

# 哈尔滨工业大学

# 实验报告

## 实验（五）

题 目 LinkLab

链接

专 业 计算机类

学 号 1180300829

班 级 1803008

学 生 余涛

指 导 教 师 吴锐

实 验 地 点 G709

实 验 日 期 2019.11.23

## 计算机科学与技术学院

## 目 录

<b>第 1 章 实验基本信息</b> .....	<b>3 -</b>
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 -
<b>第 2 章 实验预习</b> .....	<b>5 -</b>
2.1 ELF 文件格式解读 .....	5 -
2.2 程序的内存映像结构 .....	5 -
2.3 程序中符号的位置分析 .....	6 -
2.4 程序运行过程分析 .....	11 -
<b>第 3 章 各阶段的原理与方法</b> .....	<b>12 -</b>
3.1 阶段 1 的分析 .....	12 -
3.2 阶段 2 的分析.....	13 -
3.3 阶段 3 的分析.....	16 -
3.4 阶段 4 的分析.....	19 -
3.5 阶段 5 的分析.....	19 -
<b>第 4 章 总结</b> .....	<b>21 -</b>
4.1 请总结本次实验的收获.....	21 -
4.2 请给出对本次实验内容的建议 .....	21 -
<b>参考文献</b> .....	<b>22 -</b>

## 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：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。

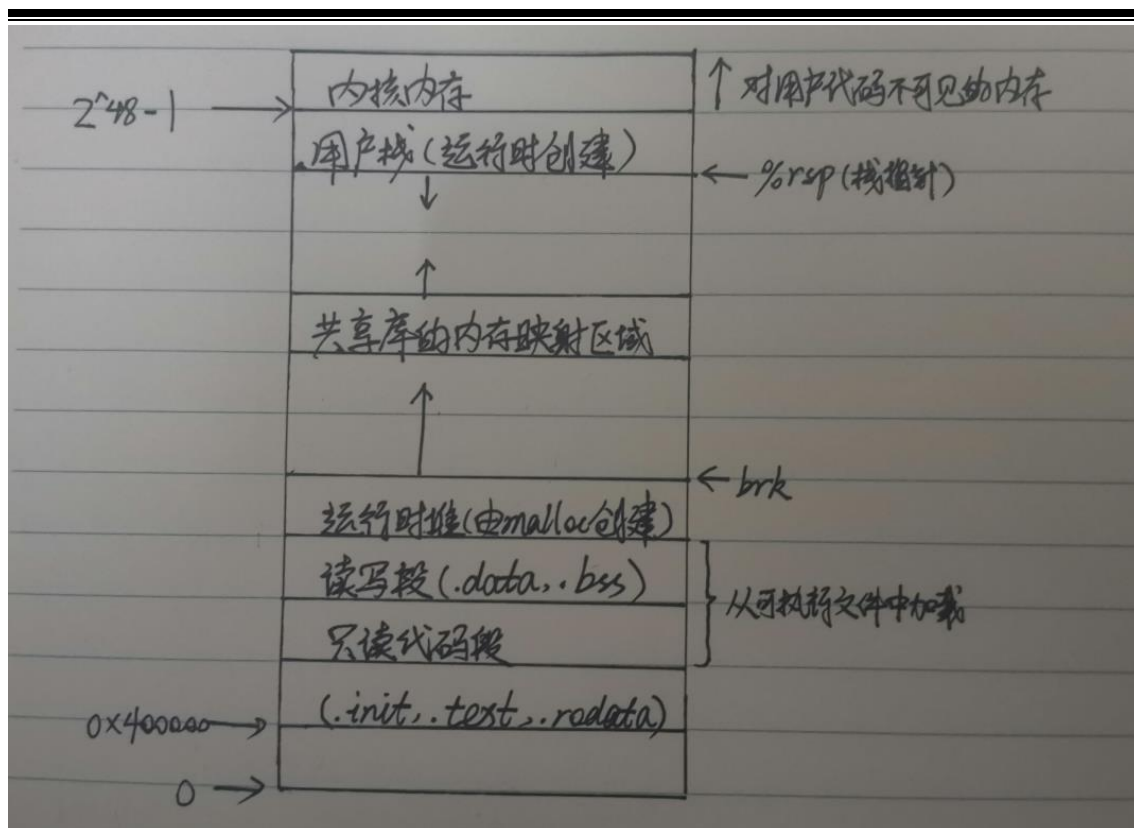
`.line`： 原始 C 源程序的行号和 `.text` 节中机器指令之间的映射

`.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、空间。并按照 Linux 下 X64 内存映像标出其所属各区（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)	exit 0x7f37aa44f3c0 139877056574400 printf 0x7f37aa46a830 139877056686128 malloc 0x7f37aa4a0a40 139877056907840 free 0x7f37aa4a11d0 139877056909776
读写段 (.data,.bss)	show_pointer 0x555e58f72155 93863707877717 useless 0x555e58f72188 93863707877768 main 0x555e58f72193 93863707877779 global 0x555e58f7502c 93863707889708 huge array 0x555e58f75040 93863707889728 big array 0x555e98f75040 93864781631552
运行时堆（由 malloc 创建）	p1 0x7f379a407010 139876787843088 p2 0x555e9b3a8670 93864819590768 p3 0x7f379a3e6010 139876787707920 p4 0x7f375a3e5010 139875713962000

	p5 0x7f36da3e4010 139873566474256
用户栈（运行时创建）	argc 0x7fff3fd1496c 140734264068460 local 0x7fff3fd14970 140734264068464 argv 0x7fff3fd14a98 140734264068760 argv[0] 7fff3fd15269 argv[1] 7fff3fd15274 argv[2] 7fff3fd15277 argv[3] 7fff3fd15282 argv[0] 0x7fff3fd15269 140734264070761 ./linkaddr argv[1] 0x7fff3fd15274 140734264070772 -u argv[2] 0x7fff3fd15277 140734264070775 1180300829 argv[3] 0x7fff3fd15282 140734264070786 余涛 env 0x7fff3fd14ac0 140734264068800 env[0] *env 0x7fff3fd15289 140734264070793 SHELL=/bin/bash env[1] *env 0x7fff3fd15299 140734264070809 SESSION_MANAGER=local/ubuntu:@/tmp/.ICE-unix/2021,unix/ubuntu:/tmp/.ICE-unix/2021 env[2] *env 0x7fff3fd152eb 140734264070891 QT_ACCESSIBILITY=1 env[3] *env 0x7fff3fd152fe 140734264070910 COLORTERM=truecolor env[4] *env 0x7fff3fd15312 140734264070930 XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/etc/xdg env[5] *env 0x7fff3fd1533f 140734264070975 XDG_MENU_PREFIX=gnome- env[6] *env 0x7fff3fd15356 140734264070998 GNOME_DESKTOP_SESSION_ID=this-is-deprecated env[7] *env 0x7fff3fd15382 140734264071042 GTK_IM_MODULE=fcitx env[8] *env 0x7fff3fd15396 140734264071062 LANGUAGE=zh_CN:en_US:en env[9] *env 0x7fff3fd153ae 140734264071086 QT4_IM_MODULE=fcitx

env[10]	*env	0x7fff3fd153c2	140734264071106	LC_ADDRESS=en_US.UTF-8
env[11]	*env	0x7fff3fd153d9	140734264071129	GNOME_SHELL_SESSION_MODE=ubuntu
env[12]	*env	0x7fff3fd153f9	140734264071161	LC_NAME=en_US.UTF-8
env[13]	*env	0x7fff3fd1540d	140734264071181	SSH_AUTH_SOCK=/run/user/1000/keyring/ssh
env[14]	*env	0x7fff3fd15436	140734264071222	XMODIFIERS=@im=fcitx
env[15]	*env	0x7fff3fd1544b	140734264071243	DESKTOP_SESSION=ubuntu
env[16]	*env	0x7fff3fd15462	140734264071266	LC_MONETARY=en_US.UTF-8
env[17]	*env	0x7fff3fd1547a	140734264071290	SSH_AGENT_PID=2125
env[18]	*env	0x7fff3fd1548d	140734264071309	GTK_MODULES=gail:atk-bridge
env[19]	*env	0x7fff3fd154a9	140734264071337	DBUS_STARTER_BUS_TYPE=session
env[20]	*env	0x7fff3fd154c7	140734264071367	XDG_SEAT=seat0
env[21]	*env	0x7fff3fd154d6	140734264071382	PWD=/home/yt1180300829/hitcs/lab5
env[22]	*env	0x7fff3fd154f9	140734264071417	XDG_SESSION_DESKTOP=ubuntu
env[23]	*env	0x7fff3fd15514	140734264071444	LOGNAME=yt1180300829
env[24]	*env	0x7fff3fd15529	140734264071465	XDG_SESSION_TYPE=x11
env[25]	*env	0x7fff3fd1553e	140734264071486	GPG_AGENT_INFO=/run/user/1000/gnupg/S.gpg-agent:0:1
env[26]	*env	0x7fff3fd15572	140734264071538	XAUTHORITY=/run/user/1000/gdm/Xauthority
env[27]	*env	0x7fff3fd1559b	140734264071579	WINDOWPATH=2
env[28]	*env	0x7fff3fd155a8	140734264071592	



	HOME=/home/yt1180300829
	env[29] *env 0x7fff3fd155c0 140734264071616
	USERNAME=yt1180300829
	env[30] *env 0x7fff3fd155d6 140734264071638
	IM_CONFIG_PHASE=2
	env[31] *env 0x7fff3fd155e8 140734264071656
	LC_PAPER=en_US.UTF-8
	env[32] *env 0x7fff3fd155fd 140734264071677
	LANG=zh_CN.UTF-8
	env[33] *env 0x7fff3fd1560e 140734264071694
	LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40
	1;31:*.tlz=01;31:*.txz=01;31:*.tzo=01;31:*.t7z=01;31:*.zip=01;31:*.z=01;31:*.dz=
	*.sar=01;31:*.rar=01;31:*.alz=01;31:*.ace=01;31:*.zoo=01;31:*.cpio=01;31:*.7z=0
	;35:*.xbm=01;35:*.xpm=01;35:*.tif=01;35:*.tiff=01;35:*.png=01;35:*.svg=01;35:*
	35:*.wmv=01;35:*.asf=01;35:*.rm=01;35:*.rmvb=01;35:*.flc=01;35:*.avi=01;35:*
	*.mka=00;36:*.mp3=00;36:*.mpc=00;36:*.ogg=00;36:*.ra=00;36:*.wav=00;36:*.o
	env[34] *env 0x7fff3fd15bf0 140734264073200
	XDG_CURRENT_DESKTOP=ubuntu:GNOME
	env[35] *env 0x7fff3fd15c11 140734264073233
	VTE_VERSION=5601
	env[36] *env 0x7fff3fd15c22 140734264073250
	GNOME_TERMINAL_SCREEN=/org/gnome/Terminal/screen/8a164301_63dd_47
	env[37] *env 0x7fff3fd15c78 140734264073336
	INVOCATION_ID=dc47daa538f044cdbbf101ece2727fd7
	env[38] *env 0x7fff3fd15ca7 140734264073383
	MANAGERPID=1990
	env[39] *env 0x7fff3fd15cb7 140734264073399
	CLUTTER_IM_MODULE=xim
	env[40] *env 0x7fff3fd15ccd 140734264073421
	LESSCLOSE=/usr/bin/lesspipe %s %s
	env[41] *env 0x7fff3fd15cef 140734264073455
	XDG_SESSION_CLASS=user
	env[42] *env 0x7fff3fd15d06 140734264073478
	TERM=xterm-256color
	env[43] *env 0x7fff3fd15d1a 140734264073498
	LC_IDENTIFICATION=en_US.UTF-8
	env[44] *env 0x7fff3fd15d38 140734264073528

	<pre> LESSOPEN=  /usr/bin/lesspipe %s env[45] *env    0x7fff3fd15d58  140734264073560 USER=yt1180300829 env[46] *env    0x7fff3fd15d6a  140734264073578 GNOME_TERMINAL_SERVICE=:1.101 env[47] *env    0x7fff3fd15d88  140734264073608 DISPLAY=:0 env[48] *env    0x7fff3fd15d93  140734264073619 SHLVL=1 env[49] *env    0x7fff3fd15d9b  140734264073627 LC_TELEPHONE=en_US.UTF-8 env[50] *env    0x7fff3fd15db4  140734264073652 QT_IM_MODULE=fcitx env[51] *env    0x7fff3fd15dc7  140734264073671 LC_MEASUREMENT=en_US.UTF-8 env[52] *env    0x7fff3fd15de2  140734264073698 XDG_VTNR=2 env[53] *env    0x7fff3fd15ded  140734264073709 XDG_SESSION_ID=2 env[54] *env    0x7fff3fd15dfe  140734264073726 DBUS_STARTER_ADDRESS=unix:path=/run/user/1000/bus,guid=7e758e95dfd60 env[55] *env    0x7fff3fd15e56  140734264073814 XDG_RUNTIME_DIR=/run/user/1000 env[56] *env    0x7fff3fd15e75  140734264073845 LC_TIME=en_US.UTF-8 env[57] *env    0x7fff3fd15e89  140734264073865 JOURNAL_STREAM=9:50936 env[58] *env    0x7fff3fd15ea0  140734264073888 XDG_DATA_DIRS=/usr/share/ubuntu:/usr/local/share:/usr/share:/var/lib/snapd/desktop env[59] *env    0x7fff3fd15ef3  140734264073971 PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games env[60] *env    0x7fff3fd15f5b  140734264074075 GDMSESSION=ubuntu env[61] *env    0x7fff3fd15f6d  140734264074093 DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus,guid=7e758e95dfd60 env[62] *env    0x7fff3fd15fc9  140734264074185 LC_NUMERIC=en_US.UTF-8 </pre>
--	--

	env[63] *env 0x7fff3fd15fe0 140734264074208 _=./linkaddr
--	---

## 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:

<\_init>

<.plt>

<puts@plt>

<\_\_stack\_chk\_fail@plt>

<\_\_printf\_chk@plt>

<free@plt>

<malloc@plt>

<\_\_cxa\_finalize@plt>

<\_start>

<deregister\_tm\_clones>

<register\_tm\_clones>

<\_\_do\_global\_ctors\_aux>

<frame\_dummy>

<useless>

<show\_pointer>

main 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，phases.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：

```
yt1180300829@ubuntu:~/hitics/lab5/linklab-1180300829$ ./linkbomb1
1180300829
```

分析与设计的过程：

1.使用 `readelf -a phase1.o` 命令查看 elf 文件的内容，可以发现字符串输出的起始地址在 .data 节中偏移量为 32 处

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	0002a4	000020	08	I	13	2	4
[ 4]	.data	PROGBITS	00000000	000080	000042	00	WA	0	0	32
[ 5]	.bss	NOBITS	00000000	0000c2	000000	00	WA	0	0	1
[ 6]	.data.rel.local	PROGBITS	00000000	0000c4	000004	00	WA	0	0	4
[ 7]	.rel.data.rel.loc	REL	00000000	0002c4	000008	08	I	13	6	4
[ 8]	.text.__x86.get_p	PROGBITS	00000000	0000c8	000004	00	AXG	0	0	1
[ 9]	.comment	PROGBITS	00000000	0000cc	000025	01	MS	0	0	1
[10]	.note.GNU-stack	PROGBITS	00000000	0000f1	000000	00		0	0	1
[11]	.eh_frame	PROGBITS	00000000	0000f4	000050	00	A	0	0	4
[12]	.rel.eh_frame	REL	00000000	0002cc	000010	08	I	13	11	4
[13]	.symtab	SYMTAB	00000000	000144	000110	10		14	12	4
[14]	.strtab	STRTAB	00000000	000254	00004d	00		0	0	1
[15]	.shstrtab	STRTAB	00000000	0002dc	00008e	00		0	0	1

2.使用命令 `gcc -m32 -o linkbomb1 main.o phase1.o` 将 main.o 和 phase.o 链接成 linkbomb1.o，然后运行 linkbomb1 程序，查看本来应该输出的字符串为 LO9Mr5SQ6U8UjQxZXtm9lidRLC3uUUVXv

```
yt1180300829@ubuntu:~/hitics/lab5/linklab-1180300829$ gcc -m32 -o linkbomb1 main.o phase1.o
yt1180300829@ubuntu:~/hitics/lab5/linklab-1180300829$ ./linkbomb1
LO9Mr5SQ6U8UjQxZXtm9l idRLC3uUUVXv
```

3.使用 hexedit 工具进入 phase1.o，我们的目的是用我们的学号 1180300829

替换掉本应该输出的.data 节里面的字符串，我们通过观察最右边的字符找到输出的字符串，用学号 1180300829 对应的 ascii 码 31 31 38 30 33 30 30 38 32 39 替换它，多余的位用 00 作为字符串的结束，然后再进行链接输出便可得到 1180300829

```

yt1180300829@ubuntu: ~/hitics/lab5/linklab-1180300829
00000000  7F 45 4C 46 01 01 01 00 00 00 00 00 01 00 03 00 .ELF.....
00000014  01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....l.....
00000028  34 00 00 00 00 00 28 00 10 00 0F 00 01 00 00 00 4.....(.....
0000003C  55 89 E5 53 83 EC 04 E8 FC FF FF FF 05 01 00 00 00 8D 90 1F U..S.....
00000050  00 00 00 83 EC 0C 52 89 C3 E8 FC FF FF FF 83 C4 10 90 8B 5D .....R.....]
00000064  FC C9 C3 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000078  00 00 00 00 00 00 00 00 41 64 5A 58 52 67 45 64 78 4B 6F 73 .....AdZXRgEdxKos
0000008C  53 34 69 47 62 49 48 6A 20 46 52 48 4E 53 68 66 44 71 76 31 $4iGbIHj FRHNShfDqv1
000000A0  31 38 30 33 30 30 38 32 39 00 00 00 00 00 00 00 00 180300829.....
000000B4  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000000C8  8B 04 24 C3 00 47 43 43 3A 20 28 55 62 75 6E 74 75 20 37 2E ..$..GCC: (Ubuntu 7.
000000DC  33 2E 30 2D 31 36 75 62 75 6E 74 75 33 29 20 37 2E 33 2E 30 3.0-16ubuntu3) 7.3.0
000000F0  00 00 00 00 14 00 00 00 00 00 00 00 01 7A 52 00 01 7C 08 01 .....zR..|..
00000104  1B 0C 04 04 88 01 00 00 20 00 00 00 1C 00 00 00 00 00 00 00 .....
00000118  2B 00 00 00 00 41 0E 08 85 02 42 0D 05 44 83 03 63 C5 C3 0C +...A...B..D..C...
0000012C  04 04 00 00 10 00 00 00 40 00 00 00 00 00 00 00 04 00 00 00 .....@.....
00000140  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000154  01 00 00 00 00 00 00 00 00 00 00 00 04 00 F1 FF 00 00 00 00 .....
00000168  00 00 00 00 00 00 00 00 03 00 02 00 00 00 00 00 00 00 00 00 .....
0000017C  00 00 00 00 03 00 04 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000190  03 00 05 00 0A 00 00 00 00 00 00 00 42 00 00 00 01 00 04 00 .....B.....
000001A4  00 00 00 00 00 00 00 00 00 00 00 00 03 00 06 00 00 00 00 00 .....
000001B8  00 00 00 00 00 00 00 00 03 00 08 00 00 00 00 00 00 00 00 00 .....
000001CC  00 00 00 00 03 00 0A 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000001E0  03 00 0B 00 00 00 00 00 00 00 00 00 00 00 00 00 03 00 09 00 .....
000001F4  00 00 00 00 00 00 00 00 00 00 00 00 03 00 01 00 13 00 00 00 .....

yt1180300829@ubuntu:~/hitics/lab5/linklab-1180300829$ ./linkbomb1
1180300829

```

## 3.2 阶段 2 的分析

程序运行结果截图：

```

yt1180300829@ubuntu:~/hitics/lab5/linklab-1180300829$ gcc -m32 -o linkbomb2 main.o phase2.o
yt1180300829@ubuntu:~/hitics/lab5/linklab-1180300829$ ./linkbomb2
1180300829

```

分析与设计的过程：

1. 将文件链接，`gcc -m32 -o linkbomb2 main.o phase.o` 得到链接后的文件 `linkbomb.o`，对 `linkbomb2` 运用 `objdump` 进行反汇编，然后查看 `YgpZFthB` 函数和 `do_phase` 函数的反汇编代码。

根据 ppt 我们知道，执行 `lea -0x1f58(%ebx),%eax` 后里面存的是我们的

学号 1180300829, YgpZFthB 函数执行 strcmp 之前向栈里压入了两个参数, 一个是 MYID, 一个是函数传入的参数。本题是要在 do\_phase 函数的 nop 部分填写代码执行压栈, 并且跳转到 YgpZFthB 函数, 所以需要找到跳转到 YgpZFthB 函数的偏移量。

```
static void OUTPUT_FUNC_NAME( const char *id ) // 该函数名对每名同学均不同
{
    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\t..." );
}
```

2. 观察反汇编代码可以分析: call 10b0 <\_\_x86.get\_pc\_thunk.bx> 和 add \$0x2dc7,%ebx 指令使得%ebx 指向了 \_GLOBAL\_OFFSET\_TABLE, call 11fd <\_\_x86.get\_pc\_thunk.ax> 和 add \$0x2d8a,%eax 指令使得 %eax 指向 \_GLOBAL\_OFFSET\_TABLE。

```
00001201 <YgpZFthB>:
1201: 55          push    %ebp
1202: 89 e5       mov     %esp,%ebp
1204: 53          push    %ebx
1205: 83 ec 04    sub     $0x4,%esp
1208: e8 a3 fe ff ff call    10b0 <__x86.get_pc_thunk.bx>
120d: 81 c3 c7 2d 00 00 add     $0x2dc7,%ebx
1213: 83 ec 08    sub     $0x8,%esp
1216: 8d 83 a8 e0 ff ff lea     -0x1f58(%ebx),%eax
121c: 50          push    %eax
121d: ff 75 08    pushl   0x8(%ebp)
1220: e8 0b fe ff ff call    1030 <strcmp@plt>
1225: 83 c4 10    add     $0x10,%esp
1228: 85 c0       test    %eax,%eax
122a: 75 10       jne     123c <YgpZFthB+0x3b>
122c: 83 ec 0c    sub     $0xc,%esp
122f: ff 75 08    pushl   0x8(%ebp)
1232: e8 09 fe ff ff call    1040 <puts@plt>
1237: 83 c4 10    add     $0x10,%esp
123a: eb 01       jmp     123d <YgpZFthB+0x3c>
123c: 90          nop
123d: 8b 5d fc    mov     -0x4(%ebp),%ebx
1240: c9          leave
1241: c3          ret

00001242 <do_phase>:
1242: 55          push    %ebp
1243: 89 e5       mov     %esp,%ebp
1245: e8 b3 ff ff ff call    11fd <__x86.get_pc_thunk.ax>
124a: 05 8a 2d 00 00 add     $0x2d8a,%eax
124f: 90          nop
1250: 90          nop
1251: 90          nop
1252: 90          nop
1253: 90          nop
1254: 90          nop
1255: 90          nop
1256: 90          nop
```

3.继续分析可以得到 `lea -0x1f58(%ebx),%eax` 语句是为了将重定位之后的 `%eax` 指向 `.dodata` 段,所以在 `do_phase` 里面添加的代码中,也需要执行该操作,让 `do_phase` 里面的 `%eax` 也指向 `.dodata` 段。

```

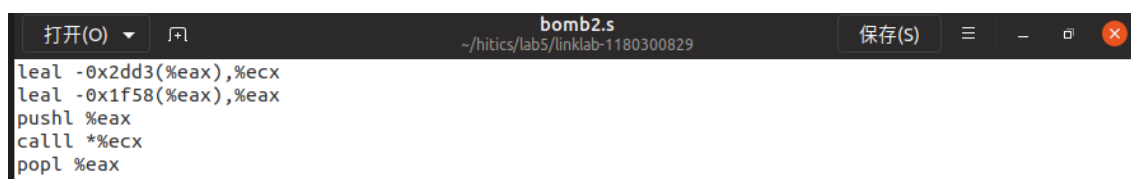
00001201 <YgpZFthB>:
1201: 55                push    %ebp
1202: 89 e5             mov     %esp,%ebp
1204: 53                push    %ebx
1205: 83 ec 04          sub     $0x4,%esp
1208: e8 a3 fe ff ff    call   10b0 <_x86.get_pc_thunk.bx>
120d: 81 c3 c7 2d 00 00 add     $0x2dc7,%ebx
1213: 83 ec 08          sub     $0x8,%esp
1216: 8d 83 a8 e0 ff ff lea     -0x1f58(%ebx),%eax
121c: 50                push    %eax
121d: ff 75 08          pushl   0x8(%ebp)
1220: e8 0b fe ff ff    call   1030 <strcmp@plt>
1225: 83 c4 10          add     $0x10,%esp
1228: 85 c0             test    %eax,%eax
122a: 75 10             jne     123c <YgpZFthB+0x3b>
122c: 83 ec 0c          sub     $0xc,%esp
122f: ff 75 08          pushl   0x8(%ebp)
1232: e8 09 fe ff ff    call   1040 <puts@plt>
1237: 83 c4 10          add     $0x10,%esp
123a: eb 01             jmp     123d <YgpZFthB+0x3c>
123c: 90                nop
123d: 8b 5d fc          mov     -0x4(%ebp),%ebx
1240: c9                leave   %eax
1241: c3                ret

00001242 <do_phase>:
1242: 55                push    %ebp
1243: 89 e5             mov     %esp,%ebp
1245: e8 b3 ff ff ff    call   11fd <_x86.get_pc_thunk.ax>
124a: 05 8a 2d 00 00    add     $0x2d8a,%eax
124f: 90                nop
1250: 90                nop
1251: 90                nop
1252: 90                nop
1253: 90                nop
1254: 90                nop
1255: 90                nop
1256: 90                nop

```

4.寻找偏移量,用 `do_phase` 里面的 `%eax` 的 `0x2d8a+124a` 得到 ELF 头的地址 `3fd4`,然后用 `3fd4` 减去 `YgpZFthB` 函数首地址 `1201` 为偏移量 `2dd3`,编写 `leal -0x2dd3(%eax),%ecx` 即可完成函数跳转。

5.于是可以得到如下的汇编代码,对其编译和反汇编得到机器码。



```

bomb2.s
~/hitics/lab5/unklab-1180300829
保存(S)

leal -0x2dd3(%eax),%ecx
leal -0x1f58(%eax),%eax
pushl %eax
calll %%ecx
popl %eax

```

6.使用 `hexedit` 进入 `phase2.o`,用得到的反汇编机器码插入 `90` 开头的第一个 `nop` 即可。

```

yt1180300829@ubuntu: ~/hitics/lab5/linklab-1180300829
yt1180300829@ubuntu:~/hitics/lab5/linklab-1180300829$ gcc -m32 -c bomb2.s
yt1180300829@ubuntu:~/hitics/lab5/linklab-1180300829$ objdump -d bomb2.o

bomb2.o:          文件格式 elf32-i386

Disassembly of section .text:

00000000 <.text>:
   0:  8d 88 2d d2 ff ff      lea    -0x2dd3(%eax),%ecx
   6:  8d 80 a8 e0 ff ff      lea    -0x1f58(%eax),%eax
   c:  50                     push   %eax
   d:  ff d1                 call   *%ecx
   f:  58                     pop    %eax

00001242 <do_phase>:
 1242:      55                     push   %ebp
 1243:      89 e5                 mov    %esp,%ebp
 1245:      e8 b3 ff ff ff      call   11fd <__x86.get_pc_thunk.ax>
 124a:      05 8a 2d 00 00        add    $0x2d8a,%eax
 124f:      90                     nop
 1250:      90                     nop
 1251:      90                     nop
 1252:      90                     nop
 1253:      90                     nop
 1254:      90                     nop
 1255:      90                     nop
 1256:      90                     nop
 1257:      90                     nop
 1258:      90                     nop
 1259:      90                     nop
 125a:      90                     nop
 125b:      90                     nop
 125c:      90                     nop
 125d:      90                     nop
 125e:      90                     nop

00000000  7F 45 4C 46 01 01 01 00 00 00 00 00 00 00 01 00 03 00 .ELF.....
00000014  01 00 00 00 00 00 00 00 00 00 00 00 58 04 00 00 00 00 00 00 00 00 .....X.....
00000028  34 00 00 00 00 00 28 00 13 00 12 00 01 00 00 00 0A 00 00 00 4.....(.....
0000003C  01 00 00 00 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 FF 75 08 E8 .....P.u...
00000064  FC FF FF FF 83 C4 10 85 C0 75 10 83 EC 0C FF 75 08 E8 FC FF .....U.....
00000078  FF FF 83 C4 10 EB 01 90 8B 5D FC C9 C3 55 89 E5 E8 FC FF FF .....]...U.....
0000008C  FF 05 01 00 00 00 8D 88 2D D2 FF FF 8D 80 A8 E0 FF FF 50 FF .....-.....P.
000000A0  D1 58 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 5D .X.....]
000000B4  C3 31 31 38 30 33 30 30 38 32 39 00 00 00 00 00 8B 04 24 C3 .1180300829.....$.
000000C8  8B 1C 24 C3 00 47 43 43 3A 20 28 55 62 75 6E 74 75 20 37 2E ..$.GCC: (Ubuntu 7.
000000DC  33 2F 30 2D 31 36 75 62 75 6F 74 75 33 29 20 37 2F 33 2F 30 3 0-16ubuntu3) 7 3 0

```

### 3.3 阶段 3 的分析

程序运行结果截图：



```

yt1180300829@ubuntu:~/hitics/lab5/linklab-1180300829$ gcc -m32 -c -o phase3_patch.o phase3_patch.c
yt1180300829@ubuntu:~/hitics/lab5/linklab-1180300829$ gcc -m32 -o linkbomb3 main.o phase3.o phase3_patch.o
yt1180300829@ubuntu:~/hitics/lab5/linklab-1180300829$ ./linkbomb3
1180300829

```

分析与设计的过程:

1.反汇编查看 do\_phase 的指令, 需要获取 cookie 的字符串, 现将 main.o 和 phase3.o 链接 gcc -m32 -o linkbomb3 main.o phase3.o, 然后对 linkbomb3 进行 gdb 调试。在 do\_phase 设置断点, 然后 run 到 do\_phase, 然后 disas do\_phase, 记下-0x17(%ebp)所在行的地址为 0x5655624f, 然后 si 单步执行, 知道执行到了 0x5655624f 位置, 然后查看(%ebp)-0x17 里面的内容

```

Dump of assembler code for function do_phase:
0x56556211 <+0>:    push    %ebp
0x56556212 <+1>:    mov     %esp,%ebp
0x56556214 <+3>:    push    %ebx
=> 0x56556215 <+4>:    sub     $0x24,%esp
0x56556218 <+7>:    call   0x565560c0 <__x86.get_pc_thunk.bx>
0x5655621d <+12>:   add     $0x2db3,%ebx
0x56556223 <+18>:   mov     %gs:0x14,%eax
0x56556229 <+24>:   mov     %eax,-0xc(%ebp)
0x5655622c <+27>:   xor     %eax,%eax
0x5655622e <+29>:   movl    $0x7a687975,-0x17(%ebp)
0x56556235 <+36>:   movl    $0x726b7673,-0x13(%ebp)
0x5655623c <+43>:   movw    $0x6f6e,-0xf(%ebp)
0x56556242 <+49>:   movb    $0x0,-0xd(%ebp)
0x56556246 <+53>:   movl    $0x0,-0x1c(%ebp)
0x5655624d <+60>:   jmp     0x5655627a <do_phase+105>
0x5655624f <+62>:   lea     -0x17(%ebp),%edx
0x56556252 <+65>:   mov     -0x1c(%ebp),%eax
0x56556255 <+68>:   add     %edx,%eax
0x56556257 <+70>:   movzbl (%eax),%eax
0x5655625a <+73>:   movzbl %al,%eax
0x5655625d <+76>:   lea     0x70(%ebx),%edx
0x56556263 <+82>:   movzbl (%edx,%eax,1),%eax

```

```

-Type <RET> for more, q to quit, c to
quit
(gdb) si
x56556218 in do_phase ()
(gdb)
x565560c0 in __x86.get_pc_thunk.bx ()
(gdb)
x565560c3 in __x86.get_pc_thunk.bx ()
(gdb)
x5655621d in do_phase ()
(gdb)
x56556223 in do_phase ()
(gdb)
x56556229 in do_phase ()
(gdb)
x5655622c in do_phase ()
(gdb)
x5655622e in do_phase ()
(gdb)
x56556235 in do_phase ()
(gdb)
x5655623c in do_phase ()
(gdb)
x56556242 in do_phase ()

```

内容如下: uyhzsvkrno

```

(gdb) x/s $ebp-0x17
0xffffcf51: "uyhzsvkrno"

```

2.查看符号表,可以得到映射数组的变量名为 nwxdCdTff, 长度为 256 个字节

```

yt1180300829@ubuntu:~/hitics/lab5/linklab-1180300829$ readelf -s phase3.o
Symbol table '.syntab' 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	nwxdCdTff
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

3.根据 ppt 提供的 phase3.c 的程序框架，可以分析出 PHASE\_COOKIE 为 uyhzsvkrno，PHASE3\_CODEBOOK 为 nwxdCdTFff

### ■ phase3.c程序框架

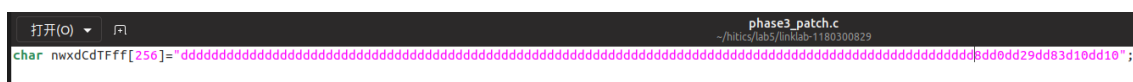
```
char PHASE3_CODEBOOK[256];
void do_phase(){
    const char char cookie[] = PHASE3_COOKIE;
    for( int i=0; i<sizeof(cookie)-1; i++ )
        printf( "%c", PHASE3_CODEBOOK[ (unsigned char)(cookie[i]) ] );
    printf( "\n" );
}
```

4, 只需要把 PHASE3\_CODEBOOK[256]数组在 COOKIE 对应的位置上改成自己的学号即可，有如下对应关系

117 121 104 122 115 118 107 114 110 111

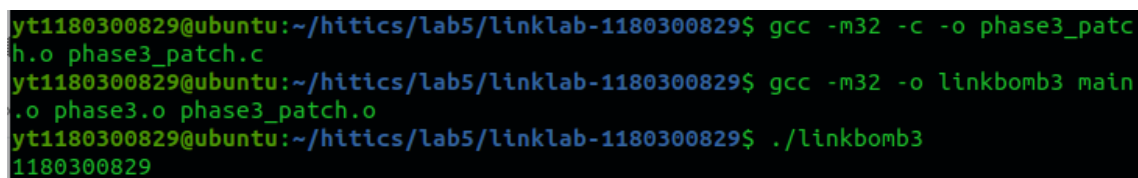
1 1 8 0 3 0 0 8 2 9

8dd0dd29dd83d10dd10



```
char nwxdCdTFff[256] = "dddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddd8dd0dd29dd83d10dd10";
```

然后将三个文件链接在一起，运行 linkbomb3:



```
yt1180300829@ubuntu:~/hitics/lab5/linklab-1180300829$ gcc -m32 -c -o phase3_patch.o phase3_patch.c
yt1180300829@ubuntu:~/hitics/lab5/linklab-1180300829$ gcc -m32 -o linkbomb3 main.o phase3.o phase3_patch.o
yt1180300829@ubuntu:~/hitics/lab5/linklab-1180300829$ ./linkbomb3
1180300829
```

## 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

学会了 hexedit 工具的使用；  
学会了将多个.o 文件链接在一起运行；  
学会了 readelf 查看 elf 头文件。

### 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>.