

# 哈尔滨工业大学

# 实验报告

## 实验（五）

题 目 LinkLab

链接

专 业 计算机

学 号 1190201423

班 级 1903004

学 生 顾海耀

指 导 教 师 史先俊

实 验 地 点 G709

实 验 日 期 2021/5/19

## 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 .....	- 11 -
3.1 阶段 1 的分析 .....	- 11 -
3.2 阶段 2 的分析 .....	- 12 -
3.3 阶段 3 的分析 .....	- 15 -
3.4 阶段 4 的分析 .....	- 19 -
3.5 阶段 5 的分析 .....	- 19 -
第 4 章 总结 .....	- 20 -
4.1 请总结本次实验的收获 .....	- 20 -
4.2 请给出对本次实验内容的建议 .....	- 20 -
参考文献 .....	- 21 -

## 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

填 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名称、地址、空间。并按照 Linux 下 X64 内存映像标出其所属各区。(为方便统一, 请用 gcc -no-pie -fno-PIC 编译与连接)

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或

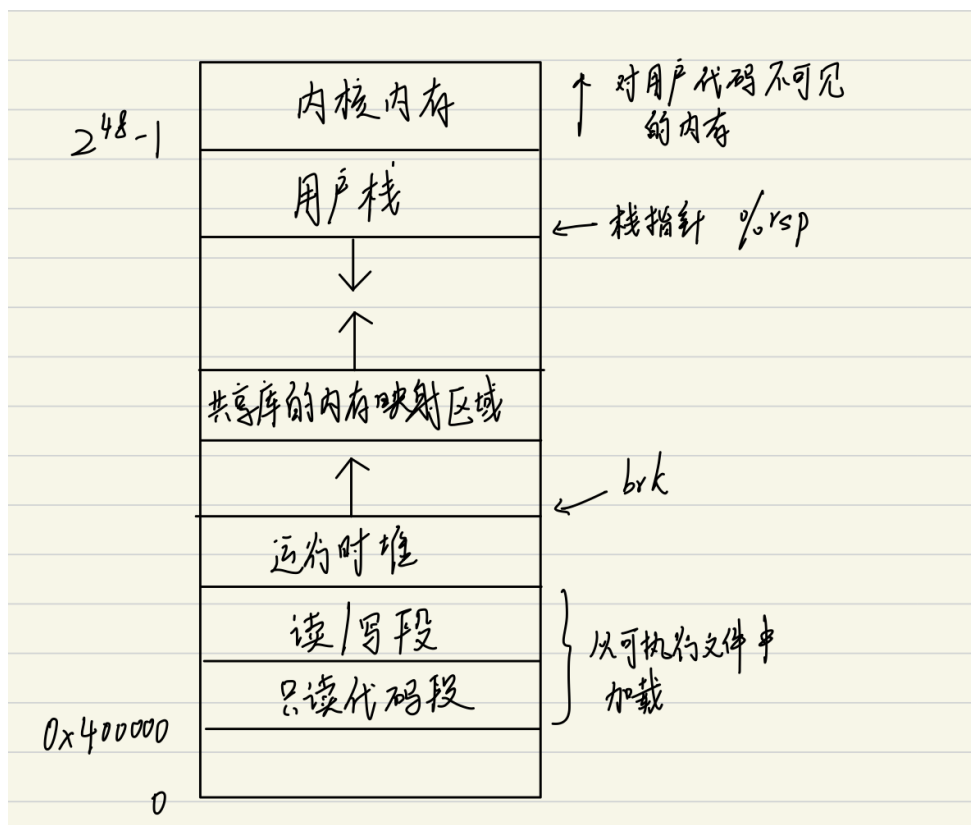
地址。(gcc 与 objdump/GDB/EDB)

## 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	字大小、字节顺序、ELF 头大小、目标文件类型、机器类型、节头部表的字节偏移、节头部表中条目的大小和数量
.text	以编译程序的机器代码
.rodata	只读数据
.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、所有初始化为 0 的全局或静态变量
.symtab	一个符号表
.rel.text	一个 .text 节中位置的列表
.rel.data	被模块引用或定义的所有全局变量的重定位信息
.debug	一个调试符号表
.line	原始 C 程序中的行号和 .text 节中的机器指令之间的映射
.strlab	一个字符串表
节头部表	描述目标文件的节

### 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)



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)

只读数据段 (.init、.text、.rodata)	<pre> .text : show_pointer 0x400762 4196194 useless 0x400757 4196183 main 0x400793 4196243 exit 0x400660 4195936 printf 0x400640 4195904 malloc 0x400650 4195920 free 0x400600 4195840 strcpy 0x400610 4195856  .rodata : pstr 0x400e20 4197920 gc 0x400e4c 4197964 cc 0x400e60 4197984 </pre>
读/写段 (.data、.bss)	<pre> .data : global 0x602080 6299776 glong 0x602088 6299784 cstr 0x6020a0 6299808 </pre>

	local static int 1 0x602110 6299920  .bss : big array 0x40602140 1080041792 huge array 0x602140 6299968 gint0 0x60212c 6299948 local static int 0 0x602130 6299952
运行时堆	p1 0x7fe7100fc010 140630383640592 p2 0x41dcd670 1104991856 p3 0x7fe7206da010 140630658228240 p4 0x7fe6d00fb010 140629309894672 p5 0x7fe6500fa010 140627162406928
用户栈	argc0x7ffcc229d4ec 140723566007532 argv 0x7ffcc229da28 140723566008872 argv[0] 7ffcc229f3da argv[1] 7ffcc229f3e8 argv[2] 7ffcc229f3eb argv[3] 7ffcc229f3f6 argv[0] 0x7ffcc229f3da 140723566015450 ./LinkAddress argv[1] 0x7ffcc229f3e8 140723566015464 -u argv[2] 0x7ffcc229f3eb 140723566015467 1190201423 argv[3] 0x7ffcc229f3f6 140723566015478 顾海耀 env 0x7ffcc229da50 140723566008912 env[0] *env 0x7ffcc229f400 140723566015488 CLUTTER_IM_MODULE=xim env[1] *env 0x7ffcc229f416 140723566015510 LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd a=01;31:*.tlz=01;31:*.txz=01;31:*.tzo=01;31:*.t7z=01;31:*.zip=01;31:*.z=01;3 01;31:*.ear=01;31:*.sar=01;31:*.rar=01;31:*.alz=01;31:*.ace=01;31:*.zoo=01;3 *.ppm=01;35:*.tga=01;35:*.xbm=01;35:*.xpm=01;35:*.tif=01;35:*.tiff=01;35:*. ;35:*.qt=01;35:*.nuv=01;35:*.wmv=01;35:*.asf=01;35:*.rm=01;35:*.rmvb=01; :*.mid=00;36:*.midi=00;36:*.mka=00;36:*.mp3=00;36:*.mpc=00;36:*.ogg=00; env[2] *env 0x7ffcc229fa02 140723566017026 LESSCLOSE=/usr/bin/lesspipe %s %s env[3] *env 0x7ffcc229fa24 140723566017060 XDG_MENU_PREFIX=gnome- env[4] *env 0x7ffcc229fa3b 140723566017083 LANG=en_US.UTF-8 env[5] *env 0x7ffcc229fa4c 140723566017100 DISPLAY=:0 env[6] *env 0x7ffcc229fa57 140723566017111 GNOME_SHELL_SESSION_MODE=ubuntu

env[7]	*env	0x7ffcc229fa77	140723566017143	COLORTERM=truecolor
env[8]	*env	0x7ffcc229fa8b	140723566017163	USERNAME=ghy1190201423
env[9]	*env	0x7ffcc229faa2	140723566017186	XDG_VTNR=2
env[10]	*env	0x7ffcc229faad	140723566017197	SSH_AUTH_SOCK=/run/user/1000/keyring/ssh
env[11]	*env	0x7ffcc229fad6	140723566017238	XDG_SESSION_ID=2
env[12]	*env	0x7ffcc229fae7	140723566017255	USER=ghy1190201423
env[13]	*env	0x7ffcc229fafa	140723566017274	DESKTOP_SESSION=ubuntu
env[14]	*env	0x7ffcc229fb11	140723566017297	QT4_IM_MODULE=fcitx
env[15]	*env	0x7ffcc229fb25	140723566017317	TEXTDOMAINDIR=/usr/share/locale/
env[16]	*env	0x7ffcc229fb46	140723566017350	GNOME_TERMINAL_SCREEN=/org/gnome/Terminal/screen/0f611ed6_1555
env[17]	*env	0x7ffcc229fb9c	140723566017436	PWD=/mnt/hgfs/hitics
env[18]	*env	0x7ffcc229fbb1	140723566017457	HOME=/home/ghy1190201423
env[19]	*env	0x7ffcc229fbca	140723566017482	TEXTDOMAIN=im-config
env[20]	*env	0x7ffcc229fbdf	140723566017503	SSH_AGENT_PID=2012
env[21]	*env	0x7ffcc229fbf2	140723566017522	QT_ACCESSIBILITY=1
env[22]	*env	0x7ffcc229fc05	140723566017541	XDG_SESSION_TYPE=x11
env[23]	*env	0x7ffcc229fc1a	140723566017562	XDG_DATA_DIRS=/usr/share/ubuntu:/usr/local/share/:/usr/share/:/var/lib/snapd
env[24]	*env	0x7ffcc229fc6f	140723566017647	XDG_SESSION_DESKTOP=ubuntu
env[25]	*env	0x7ffcc229fc8a	140723566017674	GTK_MODULES=gail:atk-bridge
env[26]	*env	0x7ffcc229fca6	140723566017702	WINDOWPATH=2
env[27]	*env	0x7ffcc229fcb3	140723566017715	TERM=xterm-256color
env[28]	*env	0x7ffcc229fcc7	140723566017735	SHELL=/bin/bash
env[29]	*env	0x7ffcc229fcd7	140723566017751	VTE_VERSION=5202
env[30]	*env	0x7ffcc229fce8	140723566017768	



	QT_IM_MODULE=fcitx env[31] *env 0x7ffcc229fcfb 140723566017787 XMODIFIERS=@im=fcitx env[32] *env 0x7ffcc229fd10 140723566017808 IM_CONFIG_PHASE=2 env[33] *env 0x7ffcc229fd22 140723566017826 XDG_CURRENT_DESKTOP=ubuntu:GNOME env[34] *env 0x7ffcc229fd43 140723566017859 GPG_AGENT_INFO=/run/user/1000/gnupg/S.gpg-agent:0:1 env[35] *env 0x7ffcc229fd77 140723566017911 GNOME_TERMINAL_SERVICE=:1.86 env[36] *env 0x7ffcc229fd94 140723566017940 XDG_SEAT=seat0 env[37] *env 0x7ffcc229fda3 140723566017955 SHLVL=1 env[38] *env 0x7ffcc229fdab 140723566017963 GDMSESSION=ubuntu env[39] *env 0x7ffcc229fdbd 140723566017981 GNOME_DESKTOP_SESSION_ID=this-is-deprecated env[40] *env 0x7ffcc229fde9 140723566018025 LOGNAME=ghy1190201423 env[41] *env 0x7ffcc229fdff 140723566018047 DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus env[42] *env 0x7ffcc229fe35 140723566018101 XDG_RUNTIME_DIR=/run/user/1000 env[43] *env 0x7ffcc229fe54 140723566018132 XAUTHORITY=/run/user/1000/gdm/Xauthority env[44] *env 0x7ffcc229fe7d 140723566018173 XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/etc/xdg env[45] *env 0x7ffcc229feaa 140723566018218 PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games env[46] *env 0x7ffcc229ff39 140723566018361 SESSION_MANAGER=local/ubuntu: @/tmp/.ICE-unix/1925,unix/ubuntu:/tmp/.ICE-unix/1925 env[47] *env 0x7ffcc229ff8b 140723566018443 LESSOPEN=  /usr/bin/lesspipe %s env[48] *env 0x7ffcc229ffab 140723566018475 GTK_IM_MODULE=fcitx env[49] *env 0x7ffcc229ffbf 140723566018495 OLDPWD=/home/ghy1190201423 env[50] *env 0x7ffcc229ffda 140723566018522 _=./LinkAddress
--	---

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)

执行前：

- <\_init>
- <free@plt>
- <strcpy@plt>
- <puts@plt>
- <\_stack\_chk\_fail@plt>
- <printf@plt>
- <malloc@plt>
- <exit@plt>
- <\_start>
- <\_dl\_relocate\_static\_pie>
- <deregister\_tm\_clones>
- <register\_tm\_clones>
- <\_\_do\_global\_ctors\_aux>
- <frame\_dummy>
- <useless>
- <show\_pointer>

执行后：

- <main>
- <\_\_libc\_csu\_init>
- <\_\_libc\_csu\_fini>
- <\_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：

```
ghy1190201423@ubuntu:/mnt/hgfs/hitacs/linklab-1190201423$ gcc -m32 -o linkbomb1 main.o phase1.o
ghy1190201423@ubuntu:/mnt/hgfs/hitacs/linklab-1190201423$ ./linkbomb1
1190201423
```

分析与设计的过程：

首先，使用 readelf 查看 phase1.o 里 elf 文件的内容：

```
Section Headers:
[Nr] Name                Type              Addr             Off             Size            ES Flg Lk Inf Al
[ 0]                      NULL              00000000          000000          000000          00  0  0  0  0
[ 1] .interp                PROGBITS          00000154          000154          000013          00  A  0  0  1
[ 2] .note.ABI-tag          NOTE              00000168          000168          000020          00  A  0  0  4
[ 3] .note.gnu.build-id     NOTE              00000188          000188          000024          00  A  0  0  4
[ 4] .gnu.hash              GNU_HASH          000001ac          0001ac          000020          04  A  5  0  4
[ 5] .dynsym                DYNSYM            000001cc          0001cc          000080          10  A  6  1  4
[ 6] .dynstr                STRTAB            0000024c          00024c          00009b          00  A  0  0  1
[ 7] .gnu.version           VERSYM            000002e8          0002e8          000010          02  A  5  0  2
[ 8] .gnu.version_r         VERNEED           000002f8          0002f8          000030          00  A  6  1  4
[ 9] .rel.dyn               REL               00000328          000328          000078          08  A  5  0  4
[10] .rel.plt               REL               000003a0          0003a0          000010          08  AI 5 22 4
[11] .init                  PROGBITS          000003b0          0003b0          000023          00  AX 0  0  4
[12] .plt                   PROGBITS          000003e0          0003e0          000030          04  AX 0  0 16
[13] .plt.got               PROGBITS          00000410          000410          000010          08  AX 0  0  8
[14] .text                  PROGBITS          00000420          000420          000202          00  AX 0  0 16
[15] .fini                  PROGBITS          00000624          000624          000014          00  AX 0  0  4
[16] .rodata                PROGBITS          00000638          000638          00007c          00  A  0  0  4
[17] .eh_frame_hdr          PROGBITS          000006b4          0006b4          00003c          00  A  0  0  4
[18] .eh_frame              PROGBITS          000006f0          0006f0          000100          00  A  0  0  4
[19] .init_array             INIT_ARRAY        00001ed0          000ed0          000004          04  WA 0  0  4
[20] .fini_array             FINI_ARRAY        00001ed4          000ed4          000004          04  WA 0  0  4
[21] .dynamic                DYNAMIC           00001ed8          000ed8          000100          08  WA 6  0  4
[22] .got                   PROGBITS          00001fd8          000fd8          000028          04  WA 0  0  4
[23] .data                  PROGBITS          00002000          001000          0000ec          00  WA 0  0 32
[24] .bss                   NOBITS            000020ec          0010ec          000004          00  WA 0  0  1
[25] .comment                PROGBITS          00000000          0010ec          000055          01  MS 0  0  1
[26] .symtab                 SYMTAB            00000000          001144          000460          10  27 45 4
[27] .strtab                 STRTAB            00000000          0015a4          00024c          00  0  0  1
[28] .shstrtab               STRTAB            00000000          0017f0          0000fc          00  0  0  1
```

看到.data 节偏移为 32，然后先进行链接，看看现在运行的结果：

```
ghy1190201423@ubuntu:/mnt/hgfs/hitacs/linklab-1190201423$ gcc -m32 -o linkbomb1 main.o phase1.o
ghy1190201423@ubuntu:/mnt/hgfs/hitacs/linklab-1190201423$ ./linkbomb1
nXOCLCLqnl9p3sN3HlfBuFpvgSNDfEcL4wrn26RyZATPOH TZD4CcPLV0oIRPG1jZ3VNuW75VC35xr00chCSGoe3W 5JRAC0GxYLjBNrGI0fyV0NnnvSLV8wLR7DKy12UosydsQHf8LuWEa091dwDLcL 3DL Z
```



```
ghy1190201423@ubuntu:/mnt/hgfs/hitlcs/linklab-1190201423$ objdump -d -r phase2.o
```

```
phase2.o:      file format elf64-x86-64
```

```
Disassembly of section .text:
```

```
0000000000000000 <wEZezGrN>:
 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
      15: R_X86_64_32 .rodata
19:  48 89 c7         mov    %rax,%rdi
1c:  e8 00 00 00 00   callq 21 <wEZezGrN+0x21>
      1d: R_X86_64_PLT32      strcmp-0x4
21:  85 c0           test   %eax,%eax
23:  75 0e           jne    33 <wEZezGrN+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 <wEZezGrN+0x31>
      2d: R_X86_64_PLT32      puts-0x4
31:  eb 01           jmp     34 <wEZezGrN+0x34>
33:  90              nop
34:  c9              leaveq
35:  c3              retq
```

```

000000000000000036 <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
41:  90                   nop
42:  90                   nop
43:  90                   nop
44:  90                   nop
45:  90                   nop
46:  90                   nop
47:  90                   nop
48:  90                   nop
49:  90                   nop
4a:  90                   nop
4b:  90                   nop
4c:  90                   nop
4d:  90                   nop
4e:  90                   nop
4f:  90                   nop
50:  90                   nop
51:  90                   nop
52:  90                   nop
53:  90                   nop
54:  90                   nop
55:  90                   nop
56:  90                   nop
57:  90                   nop
58:  90                   nop
59:  90                   nop
5a:  90                   nop
5b:  90                   nop
5c:  90                   nop
5d:  90                   nop
5e:  90                   nop
5f:  5d                   pop     %rbp
60:  c3                   retq

```

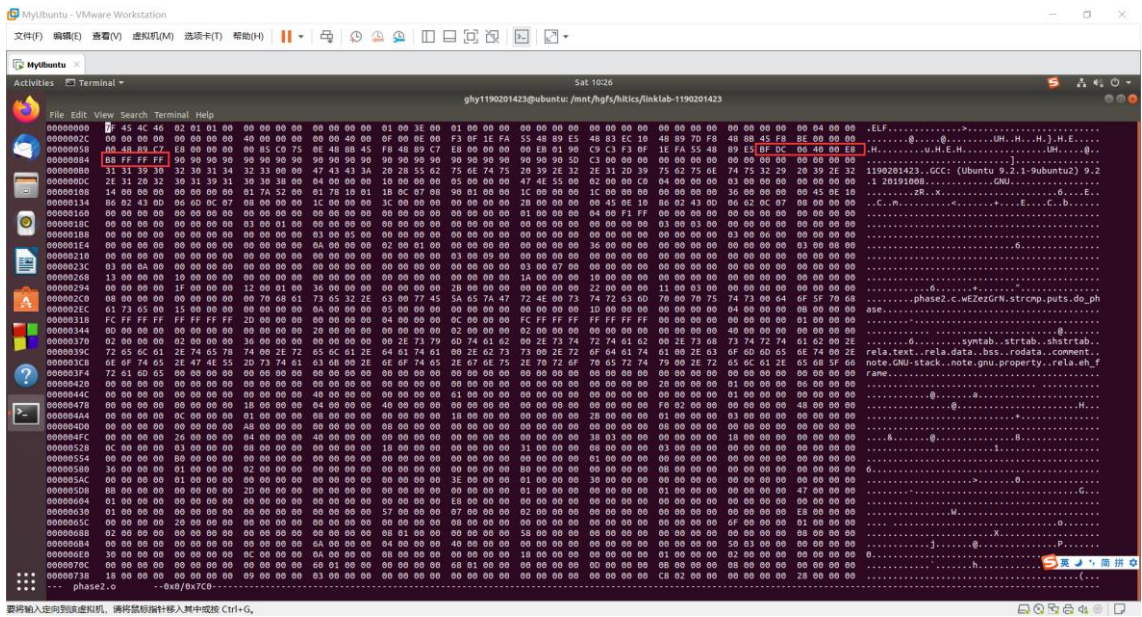
可以看到 do\_phase 有很多的 nop，我们就应该写代码来填充这些 nop，可以使用 edb：

00000000:004005ad	linkbomb2!do_phase	f3	db	0xf3	
00000000:004005ae		0f	db	0x0f	
00000000:004005af		1e	db	0x1e	
00000000:004005b0		fa	c3		
00000000:004005b1		55	pushq	%rbp	
00000000:004005b2		48 89 e5	movq	%rsp, %rbp	
00000000:004005b5		b7 dc 06 40 00	movl	\$0x4006dc, %edi	ASCII "1190201423"
00000000:004005b8		e8 b8 ff ff ff	callq	linkbomb2!wEzGrN	
00000000:004005bf		90	nop		
00000000:004005c0		90	nop		
00000000:004005c1		90	nop		
00000000:004005c2		90	nop		

使用 assembler 修改代码可以看到机器代码，然后使用 hexedit 写入 phase2.o 即可：



## 计算机系统实验报告



重新编译运行既可：

```
ghy1190201423@ubuntu:/mnt/hgfs/hitics/linklab-1190201423$ gcc -no-pie main64.o phase2.o -o linkbomb2
ghy1190201423@ubuntu:/mnt/hgfs/hitics/linklab-1190201423$ ./linkbomb2
1190201423
```

### 3.3 阶段 3 的分析

程序运行结果截图：

```
ghy1190201423@ubuntu:/mnt/hgfs/hitics/linklab-1190201423$ vim phase3_char.c
ghy1190201423@ubuntu:/mnt/hgfs/hitics/linklab-1190201423$ gcc -m32 -c -o phase3_char.o phase3_char.c
ghy1190201423@ubuntu:/mnt/hgfs/hitics/linklab-1190201423$ gcc -m32 -o linkbomb3 main.o phase3.o phase3_char.o
ghy1190201423@ubuntu:/mnt/hgfs/hitics/linklab-1190201423$ ./linkbomb3
1190201423
```

分析与设计的过程：

首先使用 readelf 获取数组名称：

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	HuyzbvTuCY
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

可以看到数组名称为 HuyzbvTuCY，接下来就要获取 cookie 的值，可以使用 gdb 来进行操作：

```
ghy1190201423@ubuntu:/mnt/hgfs/hitcs/linklab-1190201423$ gdb linkbomb3
GNU gdb (Ubuntu 8.1.1-0ubuntu1) 8.1.1
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from linkbomb3...(no debugging symbols found)...done.
(gdb) b do_phase
Breakpoint 1 at 0x631
(gdb) r
Starting program: /mnt/hgfs/hitcs/linklab-1190201423/linkbomb3
Breakpoint 1, 0x56555631 in do_phase ()
```



```
(gdb) disas do_phase
Dump of assembler code for function do_phase:
=> 0x56555631 <+0>:      endbr32
    0x56555635 <+4>:      push    %ebp
    0x56555636 <+5>:      mov     %esp,%ebp
    0x56555638 <+7>:      sub     $0x28,%esp
    0x5655563b <+10>:     mov     %gs:0x14,%eax
    0x56555641 <+16>:     mov     %eax,-0xc(%ebp)
    0x56555644 <+19>:     xor     %eax,%eax
    0x56555646 <+21>:     movl    $0x746c7a68,-0x17(%ebp)
    0x5655564d <+28>:     movl    $0x62716665,-0x13(%ebp)
    0x56555654 <+35>:     movw    $0x6a70,-0xf(%ebp)
    0x5655565a <+41>:     movb    $0x0,-0xd(%ebp)
    0x5655565e <+45>:     movl    $0x0,-0x1c(%ebp)
    0x56555665 <+52>:     jmp     0x5655568f <do_phase+94>
    0x56555667 <+54>:     lea     -0x17(%ebp),%edx
    0x5655566a <+57>:     mov     -0x1c(%ebp),%eax
    0x5655566d <+60>:     add     %edx,%eax
    0x5655566f <+62>:     movzbl  (%eax),%eax
    0x56555672 <+65>:     movzbl  %al,%eax
    0x56555675 <+68>:     movzbl  0x56557040(%eax),%eax
    0x5655567c <+75>:     movsbl  %al,%eax
    0x5655567f <+78>:     sub     $0xc,%esp
    0x56555682 <+81>:     push    %eax
    0x56555683 <+82>:     call   0xf7e43a80 <putchar>
    0x56555688 <+87>:     add     $0x10,%esp
    0x5655568b <+90>:     addl    $0x1,-0x1c(%ebp)
    0x5655568f <+94>:     mov     -0x1c(%ebp),%eax
    0x56555692 <+97>:     cmp     $0x9,%eax
    0x56555695 <+100>:    jbe     0x56555667 <do_phase+54>
    0x56555697 <+102>:    sub     $0xc,%esp
    0x5655569a <+105>:    push    $0xa
    0x5655569c <+107>:    call   0xf7e43a80 <putchar>
    0x565556a1 <+112>:    add     $0x10,%esp
    0x565556a4 <+115>:    nop
    0x565556a5 <+116>:    mov     -0xc(%ebp),%eax
    0x565556a8 <+119>:    xor     %gs:0x14,%eax
    0x565556af <+126>:    je      0x565556b6 <do_phase+133>
    0x565556b1 <+128>:    call   0xf7ee39f0 <__stack_chk_fail>
    0x565556b6 <+133>:    leave
    0x565556b7 <+134>:    ret
End of assembler dump.
```

接下来单步调试到 `lea -0x17(%ebp),%edx` 这一步因为在此`-0x17(%ebp)`的值就是 cookie 的值

```
(gdb) si
0x56555635 in do_phase ()
(gdb) si
0x56555636 in do_phase ()
(gdb) si
0x56555638 in do_phase ()
(gdb) si
0x5655563b in do_phase ()
(gdb) si
0x56555641 in do_phase ()
(gdb) si
0x56555644 in do_phase ()
(gdb) si
0x56555646 in do_phase ()
(gdb) si
0x5655564d in do_phase ()
(gdb) si
0x56555654 in do_phase ()
(gdb) si
0x5655565a in do_phase ()
(gdb) si
0x5655565e in do_phase ()
(gdb) si
0x56555665 in do_phase ()
(gdb) si
0x5655568f in do_phase ()
(gdb) x/s $ebp-0x17
0xffffd101: "hzltefqbpj"
```

得到 cookie 的值为: hzltefqbpj

于是可以对应起来:

h	z	l	t	e	f	q	b	p	j
104	122	108	116	101	102	113	98	112	106
1	1	9	0	2	0	1	4	2	3

于是构造

char

```
HuyzbvTuCY[256]="aaaaaaaaaaaaabbbbbbbbbbbbbbbccccccccccccccddddd
ddddddeeeeeeeeeeeeeeffffffffffffffgg4gg20g1g3g9ggg21hh0hhhhh1hhh
hhaaaaaaaaaaaaabbbbbbbbbbbbbbbccccccccccccccddddddeeeee
eeeeeeeeeffffffffffffffggggggggggggggghhhhhhhhhhhhhhhhhhh"
```

接下来就得到结果:

```
ghy1190201423@ubuntu:/mnt/hgfs/hitics/linklab-1190201423$ vim phase3_char.c
ghy1190201423@ubuntu:/mnt/hgfs/hitics/linklab-1190201423$ gcc -m32 -c -o phase3_char.o phase3_char.c
ghy1190201423@ubuntu:/mnt/hgfs/hitics/linklab-1190201423$ gcc -m32 -o linkbomb3 main.o phase3.o phase3_char.o
ghy1190201423@ubuntu:/mnt/hgfs/hitics/linklab-1190201423$ ./linkbomb3
1190201423
```

### 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

学习了 hexedit 工具的使用;  
学习了将多个.o 文件链接在一起运行;  
学习了 readelf 查看 elf 头文件;  
掌握了链接过程中的符号解析和重定位的过程;  
掌握了可冲定位文件邻接成可执行目标文件的方法;  
通过这次实验,清楚地了解了链接这一章的知识。

### 4.2 请给出对本次实验内容的建议

建议老师仔细讲讲后面的,前面的预习部分自己就可以完成。

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>.