哈爾濱工業大學

实验报告

实 验（五）

题　　目 LinkLab

链接

专　　业 计算机系

学　　号 1180300811

班　　级 1803008

学　　生 孙骁

指 导 教 师 吴锐

实 验 地 点 G712

实 验 日 期 2019/11/23

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

1. 理解链接的作用与工作步骤
2. 掌握 ELF 结构、符号解析与重定位的工作过程
3. 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

i7-8550U X64 CPU；1.80GHz；16G RAM；1T SSD

1.2.2 软件环境

Windows10 64 位；Vmware 15.1.0；Ubuntu 18.04 LTS

1.2.3 开发工具

Visual Studio 2019 ；CodeBlocks；gcc

1.3 实验预习

1. 上实验课前，必须认真预习实验指导书（PPT 或 PDF）
2. 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。
3. 请按顺序写出 ELF 格式的可执行目标文件的各类信息。
4. 请按照内存地址从低到高的顺序，写出 Linux 下 X64 内存映像。
5. 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。
6. 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

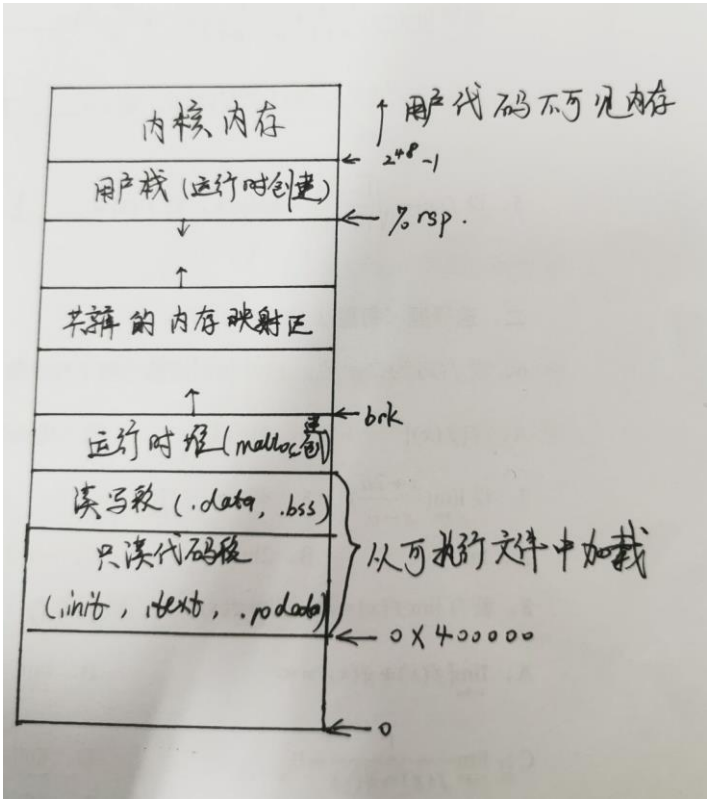
2.1 ELF 文件格式解读

请按顺序写出 ELF 格式的可执行目标文件的各类信息（5 分）

1. ELF 头
2. 段头部表：将连续的文件映射到运行时的内存段
3. .init ： 定义了_init 函数，程序初始化代码会调用它
4. .text ： 已编译程序的机器代码
5. .rodata ： 只读数据，比如 printf 语句中的格式串和开关语句的跳转表
6. .data ： 已初始化的全局和静态 C 变量
7. .bss ： 未初始化的全局和静态 C 变量
8. .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
9. .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。
10. .line ： 原始 C 源程序的行号和.text 节中机器指令之间的映射
11. .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。
12. 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、空间。并按照 Linux 下 X64 内存映像标出其所属各区（5 分）

只读段（.init , .text , .rodata）：

```
exit    0x7f75da245120  140144147714336
printf  0x7f75da266e80  140144147852928
malloc  0x7f75da299070  140144148058224
free    0x7f75da299950  140144148060496
```

读写段（.data .bss）：

```
big array       0x555fbd927040  93869690744896
huge array      0x555f7d927040  93868617003072
```

```
global  0x555f7d92702c  93868617003052
```

```
show_pointer    0x555f7d72581a  93868614899738
useless 0x555f7d72584d  93868614899789
main    0x555f7d725858  93868614899800
```

运行时堆：

```
p1      0x7f75ca201010  140143879000080
p2      0x555fbedac670  93869712262768
p3      0x7f75da7de010  140144153583632
p4      0x7f758a200010  140142805254160
p5      0x7f750a1ff010  140140657766416
```

用户栈：

env0x7fff7a68c000 140735247073280

env[0] *env 0x7fff7a68d18b 140735247077771

CLUTTER_IM_MODULE=xim

env[1] *env 0x7fff7a68d1a1 140735247077793

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:*.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[2] *env 0x7fff7a68d78d 140735247079309

LC_MEASUREMENT=zh_CN.UTF-8

env[3] *env 0x7fff7a68d7a8 140735247079336

LESSCLOSE=/usr/bin/lesspipe %s %s

env[4] *env 0x7fff7a68d7ca 140735247079370

LC_PAPER=zh_CN.UTF-8

env[5] *env 0x7fff7a68d7df 140735247079391

LC_MONETARY=zh_CN.UTF-8

env[6] *env 0x7fff7a68d7f7 140735247079415
XDG_MENU_PREFIX=gnome-
env[7] *env 0x7fff7a68d80e 140735247079438
LANG=zh_CN.UTF-8
env[8] *env 0x7fff7a68d81f 140735247079455
MANAGERPID=1573
env[9] *env 0x7fff7a68d82f 140735247079471
DISPLAY=:0
env[10]*env 0x7fff7a68d83a 140735247079482
INVOCATION_ID=c699e7cb256d455b8072b8a4ad3a3461
env[11]*env 0x7fff7a68d869 140735247079529
GNOME_SHELL_SESSION_MODE=ubuntu
env[12]*env 0x7fff7a68d889 140735247079561
COLORTERM=truecolor
env[13]*env 0x7fff7a68d89d 140735247079581
ZEITGEIST_DATA_PATH=/home/demerzel/.local/zeitgeist
env[14]*env 0x7fff7a68d8d7 140735247079639
USERNAME=demerzel
env[15]*env 0x7fff7a68d8e9 140735247079657
XDG_VTNR=2
env[16]*env 0x7fff7a68d8f4 140735247079668
SSH_AUTH_SOCK=/run/user/1000/keyring/ssh
env[17]*env 0x7fff7a68d91d 140735247079709
MANDATORY_PATH=/usr/gconf/ubuntu.mandatory.path
env[18]*env 0x7fff7a68d953 140735247079763
LC_NAME=zh_CN.UTF-8
env[19]*env 0x7fff7a68d967 140735247079783
XDG_SESSION_ID=2
env[20]*env 0x7fff7a68d978 140735247079800
USER=demerzel
env[21]*env 0x7fff7a68d986 140735247079814
DESKTOP_SESSION=ubuntu
env[22]*env 0x7fff7a68d99d 140735247079837
QT4_IM_MODULE=fcitx
env[23]*env 0x7fff7a68d9b1 140735247079857
TEXTDOMAINDIR=/usr/locale/
env[24]*env 0x7fff7a68d9d2 140735247079890
GNOME_TERMINAL_SCREEN=/org/gnome/Terminal/screen/04147f59_c85b_4

1ce_82e1_d43bdfe78d2f

env[25]*env 0x7fff7a68da28 140735247079976
DEFAULTS_PATH=/usr/gconf/ubuntu.default.path
env[26]*env 0x7fff7a68da5b 140735247080027
PWD=/home/demerzel/ccode/lab5
env[27]*env 0x7fff7a68da79 140735247080057
HOME=/home/demerzel
env[28]*env 0x7fff7a68da8d 140735247080077
JOURNAL_STREAM=9:45390
env[29]*env 0x7fff7a68daa4 140735247080100
TEXTDOMAIN=im-config
env[30]*env 0x7fff7a68dab9 140735247080121
SSH_AGENT_PID=1684
env[31]*env 0x7fff7a68dacc 140735247080140
QT_ACCESSIBILITY=1
env[32]*env 0x7fff7a68dadf 140735247080159
XDG_SESSION_TYPE=x11
env[33]*env 0x7fff7a68daf4 140735247080180
XDG_DATA_DIRS=/usr/ubuntu:/usr/local:/usr:arb/snapd/desktop
env[34]*env 0x7fff7a68db47 140735247080263
XDG_SESSION_DESKTOP=ubuntu
env[35]*env 0x7fff7a68db62 140735247080290
LC_ADDRESS=zh_CN.UTF-8
env[36]*env 0x7fff7a68db79 140735247080313
DBUS_STARTER_ADDRESS=unix:path=/run/user/1000s,guid=dd7c54e2bd5799
a6abd621745de52351
env[37]*env 0x7fff7a68dbd1 140735247080401
LC_NUMERIC=zh_CN.UTF-8
env[38]*env 0x7fff7a68dbe8 140735247080424
GTK_MODULES=gail:atk-bridge
env[39]*env 0x7fff7a68dc04 140735247080452
WINDOWPATH=2
env[40]*env 0x7fff7a68dc11 140735247080465
TERM=xterm-256color
env[41]*env 0x7fff7a68dc25 140735247080485
VTE_VERSION=5202
env[42]*env 0x7fff7a68dc36 140735247080502
SHELL=/bin/bash

env[43]*env 0x7fff7a68dc46 140735247080518
QT_IM_MODULE=fcitx
env[44]*env 0x7fff7a68dc59 140735247080537
XMODIFIERS=@im=fcitx
env[45]*env 0x7fff7a68dc6e 140735247080558
IM_CONFIG_PHASE=2
env[46]*env 0x7fff7a68dc80 140735247080576
DBUS_STARTER_BUS_TYPE=session
env[47]*env 0x7fff7a68dc9e 140735247080606
XDG_CURRENT_DESKTOP=ubuntu:GNOME
env[48]*env 0x7fff7a68dcbf 140735247080639
GPG_AGENT_INFO=/run/user/1000/gnupg/S.gpg-agent:0:1
env[49]*env 0x7fff7a68dcf3 140735247080691
GNOME_TERMINAL_SERVICE=:1.125
env[50]*env 0x7fff7a68dd11 140735247080721
SHLVL=1
env[51]*env 0x7fff7a68dd19 140735247080729
XDG_SEAT=seat0
env[52]*env 0x7fff7a68dd28 140735247080744
LANGUAGE=zh_CN:zh:en_US:en
env[53]*env 0x7fff7a68dd43 140735247080771
LC_TELEPHONE=zh_CN.UTF-8
env[54]*env 0x7fff7a68dd5c 140735247080796
GDMSESSION=ubuntu
env[55]*env 0x7fff7a68dd6e 140735247080814
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
env[56]*env 0x7fff7a68dd9a 140735247080858
LOGNAME=demerzel
env[57]*env 0x7fff7a68ddab 140735247080875
DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000s,guid=dd7c54e2bd
5799a6abd621745de52351
env[58]*env 0x7fff7a68de07 140735247080967
XDG_RUNTIME_DIR=/run/user/1000
env[59]*env 0x7fff7a68de26 140735247080998
XAUTHORITY=/run/user/1000/gdm/Xauthority
env[60]*env 0x7fff7a68de4f 140735247081039
XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/etc/xdg
env[61]*env 0x7fff7a68de7c 140735247081084

PATH=/home/demerzel/.local/umake/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin

env[62]*env 0x7fff7a68df0a 140735247081226

LC_IDENTIFICATION=zh_CN.UTF-8

env[63]*env 0x7fff7a68df28 140735247081256

SESSION_MANAGER=local/demerzel-virtual-machine:@/tmp/.ICE-unix/1603,unix/demerzel-virtual-machine:/tmp/.ICE-unix/1603

env[64]*env 0x7fff7a68df9e 140735247081374

LESSOPEN=| /usr/bin/lesspipe %s

env[65]*env 0x7fff7a68dfbe 140735247081406

GTK_IM_MODULE=fcitx

env[66]*env 0x7fff7a68dfd2 140735247081426

LC_TIME=zh_CN.UTF-8

env[67]*env 0x7fff7a68dfe6 140735247081446

_=./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）

main 前：

0x00005555555546a8 <_init>

0x00005555555546d0<puts@plt>

0x00005555555546e0<__stack_chk_fail@plt>

0x00005555555546f0<free@plt>

0x00005555555546f8<printf@plt>

0x0000555555554700<malloc@plt>

0x0000555555554708<__cxa_finalize@plt>

0x0000555555554710<_start>

0x0000555555554740<deregister_tm_clones>

0x0000555555554780<register_tm_clones>

0x00005555555547d0<__do_global_dtors_aux>

0x0000555555554810<frame_dummy>

0x000055555555481a<show_pointer>

0x000055555555484d <useless>

main 后：

0x0000555555554858<main>
0x0000555555554c40<__libc_csu_init>
0x0000555555554cb0<__libc_csu_fini>
0x0000555555554cb4 <_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：

```
demerzel@demerzel-virtual-machine:~/ccode/lab5$ gcc -m32 -o linkbomb1 main.o phase1.o
demerzel@demerzel-virtual-machine:~/ccode/lab5$ ./linkbomb1
1180300811
```

分析与设计的过程：

将 main.o 与 phase1.c 链接后，直接运行，结果显示为乱码。

```
demerzel@demerzel-virtual-machine:~/ccode/lab5$ gcc -m32 -o linkbomb1 main.o phase1.o
demerzel@demerzel-virtual-machine:~/ccode/lab5$ ./linkbomb1
W2      GUYBCs6NELJ     KFSaRwzXL8DBmRL5jQ7QnAgRp00aTBQFROtv0yGGFFplKelOe80KO9kQds9xTINAznUHJMMi8P5Ny2 TybG
sPpzP4eAFtuXzDEuGhCn1HE
```

按照实验要求，要求输出学号。反汇编 phase1.o

```
00000000 <do_phase>:
   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 <do_phase+0x8>
   c:   05 01 00 00 00          add    $0x1,%eax
  11:   8d 90 06 00 00 00       lea    0x6(%eax),%edx
  17:   83 ec 0c                sub    $0xc,%esp
  1a:   52                      push   %edx
  1b:   89 c3                   mov    %eax,%ebx
  1d:   e8 fc ff ff ff          call   1e <do_phase+0x1e>
  22:   83 c4 10                add    $0x10,%esp
  25:   90                      nop
  26:   8b 5d fc                mov    -0x4(%ebp),%ebx
  29:   c9                      leave
  2a:   c3                      ret
```

可见在 do_phase 中，printf 被优化为 puts

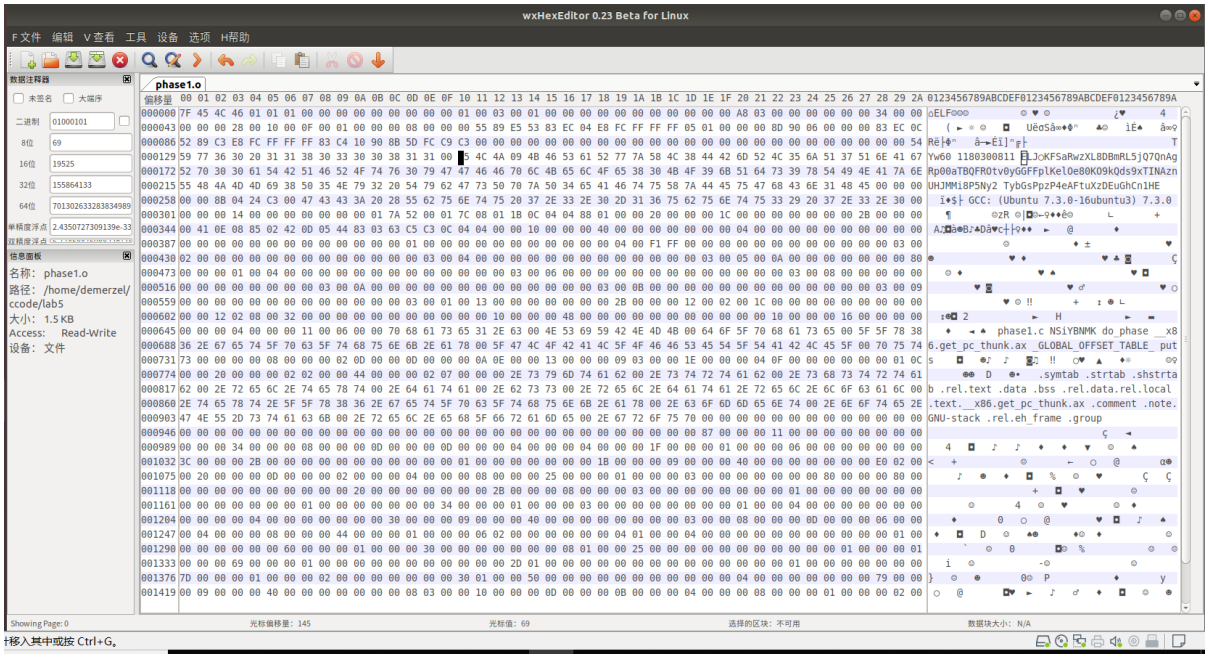
```
00000585 <do_phase>:
 585:   55                      push   %ebp
 586:   89 e5                   mov    %esp,%ebp
 588:   53                      push   %ebx
 589:   83 ec 04                sub    $0x4,%esp
 58c:   e8 f0 ff ff ff          call   581 <__x86.get_pc_thunk.ax>
 591:   05 47 1a 00 00          add    $0x1a47,%eax
 596:   8d 90 4e 00 00 00       lea    0x4e(%eax),%edx
 59c:   83 ec 0c                sub    $0xc,%esp
 59f:   52                      push   %edx
 5a0:   89 c3                   mov    %eax,%ebx
 5a2:   e8 19 fe ff ff          call   3c0 <puts@plt>
 5a7:   83 c4 10                add    $0x10,%esp
 5aa:   90                      nop
 5ab:   8b 5d fc                mov    -0x4(%ebp),%ebx
 5ae:   c9                      leave
 5af:   c3                      ret
```

因为字符串常数因此被保存在.data 节数据段中。因此只需要在 phase1 中查找到相应的字符串更改为学号。

首先 readelf -a phase1.o 查看 elf 文件内容，根据节头信息可知，字符串输出起始地址在 .data 节中偏移量为 32 的位置。

```
节头:
  [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 0002e0 000020 08   I 13   2  4
  [ 4] .data             PROGBITS        00000000 000080 000080 00  WA  0   0 32
  [ 5] .bss              NOBITS          00000000 000100 000000 00  WA  0   0  1
  [ 6] .data.rel.local   PROGBITS        00000000 000100 000004 00  WA  0   0  4
  [ 7] .rel.data.rel.loc REL             00000000 000300 000008 08   I 13   6  4
  [ 8] .text.__x86.get_p PROGBITS        00000000 000104 000004 00 AXG  0   0  1
  [ 9] .comment          PROGBITS        00000000 000108 000025 01  MS  0   0  1
  [10] .note.GNU-stack   PROGBITS        00000000 00012d 000000 00      0   0  1
  [11] .eh_frame         PROGBITS        00000000 000130 000050 00   A  0   0  4
  [12] .rel.eh_frame     REL             00000000 000308 000010 08   I 13  11  4
  [13] .symtab           SYMTAB          00000000 000180 000110 10     14  12  4
  [14] .strtab           STRTAB          00000000 000290 00004d 00      0   0  1
  [15] .shstrtab         STRTAB          00000000 000318 00008e 00      0   0  1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings), I (info),
  L (link order), O (extra OS processing required), G (group), T (TLS),
```

用 wxHexeditor 打开 phase1.o，并将 1180300811\0 插入指定位置



重新链接后结果如上。

3.2 阶段 2 的分析

程序运行结果截图：

```
demerzel@demerzel-virtual-machine:~/ccode/lab5$ gcc -m32 -o linkbomb2 main.o phase2.o
demerzel@demerzel-virtual-machine:~/ccode/lab5$ ./linkbomb2
1180300811
```

分析与设计的过程：

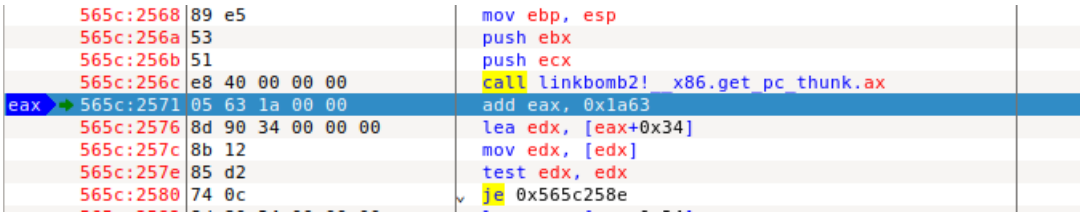
通过 gcc -m32 -o linkbomb2 main.o phase2.o 以及 odjdump -d linkbomb2 > linkbomb2.S 后，在.text 字节中找到比较字符串的位置：

```
000005b5 <xdkuJjfQ>:
 5b5:   55                      push   %ebp
 5b6:   89 e5                   mov    %esp,%ebp
 5b8:   53                      push   %ebx
 5b9:   83 ec 04                sub    $0x4,%esp
 5bc:   e8 9f fe ff ff          call   460 <__x86.get_pc_thunk.bx>
 5c1:   81 c3 13 1a 00 00       add    $0x1a13,%ebx
 5c7:   83 ec 08                sub    $0x8,%esp
 5ca:   8d 83 50 e7 ff ff       lea    -0x18b0(%ebx),%eax
 5d0:   50                      push   %eax
 5d1:   ff 75 08                pushl  0x8(%ebp)
 5d4:   e8 07 fe ff ff          call   3e0 <strcmp@plt>
 5d9:   83 c4 10                add    $0x10,%esp
 5dc:   85 c0                   test   %eax,%eax
 5de:   75 10                   jne    5f0 <xdkuJjfQ+0x3b>
 5e0:   83 ec 0c                sub    $0xc,%esp
 5e3:   ff 75 08                pushl  0x8(%ebp)
 5e6:   e8 05 fe ff ff          call   3f0 <puts@plt>
 5eb:   83 c4 10                add    $0x10,%esp
 5ee:   eb 01                   jmp    5f1 <xdkuJjfQ+0x3c>
 5f0:   90                      nop
 5f1:   8b 5d fc                mov    -0x4(%ebp),%ebx
 5f4:   c9                      leave
 5f5:   c3                      ret
```

在<do_phase>中发现一个名为 x86.get_PC_thunk.ax 的 call 调用，跟进发现其中的执行逻辑是：

```
000005b1 <__x86.get_pc_thunk.ax>:
 5b1:   8b 04 24                mov    (%esp),%eax
 5b4:   c3                      ret
```

此函数可以通过被写的寄存器来通过偏移量访问 global 类型， call 调用之后此时 PC 指向下一条指令，同时将这条指令的地址压入栈中，进入 x86.get_PC_thunk.ax 之后，将栈顶的值赋值给指定的寄存器（后缀 ax 代表是%eax），这时候指定寄存器中就放着可以用来相对寻址的下一条指令的位置。



call __x86.get_pc_thunk.ax

add $0x1a63, %eax

实现了将%eax 指向_GLOBAL_OFFSET_TABLE_的功能，_GLOBAL_OFFSET_TABLE_用来定位 global 变量的真实（运行时）地址，对于上图的

40 00 00 00 和 63 1a 00 00

实现了将%eax 指向_GLOBAL_OFFSET_TABLE_的功能，_GLOBAL_OFFSET_TABLE_用来定位 global 变量的真实（运行时）地址，对于上图的

40 00 00 00 和 63 1a 00 00

可以看见在<xdkuJjfq>函数中，

call 460<__x86.get_pc_thunk.bx>

add $0x1a13,%ebx

leal -0x18b0(%ebx),%eax

前两步将%ebx 指向_GLOBAL_OFFSET_TABLE_，后一步也是一个重定向之后确定的值，重定向之后%eax 指向了.rodata，就是 MYID。所以只需要将%eax 也指向.rodata 就可以了。在 do_phase 的 nop 之前 eax 也已经指向了_GLOBAL_OFFSET_TABLE_，所以只需要

leal -0x18b0(%eax),%eax

就在 do_phase 中也使%eax 指向了.rodata，将之作为参数压栈，然后 call 指令执行相对跳转，使 eax 出栈“恢复现场”。

编写汇编代码如下：

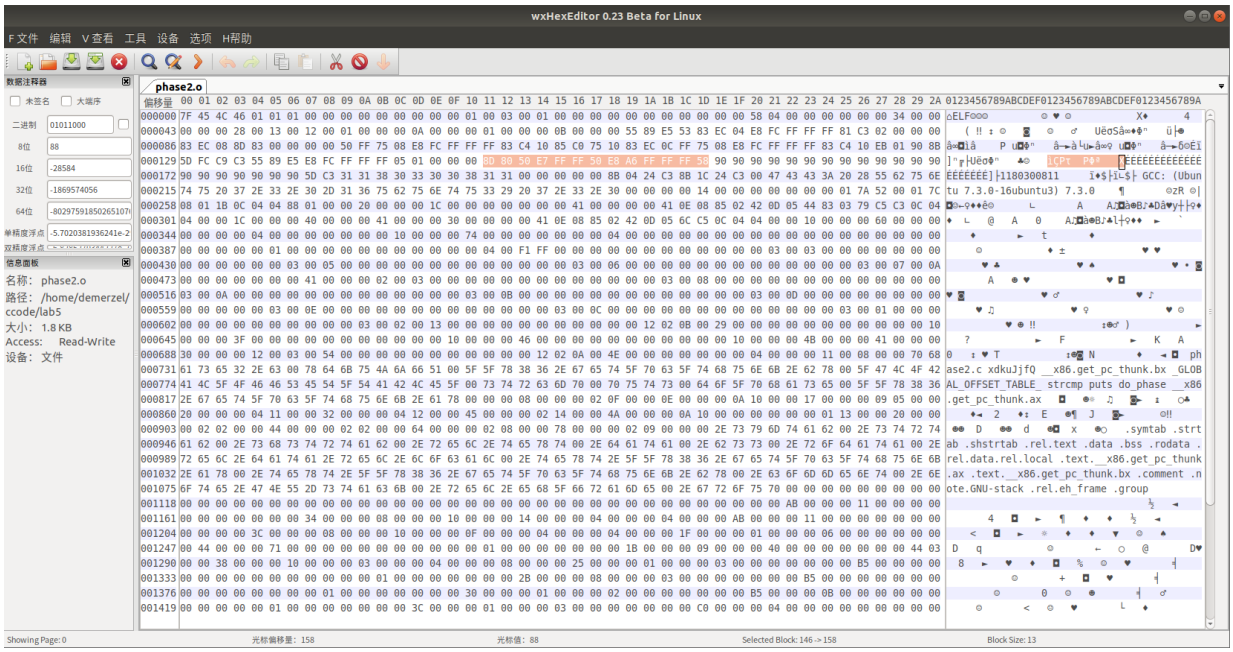
```
lea -0x18b0(%eax),%eax
push %eax
call 0xfff0dfa6
pop %eax
```

编译、反汇编后得到以下字节：

```
00000000 <.text>:
   0:   8d 80 50 e7 ff ff       lea    -0x18b0(%eax),%eax
   6:   50                      push   %eax
   7:   e8 a6 ff ff ff          call   0xffffffa6
   c:   58                      pop    %eax
```

得到插入值：8d 80 50 e7 ff ff 50 e8 a6 ff ff ff 58

用 wxHexeditor 打开 phase1.o，在第一个 90 处插入字节码，重新链接后得到上图输出结果。



3.3 阶段 3 的分析

程序运行结果截图：

```
demerzel@demerzel-virtual-machine:~/ccode/lab5$  gcc -m32 -c phase3_patch.c
demerzel@demerzel-virtual-machine:~/ccode/lab5$  gcc -m32 -o linkbomb3 main.o phase3.o phase3_patch.o
demerzel@demerzel-virtual-machine:~/ccode/lab5$ ./linkbomb3
1180300811
```

分析与设计的过程：

使用 readelf 确定 PPT 中的 PHASE3_CODEBOOK 数组对应的名称。

readelf -a phase3.o 得到：

```
Symbol table '.symtab' contains 18 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    2
     3: 00000000     0 SECTION LOCAL  DEFAULT    4
     4: 00000000     0 SECTION LOCAL  DEFAULT    5
     5: 00000000     0 SECTION LOCAL  DEFAULT    6
     6: 00000000     0 SECTION LOCAL  DEFAULT    8
     7: 00000000     0 SECTION LOCAL  DEFAULT   10
     8: 00000000     0 SECTION LOCAL  DEFAULT   11
     9: 00000000     0 SECTION LOCAL  DEFAULT    9
    10: 00000000     0 SECTION LOCAL  DEFAULT    1
    11: 00000020   256 OBJECT  GLOBAL DEFAULT  COM eintuxr0qZ
    12: 00000000   149 FUNC    GLOBAL DEFAULT    2 do_phase
    13: 00000000     0 FUNC    GLOBAL HIDDEN     8 __x86.get_pc_thunk.bx
    14: 00000000     0 NOTYPE  GLOBAL DEFAULT  UND _GLOBAL_OFFSET_TABLE_
    15: 00000000     0 NOTYPE  GLOBAL DEFAULT  UND putchar
    16: 00000000     0 NOTYPE  GLOBAL HIDDEN   UND __stack_chk_fail_local
    17: 00000000     4 OBJECT  GLOBAL DEFAULT    6 phase
```

可见，PHASE3_CODEBOOK 的实际名称是 eintuxr0qZ

phase3 链接并反汇编后，观察 do_phase 函数，得到以下循环部分：

```
00000605 <do_phase>:
 605:   55                      push   %ebp
 606:   89 e5                   mov    %esp,%ebp
 608:   53                      push   %ebx
 609:   83 ec 24                sub    $0x24,%esp
 60c:   e8 9f fe ff ff          call   4b0 <__x86.get_pc_thunk.bx>
 611:   81 c3 bf 19 00 00       add    $0x19bf,%ebx
 617:   65 a1 14 00 00 00       mov    %gs:0x14,%eax
 61d:   89 45 f4                mov    %eax,-0xc(%ebp)
 620:   31 c0                   xor    %eax,%eax
 622:   c7 45 e9 68 6e 6c 69    movl   $0x696c6e68,-0x17(%ebp)
 629:   c7 45 ed 76 64 71 66    movl   $0x66716476,-0x13(%ebp)
 630:   66 c7 45 f1 61 70       movw   $0x7061,-0xf(%ebp)
 636:   c6 45 f3 00             movb   $0x0,-0xd(%ebp)
 63a:   c7 45 e4 00 00 00 00    movl   $0x0,-0x1c(%ebp)
 641:   eb 2b                   jmp    66e <do_phase+0x69>
 643:   8d 55 e9                lea    -0x17(%ebp),%edx
 646:   8b 45 e4                mov    -0x1c(%ebp),%eax
 649:   01 d0                   add    %edx,%eax
 64b:   0f b6 00                movzbl (%eax),%eax
 64e:   0f b6 c0                movzbl %al,%eax
 651:   8d 93 70 00 00 00       lea    0x70(%ebx),%edx
 657:   0f b6 04 02             movzbl (%edx,%eax,1),%eax
 65b:   0f be c0                movsbl %al,%eax
 65e:   83 ec 0c                sub    $0xc,%esp
 661:   50                      push   %eax
 662:   e8 e9 fd ff ff          call   450 <putchar@plt>
 667:   83 c4 10                add    $0x10,%esp
 66a:   83 45 e4 01             addl   $0x1,-0x1c(%ebp)
 66e:   8b 45 e4                mov    -0x1c(%ebp),%eax
 671:   83 f8 09                cmp    $0x9,%eax
 674:   76 cd                   jbe    643 <do_phase+0x3e>
 676:   83 ec 0c                sub    $0xc,%esp
 679:   6a 0a                   push   $0xa
 67b:   e8 d0 fd ff ff          call   450 <putchar@plt>
 680:   83 c4 10                add    $0x10,%esp
 683:   90                      nop
 684:   8b 45 f4                mov    -0xc(%ebp),%eax
 687:   65 33 05 14 00 00 00    xor    %gs:0x14,%eax
 68e:   74 05                   je     695 <do_phase+0x90>
 690:   e8 7b 00 00 00          call   710 <__stack_chk_fail_local>
 695:   8b 5d fc                mov    -0x4(%ebp),%ebx
 698:   c9                      leave
 699:   c3                      ret
```

再查看 %ebp-0x17 出的内容，即可得到 COOKIE 字符串。

```
(gdb) x/s $ebp-0x17
0xffffce71:     "hnlivdqfap"
```

得到 cookie 字符串所对应的 ASCII 码值为：104 110 108 105 118 100 113 102 97 112 ，根据程序输出的顺序，只需要在 eintuxr0qZ [] 数组中，并且 COOKIE 的 ASCII 码对应的位置输入学号即可，其余位置不输出，可随便填入

构造字符串如下：

```
char eintuxrOqZ [256] =
"hhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhh

ıhhhhhhhhhhhhh1hh0h8h10hh8h1h10hhhh3hhhhhhhhhhhhhhhhhhr
```

之后编译，链接后输出如上。

3.4 阶段 4 的分析

程序运行结果截图：

分析与设计的过程：

通过 edb 获得 cookie 的值如下：

```
ff86:0690 514a41fc  AJQ
ff86:0694 55424f50 POBU
ff86:0698 0046544c LTF.
```

查看 switch 处主体代码：

```
646:   8d 55 e9              lea    -0x17(%ebp),%edx
649:   8b 45 e4              mov    -0x1c(%ebp),%eax
64c:   01 d0                 add    %edx,%eax
64e:   0f b6 00              movzbl (%eax),%eax
651:   88 45 e3              mov    %al,-0x1d(%ebp)
654:   0f be 45 e3           movsbl -0x1d(%ebp),%eax
658:   83 e8 41              sub    $0x41,%eax
65b:   83 f8 19              cmp    $0x19,%eax
65e:   0f 87 b5 00 00 00     ja     719 <.L30+0x5>
664:   c1 e0 02              shl    $0x2,%eax
667:   8b 84 18 a4 e8 ff ff  mov    -0x175c(%eax,%ebx,1),%eax
66e:   01 d8                 add    %ebx,%eax
670:   ff e0                 jmp    *%eax
```

得知代码的主要功能就是计算地址值到 eax，然后跳转到 eax 所指向的地址。eax 的计算公式为*（（%eax-0x41）<<2+%ebx-0x176c），这里是将 cookie 值作为索引映射到 switch 跳转表，通过 %eax=%ebx+偏移量 获得存储在.text 段中的 case 代码段地址，之后跳转执行。

查看跳转表如下：

```
00000672 <.L4>:
 672:   c6 45 e3 53             movb   $0x53,-0x1d(%ebp)
 676:   e9 9e 00 00 00          jmp    719 <.L30+0x5>

0000067b <.L6>:
 67b:   c6 45 e3 39             movb   $0x39,-0x1d(%ebp)
 67f:   e9 95 00 00 00          jmp    719 <.L30+0x5>

00000684 <.L7>:
 684:   c6 45 e3 34             movb   $0x34,-0x1d(%ebp)
 688:   e9 8c 00 00 00          jmp    719 <.L30+0x5>

0000068d <.L8>:
 68d:   c6 45 e3 31             movb   $0x31,-0x1d(%ebp)
 691:   e9 83 00 00 00          jmp    719 <.L30+0x5>

00000696 <.L9>:
 696:   c6 45 e3 45             movb   $0x45,-0x1d(%ebp)
 69a:   eb 7d                   jmp    719 <.L30+0x5>

0000069c <.L10>:
 69c:   c6 45 e3 38             movb   $0x38,-0x1d(%ebp)
 6a0:   eb 77                   jmp    719 <.L30+0x5>

000006a2 <.L11>:
 6a2:   c6 45 e3 33             movb   $0x33,-0x1d(%ebp)
 6a6:   eb 71                   jmp    719 <.L30+0x5>

000006a8 <.L12>:
 6a8:   c6 45 e3 71             movb   $0x71,-0x1d(%ebp)
 6ac:   eb 6b                   jmp    719 <.L30+0x5>

000006ae <.L13>:
 6ae:   c6 45 e3 35             movb   $0x35,-0x1d(%ebp)
 6b2:   eb 65                   jmp    719 <.L30+0x5>

000006b4 <.L14>:
 6b4:   c6 45 e3 45             movb   $0x45,-0x1d(%ebp)
 6b8:   eb 5f                   jmp    719 <.L30+0x5>

000006ba <.L15>:
 6ba:   c6 45 e3 6c             movb   $0x6c,-0x1d(%ebp)
 6be:   eb 59                   jmp    719 <.L30+0x5>


000006ba <.L15>:
 6ba:   c6 45 e3 6c             movb   $0x6c,-0x1d(%ebp)
 6be:   eb 59                   jmp    719 <.L30+0x5>

000006c0 <.L16>:
 6c0:   c6 45 e3 78             movb   $0x78,-0x1d(%ebp)
 6c4:   eb 53                   jmp    719 <.L30+0x5>

000006c6 <.L17>:
 6c6:   c6 45 e3 60             movb   $0x60,-0x1d(%ebp)
 6ca:   eb 4d                   jmp    719 <.L30+0x5>

000006cc <.L18>:
 6cc:   c6 45 e3 30             movb   $0x30,-0x1d(%ebp)
 6d0:   eb 47                   jmp    719 <.L30+0x5>

000006d2 <.L19>:
 6d2:   c6 45 e3 7a             movb   $0x7a,-0x1d(%ebp)
 6d6:   eb 41                   jmp    719 <.L30+0x5>

000006d8 <.L20>:
 6d8:   c6 45 e3 79             movb   $0x79,-0x1d(%ebp)
 6dc:   eb 3b                   jmp    719 <.L30+0x5>

000006de <.L21>:
 6de:   c6 45 e3 36             movb   $0x36,-0x1d(%ebp)
 6e2:   eb 35                   jmp    719 <.L30+0x5>

000006e4 <.L22>:
 6e4:   c6 45 e3 37             movb   $0x37,-0x1d(%ebp)
 6e8:   eb 2f                   jmp    719 <.L30+0x5>

000006ea <.L23>:
 6ea:   c6 45 e3 7a             movb   $0x7a,-0x1d(%ebp)
 6ee:   eb 29                   jmp    719 <.L30+0x5>

000006f0 <.L24>:
 6f0:   c6 45 e3 46             movb   $0x46,-0x1d(%ebp)
 6f4:   eb 23                   jmp    719 <.L30+0x5>

000006f6 <.L25>:
 6f6:   c6 45 e3 4c             movb   $0x4c,-0x1d(%ebp)
 6fa:   eb 1d                   jmp    719 <.L30+0x5>
```

```
000006fc <.L26>:
 6fc:   c6 45 e3 4a             movb   $0x4a,-0x1d(%ebp)
 700:   eb 17                   jmp    719 <.L30+0x5>

00000702 <.L27>:
 702:   c6 45 e3 66             movb   $0x66,-0x1d(%ebp)
 706:   eb 11                   jmp    719 <.L30+0x5>

00000708 <.L28>:
 708:   c6 45 e3 71             movb   $0x71,-0x1d(%ebp)
 70c:   eb 0b                   jmp    719 <.L30+0x5>

0000070e <.L29>:
 70e:   c6 45 e3 41             movb   $0x41,-0x1d(%ebp)
 712:   eb 05                   jmp    719 <.L30+0x5>

00000714 <.L30>:
 714:   c6 45 e3 32             movb   $0x32,-0x1d(%ebp)
 718:   90                      nop
```

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

本次链接实验让我对 c 文件链接的具体原理和强弱符号有了更深的理解，也进一步了解了 gdb、edb 等工具的使用方法。

4.2 请给出对本次实验内容的建议

phase2 相对于 phase3 难度过大，建议 phase2 与 phase3 互换，之后降低前两个阶段的分数，让更多人去做后面的实验。

参考文献

[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.