898008 XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/etc/xdg env[5] *env 0x7fff31eb6385 140734030898053 XDG_MENU_PREFIX=gnome- env[6] *env 0x7fff31eb639c 140734030898076 GNOME_DESKTOP_SESSION_ID=this-is-deprecated env[7] *env 0x7fff31eb63c8 140734030898120 GTK_IM_MODULE=fcitx env[8] *env 0x7fff31eb63dc 140734030898140 LANGUAGE=zh_CN:en env[9] *env 0x7fff31eb63ee 140734030898158 QT4_IM_MODULE=fcitx env[10] *env 0x7fff31eb6402 140734030898178 LC_ADDRESS=zh_CN.UTF-8 env[11] *env 0x7fff31eb6419 140734030898201 GNOME_SHELL_SESSION_MODE=ubuntu env[12] *env 0x7fff31eb6439 140734030898233 LC_NAME=zh_CN.UTF-8 env[13] *env 0x7fff31eb644d 140734030898253 SSH_AUTH_SOCK=/run/user/1000/keyring/ssh env[14] *env 0x7fff31eb6476 140734030898294

	XMODIFIERS=@im=fcitx env[15] *env 0x7fff31eb648b 140734030898315 DESKTOP_SESSION=ubuntu env[16] *env 0x7fff31eb64a2 140734030898338 LC_MONETARY=zh_CN.UTF-8 env[17] *env 0x7fff31eb64ba 140734030898362 SSH_AGENT_PID=1767 env[18] *env 0x7fff31eb64cd 140734030898381 GTK_MODULES=gail:atk-bridge env[19] *env 0x7fff31eb64e9 140734030898409 DBUS_STARTER_BUS_TYPE=session env[20] *env 0x7fff31eb6507 140734030898439 PWD=/mnt/hgfs/d env[21] *env 0x7fff31eb6517 140734030898455 LOGNAME=yuanye env[22] *env 0x7fff31eb6526 140734030898470 XDG_SESSION_DESKTOP=ubuntu env[23] *env 0x7fff31eb6541 140734030898497 XDG_SESSION_TYPE=x11 env[24] *env 0x7fff31eb6556 140734030898518 GPG_AGENT_INFO=/run/user/1000/gnupg/S.gpg-agent:0:1 env[25] *env 0x7fff31eb658a 140734030898570 XAUTHORITY=/run/user/1000/gdm/Xauthority

env[26] *env 0x7fff31eb65b3 140734030898611

WINDOWPATH=2

env[27] *env 0x7fff31eb65c0 140734030898624

HOME=/home/yuanye

env[28] *env 0x7fff31eb65d2 140734030898642

USERNAME=yuanye

env[29] *env 0x7fff31eb65e2 140734030898658

IM_CONFIG_PHASE=1

env[30] *env 0x7fff31eb65f4 140734030898676

LC_PAPER=zh_CN.UTF-8

env[31] *env 0x7fff31eb6609 140734030898697

LANG=zh_CN.UTF-8

env[32] *env 0x7fff31eb661a 140734030898714

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:*.xw

	d=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[33] *env 0x7fff31eb6bfc 140734030900220 XDG_CURRENT_DESKTOP=ubuntu:GNOME env[34] *env 0x7fff31eb6c1d 140734030900253 VTE_VERSION=6003 env[35] *env 0x7fff31eb6c2e 140734030900270 GNOME_TERMINAL_SCREEN=/org/gnome/Terminal/screen/0581008d_9cec_4e04_b01c_a6976c06baac env[36] *env 0x7fff31eb6c84 140734030900356 INVOCATION_ID=21335b21bb5e41e4a45269eddc0d084c env[37] *env 0x7fff31eb6cb3 140734030900403 MANAGERPID=1593 env[38] *env 0x7fff31eb6cc3 140734030900419 CLUTTER_IM_MODULE=fcitx env[39] *env 0x7fff31eb6cdb 140734030900443 LESSCLOSE=/usr/bin/lesspipe %s %s env[40] *env 0x7fff31eb6cfd 140734030900477 XDG_SESSION_CLASS=user env[41] *env 0x7fff31eb6d14 140734030900500 TERM=xterm-256color env[42] *env 0x7fff31eb6d28 140734030900520

	LC_IDENTIFICATION=zh_CN.UTF-8 env[43] *env 0x7fff31eb6d46 140734030900550 LESSOPEN=| /usr/bin/lesspipe %s env[44] *env 0x7fff31eb6d66 140734030900582 USER=yuanye env[45] *env 0x7fff31eb6d72 140734030900594 GNOME_TERMINAL_SERVICE=:1.98 env[46] *env 0x7fff31eb6d8f 140734030900623 DISPLAY=:0 env[47] *env 0x7fff31eb6d9a 140734030900634 SHLVL=1 env[48] *env 0x7fff31eb6da2 140734030900642 LC_TELEPHONE=zh_CN.UTF-8 env[49] *env 0x7fff31eb6dbb 140734030900667 QT_IM_MODULE=fcitx env[50] *env 0x7fff31eb6dce 140734030900686 LC_MEASUREMENT=zh_CN.UTF-8 env[51] *env 0x7fff31eb6de9 140734030900713 DBUS_STARTER_ADDRESS=unix:path=/run/user/1000/bus,guid=bb5d5914d54f13f81446171560b4e21d env[52] *env 0x7fff31eb6e41 140734030900801 PAPERSIZE=a4 env[53] *env 0x7fff31eb6e4e 140734030900814

	XDG_RUNTIME_DIR=/run/user/1000 env[54] *env 0x7fff31eb6e6d 140734030900845 LC_TIME=zh_CN.UTF-8 env[55] *env 0x7fff31eb6e81 140734030900865 JOURNAL_STREAM=8:51477 env[56] *env 0x7fff31eb6e98 140734030900888 XDG_DATA_DIRS=/usr/share/ubuntu:/usr/local/share/:/usr/share/:/var/lib/snapd/desktop env[57] *env 0x7fff31eb6eed 140734030900973 PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin env[58] *env 0x7fff31eb6f55 140734030901077 GDMSESSION=ubuntu env[59] *env 0x7fff31eb6f67 140734030901095 DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus,guid=bb5d5914d54f13f81446171560b4e21d env[60] *env 0x7fff31eb6fc3 140734030901187 LC_NUMERIC=zh_CN.UTF-8 env[61] *env 0x7fff31eb6fda 140734030901210 _=./LinkAddress

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）

main 执行前：

malloc@plt()
free@plt()
_init()
_start()
__libc_csu_init()
frame_dummy()
register_tm_clones ()
main()

main 执行后

useless ()
show_pointer ()
printf@plt ()
malloc@plt ()
puts@plt ()
free@plt ()
__do_global_dtors_aux ()
deregister_tm_clones ()
_fini ()

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：

```
yuanye@1190200122-yuanye:/mnt/hgfs/d$ gcc -m32 -o linkbomb1 main.o phase1.o
yuanye@1190200122-yuanye:/mnt/hgfs/d$ ./linkbomb1
1190200122
```

分析与设计的过程：

我们将 main 与 phase1 链接后运行得到如下内容

```
yuanye@1190200122-yuanye:/mnt/hgfs/d$ gcc -m32 -o linkbomb1 main.o phase1.o
yuanye@1190200122-yuanye:/mnt/hgfs/d$ ./linkbomb1
F9Ha0HO UCLs    lLcj27wTTHstB miQfp3hV  B8W7KD26IbGDZnPpDCnOyFBrunBczEgNIdVyxZwz
hMHZ    RdRIeHfJwwj8m7twj0XIizJ7vaTRydpXf7Ni0tXC0mGL5aExVYVUPxYxZj      XnVTEDB4
PqaxuPzc1KzVRtnJ0BF1KUEC4Cf8887f9H32WaMfsn2wfzP
```

显然我们只需要将 phase1 中的该字符串修改为自己的学号即可。

首先我们执行 readelf -S phase1.o 查看.data 节的偏移量位 0x60

```
节头：
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .text             PROGBITS        00000000 000034 00001e 00  AX  0   0  1
  [ 2] .rel.text         REL             00000000 0002a4 000010 08   I 11   1  4
  [ 3] .data             PROGBITS        00000000 000060 0000cc 00  WA  0   0 32
  [ 4] .rel.data         REL             00000000 0002b4 000008 08   I 11   3  4
  [ 5] .bss              NOBITS          00000000 00012c 000000 00  WA  0   0  1
  [ 6] .comment          PROGBITS        00000000 00012c 00002d 01  MS  0   0  1
  [ 7] .note.GNU-stack   PROGBITS        00000000 000159 000000 00      0   0  1
  [ 8] .note.gnu.propert NOTE            00000000 00015c 00001c 00   A  0   0  4
  [ 9] .eh_frame         PROGBITS        00000000 000178 000038 00   A  0   0  4
```

然后执行 readelf -r phase1.o 查看重定位节，puts 的偏移量为 0x14

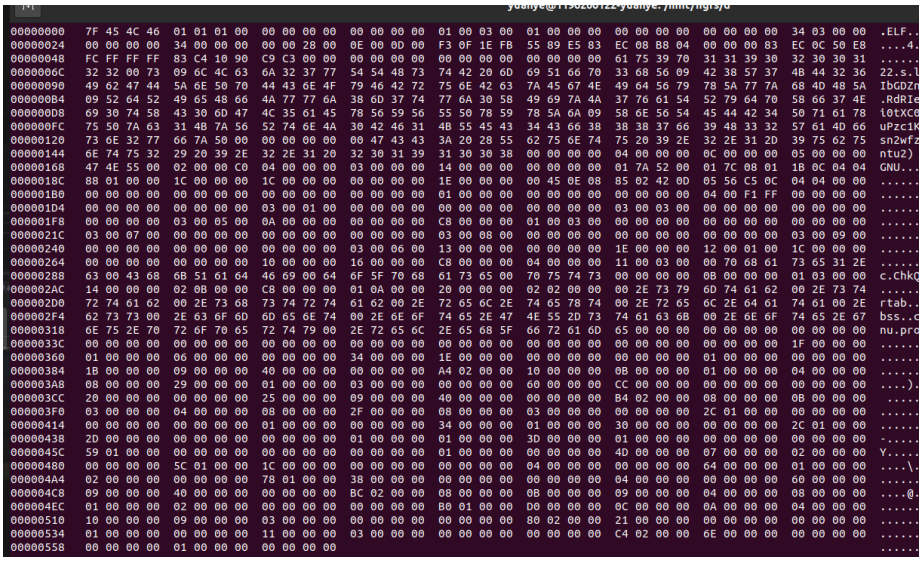
```
1
2  重定位节 '.rel.text' at offset 0x2a4 contains 2 entries:
3   偏移量      信息     类型              符号值      符号名称
4  0000000b  00000301 R_386_32           00000000   .data
5  00000014  00000b02 R_386_PC32         00000000   puts
6
7  重定位节 '.rel.data' at offset 0x2b4 contains 1 entry:
8   偏移量      信息     类型              符号值      符号名称
9  000000c8  00000a01 R_386_32           00000000   do_phase
10
11 重定位节 '.rel.eh_frame' at offset 0x2bc contains 1 entry:
12  偏移量      信息     类型              符号值      符号名称
13 00000020  00000202 R_386_PC32         00000000   .text
14
```

我们将 phase1.o 的反汇编文件打开可以看到在调用偏移量为 0x14 的函数，也就是 puts 之前传递的参数为 0x4。

```
00000000 <do_phase>:
   0:   f3 0f 1e fb             endbr32
   4:   55                      push   %ebp
   5:   89 e5                   mov    %esp,%ebp
   7:   83 ec 08                sub    $0x8,%esp
   a:   b8 04 00 00 00          mov    $0x4,%eax
   f:   83 ec 0c                sub    $0xc,%esp
  12:   50                      push   %eax
  13:   e8 fc ff ff ff          call   14 <do_phase+0x14>
  18:   83 c4 10                add    $0x10,%esp
  1b:   90                      nop
  1c:   c9                      leave
  1d:   c3                      ret
```

因此输出的字符串的起始位置应该为 0x60+0x4 = 0x64。

我们用 HEXEdit 打开 phase.1 可以看到如下：



我们很快就可以找到未经过修改之前的字符串对应的位置，去除对应长度的部分将其修改为“1190200122\0”即可。

3.2 阶段 2 的分析

程序运行结果截图：

```
yuanye@1190200122-yuanye:/mnt/hgfs/d$ gcc -no-pie -m32 -o linkbomb2 main.o phase2.o
yuanye@1190200122-yuanye:/mnt/hgfs/d$ ./linkbomb2
1190200122
```

分析与设计的过程：

首先我们使用 objdump 查看 phase_2 的反汇编代码。

```
00000000 <uryJIant>:
   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 <uryJIant+0x16>
  1a: 83 c4 10              add    $0x10,%esp
  1d: 85 c0                 test   %eax,%eax
  1f: 75 10                 jne    31 <uryJIant+0x31>
  21: 83 ec 0c              sub    $0xc,%esp
  24: ff 75 08              pushl  0x8(%ebp)
  27: e8 fc ff ff ff        call   28 <uryJIant+0x28>
  2c: 83 c4 10              add    $0x10,%esp
  2f: eb 01                 jmp    32 <uryJIant+0x32>
  31: 90                    nop
  32: c9                    leave
  33: c3                    ret

00000034 <do_phase>:
  34: f3 0f 1e fb           endbr32
  38: 55                    push   %ebp
  39: 89 e5                 mov    %esp,%ebp
  3b: 90                    nop
  3c: 90                    nop
  3d: 90                    nop
  3e: 90                    nop
  3f: 90                    nop
  40: 90                    nop
```

我们可以知道 uryJIant 为我们的输出函数，由 ppt 可知，我们需要让传进去的 *id 与 MYID 相同即可，因此我们需要利用 gdb 知道 MYID 的位置。

```
(gdb) disassemble uryJIant
Dump of assembler code for function uryJIant:
   0x080491fa <+0>:     endbr32
   0x080491fe <+4>:     push   %ebp
   0x080491ff <+5>:     mov    %esp,%ebp
   0x08049201 <+7>:     sub    $0x8,%esp
   0x08049204 <+10>:    sub    $0x8,%esp
   0x08049207 <+13>:    push   $0x804a07c
   0x0804920c <+18>:    pushl  0x8(%ebp)
   0x0804920f <+21>:    call   0x8049070 <strcmp@plt>
   0x08049214 <+26>:    add    $0x10,%esp
   0x08049217 <+29>:    test   %eax,%eax
   0x08049219 <+31>:    jne    0x804922b <uryJIant+49>
   0x0804921b <+33>:    sub    $0xc,%esp
   0x0804921e <+36>:    pushl  0x8(%ebp)
   0x08049221 <+39>:    call   0x8049080 <puts@plt>
   0x08049226 <+44>:    add    $0x10,%esp
   0x08049229 <+47>:    jmp    0x804922c <uryJIant+50>
   0x0804922b <+49>:    nop
   0x0804922c <+50>:    leave
   0x0804922d <+51>:    ret
End of assembler dump.
(gdb) x/s 0x804a07c
0x804a07c:      "1190200122"
```

由上图我们可以知道 MYID 的位置即为 0x804a07c。当我们在编译命令里加上 -no-pie 之后我们链接之后的 MYID 位置不会发生变化。因此只需要在调用输出函数之前将传递的参数赋值为 0x804a07c 即可。另外我们需要利用 PC 相对寻址的办法寻找输出函数的相对地址，结合所需要构造的语句的字节数我们可以计算输出的相对地址为-0x45，因此我们可以构造汇编代码。

a.s

```
1  push $0x804a07c
2  call -0x45
3  add $0x4,%esp
4
```

通过执行 gcc -c a.s 得到 a.o，然后用 objdump 反汇编得到 16 进制代码。

```
yuanye@1190200122-yuanye:/mnt/hgfs/d$ gcc -m32 -c a.s
yuanye@1190200122-yuanye:/mnt/hgfs/d$ objdump -d a.o

a.o:     文件格式 elf32-i386


Disassembly of section .text:

00000000 <.text>:
   0:   68 7c a0 04 08          push   $0x804a07c
   5:   e8 b7 ff ff ff          call   0xffffffc1
   a:   83 c4 04                add    $0x4,%esp
```

将该 16 进制代码替换 nop，修正-0x45 的补码后重新链接并运行。

```
yuanye@1190200122-yuanye:/mnt/hgfs/d$ gcc -no-pie -m32 -o bb main.o phase2.o
yuanye@1190200122-yuanye:/mnt/hgfs/d$ ./bb
1190200122
```

3.3 阶段 3 的分析

程序运行结果截图：

```
yuanye@1190200122-yuanye:/mnt/hgfs/d$ gcc -m32 -c -o phase3_patch.o a.c
yuanye@1190200122-yuanye:/mnt/hgfs/d$ gcc -m32 -o linkbomb3 main.o phase3.o phase3_patch.o
yuanye@1190200122-yuanye:/mnt/hgfs/d$ ./linkbomb3
1190200122
```

分析与设计的过程：

通过 readelf -s phase3.o 查看 PHASE3_CODEBOOK 的名字。

```
yuanye@1190200122-yuanye:/mnt/hgfs/d$ readelf -s phase3.o

Symbol table '.symtab' contains 14 entries:
   Num:    Value  Size Type    Bind   Vis      Ndx Name
     0: 00000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 00000000     0 FILE    LOCAL  DEFAULT  ABS phase3.c
     2: 00000000     0 SECTION LOCAL  DEFAULT    1
     3: 00000000     0 SECTION LOCAL  DEFAULT    3
     4: 00000000     0 SECTION LOCAL  DEFAULT    5
     5: 00000000     0 SECTION LOCAL  DEFAULT    7
     6: 00000000     0 SECTION LOCAL  DEFAULT    8
     7: 00000000     0 SECTION LOCAL  DEFAULT    9
     8: 00000000     0 SECTION LOCAL  DEFAULT    6
     9: 00000020   256 OBJECT  GLOBAL DEFAULT  COM UEXjjBHcRo
    10: 00000000   135 FUNC    GLOBAL DEFAULT    1 do_phase
    11: 00000000     0 NOTYPE  GLOBAL DEFAULT  UND putchar
    12: 00000000     0 NOTYPE  GLOBAL DEFAULT  UND __stack_chk_fail
    13: 00000000     4 OBJECT  GLOBAL DEFAULT    3 phase
```

很显然 PHASE3_CODEBOOK 的名字为 UEXjjBHcRo。

然后我们通过 gdb 查看 cookie。

```
(gdb) disassemble
Dump of assembler code for function do_phase:
=> 0x56556251 <+0>:     endbr32
   0x56556255 <+4>:     push   %ebp
   0x56556256 <+5>:     mov    %esp,%ebp
   0x56556258 <+7>:     sub    $0x28,%esp
   0x5655625b <+10>:    mov    %gs:0x14,%eax
   0x56556261 <+16>:    mov    %eax,-0xc(%ebp)
   0x56556264 <+19>:    xor    %eax,%eax
   0x56556266 <+21>:    movl   $0x70717976,-0x17(%ebp)
   0x5655626d <+28>:    movl   $0x62676861,-0x13(%ebp)
   0x56556274 <+35>:    movw   $0x776f,-0xf(%ebp)
   0x5655627a <+41>:    movb   $0x0,-0xd(%ebp)
   0x5655627e <+45>:    movl   $0x0,-0x1c(%ebp)
   0x56556285 <+52>:    jmp    0x565562af <do_phase+94>
   0x56556287 <+54>:    lea    -0x17(%ebp),%edx
   0x5655628a <+57>:    mov    -0x1c(%ebp),%eax
   0x5655628d <+60>:    add    %edx,%eax
   0x5655628f <+62>:    movzbl (%eax),%eax
   0x56556292 <+65>:    movzbl %al,%eax
   0x56556295 <+68>:    movzbl 0x56559040(%eax),%eax
   0x5655629c <+75>:    movsbl %al,%eax
   0x5655629f <+78>:    sub    $0xc,%esp
   0x565562a2 <+81>:    push   %eax
   0x565562a3 <+82>:    call   0xf7e3d350 <putchar>
   0x565562a8 <+87>:    add    $0x10,%esp
   0x565562ab <+90>:    addl   $0x1,-0x1c(%ebp)
   0x565562af <+94>:    mov    -0x1c(%ebp),%eax
   0x565562b2 <+97>:    cmp    $0x9,%eax
   0x565562b5 <+100>:   jbe    0x56556287 <do_phase+54>
   0x565562b7 <+102>:   sub    $0xc,%esp
   0x565562ba <+105>:   push   $0xa
   0x565562bc <+107>:   call   0xf7e3d350 <putchar>
   0x565562c1 <+112>:   add    $0x10,%esp
   0x565562c4 <+115>:   nop
   0x565562c5 <+116>:   mov    -0xc(%ebp),%eax
   0x565562c8 <+119>:   xor    %gs:0x14,%eax
   0x565562cf <+126>:   je     0x565562d6 <do_phase+133>
   0x565562d1 <+128>:   call   0xf7ee06c0 <__stack_chk_fail>
   0x565562d6 <+133>:   leave
   0x565562d7 <+134>:   ret
```

结合反汇编代码与 ppt 我们可以不难发现，%eax 储存的是下标 i，而-0x17(%ebp)即为 cookie 的首地址，我们将其输出。

```
(gdb) x/10d $ebp-0x17
0xffffcff1:     118     121     113     112     97      104     103     98
0xffffcff9:     111     119
```

这是个数字对应在 PHASE3_CODEBOOK 的映射以为我们要输出的内容，因此我们需要构建一个强符号 UEXjjBHcRo 并使其将上面十个数字映射到学号上即可。

phase3_patch.c 内容如下：

char

UEXjjBHcRo[256]="00000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000210000000000002090
0001201000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000
000000000";

编译、链接、运行即可。

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.