

哈爾濱工業大學

实验报告

实 验（五）

题 目 LinkLab

链接

专 业 计算机

学 号 1180300308

班 级 03003

学 生 刘义

指 导 教 师 史先俊

实 验 地 点 G712

实 验 日 期 2019 年 11 月 20 日

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	- 5 -
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）	- 5 -
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）	- 5 -
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。	- 6 -
（5 分）	- 6 -
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）	- 9 -
第 3 章 各阶段的原理与方法	- 10 -
3.1 阶段 1 的分析.....	- 10 -
3.2 阶段 2 的分析	- 11 -
3.3 阶段 3 的分析	- 13 -
3.4 阶段 4 的分析	- 15 -
3.5 阶段 5 的分析	- 16 -
第 4 章 总结	- 17 -
4.1 请总结本次实验的收获.....	- 17 -
4.2 请给出对本次实验内容的建议.....	- 17 -
参考文献	- 18 -

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。
- 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。

■ 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 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 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)
⇕
⇕
共享栈的内存映射区域
⇕
运行时堆 (由 malloc 创建)
读/写段 (.data, .bss)

只读代码段 (.init,.text,.rodata)

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)

所属区	符号、地址、空间（从小到大）
0(null)	p5 (nil) 0
只读代码段 (.init , .text , .rodata)	show_pointer 0x5612669dd875 94637031020661 useless 0x5612669dd86a 94637031020650 main 0x5612669dd8a8 94637031020712
读/写段 (.data , .bss)	big array 0x5612a6bdf0e0 94638106865888 huge array 0x561266bdf0e0 94637033124064 global 0x561266bdf020 94637033123872 p2 0x7f939e58b010 140271993532432
运行时堆	p1 0x7f938dfb3010 140271718969360 p3 0x7f939e56a010 140271993397264 p4 0x7f934dfb2010 140270645223440
共享库的内存映射区域	exit 0x7f939dff7120 140271987683616 printf 0x7f939e018e80 140271987822208 malloc 0x7f939e04b070 140271988027504 free 0x7f939e04b950 140271988029776 strcpy 0x7f939e06a540 140271988155712
用户栈（运行时创建）	argc 0x7ffedd9b47fc 140732616361980 argv 0x7ffedd9b4d38 140732616363320 argv[0] 7ffedd9b5288 argv[1] 7ffedd9b5296 argv[2] 7ffedd9b5299 argv[3] 7ffedd9b52a4 argv[0] 0x7ffedd9b5288 140732616364680 ./LinkAddress argv[1] 0x7ffedd9b5296 140732616364694 -u argv[2] 0x7ffedd9b5299 140732616364697 1180300308 argv[3] 0x7ffedd9b52a4 140732616364708 刘义 env 0x7ffedd9b4d60 140732616363360

env[0]	*env 0x7ffedd9b52ab 140732616364715
CLUTTER_IM_MODULE=xim	
env[1]	*env 0x7ffedd9b52c1 140732616364737
env[2]	*env 0x7ffedd9b58ad 140732616366253
LC_MEASUREMENT=zh_CN.UTF-8	
env[3]	*env 0x7ffedd9b58c8 140732616366280
LESSCLOSE=/usr/bin/lesspipe %s %s	
env[4]	*env 0x7ffedd9b58ea 140732616366314
LC_PAPER=zh_CN.UTF-8	
env[5]	*env 0x7ffedd9b58ff 140732616366335
LC_MONETARY=zh_CN.UTF-8	
env[6]	*env 0x7ffedd9b5917 140732616366359
XDG_MENU_PREFIX=gnome-	
env[7]	*env 0x7ffedd9b592e 140732616366382
LANG=zh_CN.UTF-8	
env[8]	*env 0x7ffedd9b593f 140732616366399
DISPLAY=:0	
env[9]	*env 0x7ffedd9b594a 140732616366410
GNOME_SHELL_SESSION_MODE=ubuntu	
env[10]	*env 0x7ffedd9b596a 140732616366442
COLORTERM=truecolor	
env[11]	*env 0x7ffedd9b597e 140732616366462
USERNAME=1180300308 刘义	
env[12]	*env 0x7ffedd9b5998 140732616366488
XDG_VTNR=2	
env[13]	*env 0x7ffedd9b59a3 140732616366499
SSH_AUTH_SOCK=/run/user/1000/keyring/ssh	
env[14]	*env 0x7ffedd9b59cc 140732616366540
LC_NAME=zh_CN.UTF-8	
env[15]	*env 0x7ffedd9b59e0 140732616366560
XDG_SESSION_ID=2	
env[16]	*env 0x7ffedd9b59f1 140732616366577
USER=1180300308 刘义	
env[17]	*env 0x7ffedd9b5a07 140732616366599
DESKTOP_SESSION=ubuntu	
env[18]	*env 0x7ffedd9b5a1e 140732616366622
QT4_IM_MODULE=fcitx	
env[19]	*env 0x7ffedd9b5a32 140732616366642
TEXTDOMAINDIR=/usr/share/locale/	
env[20]	*env 0x7ffedd9b5a53 140732616366675
env[21]	*env 0x7ffedd9b5aa9 140732616366761
PWD=/home/dongbaiyue/csapp/program/csapp.lab5/linklab-1180300308	
env[22]	*env 0x7ffedd9b5aea 140732616366826
HOME=/home/dongbaiyue	
env[23]	*env 0x7ffedd9b5b00 140732616366848
TEXTDOMAIN=im-config	
env[24]	*env 0x7ffedd9b5b15 140732616366869
SSH_AGENT_PID=1420	
env[25]	*env 0x7ffedd9b5b28 140732616366888
QT_ACCESSIBILITY=1	
env[26]	*env 0x7ffedd9b5b3b 140732616366907
XDG_SESSION_TYPE=x11	
env[27]	*env 0x7ffedd9b5b50 140732616366928
env[28]	*env 0x7ffedd9b5ba5 140732616367013
XDG_SESSION_DESKTOP=ubuntu	
env[29]	*env 0x7ffedd9b5bc0 140732616367040
LC_ADDRESS=zh_CN.UTF-8	
env[30]	*env 0x7ffedd9b5bd7 140732616367063
GJS_DEBUG_OUTPUT=stderr	
env[31]	*env 0x7ffedd9b5bef 140732616367087
LC_NUMERIC=zh_CN.UTF-8	
env[32]	*env 0x7ffedd9b5c06 140732616367110

	<pre> GTK_MODULES=gail:atk-bridge env[33] *env 0x7ffedd9b5c22 140732616367138 PAPERSIZE=a4 env[34] *env 0x7ffedd9b5c2f 140732616367151 WINDOWPATH=2 env[35] *env 0x7ffedd9b5c3c 140732616367164 TERM=xterm-256color env[36] *env 0x7ffedd9b5c50 140732616367184 SHELL=/bin/bash env[37] *env 0x7ffedd9b5c60 140732616367200 VTE_VERSION=5202 env[38] *env 0x7ffedd9b5c71 140732616367217 QT_IM_MODULE=fcitx env[39] *env 0x7ffedd9b5c84 140732616367236 XMODIFIERS=@im=fcitx env[40] *env 0x7ffedd9b5c99 140732616367257 IM_CONFIG_PHASE=2 env[41] *env 0x7ffedd9b5cab 140732616367275 XDG_CURRENT_DESKTOP=ubuntu:GNOME env[42] *env 0x7ffedd9b5ccc 140732616367308 GPG_AGENT_INFO=/run/user/1000/gnupg/S.gpg-agent:0:1 env[43] *env 0x7ffedd9b5d00 140732616367360 GNOME_TERMINAL_SERVICE=:1.263 env[44] *env 0x7ffedd9b5d1e 140732616367390 XDG_SEAT=seat0 env[45] *env 0x7ffedd9b5d2d 140732616367405 SHLVL=1 env[46] *env 0x7ffedd9b5d35 140732616367413 LANGUAGE=zh_CN:en_US:en env[47] *env 0x7ffedd9b5d4d 140732616367437 LC_TELEPHONE=zh_CN.UTF-8 env[48] *env 0x7ffedd9b5d66 140732616367462 GDMSESSION=ubuntu env[49] *env 0x7ffedd9b5d78 140732616367480 GNOME_DESKTOP_SESSION_ID=this-is-deprecated env[50] *env 0x7ffedd9b5da4 140732616367524 LOGNAME=1180300308 刘义 env[51] *env 0x7ffedd9b5dbd 140732616367549 DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus env[52] *env 0x7ffedd9b5df3 140732616367603 XDG_RUNTIME_DIR=/run/user/1000 env[53] *env 0x7ffedd9b5e12 140732616367634 XAUTHORITY=/run/user/1000/gdm/Xauthority env[54] *env 0x7ffedd9b5e3b 140732616367675 XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/etc/xdg env[55] *env 0x7ffedd9b5e68 140732616367720 env[56] *env 0x7ffedd9b5ed0 140732616367824 LC_IDENTIFICATION=zh_CN.UTF-8 env[57] *env 0x7ffedd9b5eee 140732616367854 GJS_DEBUG_TOPICS=JS ERROR;JS LOG env[58] *env 0x7ffedd9b5f0f 140732616367887 env[59] *env 0x7ffedd9b5f61 140732616367969 LESSOPEN= /usr/bin/lesspipe %s env[60] *env 0x7ffedd9b5f81 140732616368001 GTK_IM_MODULE=fcitx env[61] *env 0x7ffedd9b5f95 140732616368021 LC_TIME=zh_CN.UTF-8 env[62] *env 0x7ffedd9b5fa9 140732616368041 OLDPWD=/home/dongbaiyue/csapp/program/csapp.lab5 env[63] *env 0x7ffedd9b5fda 140732616368090 _=/LinkAddress </pre>
--	--

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)

时间段	执行的子程序
执行 main 函数前	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
执行 main 函数后	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：

```
1180300308刘义@ubuntu:~/csapp/program/csapp.lab5/linklab-1180300308$ gcc -m32 main.o phase1.o -o linkbomb
1180300308刘义@ubuntu:~/csapp/program/csapp.lab5/linklab-1180300308$ ls
a.txt      linkbomb2      linkbomb.obj   phase2.o      phase3_patch.o
linkaddr.c linkbomb3      main.o         phase2.obj     phase4.o
LinkAddress linkbomb3.idb  phase1.elf     phase3.o       phase5.o
linkbomb   linkbomb3.obj  phase1.o       phase3_patch.c
1180300308刘义@ubuntu:~/csapp/program/csapp.lab5/linklab-1180300308$ ./linkbomb
1180300308
```

分析与设计的过程：

首先，尝试将 main.o 与未修改的 phase1.o 链接生成可执行文件，运行：

```
1180300308刘义@ubuntu:~/csapp/program/csapp.lab5/linklab-1180300308$ gcc -m32 -o linkbomb main.o phase1.o
1180300308刘义@ubuntu:~/csapp/program/csapp.lab5/linklab-1180300308$ ./linkbomb
JL6UI5BkB Wn50aNYeR NX8oRLR57c4ikk43 h9W5lXUigXeIU67gxhZ
```

这是一串没有什么特殊意义字符串，只需将该字符串的前部替换为学号 1180300308，最终即可在屏幕上输出我的学号。

printf(“%s\n”,s)输出函数最终会被优化为 puts(s)，s 为字符串常量因此被保存在 phase1.o 的.data 节数据段中。

用 hexedit 打开 phase1.o，在上图字符串对应的二进制区域，将其修改为“31 31 38 30 33 30 30 33 30 38 00”，即“1180300308”，最后的“00”字节表示字符串结束。修改后截图如下：

```
00000070  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000080  33 4D 55 56 4B 69 30 59 57 66 50 4F 64 4F 76 68 3MUVKi0YWfPodOvh
00000090  75 62 76 79 43 53 6A 6F 50 55 4F 4A 74 51 55 52 ubvyCSjoPU0JtQUR
000000A0  4F 6E 52 62 4D 41 5A 49 65 42 20 7A 4E 4D 77 4E OnRbMAZIEB zNMwN
000000B0  35 31 31 38 30 33 30 30 33 30 38 00 6E 35 4F 61 51180300308.n50a
000000C0  4E 59 65 52 09 4E 58 38 6F 52 4C 52 35 37 63 34 NYeR.NX8oRLR57c4
000000D0  69 6B 6B 34 33 20 68 39 57 35 6C 58 55 69 67 58 ikk43 h9W5lXUigX
```

3.2 阶段 2 的分析

程序运行结果截图：

```
1180300308刘义@ubuntu:~/csapp/program/csapp.lab5/linklab-1180300308$ gcc -m32 main.o phase2.o -o linkbomb2
1180300308刘义@ubuntu:~/csapp/program/csapp.lab5/linklab-1180300308$ ./linkbomb2
1180300308
```

分析与设计的过程：

首先分析 phase2.c 的程序框架：

□ 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 );
}

void do_phase() {
    // 在代码节中预留存储位置供学生插入完成功能的必要指令
    asm( "nop\n\tnop\n\tnop\n\tnop\n\tnop\n\tnop\n\tnop\n\tnop\n\tnop\n\tnop\n\t..." );
}
```

do_phase 是入口函数，要输出学号，需要在 do_phase 中调用包含 printf 的函数，并让 id = MYID。

将 main.o 与未修改的 phase2.o 链接生成可执行文件，在 objdump 反汇编，查看代码如下：

```

000005b5 <fkJmyWeb>:
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 <fkJmyWeb+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 <fkJmyWeb+0x3c>
5f0: 90          nop
5f1: 8b 5d fc    mov     -0x4(%ebp),%ebx
5f4: c9          leave
5f5: c3          ret

000005f6 <do_phase>:
5f6: 55          push    %ebp
5f7: 89 e5       mov     %esp,%ebp
5f9: e8 b3 ff ff ff call    5b1 <__x86.get_pc_thunk.ax>
5fe: 05 d6 19 00 00 add     $0x19d6,%eax
603: 90          nop
604: 90          nop

```

包含 printf 的函数为 fkJmyWeb。

函数 fkJmyWeb 中的:

```

5bc: e8 9f fe ff ff call    460 <__x86.get_pc_thunk.bx>
5c1: 81 c3 13 1a 00 00 add     $0x1a13,%ebx

```

和函数 do_phase 的:

```

5f9: e8 b3 ff ff ff call    5b1 <__x86.get_pc_thunk.ax>
5fe: 05 d6 19 00 00 add     $0x19d6,%eax

```

分别是使 %ebx 和 %eax 指向 _GLOBAL_OFFSET_TABLE_。

然后，函数 fkJmyWeb 中的:

```

5ca: 8d 83 50 e7 ff ff lea     -0x18b0(%ebx),%eax

```

是使%ebx 指向了.rodata，即 MYID。

要使 id = MYID，可以让函数 do_phase 中的%eax 指向 MYID，然后将%eax 压栈，再调用函数 fkJmyWeb，以作为函数 fkJmyWeb 的参数 id。编写汇编代码如下：

```
lea -0x18b0(%eax),%eax //让%eax指向MYID
push %eax //将%eax压栈
call 0xffffffa6 //相对偏移-0x5a
pop %eax //恢复现场
```

生成.o 文件后反汇编得到相应二进制字节

```
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
```

使用 readelf 得到 phase2.o 的 elf 信息，查看节头，.text 相对于 elf 文件头的偏移地址 0x44：

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		16	20	4
[2]	.group	GROUP	00000000	00003c	000008	04		16	15	4
[3]	.text	PROGBITS	00000000	000044	000071	00	AX	0	0	1

do_phase 中第一个 nop 字节相对于.text 节的偏移量 0x4e，nop 在 phase2.o 中的起始地址为 0x44+0x4e=0x92。

使用 hexedit 打开 phase2.o，根据偏移量修改相应字节为要注入的代码：

```
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 80 50 E7 FF FF 50 E8 A6 FF FF FF 58 90  ....P...P....X.
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 30 33 30 30 33 30 38 00  ...].1180300308.
000000C0  00 00 00 00 8B 04 24 C3 8B 1C 24 C3 00 47 43 43  ....$...$..GCC
```

3.3 阶段 3 的分析

程序运行结果截图：

```
1180300308刘义@ubuntu:~/csapp/program/csapp.lab5/linklab-1180300308$ gcc -m32 -c
phase3_patch.c -o phase3_patch.o
1180300308刘义@ubuntu:~/csapp/program/csapp.lab5/linklab-1180300308$ gcc -m32 ma
in.o phase3.o phase3_patch.o -o linkbomb3
1180300308刘义@ubuntu:~/csapp/program/csapp.lab5/linklab-1180300308$ ./linkbomb3
1180300308
```

分析与设计的过程：

首先分析 phase3.c 的程序框架

□ 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" );
}
```

cookie 字符串由一组英文字母组成，总长度与学号字符串相同。phase3.c 将该字符串的英文字符对应的 ASCII 编码对应的字符数组的相应元素输出。

phase3.c 中的字符数组未初始化、为弱符号，只需在新建的 phase3_patch.c 文件中定义相应的强符号即可。

链接 phase3.o 与 main.o 生成可执行文件，查看其反汇编文件，do_phase:

```

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 75 65 6f 77 movl    $0x776f6575,-0x17(%ebp)
629: c7 45 ed 79 78 6d 68 movl    $0x686d7879,-0x13(%ebp)
630: 66 c7 45 f1 6c 69 movw    $0x696c,-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>

```

分析可知，字符串 cookie 在 %ebp-0x17 的位置，gdb 调试打印出其值：

```
(gdb) x/10cb $ebp-0x17
0xffffcf61:    117 'u' 101 'e' 111 'o' 119 'w' 121 'y' 120 'x' 109 'm' 104 'h'
0xffffcf69:    108 'l' 105 'i'
```

字符数组在 `ebx+0x70`（即 `edx`）的位置，打印其值：

```
(gdb) x/10cb $edx
0x56557040 <YNDMcIjLSC>:      0 '\000'      0 '\000'      0 '\000'      0 '\000'
                                0 '\000'      0 '\000'      0 '\000'      0 '\000'
                                0 '\000'      0 '\000'      0 '\000'      0 '\000'
```

字符数组未初始化，其值默认为 0，字符数组名为：YNDMcIjLSC。

创建 `phase3_patch.c` 文件，定义强符号 `YNDMcIjLSC`，要使程序输出学号，需要使字符串 `cookie` 对应的字符数组元素分别为 `1180300308`，比如：`YNDMcIjLSC[117] = '1'`。

[illegible]

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

- 1) ELF 文件结构
- 2) 更深入地理解了链接的原理细节

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>.