哈爾濱工業大學

实验报告

实 验（五）

题　　目 LinkLab

链接

专　　业 计算机科学与技术

学　　号 1171000410

班　　级 1703005

学　　生 强文杰

指 导 教 师 吴锐

实 验 地 点 G712

实 验 日 期 2018.11.18

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

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。

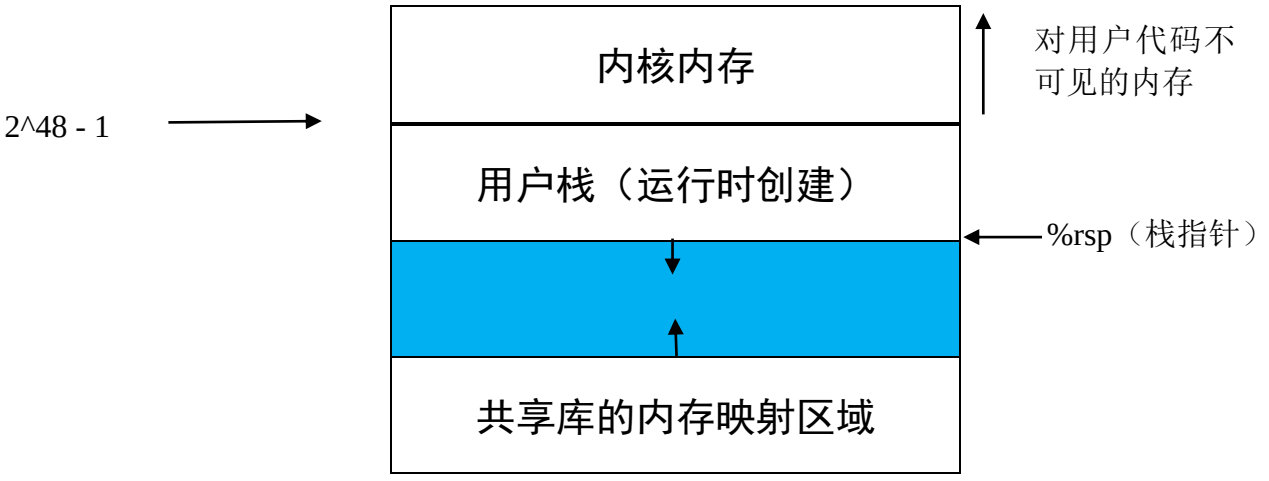
请按顺序写出 LinkAddress 从开始执行到 main 前/后执行的子程序的名字。(gcc 与 objdump/GDB/EDB)

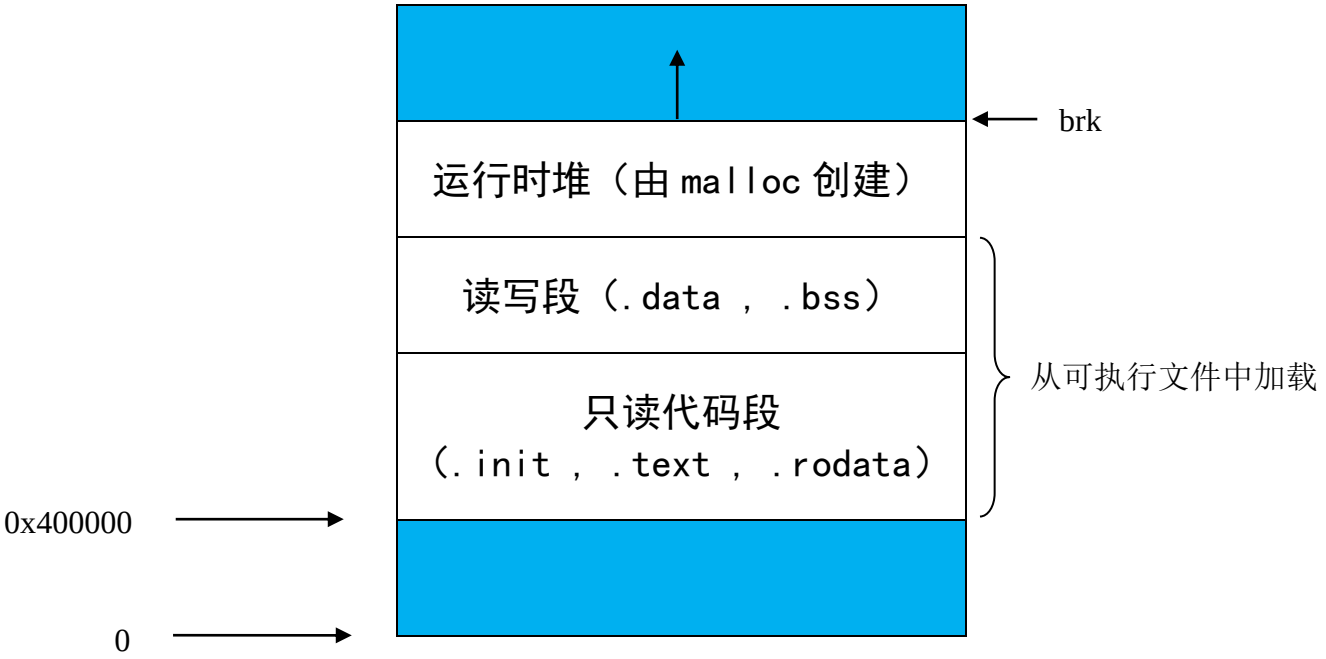
第 2 章 实验预习

2.1 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：将连续的文件映射到运行时的内存段
. init ： 定义了_init 函数，程序初始化代码会调用它
. 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 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）





（注：Linux x86-64 运行时内存映像。没有展示出由于段对其要求和地址空间随机化（ASLR）造成的空隙。区域大小不成比例）

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	各符号的地址、空间（地址从小到大）
只读代码段（.init , .text , .rodata）	exit 0x400630 4195888 printf 0x400600 4195840 malloc 0x400620 4195872 free 0x4005d0 4195792
读写段（.data .bss）	show_pointer 0x400746 4196166 useless 0x400777 4196215 main 0x400782 4196226 global 0x60206c 6299756 huge array 0x602080 6299776 big array 0x40602080 1080041600
运行时堆（由 malloc 创建）	p1 0x7fd9f7587010 140574134398992 p2 0x435e2420 1130243104

	p3 0x7fda07b38010 140574408802320 p4 0x7fd9b7586010 140573060653072 p5 (nil)0
用户栈（运行时创建）	argc0x7ffe1beb339c 140729366819740 local 0x7ffe1beb33a0 140729366819744 argv 0x7ffe1beb34c8 140729366820040 argv[0] 7ffe1beb52ec argv[1] 7ffe1beb52fa argv[2] 7ffe1beb52fd argv[3] 7ffe1beb5308 argv[0] 0x7ffe1beb52ec 140729366827756 ./linkaddress argv[1] 0x7ffe1beb52fa 140729366827770 -u argv[2] 0x7ffe1beb52fd 140729366827773 1171000410 argv[3] 0x7ffe1beb5308 140729366827784 强文杰 env 0x7ffe1beb34f0 140729366820080 env[0] *env 0x7ffe1beb5312 140729366827794 XDG_VTNR=7 env[1] *env 0x7ffe1beb531d 140729366827805 XDG_SESSION_ID=c2 env[2] *env 0x7ffe1beb532f 140729366827823 CLUTTER_IM_MODULE=xim env[3] *env 0x7ffe1beb5345 140729366827845 XDG_GREETER_DATA_DIR=/var/lib/lightdm-data/qwj env[4] *env 0x7ffe1beb5374 140729366827892 GPG_AGENT_INFO=/home/qwj/.gnupg/S.gpg-agent:0:1 env[5] *env 0x7ffe1beb53a4 140729366827940 SHELL=/bin/bash env[6] *env 0x7ffe1beb53b4 140729366827956 TERM=xterm-256color env[7] *env 0x7ffe1beb53c8 140729366827976 VTE_VERSION=4205 env[8] *env 0x7ffe1beb53d9 140729366827993 QT_LINUX_ACCESSIBILITY_ALWAYS_ON=1 env[9] *env 0x7ffe1beb53fc 140729366828028 WINDOWID=52429582 env[10] *env 0x7ffe1beb540e 140729366828046 UPSTART_SESSION=unix:abstract=/com/ubuntu/upstart-session/1000/1676 env[11] *env 0x7ffe1beb5452 140729366828114 GNOME_KEYRING_CONTROL= env[12] *env 0x7ffe1beb5469 140729366828137 GTK_MODULES=gail:atk-bridge:unity-gtk-module env[13] *env 0x7ffe1beb5496 140729366828182 USER=qwj

	env[14] *env 0x7ffe1beb549f 140729366828191 env[15] *env 0x7ffe1beb5a27 140729366829607 QT_ACCESSIBILITY=1 env[16] *env 0x7ffe1beb5a3a 140729366829626 XDG_SESSION_PATH=/org/freedesktop/DisplayManager/Session0 env[17] *env 0x7ffe1beb5a74 140729366829684 XDG_SEAT_PATH=/org/freedesktop/DisplayManager/Seat0 env[18] *env 0x7ffe1beb5aa8 140729366829736 SSH_AUTH_SOCK=/run/user/1000/keyring/ssh env[19] *env 0x7ffe1beb5ad1 140729366829777 DEFAULTS_PATH=/usr/share/gconf/ubuntu.default.path env[20] *env 0x7ffe1beb5b04 140729366829828 XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/usr/share/upstart/xdg:/etc/xdg env[21] *env 0x7ffe1beb5b48 140729366829896 PATH=/home/qwj/bin:/home/qwj/.local/bin:/usr/local/sbin:/usr/local/bin:/usr/sbi env[22] *env 0x7ffe1beb5bd3 140729366830035 DESKTOP_SESSION=ubuntu env[23] *env 0x7ffe1beb5bea 140729366830058 QT_IM_MODULE=fcitx env[24] *env 0x7ffe1beb5bfd 140729366830077 QT_QPA_PLATFORMTHEME=appmenu-qt5 env[25] *env 0x7ffe1beb5c1e 140729366830110 XDG_SESSION_TYPE=x11 env[26] *env 0x7ffe1beb5c33 140729366830131 PWD=/home/qwj/hitics/linklab-1171000410 env[27] *env 0x7ffe1beb5c5b 140729366830171 JOB=gnome-session env[28] *env 0x7ffe1beb5c6d 140729366830189 XMODIFIERS=@im=fcitx env[29] *env 0x7ffe1beb5c82 140729366830210 GNOME_KEYRING_PID= env[30] *env 0x7ffe1beb5c95 140729366830229 LANG=zh_CN.UTF-8 env[31] *env 0x7ffe1beb5ca6 140729366830246 GDM_LANG=zh_CN env[32] *env 0x7ffe1beb5cb5 140729366830261 MANDATORY_PATH=/usr/share/gconf/ubuntu.mandatory.path env[33] *env 0x7ffe1beb5ceb 140729366830315 IM_CONFIG_PHASE=1 env[34] *env 0x7ffe1beb5cfd 140729366830333 COMPIZ_CONFIG_PROFILE=ubuntu env[35] *env 0x7ffe1beb5d1a 140729366830362 GDMSESSION=ubuntu env[36] *env 0x7ffe1beb5d2c 140729366830380 SESSIONTYPE=gnome-session env[37] *env 0x7ffe1beb5d46 140729366830406 GTK2_MODULES=overlay-scrollbar

	env[38] *env 0x7ffe1beb5d65 140729366830437 HOME=/home/qwj env[39] *env 0x7ffe1beb5d74 140729366830452 XDG_SEAT=seat0 env[40] *env 0x7ffe1beb5d83 140729366830467 SHLVL=1 env[41] *env 0x7ffe1beb5d8b 140729366830475 LANGUAGE=zh_CN:zh env[42] *env 0x7ffe1beb5d9d 140729366830493 GNOME_DESKTOP_SESSION_ID=this-is-deprecated env[43] *env 0x7ffe1beb5dc9 140729366830537 UPSTART_INSTANCE= env[44] *env 0x7ffe1beb5ddb 140729366830555 UPSTART_EVENTS=started starting env[45] *env 0x7ffe1beb5dfb 140729366830587 XDG_SESSION_DESKTOP=ubuntu env[46] *env 0x7ffe1beb5e16 140729366830614 LOGNAME=qwj env[47] *env 0x7ffe1beb5e22 140729366830626 QT4_IM_MODULE=fcitx env[48] *env 0x7ffe1beb5e36 140729366830646 XDG_DATA_DIRS=/usr/share/ubuntu:/usr/share/gnome:/usr/local/share:/usr/sh env[49] *env 0x7ffe1beb5e9a 140729366830746 DBUS_SESSION_BUS_ADDRESS=unix:abstract=/tmp/dbus-7GaiE8N1nK env[50] *env 0x7ffe1beb5ed6 140729366830806 LESSOPEN=| /usr/bin/lesspipe %s env[51] *env 0x7ffe1beb5ef6 140729366830838 INSTANCE=Unity env[52] *env 0x7ffe1beb5f05 140729366830853 UPSTART_JOB=unity-settings-daemon env[53] *env 0x7ffe1beb5f27 140729366830887 XDG_RUNTIME_DIR=/run/user/1000 env[54] *env 0x7ffe1beb5f46 140729366830918 DISPLAY=:0 env[55] *env 0x7ffe1beb5f51 140729366830929 XDG_CURRENT_DESKTOP=Unity env[56] *env 0x7ffe1beb5f6b 140729366830955 GTK_IM_MODULE=fcitx env[57] *env 0x7ffe1beb5f7f 140729366830975 LESSCLOSE=/usr/bin/lesspipe %s %s env[58] *env 0x7ffe1beb5fa1 140729366831009 XAUTHORITY=/home/qwj/.Xauthority env[59] *env 0x7ffe1beb5fc2 140729366831042 OLDPWD=/home/qwj/hitics env[60] *env 0x7ffe1beb5fda 140729366831066 _=./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）

main 执行前：

Breakpoint 1 at 0x400598
<function, no debug info> _init;
Breakpoint 2 at 0x4005d0
<function, no debug info> free@plt;
Breakpoint 3 at 0x4005e0
<function, no debug info> puts@plt;
Breakpoint 4 at 0x4005f0
<function, no debug info> __stack_chk_fail@plt;
Breakpoint 5 at 0x400600
<function, no debug info> printf@plt;
Breakpoint 6 at 0x400610
<function, no debug info> __libc_start_main@plt;
Breakpoint 7 at 0x400620
<function, no debug info> malloc@plt;
Breakpoint 8 at 0x400630
<function, no debug info> exit@plt;
Breakpoint 9 at 0x400650
<function, no debug info> _start;
Breakpoint 10 at 0x400680
<function, no debug info> deregister_tm_clones;
Breakpoint 11 at 0x4006c0
<function, no debug info> register_tm_clones;
Breakpoint 12 at 0x400700
<function, no debug info> __do_global_dtors_aux;
Breakpoint 13 at 0x400720
<function, no debug info> frame_dummy;
Breakpoint 14 at 0x40074a
<function, no debug info> show_pointer;
Breakpoint 15 at 0x40077b
<function, no debug info> useless;

main 执行后：

Breakpoint 16 at 0x400786
<function, no debug info> main;
Breakpoint 17 at 0x400b10
<function, no debug info> __libc_csu_init;
Breakpoint 18 at 0x400b80
<function, no debug info> __libc_csu_fini;
Breakpoint 19 at 0x400b84
<function, no debug info> _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：

```
qwj@qwj-virtual-machine:~/hitics/linklab-1171000410$ ./linkbomb1
1171000410
```

分析与设计的过程：

```
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 0002c0 000020 08   I 13   2  4
  [ 4] .data             PROGBITS        00000000 000080 00005f 00  WA  0   0 32
  [ 5] .bss              NOBITS          00000000 0000df 000000 00  WA  0   0  1
  [ 6] .data.rel.local   PROGBITS        00000000 0000e0 000004 00  WA  0   0  4
  [ 7] .rel.data.rel.loc REL             00000000 0002e0 000008 08   I 13   6  4
  [ 8] .text.__x86.get_p PROGBITS        00000000 0000e4 000004 00 AXG  0   0  1
  [ 9] .comment          PROGBITS        00000000 0000e8 000025 01  MS  0   0  1
  [10] .note.GNU-stack   PROGBITS        00000000 00010d 000000 00      0   0  1
  [11] .eh_frame         PROGBITS        00000000 000110 000050 00   A  0   0  4
  [12] .rel.eh_frame     REL             00000000 0002e8 000010 08   I 13  11  4
  [13] .symtab           SYMTAB          00000000 000160 000110 10     14  12  4
  [14] .strtab           STRTAB          00000000 000270 00004d 00      0   0  1
  [15] .shstrtab         STRTAB          00000000 0002f8 00008e 00      0   0  1
```

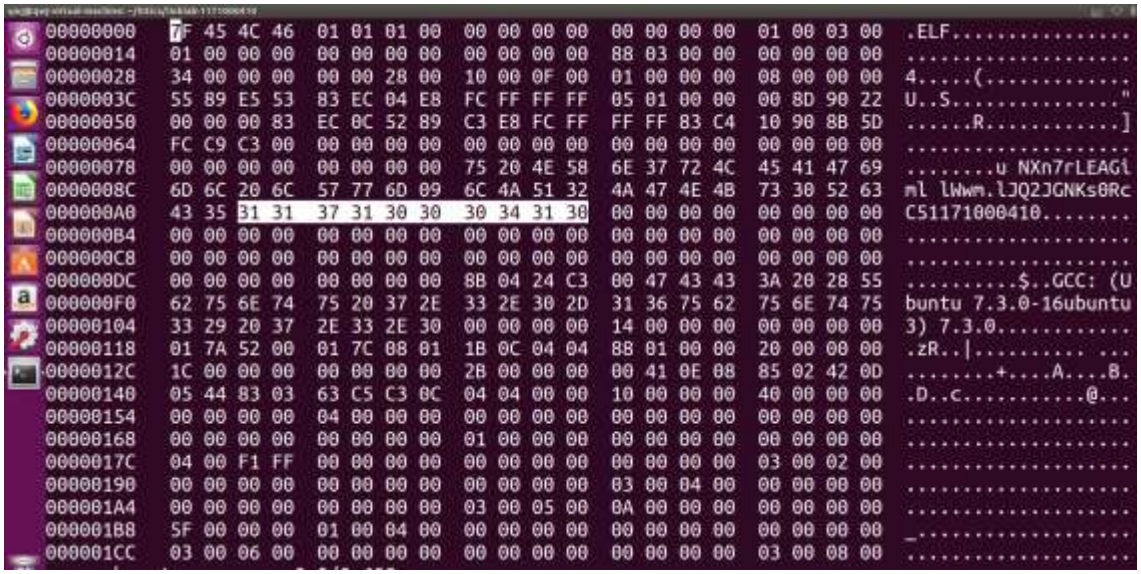
首先 readelf -a phase1.o 查看 elf 文件内容，根据节头信息可知，字符串输出起始地址在 .data 节中偏移量为 32 的位置。

```
qwj@qwj-virtual-machine:~/hitics/linklab-1171000410$ ./linkbomb1
X3uTxl4lOMhOo4OXvzwiz6pog8UwCCHEcWO8k3dX1JlGT6F8AaUU5Bk6orq9
```

然后先 gcc -m32 -o linkbomb1 main.o phase1.o 链接后，运行 linkbomb 程序，查看输出的字符串如上图。

最后使用 hexedit 工具，进入 phase1.o ，之后有两种方法：一是直接找到以上的字符串，并将其对应的 ascii 编码修改；二是根据以上的分析，找到 .data 节偏移量为 32 的位置，并修改 ascii 码（很明显两种方法的效果相同）。

学号 1171000410 对应的 ascii 码为：31 31 37 31 30 30 30 34 31 30 ，最后再以 00 表示字符串结束，于是得到以下的修改结果。再次链接并且运行 linkbomb1 即可输出 1171000410



3.2 阶段 2 的分析

程序运行结果截图：

```
qwj@qwj-virtual-machine:~/hitics/linklab-1171000410$ gcc -m32 -o linkbomb2 main.o phase2.o
qwj@qwj-virtual-machine:~/hitics/linklab-1171000410$ ./linkbomb2
1171000410
```

分析与设计的过程：

首先将文件链接，gcc -m32 -o linkbomb2 main.o phase.o ，然后使用 gdb 对 qmNFFwCm 函数进行调试。

```
   0x08048493 <+0>:     push   %ebp
   0x08048494 <+1>:     mov    %esp,%ebp
   0x08048496 <+3>:     push   %ebx
   0x08048497 <+4>:     sub    $0x4,%esp
   0x0804849a <+7>:     call   0x8048370 <__x86.get_pc_thunk.bx>
   0x0804849f <+12>:    add    $0x1b61,%ebx
   0x080484a5 <+18>:    sub    $0x8,%esp
=> 0x080484a8 <+21>:    lea    -0x19fc(%ebx),%eax
   0x080484ae <+27>:    push   %eax
   0x080484af <+28>:    pushl  0x8(%ebp)
   0x080484b2 <+31>:    call   0x8048300 <strcmp@plt>
   0x080484b7 <+36>:    add    $0x10,%esp
   0x080484ba <+39>:    test   %eax,%eax
   0x080484bc <+41>:    jne    0x80484ce <qmNFFwCm+59>
   0x080484be <+43>:    sub    $0xc,%esp
   0x080484c1 <+46>:    pushl  0x8(%ebp)
   0x080484c4 <+49>:    call   0x8048310 <puts@plt>
   0x080484c9 <+54>:    add    $0x10,%esp
   0x080484cc <+57>:    jmp    0x80484cf <qmNFFwCm+60>
   0x080484ce <+59>:    nop
   0x080484cf <+60>:    mov    -0x4(%ebp),%ebx
   0x080484d2 <+63>:    leave
   0x080484d3 <+64>:    ret
End of assembler dump.
(gdb) ni
0x080484ae in qmNFFwCm ()
(gdb) x/s $eax
0x8048604:      "1171000410"
```

单步执行到如图所示的位置，查看寄存器%eax，发现存的正是我们的学号。

```
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\t..." );
}
```

再分析 strcmp 函数，根据 ppt 给的提示，执行 strcmp 之前向栈中压入了两个参数，一个是 MYID，另一个则是函数传入的参数。因此，整道题的逻辑就是，在 do_phase 函数的 nop 部分，执行压栈，并且跳转到 qmNFFwCm 函数。

根据查询， call 0x804848f <__x86.get_pc_thunk.ax> 和 add $0x1b24,%eax 指令，实现了%eax 指向_GLOBAL_OFFSET_TABLE，同样的 call 0x8048370 <__x86.get_pc_thunk.bx> 和 add $0x1b61,%ebx 指令，实现了%ebx 指向 _GLOBAL_OFFSET_TABLE。

又因为%ebx 之后还执行了 lea -0x19fc(%ebx),%eax ，目的是重定位之后使%eax 指向 .rodata ，因此在 nop 处填写的汇编代码中，需要同样的操作，使 do_phase 中%eax 也指向 .rodata ，操作为 lea -0x19fc(%eax),%eax 。

最后，我们需要使用相对寻址的方式，使 do_phase 函数跳转到 qmNFFwCm 函数 。

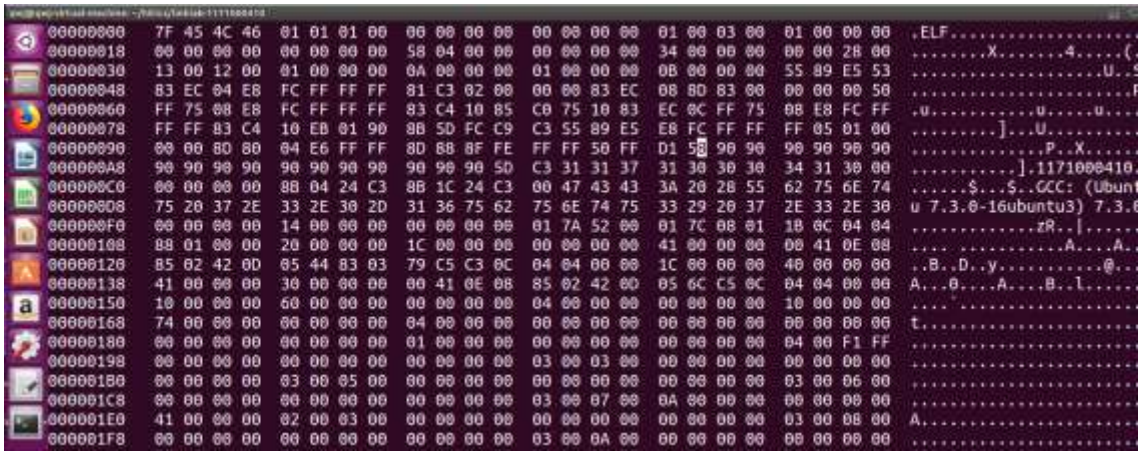
```
(gdb) disas qmNFFwCm
Dump of assembler code for function qmNFFwCm:
   0x08048493 <+0>:     push   %ebp
   0x08048494 <+1>:     mov    %esp,%ebp
   0x08048496 <+3>:     push   %ebx
   0x08048497 <+4>:     sub    $0x4,%esp
   0x0804849a <+7>:     call   0x8048370 <__x86.get_pc_thunk.bx>
   0x0804849f <+12>:    add    $0x1b61,%ebx
   0x080484a5 <+18>:    sub    $0x8,%esp
   0x080484a8 <+21>:    lea    -0x19fc(%ebx),%eax
   0x080484ae <+27>:    push   %eax
   0x080484af <+28>:    pushl  0x8(%ebp)
   0x080484b2 <+31>:    call   0x8048300 <strcmp@plt>
   0x080484b7 <+36>:    add    $0x10,%esp
   0x080484ba <+39>:    test   %eax,%eax
   0x080484bc <+41>:    jne    0x80484ce <qmNFFwCm+59>
   0x080484be <+43>:    sub    $0xc,%esp
   0x080484c1 <+46>:    pushl  0x8(%ebp)
   0x080484c4 <+49>:    call   0x8048310 <puts@plt>
   0x080484c9 <+54>:    add    $0x10,%esp
   0x080484cc <+57>:    jmp    0x80484cf <qmNFFwCm+60>
   0x080484ce <+59>:    nop
   0x080484cf <+60>:    mov    -0x4(%ebp),%ebx
   0x080484d2 <+63>:    leave
```

```
(gdb) reg
Undefined command: "reg".  Try "help".
(gdb) info reg
eax            0x8048604        134514180
ecx            0xffffd060       -12192
edx            0x80484d4        134513876
ebx            0x0      0
esp            0xffffd034       0xffffd034
ebp            0xffffd038       0xffffd038
esi            0xf7fb5000       -134524928
edi            0xf7fb5000       -134524928
eip            0x80484e8        0x80484e8 <do_phase+20>
eflags         0x216    [ PF AF IF ]
cs             0x23     35
ss             0x2b     43
ds             0x2b     43
es             0x2b     43
fs             0x0      0
gs             0x63     99
(gdb) q
```

使用 gdb 调试可知 qmNFFwCm 函数的地址为 0x08048493 ，而 do_phase 函数执行到 lea -0x19fc(%eax),%eax 时%eax 的值为 0x8048604 ，根据二者的差值，即可跳转%eax 减去 0x171 处的地址。

于是汇编代码如下：

最后将得到的反汇编代码使用 hexedit 插入第一个 nop 处即可。



3.3 阶段 3 的分析

程序运行结果截图：

```
qwj@qwj-virtual-machine:~/hitics/linklab-1171000410$ gcc -m32 -c -o phase3_patch
.o phase3_patch.c
qwj@qwj-virtual-machine:~/hitics/linklab-1171000410$ gcc -m32 -o linkbomb3 main.
o phase3.o phase3_patch.o
qwj@qwj-virtual-machine:~/hitics/linklab-1171000410$ ./linkbomb3
1171000410
```

分析与设计的过程：

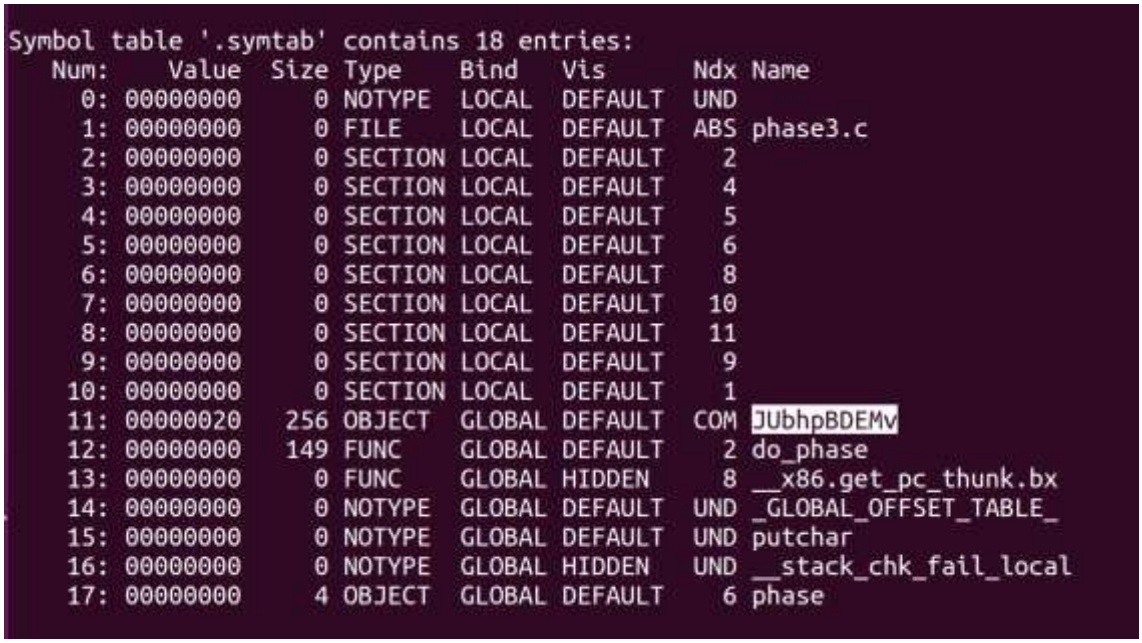
首先，分析 do_phase 的反汇编指令，获取 COOKIE 字符串。gcc -m32 -o linkbomb3 main.o phase3.o 链接之后，使用 gdb 进行调试， 通过调试指令 si 即可进入 do_phase 函数，通过分析 do_phase 函数，可得到如下的循环部分：

```
   0x0804851e <+43>:    movw   $0x6561,-0xf(%ebp)
   0x08048524 <+49>:    movb   $0x0,-0xd(%ebp)
   0x08048528 <+53>:    movl   $0x0,-0x1c(%ebp)
   0x0804852f <+60>:    jmp    0x804855c <do_phase+105>
   0x08048531 <+62>:    lea    -0x17(%ebp),%edx
   0x08048534 <+65>:    mov    -0x1c(%ebp),%eax
   0x08048537 <+68>:    add    %edx,%eax
   0x08048539 <+70>:    movzbl (%eax),%eax
   0x0804853c <+73>:    movzbl %al,%eax
   0x0804853f <+76>:    lea    0x804a060,%edx
   0x08048545 <+82>:    movzbl (%edx,%eax,1),%eax
   0x08048549 <+86>:    movsbl %al,%eax
---Type <return> to continue, or q <return> to quit---r
   0x0804854c <+89>:    sub    $0xc,%esp
=> 0x0804854f <+92>:    push   %eax
   0x08048550 <+93>:    call   0x8048380 <putchar@plt>
   0x08048555 <+98>:    add    $0x10,%esp
   0x08048558 <+101>:   addl   $0x1,-0x1c(%ebp)
   0x0804855c <+105>:   mov    -0x1c(%ebp),%eax
   0x0804855f <+108>:   cmp    $0x9,%eax
   0x08048562 <+111>:   jbe    0x8048531 <do_phase+62>
   0x08048564 <+113>:   sub    $0xc,%esp
   0x08048567 <+116>:   push   $0xa
   0x08048569 <+118>:   call   0x8048380 <putchar@plt>
   0x0804856e <+123>:   add    $0x10,%esp
```

再查看 %ebp-0x17 出的内容，即可得到 COOKIE 字符串。

```
(gdb) x/s $ebp-0x17
0xffffcfc1:     "mdblgtcjae"
```

第二步. 根据符号表，找到映射数组的变量名。



```
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 JUbhpBDEMv
    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
```

变量类型为 COM ,长度为 256 个字节 ，并且就是 do_phase 框架中的 PHASE3_CODEBOOK 。

由此不难写出 phase3.c 程序框架如下：

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

第三步。

得到 cookie 字符串所对应的 ASCII 码值为：109 100 98 108 103 116 99 106 97 101 ，根据程序输出的顺序，我们只需要在 JUbhpBDEMv[] 数组中，并且 COOKIE 的 ASCII 码对应的位置输入自己的学号即可，其余位置不输出，可随便填入

phase_patch.c 中的代码如下：

最后再将 phase_patch.c 编译的文件 phase_patch.o 和 main.o 和 phase3.o 链接在一起，运行 linkbomb3 即可显示学号 1171000410 .

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

对链接的作用和工作步骤有了更深的理解；
学会使用 readelf 工具查看 elf 可重定位目标文件；
学会使用 hexedit 对 .o 文件进行修改
掌握了链接过程中的符号解析和重定位的过程

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.