

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 1190201215

班 级 1903007

学 生 冯开来

指 导 教 师 吴锐

实 验 地 点 G709

实 验 日 期 2021.5.17

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	- 4 -
2.1 ELF 文件格式解读	- 4 -
2.2 程序的内存映像结构	- 4 -
2.3 程序中符号的位置分析	- 5 -
2.4 程序运行过程分析	- 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	- 15 -
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、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，其条目是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	各符号地址、空间
只读代码段 (.init, .text, .rodata)	exit 0x7f2c2efb4bc0 139827743509440 printf 0x7f2c2efcfe10 139827743620624 malloc 0x7f2c2f008260 139827743851104 free 0x7f2c2f008850 139827743852624
运行时堆 (由 malloc 创建)	p1 0x7f2c1ef6a010 139827474767888 p2 0x5557d34936b0 93835695306416 p3 0x7f2c1ef49010 139827474632720 p4 0x7f2bdef48010 139826400886800 p5 0x7f2b5ef47010 139824253399056
用户栈 (运行时创建)	argc 0x7ffed2b053cc 140732433191884 argv 0x7ffed2b05508 140732433192200 argv[0] 7ffed2b0636f argv[1] 7ffed2b06377 argv[2] 7ffed2b0637a argv[3] 7ffed2b06385 argv[0] 0x7ffed2b0636f 140732433195887 ./a.out argv[1] 0x7ffed2b06377 140732433195895 -u argv[2] 0x7ffed2b0637a 140732433195898 1190201215 argv[3] 0x7ffed2b06385 140732433195909 冯开来 env 0x7ffed2b05530 140732433192240 env[0] *env 0x7ffed2b0638f 140732433195919 SHELL=/bin/bash env[1] *env 0x7ffed2b0639f 140732433195935 SESSION_MANAGER=local/ubuntu:@/tmp/.ICE-unix/1886 ,unix/ubuntu:/tmp/.ICE-unix/1886 env[2] *env 0x7ffed2b063f1 140732433196017 QT_ACCESSIBILITY=1

	env[3] *env 0x7ffed2b06404 140732433196036 COLORTERM=truecolor
	env[4] *env 0x7ffed2b06418 140732433196056 XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/etc/xdg
	env[5] *env 0x7ffed2b06445 140732433196101 XDG_MENU_PREFIX=gnome-
	env[6] *env 0x7ffed2b0645c 140732433196124 GNOME_DESKTOP_SESSION_ID=this-is-deprecated
	env[7] *env 0x7ffed2b06488 140732433196168 LANGUAGE=zh_CN:en_US:en
	env[8] *env 0x7ffed2b064a0 140732433196192 LC_ADDRESS=zh_CN.UTF-8
	env[9] *env 0x7ffed2b064b7 140732433196215 GNOME_SHELL_SESSION_MODE=ubuntu
	env[10] *env 0x7ffed2b064d7 140732433196247 LC_NAME=zh_CN.UTF-8
	env[11] *env 0x7ffed2b064eb 140732433196267 SSH_AUTH_SOCK=/run/user/1000/keyring/ssh
	env[12] *env 0x7ffed2b06514 140732433196308 XMODIFIERS=@im=ibus
	env[13] *env 0x7ffed2b06528 140732433196328 DESKTOP_SESSION=ubuntu
	env[14] *env 0x7ffed2b0653f 140732433196351 LC_MONETARY=zh_CN.UTF-8
	env[15] *env 0x7ffed2b06557 140732433196375 SSH_AGENT_PID=1851
	env[16] *env 0x7ffed2b0656a 140732433196394 GTK_MODULES=gail:atk-bridge
	env[17] *env 0x7ffed2b06586 140732433196422 PWD=/home/fkl1190201215
	env[18] *env 0x7ffed2b0659e 140732433196446 LOGNAME=fkl1190201215
	env[19] *env 0x7ffed2b065b4 140732433196468 XDG_SESSION_DESKTOP=ubuntu
	env[20] *env 0x7ffed2b065cf 140732433196495 XDG_SESSION_TYPE=x11
	env[21] *env 0x7ffed2b065e4 140732433196516

	GPG_AGENT_INFO=/run/user/1000/gnupg/S.gpg-agent:0:1 env[22] *env 0x7ffed2b06618 140732433196568 XAUTHORITY=/run/user/1000/gdm/Xauthority env[23] *env 0x7ffed2b06641 140732433196609 WINDOWPATH=2 env[24] *env 0x7ffed2b0664e 140732433196622 HOME=/home/fkl1190201215 env[25] *env 0x7ffed2b06667 140732433196647 USERNAME=fkl1190201215 env[26] *env 0x7ffed2b0667e 140732433196670 IM_CONFIG_PHASE=1 env[27] *env 0x7ffed2b06690 140732433196688 LC_PAPER=zh_CN.UTF-8 env[28] *env 0x7ffed2b066a5 140732433196709 LANG=zh_CN.UTF-8 env[29] *env 0x7ffed2b066b6 140732433196726 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;
--	---

	<pre> 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[30] *env 0x7ffc8e874c98 140722699717784 XDG_CURRENT_DESKTOP=ubuntu:GNOME env[31] *env 0x7ffc8e874cb9 140722699717817 VTE_VERSION=6003 env[32] *env 0x7ffc8e874cca 140722699717834 GNOME_TERMINAL_SCREEN=/org/gnome/Terminal /screen/19808354_3169_424d_acb1_247349d408bc env[33] *env 0x7ffc8e874d20 140722699717920 INVOCATION_ID=95b30905a0c44ceea2c449c377971e9f env[34] *env 0x7ffc8e874d4f 140722699717967 MANAGERPID=1680 env[35] *env 0x7ffc8e874d5f 140722699717983 LESSCLOSE=/usr/bin/lesspipe %s %s env[36] *env 0x7ffc8e874d81 140722699718017 XDG_SESSION_CLASS=user env[37] *env 0x7ffc8e874d98 140722699718040 TERM=xterm-256color env[38] *env 0x7ffc8e874dac 140722699718060 LC_IDENTIFICATION=zh_CN.UTF-8 env[39] *env 0x7ffc8e874dca 140722699718090 LESSOPEN= /usr/bin/lesspipe %s env[40] *env 0x7ffc8e874dea 140722699718122 USER=fk11190201215 env[41] *env 0x7ffc8e874dfd 140722699718141 GNOME_TERMINAL_SERVICE=:1.86 env[42] *env 0x7ffc8e874e1a 140722699718170 DISPLAY=:0 env[43] *env 0x7ffc8e874e25 140722699718181 </pre>
--	--

	SHLVL=1 env[44] *env 0x7ffc8e874e2d 140722699718189 LC_TELEPHONE=zh_CN.UTF-8 env[45] *env 0x7ffc8e874e46 140722699718214 QT_IM_MODULE=ibus env[46] *env 0x7ffc8e874e58 140722699718232 LC_MEASUREMENT=zh_CN.UTF-8 env[47] *env 0x7ffc8e874e73 140722699718259 PAPERSIZE=a4 env[48] *env 0x7ffc8e874e80 140722699718272 XDG_RUNTIME_DIR=/run/user/1000 env[49] *env 0x7ffc8e874e9f 140722699718303 LC_TIME=zh_CN.UTF-8 env[50] *env 0x7ffc8e874eb3 140722699718323 JOURNAL_STREAM=8:51047 env[51] *env 0x7ffc8e874eca 140722699718346 XDG_DATA_DIRS=/usr/share/ubuntu:/usr/local/share/ :/usr/share:/var/lib/snapd/desktop env[52] *env 0x7ffc8e874f1f 140722699718431 PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin: /sbin:/bin:/usr/games:/usr/local/games:/snap/bin env[53] *env 0x7ffc8e874f87 140722699718535 GDMSESSION=ubuntu env[54] *env 0x7ffc8e874f99 140722699718553 DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user /1000/bus env[55] *env 0x7ffc8e874fcf 140722699718607 LC_NUMERIC=zh_CN.UTF-8 env[56] *env 0x7ffc8e874fe6 140722699718630 _=./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 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，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：

```

fkl1190201215@ubuntu:~$ gcc -m32 -o linkbomb1 main.o phase1.o
fkl1190201215@ubuntu:~$ ./linkbomb1
1190201215

```

分析与设计的过程：

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

```

节头:
[Nr] Name                Type              Addr             Off             Size            ES Flg Lk Inf Al
[ 0]                      NULL              00000000         000000          000000          00  0  0  0  0
[ 1] .text                  PROGBITS          00000000         000034          00001e          00  AX  0  0  1
[ 2] .rel.text              REL               00000000         0002ac          000010          08  I 11  1  4
[ 3] .data                  PROGBITS          00000000         000060          0000d4          00  WA  0  0 32
[ 4] .rel.data              REL               00000000         0002bc          000008          08  I 11  3  4
[ 5] .bss                   NOBITS            00000000         000134          000000          00  WA  0  0  1
[ 6] .comment                PROGBITS          00000000         000134          00002d          01  MS  0  0  1
[ 7] .note.gnu-stack         PROGBITS          00000000         000161          000000          00  0  0  0  1
[ 8] .note.gnu.property      NOTE              00000000         000164          00001c          00  A  0  0  4
[ 9] .eh_frame               PROGBITS          00000000         000180          000038          00  A  0  0  4
[10] .rel.eh_frame            REL               00000000         0002c4          000008          08  I 11  9  4
[11] .symtab                  SYMTAB            00000000         0001b8          0000d0          10  12 10  4
[12] .strtab                  STRTAB            00000000         000288          000021          00  0  0  0  1
[13] .shstrtab                STRTAB            00000000         0002cc          00000e          00  0  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),
C (compressed), x (unknown), o (OS specific), E (exclude)

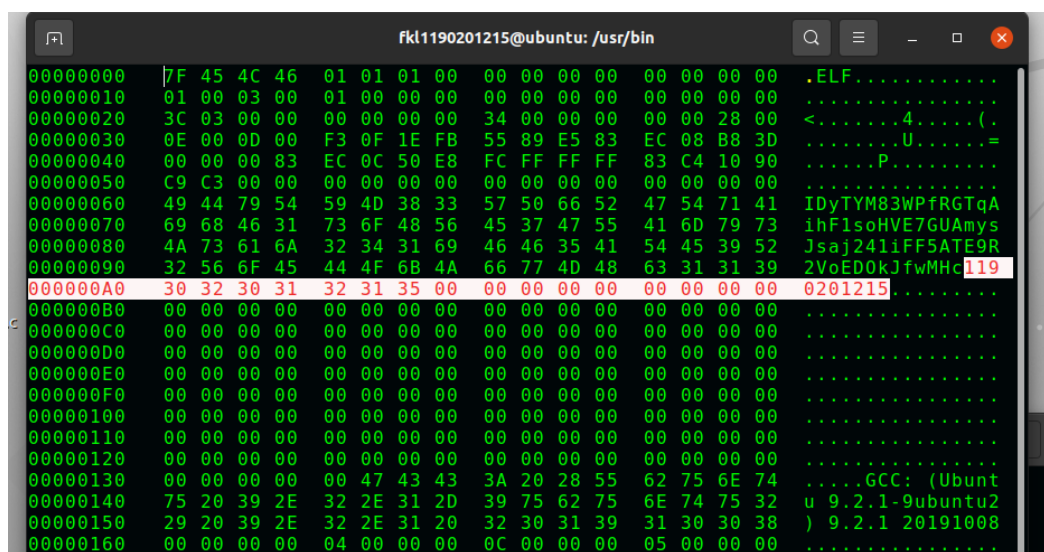
```

将 main.o 和 phase1.o 链接得到 linkbomb1.o，运行之和得到字符串为

```
ject module(s) with the main module.
fkl1190201215@ubuntu:~$ gcc -m32 -o linkbomb main.o phase1.o
fkl1190201215@ubuntu:~$ ./linkbomb
jsuW5dDSd5ytHtlbi4wmsPKPCLv6b6aFlxftLcxeYkMhI6C01GcJGTq8e75h9woEl92WNKCU0qwCYMC
Zqn8koPLiTB QioEL6UzXPwDQudR4gs20BWUpX3PLAttH0UCHUWIXwYg8We79L1 X
fkl1190201215@ubuntu:~$
```

得到一串乱码，开头是 jsu，我们一会需要用自己的学号替代这些乱码。

使用 hexedit 工具进入 phase1.o，我学号的 ascii 值是 31 31 39 30 32 30 31 32 31 35，剩余的位数用 00 作为字符串的结束，然后从 jsu 所在的数据处开始修改



保存之后重新链接，得到结果

```
fkl1190201215@ubuntu:~$ gcc -m32 -o linkbomb1 main.o phase1.o
fkl1190201215@ubuntu:~$ ./linkbomb1
1190201215
```

3.2 阶段 2 的分析

程序运行结果截图：

```
程序截图 (核心已转储)
fkl1190201215@ubuntu:~$ gcc -m32 -no-pie -o linkbomb2 main.o phase2.o
fkl1190201215@ubuntu:~$ ./linkbomb2
1190201215
```

分析与设计的过程：

链接之后查看反汇编地址，注意链接的时候要加上 -no-pie 不然得到的不是实际地址，因为这个我磨了好久。主要是查看我的 AYvKHsFa 函数。分析函数并用 gdb 调试。我们直到 push 0x804a07c 中存的是我的学号 1190201215，AYvKHsFa 函数执行 strcmp 前想栈里压入了两个参数，一个是我的学号，一

个是调用函数前 push 的数。本题目的就是要在 do_phase 函数的 nop 中压入我们需要的数，并跳转到 AYvKHsFa 函数。

根据反汇编我们知道，存放我学号的地址是 0x804a07c，函数地址是 0x80491fa。

```

080491fa <AYvKHsFa>:
80491fa:  f3 0f 1e fb      endbr32
80491fe:  55               push    %ebp
80491ff:  89 e5            mov     %esp,%ebp
8049201:  83 ec 08         sub     $0x8,%esp
8049204:  83 ec 08         sub     $0x8,%esp
8049207:  68 7c a0 04 08   push    $0x804a07c
804920c:  ff 75 08         pushl   0x8(%ebp)
804920f:  e8 5c fe ff ff   call    8049070 <strcmp@plt>
8049214:  83 c4 10         add     $0x10,%esp
8049217:  85 c0            test    %eax,%eax
8049219:  75 10            jne     804922b <AYvKHsFa+0x31>
804921b:  83 ec 0c         sub     $0xc,%esp
804921e:  ff 75 08         pushl   0x8(%ebp)
8049221:  e8 5a fe ff ff   call    8049080 <puts@plt>
8049226:  83 c4 10         add     $0x10,%esp
8049229:  eb 01            jmp     804922c <AYvKHsFa+0x32>
804922b:  90               nop
804922c:  c9               leave
804922d:  c3               ret

```

于是我们编写如下汇编代码，意思就是把存入我们学号的内存压栈，然后调用函数

```

1 push $0x804a07c
2 mov $0x80491fa, %ecx
3 call *%ecx
4 pop %ecx

```

用指令 gcc -m32 -c bomb2.s 得到机器代码

```

defined
fkl1190201215@ubuntu:~$ gcc -m32 -c bomb2.s
fkl1190201215@ubuntu:~$ objdump -d bomb2.o

bomb2.o:          文件格式 elf32-i386

Disassembly of section .text:

00000000 <.text>:
 0:  68 7c a0 04 08   push    $0x804a07c
 5:  b9 fa 91 04 08   mov     $0x80491fa,%ecx
 a:  ff d1           call    *%ecx
 c:  59             pop     %ecx

```

最后用 hexedit 打开 phase2.o 找到 nop 指令（一堆 90），然后用我们得到的机

器代码覆盖

```

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  04 03 00 00 00 00 00 00 34 00 00 00 00 00 28 00 .....4.....(
00000030  0F 00 0E 00 F3 0F 1E FB 55 89 E5 83 EC 08 83 EC .....U.....
00000040  08 68 00 00 00 00 FF 75 08 E8 FC FF FF FF 83 C4 .h.....U.....
00000050  10 85 C0 75 10 83 EC 0C FF 75 08 E8 FC FF FF FF ..u.....u.....
00000060  83 C4 10 EB 01 90 C9 C3 F3 0F 1E FB 55 89 E5 90 .....U...
00000070  68 7C A0 04 08 B9 FA 91 04 08 FF D1 59 90 90 90 h|.....Y...
00000080  90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 .....
00000090  5D C3 00 00 00 00 00 00 31 31 39 30 32 30 31 32 ].....11902012
000000A0  31 35 00 00 47 43 43 3A 20 28 55 62 75 6E 74 75 15..GCC: (Ubuntu
000000B0  20 39 2E 32 2E 31 2D 39 75 62 75 6E 74 75 32 29 9.2.1-9ubuntu2)
000000C0  20 39 2E 32 2E 31 2D 32 30 31 39 31 30 30 38 00 9.2.1 20191008.
000000D0  04 00 00 00 0C 00 00 00 05 00 00 00 47 4E 55 00 .....GNU.
000000E0  02 00 00 C0 04 00 00 00 03 00 00 00 14 00 00 00 .....
000000F0  00 00 00 00 01 7A 52 00 01 7C 08 01 1B 0C 04 04 .....zR...|.....
00000100  88 01 00 00 1C 00 00 00 1C 00 00 00 00 00 00 00 .....
00000110  34 00 00 00 00 45 0E 08 85 02 42 0D 05 6C C5 0C 4....E...B..l..
00000120  04 04 00 00 1C 00 00 00 3C 00 00 00 34 00 00 00 .....<...4...
00000130  2A 00 00 00 00 45 0E 08 85 02 42 0D 05 62 C5 0C *....E...B..b..
00000140  04 04 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000150  00 00 00 00 01 00 00 00 00 00 00 00 00 00 00 00 .....
00000160  04 00 F1 FF 00 00 00 00 00 00 00 00 00 00 00 00 .....
--- phase2.o --0x7D/0x55C-----

```

最后运行进行链接和运行一直不成功，直到有一次加了 `no-pie`，我才成功，我估计这也是因为链接的时候不是实际地址，所以 `push` 和 `call` 都有问题，总之最后真是好不容易才成功。

```

fkl1190201215@ubuntu:~$ gcc -m32 -no-pie -o linkbomb2 main.o phase2.o
fkl1190201215@ubuntu:~$ ./linkbomb2
1190201215

```

3.3 阶段 3 的分析

程序运行结果截图：

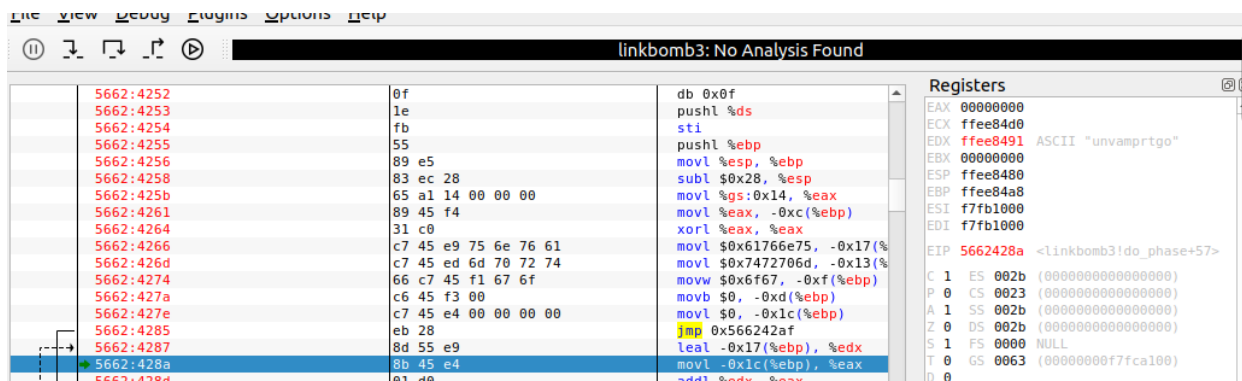
```

fkl1190201215@ubuntu:~$ gcc -m32 -c phase3 patch.c
fkl1190201215@ubuntu:~$ gcc -m32 -o linkbomb3 main.o phase3.o phase3_patch.o
fkl1190201215@ubuntu:~$ ./linkbomb3
1190201215
fkl1190201215@ubuntu:~$

```

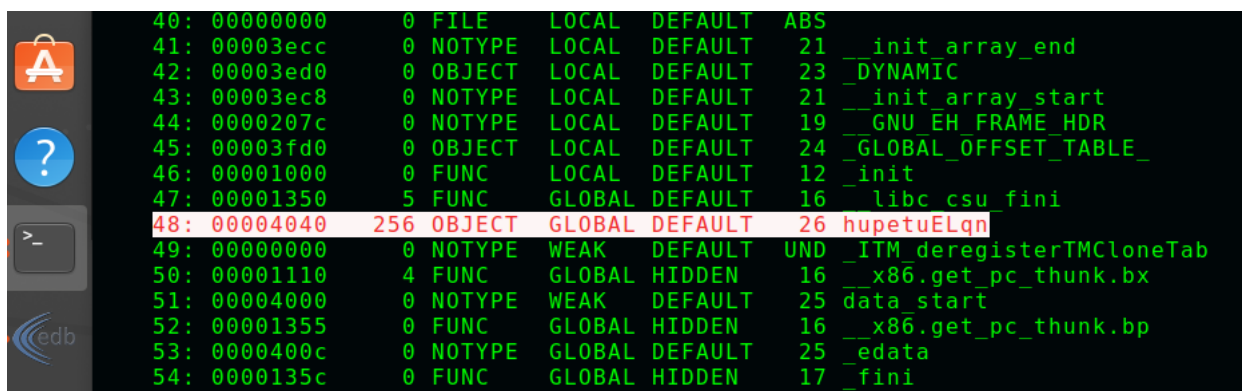
分析与设计的过程:

同上面两个 phase 一样,链接之后用 edb 进行调试,在 do_phase 中-0x17(%ebp),通过 edb 右上角查看得到一串 cookie 值。



图中右上角红色 ffee8491, unvamptrgo。

查看符号表,得到映射的数组变量名为 hupetuELqn, 长度为 256 个字节



根据 ppt 提供的 phase3.c 的框架,可以分析处 PHASE_COOKIE 为 unvamptrgo, PHASE3_CODEBOOK 为 hupetuELqn。

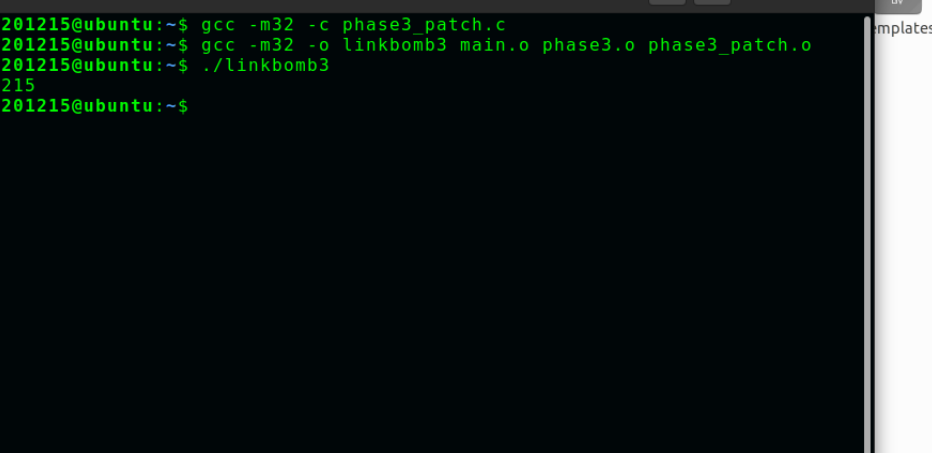
■ phase3.c程序框架

```
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" );
}
```

只需要将 PHASE3_CODEBOOK[256]数组在 COOKIE 对应的位置上改成自己

(16 进制)

1 1 9 0 2 0 1 2 1 5

[illegible]

```
fk1190201215@ubuntu: ~  
fk1190201215@ubuntu:~$ gcc -m32 -c phase3_patch.c  
fk1190201215@ubuntu:~$ gcc -m32 -o linkbomb3 main.o phase3.o phase3_patch.o  
fk1190201215@ubuntu:~$ ./linkbomb3  
1190201215  
fk1190201215@ubuntu:~$
```

程序运行结果截图:

阶段 5 的分析

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

学会了如何使用 hexedit 等工具
学会了如何将多个.o 文件链接在一起
强化了强符合和弱符号的概念
学会用 readelf 查看 elf 头文件

4.2 请给出对本次实验内容的建议

无

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>.