哈爾濱工業大學

实验报告

实 验（五）

题　　目　LinkLab

链接

专　　业　计算学部

学　　号　1190200708

班　　级　1903008

学　　生　熊峰

指 导 教 师　吴锐

实 验 地 点　G709

实 验 日 期　2021.05.17

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

X86-64 CPU；3.60GHz；16G RAM；256G SSD；1T SSD

1.2.2 软件环境

Win 10

Ubuntu 20.04.2 LTS

1.2.3 开发工具

Visual Studio 2019；Vim；GCC；GDB；Code::Blocks; CLion 2020.3.1 x64;EDB

1.3 实验预习

上实验课前，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。

按顺序写出 ELF 格式的可执行目标文件的各类信息。

按照内存地址从低到高的顺序，写出 Linux 下 X64 内存映像。

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。

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

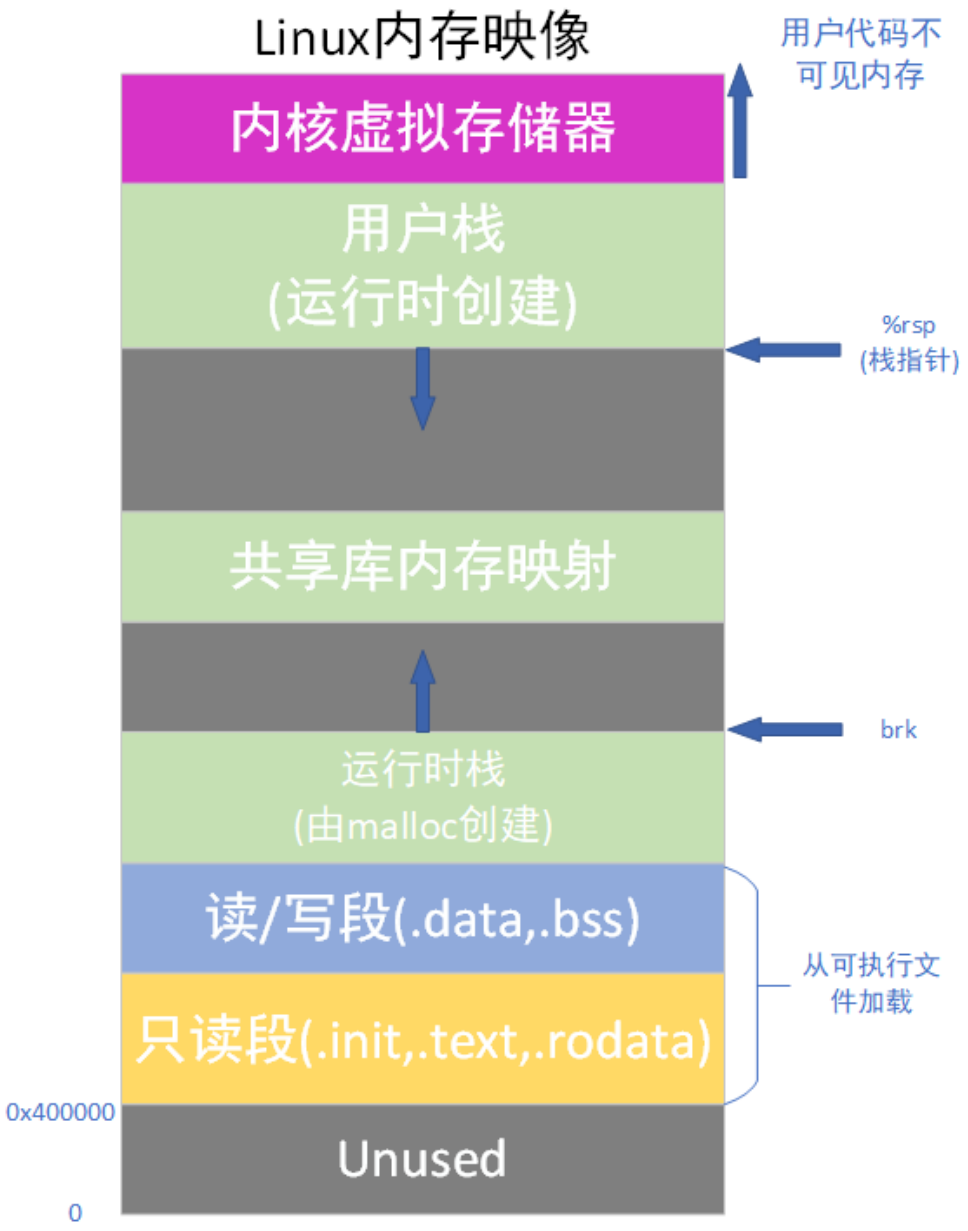
2.1 ELF 文件格式解读

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	ELF 头中字段 e_entry 给出执行程序时第一条指令的地址
程序头表	也称段头表，是一个结构数组，描述可执行文件中的节与虚拟空间中的存储段之间的映射关系。
.init 节	定义_init 函数，该函数用来进行可执行目标文件开始执行时的初始化工作。
.text 节	编译后的代码部分。
.rodata 节	只读数据，如 printf 格式串、switch 跳转表等。在程序装载时，程序中的制度数据一般会被装入进程空间中那些只读的段中。
.data 节	已初始化的全局变量和静态 C 变量。
.bss 节	未初始化全局变量和静态 C 变量。
.symtab 节	存放函数和全局变量（符号表）信息，他不包括局部变量。
.debug 节	调试用符号表，内容格式没有统一规定。
.strtab 节	包含 symtab 和 debug 节中符号及节名。
.line 节	用于调试的节，包含那些调试符号的行号，为程序指令码与源文件的行号建立起联系，内容格式没有统一规定。
节头表	每个节的节名、偏移和大小。

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
只读代码段(.init,.text,.rodata)	show_pointer ©0x562cb5549199 94750020768153 useless 0x562cb55491d0 94750020768208 main 0x562cb55491df 94750020768223

	exit 0x7f8eba5c6bc0 140250988702656 printf 0x7f8eba5e1e10 140250988813840 malloc 0x7f8eba61a260 140250989044320 free 0x7f8eba61a850 140250989045840
读/写段(.data,.bss)	global 0x562cb554c02c 94750020780076 big array 0x562cf554c040 94751094521920 huge array 0x562cb554c040 94750020780096
运行时栈	p1 0x7f8eaa57c010 140250719961104 p2 0x562cf7da86b0 94751136843440 p3 0x7f8eaa55b010 140250719825936 p4 0x7f8e6a55a010 140249646080016 p5 0x7f8dea559010 140247498592272
用户栈	argc 0x7ffd325ac0dc 140725448261852 argv 0x7ffd325ac218 140725448262168 argv[0] 7ffd325ae187 argv[1] 7ffd325ae195 argv[2] 7ffd325ae198 argv[3] 7ffd325ae1a3 argv[0] 0x7ffd325ae187 140725448270215 ./LinkAddress argv[1] 0x7ffd325ae195 140725448270229 -u argv[2] 0x7ffd325ae198 140725448270232 1190200708 argv[3] 0x7ffd325ae1a3 140725448270243 熊峰 env 0x7ffd325ac240 140725448262208 env[0] *env 0x7ffd325ae1aa 140725448270250 SHELL=/bin/bash env[1] *env 0x7ffd325ae1ba 140725448270266 WSL_DISTRO_NAME=Ubuntu env[2] *env 0x7ffd325ae1d1 140725448270289 WT_SESSION=eac5fdec-fd6c-440a-bb67-18f99dfe72a8 env[3] *env 0x7ffd325ae201 140725448270337 NAME=xf1190200708 env[4] *env 0x7ffd325ae213 140725448270355 PWD=/mnt/c/Users/Alienware/Desktop/share/Lab5

```
env[5]  *env    0x7ffd325ae241  140725448270401
LOGNAME=baileys
env[6]  *env    0x7ffd325ae251  140725448270417
HOME=/home/baileys
env[7]  *env    0x7ffd325ae264  140725448270436
LANG=C.UTF-8
env[8]  *env    0x7ffd325ae271  140725448270449
WSL_INTEROP=/run/WSL/8_interop
env[9]  *env    0x7ffd325ae290  140725448270480
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=0
1;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:*.g
z=01;31:*.lrz=01;31:*.lz=01;31:*.lzo=01;31:*.xz=01;3
1:*.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;3
1:*.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=0
1;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:*.t
ga=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:*.m
p4=01;35:*.m4v=01;35:*.mp4v=01;35:*.vob=01;35:*.qt=0
1;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:*.f
lv=01;35:*.gl=01;35:*.dl=01;35:*.xcf=01;35:*.xwd=01;
=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[10] *env    0x7ffd325ae872  140725448271986
LESSCLOSE=/usr/bin/lesspipe %s %s
env[11] *env    0x7ffd325ae894  140725448272020
TERM=xterm-256color
env[12] *env    0x7ffd325ae8a8  140725448272040
LESSOPEN=| /usr/bin/lesspipe %s
env[13] *env    0x7ffd325ae8c8  140725448272072
USER=baileys
env[14] *env    0x7ffd325ae8d5  140725448272085
SHLVL=1
```

```
env[15] *env    0x7ffd325ae8dd  140725448272093
WSLENV=WT_SESSION::WT_PROFILE_ID
env[16] *env    0x7ffd325ae8fe  140725448272126
XDG_DATA_DIRS=/usr/local/share:/usr/share:/var/lib/s
napd/desktop
env[17] *env    0x7ffd325ae93f  140725448272191
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/b
in:/sbin:/bin:/usr/games:/usr/local/games:/mnt/d/Rto
ols/bin:/mnt/d/VMware/VMware Workstation/bin/:/mnt/c
/Program Files/Common Files/Oracle/Java/javapath:/mn
t/c/Program Files (x86)/Intel/Intel(R) Management En
gine Components/iCLS/:/mnt/c/Program Files/Intel/Int
el(R) Management Engine Components/iCLS/:/mnt/c/Wind
ows/system32:/mnt/c/Windows:/mnt/c/Windows/System32/
Wbem:/mnt/c/Windows/System32/WindowsPowerShell/v1.0/
:/mnt/c/Windows/System32/OpenSSH/:/mnt/c/Program Fil
es (x86)/Intel/Intel(R) Management Engine Components
/DAL:/mnt/c/Program Files/Intel/Intel(R) Management
Engine Components/DAL:/mnt/c/Program Files (x86)/NVI
DIA Corporation/PhysX/Common:/mnt/c/Program Files/NV
IDIA Corporation/NVIDIA NGX:/mnt/c/WINDOWS/system32:
/mnt/c/WINDOWS:/mnt/c/WINDOWS/System32/Wbem:/mnt/c/W
INDOWS/System32/WindowsPowerShell/v1.0/:/mnt/c/WINDO
WS/System32/OpenSSH/:/mnt/c/Program Files/NVIDIA Cor
poration/NVIDIA NvDLISR:/mnt/c/Users/Alienware/AppDa
ta/Local/Programs/Python/Python37:/mnt/c/Users/Alien
ware/AppData/Local/Programs/Python/Python37/Scripts:
/mnt/c/Program Files/CMake/bin:/mnt/d/MinGW-w64/ming
w64:/mnt/d/MinGW-w64/mingw64/bin:/mnt/c/Program File
s/Git/cmd:/mnt/c/Program Files/dotnet/:/mnt/c/Progra
m Files/Microsoft SQL Server/130/Tools/Binn/:/mnt/c/
Program Files/Microsoft SQL Server/Client SDK/ODBC/1
70/Tools/Binn/:/mnt/d/nodejs/:/mnt/c/Users/Alienware
/AppData/Local/Programs/Python/Python38:/mnt/c/Users
/Alienware/AppData/Local/Microsoft/WindowsApps:/mnt/
d/Microsoft VS Code/bin:/mnt/c/Users/Alienware/.dotn
et/tools:/mnt/c/Users/Alienware/AppData/Roaming/npm:
/snap/bin
env[18] *env    0x7ffd325aef95  140725448273813
HOSTTYPE=x86_64
env[19] *env    0x7ffd325aefa5  140725448273829
WT_PROFILE_ID={61c54bbd-c2c6-5271-96e7-009a87ff44bf}

env[20] *env    0x7ffd325aefda  140725448273882
_=./LinkAddress
```

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）

通过 objdump -d linkaddress | grep "<.*>:"指令，查看程序中的函数。

```
baileys@xf1190200708:/mnt/c/Users/Alienware/Desktop/share/Lab5$ objdump -d LinkAddress | grep "<.*>:"
0000000000001000 <_init>:
0000000000001020 <.plt>:
0000000000001050 <free@plt>:
0000000000001060 <printf@plt>:
0000000000001070 <malloc@plt>:
0000000000001080 <__cxa_finalize@plt>:
0000000000001090 <puts@plt>:
00000000000010a0 <__stack_chk_fail@plt>:
00000000000010b0 <_start>:
00000000000010e0 <deregister_tm_clones>:
0000000000001110 <register_tm_clones>:
0000000000001150 <__do_global_dtors_aux>:
0000000000001190 <frame_dummy>:
0000000000001199 <show_pointer>:
00000000000011d0 <useless>:
00000000000011df <main>:
00000000000015c0 <__libc_csu_init>:
0000000000001630 <__libc_csu_fini>:
0000000000001638 <_fini>:
```

通过加断点调试程序，可得执行情况。

```
(gdb) b _init
Breakpoint 1 at 0x1000
(gdb) b free@plt
Breakpoint 2 at 0x1050
(gdb) b printf@plt
Breakpoint 3 at 0x1060
(gdb) b malloc@plt
Breakpoint 4 at 0x1070
(gdb) b __cxa_finalize@plt
Breakpoint 5 at 0x1080
(gdb) b puts@plt
Breakpoint 6 at 0x1090
(gdb) b __stack_chk_fail@plt
Breakpoint 7 at 0x10a0
(gdb) b _start
Breakpoint 8 at 0x10b0
(gdb) b deregister_tm_clones
Breakpoint 9 at 0x10e0
(gdb) b register_tm_clones
Breakpoint 10 at 0x1110
(gdb) b __do_global_dtors_aux
Breakpoint 11 at 0x1150
(gdb) b frame_dummy
Breakpoint 12 at 0x1190
(gdb) b show_pointer
Breakpoint 13 at 0x1199
(gdb) b useless
Breakpoint 14 at 0x11d0
(gdb) b main
Breakpoint 15 at 0x11df
(gdb) b __libc_csu_init
Breakpoint 16 at 0x15c0
(gdb) b __libc_csu_fini
Breakpoint 17 at 0x1630
(gdb) b _fini
Breakpoint 18 at 0x1638
(gdb)
```

并使用 GDB 工具调试。

由此可知在 main 前调用的函数，及在 main 后调用的函数如下：

main 前:

malloc@plt 、 free@plt 、 _init 、 _start 、 __libc_csu_init 、 frame_dummy 、 register_tm_clones.

main 后：

useless 、 show_pointer 、 printf@plt 、 malloc@plt 、 puts@plt 、 free@plt 、 __do_global_dtors_aux、__cxa_finalize@plt、_fini

具体函数如下：

malloc@plt
malloc@plt
malloc@plt
malloc@plt
malloc@plt
malloc@plt
malloc@plt
free@plt
malloc@plt
malloc@plt
malloc@plt
malloc@plt
malloc@plt
malloc@plt
malloc@plt
malloc@plt
malloc@plt
malloc@plt
malloc@plt
malloc@plt
_init
_start
__libc_csu_init
_init
frame_dummy
register_tm_clones
main
useless
show_pointer
printf@plt
malloc@plt

printf@plt
show_pointer
printf@plt
puts@plt
printf@plt
show_pointer
printf@plt
puts@plt
printf@plt
show_pointer
printf@plt
puts@plt
printf@plt
show_pointer
printf@plt
puts@plt
printf@plt
show_pointer
printf@plt
puts@plt
printf@plt
show_pointer
printf@plt
puts@plt
printf@plt
show_pointer
printf@plt
puts@plt
printf@plt
show_pointer
printf@plt
puts@plt
printf@plt
show_pointer
printf@plt
puts@plt

printf@plt
show_pointer
printf@plt
puts@plt
printf@plt
show_pointer
printf@plt
puts@plt
printf@plt
show_pointer
printf@plt
puts@plt
printf@plt
show_pointer
printf@plt
puts@plt
printf@plt
show_pointer
printf@plt
puts@plt
printf@plt
show_pointer
printf@plt
puts@plt
printf@plt
show_pointer
printf@plt
puts@plt
printf@plt
show_pointer
printf@plt
puts@plt
printf@plt
show_pointer
printf@plt
puts@plt

printf@plt
show_pointer
printf@plt
puts@plt
printf@plt
show_pointer
printf@plt
puts@plt
printf@plt
show_pointer
printf@plt
puts@plt
printf@plt
show_pointer
printf@plt
puts@plt
printf@plt
show_pointer
printf@plt
puts@plt
malloc@plt
malloc@plt
malloc@plt
malloc@plt
malloc@plt
show_pointer
printf@plt
show_pointer
printf@plt
show_pointer
printf@plt
show_pointer
printf@plt
show_pointer
printf@plt
show_pointer

printf@plt
printf@plt
printf@plt
printf@plt
printf@plt
show_pointer
printf@plt
puts@plt
show_pointer
printf@plt
puts@plt
show_pointer
printf@plt
puts@plt
show_pointer
printf@plt
puts@plt
show_pointer
printf@plt
puts@plt
show_pointer
printf@plt
show_pointer
printf@plt
show_pointer
printf@plt
show_pointer
printf@plt
show_pointer
printf@plt
show_pointer
printf@plt
show_pointer
printf@plt
show_pointer
printf@plt
show_pointer
printf@plt

show_pointer
printf@plt
show_pointer
printf@plt
show_pointer
printf@plt
show_pointer
printf@plt
free@plt
free@plt
free@plt
free@plt
__do_global_dtors_aux
__cxa_finalize@plt
deregister_tm_clones
_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：

```
baileys@xf1190200708:/mnt/c/Users/Alienware/Desktop/share/Lab5/64$ gcc -o linkbomb1 main.o phase1.o -no-pie
baileys@xf1190200708:/mnt/c/Users/Alienware/Desktop/share/Lab5/64$ ./linkbomb1
1190200708
```

分析与设计的过程：

```
gcc -o linkbomb1 main.o phase1.o -no-pie
objdump -d linkbomb1 > linkbomb1.s
```

首先通过 gcc -o linkbomb1 main.o phase1.o -no-pie 指令，将两个文件链接，并对所得到的文件反汇编，并保存到 linkbomb1.s。

```
0000000000401136 <main>:
  401136: f3 0f 1e fa           endbr64
  40113a: 55                    push   %rbp
  40113b: 48 89 e5              mov    %rsp,%rbp
  40113e: 48 83 ec 10           sub    $0x10,%rsp
  401142: 89 7d fc              mov    %edi,-0x4(%rbp)
  401145: 48 89 75 f0           mov    %rsi,-0x10(%rbp)
  401149: 48 8b 05 a8 2f 00 00  mov    0x2fa8(%rip),%rax        # 4040f8 <phase>
  401150: 48 85 c0              test   %rax,%rax
  401153: 74 10                 je     401165 <main+0x2f>
  401155: 48 8b 15 9c 2f 00 00  mov    0x2f9c(%rip),%rdx        # 4040f8 <phase>
  40115c: b8 00 00 00 00        mov    $0x0,%eax
  401161: ff d2                 callq  *%rdx
  401163: eb 0a                 jmp    40116f <main+0x39>
  401165: bf 08 20 40 00        mov    $0x402008,%edi
  40116a: e8 d1 fe ff ff        callq  401040 <puts@plt>
  40116f: b8 00 00 00 00        mov    $0x0,%eax
  401174: c9                    leaveq
  401175: c3                    retq

0000000000401176 <do_phase>:
  401176: f3 0f 1e fa           endbr64
  40117a: 55                    push   %rbp
  40117b: 48 89 e5              mov    %rsp,%rbp
  40117e: b8 6a 40 40 00        mov    $0x40406a,%eax
  401183: 48 89 c7              mov    %rax,%rdi
  401186: e8 b5 fe ff ff        callq  401040 <puts@plt>
  40118b: 90                    nop
  40118c: 5d                    pop    %rbp
  40118d: c3                    retq
  40118e: 66 90                 xchg   %ax,%ax
```

通过读取 linkbomb1.s 的代码，在 0x401186 处，由 mov $0x40406a,%eax 此时向 puts 中传递的参数为 0x40406a.

通过 readelf -S phase1.o 可以发现 phase1.o 中存在 qB0cMnCY 的字符串。

```
Symbol table '.symtab' contains 13 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
     0: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS phase1.c
     2: 0000000000000000     0 SECTION LOCAL  DEFAULT    1
     3: 0000000000000000     0 SECTION LOCAL  DEFAULT    3
     4: 0000000000000000     0 SECTION LOCAL  DEFAULT    5
     5: 0000000000000000   180 OBJECT  LOCAL  DEFAULT    3 qB0cMnCY
     6: 0000000000000000     0 SECTION LOCAL  DEFAULT    7
     7: 0000000000000000     0 SECTION LOCAL  DEFAULT    8
     8: 0000000000000000     0 SECTION LOCAL  DEFAULT    9
     9: 0000000000000000     0 SECTION LOCAL  DEFAULT    6
    10: 0000000000000000    24 FUNC    GLOBAL DEFAULT    1 do_phase
    11: 0000000000000000     0 NOTYPE  GLOBAL DEFAULT  UND puts
    12: 00000000000000b8     8 OBJECT  GLOBAL DEFAULT    3 phase
```

```
baileys@xf1190200708:/mnt/c/Users/Alienware/Desktop/share/Lab5/64$ objdump linkbomb1 -s -j .data

linkbomb1:     file format elf64-x86-64

Contents of section .data:
 404020 00000000 00000000 00000000 00000000  ................
 404030 00000000 00000000 00000000 00000000  ................
 404040 5264436b 64613563 56727a67 72203353  RdCkda5cVrzgr 3S
 404050 44326f79 37387971 09465471 43537752  D2oy78yq.FTqCSwR
 404060 37326a5a 67525953 32615167 76095368  72jZgRYS2aQgv.Sh
 404070 7a6d4e37 30414f7a 57635250 617a7057  zmN70AOzWcRPazpW
 404080 36583031 6144554b 58646d30 72773764  6X01aDUKXdm0rw7d
 404090 536d4336 79334c34 544d6753 50595339  SmC6y3L4TMgSPYS9
 4040a0 73637655 38616b20 42303441 45475961  scvU8ak B04AEGYa
 4040b0 38353448 794c6245 66314645 53205978  854HyLbEf1FES Yx
 4040c0 4d50694f 69714862 44567071 73712043  MPiOiqHbDVpqsq C
 4040d0 69696230 4e524859 366a7a34 62706e78  iib0NRHY6jz4bpnx
 4040e0 76434a6e 65575471 544c5a59 746e6c41  vCJneWTqTLZYtnlA
 4040f0 6b314b00 00000000 76114000 00000000  k1K.....v.@.....
baileys@xf1190200708:/mnt/c/Users/Alienware/Desktop/share/Lab5/64$ objdump phase1.o -s -j .data

phase1.o:     file format elf64-x86-64

Contents of section .data:
 0000 5264436b 64613563 56727a67 72203353  RdCkda5cVrzgr 3S
 0010 44326f79 37387971 09465471 43537752  D2oy78yq.FTqCSwR
 0020 37326a5a 67525953 32615167 76095368  72jZgRYS2aQgv.Sh
 0030 7a6d4e37 30414f7a 57635250 617a7057  zmN70AOzWcRPazpW
 0040 36583031 6144554b 58646d30 72773764  6X01aDUKXdm0rw7d
 0050 536d4336 79334c34 544d6753 50595339  SmC6y3L4TMgSPYS9
 0060 73637655 38616b20 42303441 45475961  scvU8ak B04AEGYa
 0070 38353448 794c6245 66314645 53205978  854HyLbEf1FES Yx
 0080 4d50694f 69714862 44567071 73712043  MPiOiqHbDVpqsq C
 0090 69696230 4e524859 366a7a34 62706e78  iib0NRHY6jz4bpnx
 00a0 76434a6e 65575471 544c5a59 746e6c41  vCJneWTqTLZYtnlA
 00b0 6b314b00 00000000 00000000 00000000  k1K.............
baileys@xf1190200708:/mnt/c/Users/Alienware/Desktop/share/Lab5/64$
```

通过 objdump linkbomb1 -s -j .data 和 objdump phase1 -s -j .data，可以观察到 phase1.o 中的.data 节相对 linkbomb1 中的.data 节的偏移量为 0x20。故此时实际被读取字符串在 phase1.o 中的偏移量为 0x40406a - 0x404040 = 0x2a。

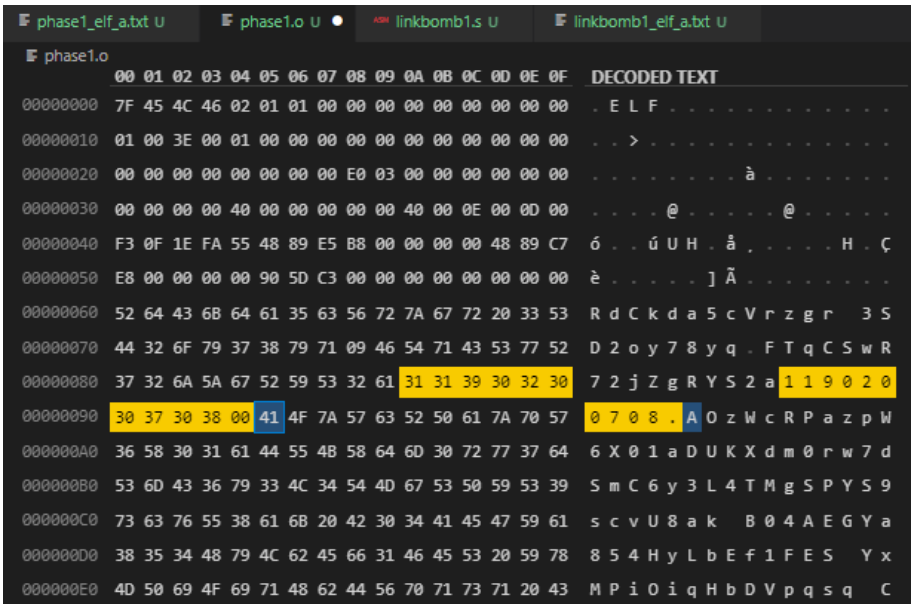
```
0000000000000020 0000000000000008 WA       0     0     8
[25] .data             PROGBITS         0000000000404020  00003020
     00000000000000e0  0000000000000000  WA       0     0     32
```

由偏移量为 0x2a，因此，将 phase1.o 中的.data 区域的起始位置偏移 0x2a 处修改为 1190200708’\0’即可满足题目要求。

通过 readelf -S phase1.o 查看 phase1.o 的节头表。

```
baileys@xf1190200708:/mnt/c/Users/Alienware/Desktop/share/Lab5/64$ readelf -S phase1.o
There are 14 section headers, starting at offset 0x3e0:

Section Headers:
  [Nr] Name              Type             Address           Offset
       Size              EntSize          Flags  Link  Info  Align
  [ 0]                   NULL             0000000000000000  00000000
       0000000000000000  0000000000000000           0     0     0
  [ 1] .text             PROGBITS         0000000000000000  00000040
       0000000000000018  0000000000000000  AX       0     0     1
  [ 2] .rela.text        RELA             0000000000000000  00000308
       0000000000000030  0000000000000018   I      11     1     8
  [ 3] .data             PROGBITS         0000000000000000  00000060
       00000000000000c0  0000000000000000  WA       0     0     32
```

故根据计算此时输出的字符串的起始地址应为.data 区域在 phase1.o 中的偏移量 0x60 加上被读取字符串在.data 区域处的偏移量 0x2a，即 0x60+0x2a=0x8a。将 phase1.o 的 0x8a 处字符开始修改为 1190200708’\0’即可实现。通过 VSCode 中的 hexedit 插件，可以对 0x8a 处的字符完成修改。



将 0x8a 处的字符修改为 1190200708’\0’.保存并重新链接。

```
baileys@xf1190200708:/mnt/c/Users/Alienware/Desktop/share/Lab5/64$ gcc -o linkbomb1 main.o phase1.o -no-pie
baileys@xf1190200708:/mnt/c/Users/Alienware/Desktop/share/Lab5/64$ ./linkbomb1
1190200708
```

此时显示为个人的学号 1190200708.

3.2 阶段 2 的分析

程序运行结果截图：

```
baileys@xf1190200708:/mnt/c/Users/Alienware/Desktop/share/Lab5/64$ gcc -o linkbomb2 main.o
phase2.o -no-pie
baileys@xf1190200708:/mnt/c/Users/Alienware/Desktop/share/Lab5/64$ ./linkbomb2
1190200708
baileys@xf1190200708:/mnt/c/Users/Alienware/Desktop/share/Lab5/64$
```

分析与设计的过程：

```
gcc -o linkbomb2 main.o phase2.o -no-pie
objdump -d linkbomb2>linkbomb2.s
```

首先将 main.o 与 phase2.o 链接起来，并使用 objdump -d linkbomb2 > linkbomb2.s 指令，将 linkbomb2 的汇编代码保存到 linkbomb2.s 中。

```
  60:   c3                      retq
baileys@xf1190200708:/mnt/c/Users/Alienware/Desktop/share/Lab5/64$ readelf -S phase2.o
There are 15 section headers, starting at offset 0x400:

Section Headers:
  [Nr] Name              Type             Address           Offset
       Size              EntSize          Flags  Link  Info  Align
  [ 0]                   NULL             0000000000000000  00000000
       0000000000000000  0000000000000000           0     0     0
  [ 1] .text             PROGBITS         0000000000000000  00000040
       0000000000000061  0000000000000000  AX       0     0     1
  [ 2] .rela.text        RELA             0000000000000000  000002f0
       0000000000000048  0000000000000018   I      12     1     8
  [ 3] .data             PROGBITS         0000000000000000  000000a8
```

通过 readelf -S phase2.o 指令，查看 phase2.o 的代码段，可知.text 的起始位置为 0x40。

```
Disassembly of section .text:

0000000000000000 <HPZrkXfZ>:
   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
  19:   48 89 c7                mov    %rax,%rdi
  1c:   e8 00 00 00 00          callq  21 <HPZrkXfZ+0x21>
  21:   85 c0                   test   %eax,%eax
  23:   75 0e                   jne    33 <HPZrkXfZ+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 <HPZrkXfZ+0x31>
  31:   eb 01                   jmp    34 <HPZrkXfZ+0x34>
  33:   90                      nop
  34:   c9                      leaveq
  35:   c3                      retq

0000000000000036 <do_phase>:
  36:   f3 0f 1e fa             endbr64
  3a:   55                      push   %rbp
```

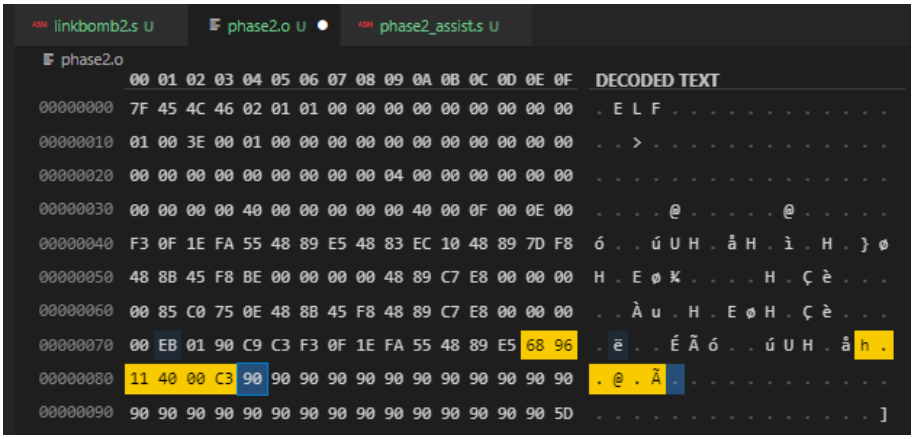
通过 objdump -d phase2.o 指令查看当前两个函数的位置的相对偏移量。由上图可知，phase2.o 的 0x40 处的代码为 HPZrkXfZ 函数，.text 偏移量 0x40 加上 do_phase 函数的偏移量 0x36，即 0x40 + 0x36 = 0x76 处为 do_phase 函数，通过 hexedit 进入 phase2.o 中修改。

首先编写 phase2_assist.s 程序，进入 HPZrkXfZ 函数中。

```
pushq $0x0000000000401196

retq
```

```
baileys@xf1190200708:/mnt/c/Users/Alienware/Desktop/share/Lab5/64$ objdump -d phase2_assist.o

phase2_assist.o:     file format elf64-x86-64


Disassembly of section .text:

0000000000000000 <.text>:
   0:   68 96 11 40 00          pushq  $0x401196
   5:   c3                      retq
```

将 phase2.o 中函数 do_phase 中 nop 指令先修改为 68 96 11 40 00 c3，使函数跳转到 HPZrkXfZ 函数。



将修改后的程序与 main.o 链接，并调试。

```
(gdb) b HPZrkXfZ
Breakpoint 1 at 0x401196
(gdb) r
Starting program: /mnt/c/Users/Alienware/Desktop/share/Lab5/64/linkbomb2

Breakpoint 1, 0x0000000000401196 in HPZrkXfZ ()
(gdb) disass
Dump of assembler code for function HPZrkXfZ:
=> 0x0000000000401196 <+0>:     endbr64
   0x000000000040119a <+4>:     push   %rbp
   0x000000000040119b <+5>:     mov    %rsp,%rbp
   0x000000000040119e <+8>:     sub    $0x10,%rsp
   0x00000000004011a2 <+12>:    mov    %rdi,-0x8(%rbp)
   0x00000000004011a6 <+16>:    mov    -0x8(%rbp),%rax
   0x00000000004011aa <+20>:    mov    $0x40207c,%esi
   0x00000000004011af <+25>:    mov    %rax,%rdi
   0x00000000004011b2 <+28>:    callq  0x401060 <strcmp@plt>
   0x00000000004011b7 <+33>:    test   %eax,%eax
   0x00000000004011b9 <+35>:    jne    0x4011c9 <HPZrkXfZ+51>
   0x00000000004011bb <+37>:    mov    -0x8(%rbp),%rax
   0x00000000004011bf <+41>:    mov    %rax,%rdi
   0x00000000004011c2 <+44>:    callq  0x401050 <puts@plt>
   0x00000000004011c7 <+49>:    jmp    0x4011ca <HPZrkXfZ+52>
   0x00000000004011c9 <+51>:    nop
   0x00000000004011ca <+52>:    leaveq
   0x00000000004011cb <+53>:    retq
End of assembler dump.
(gdb) x/s 0x40207c
0x40207c:       "1190200708"
```

发现其将%rdi 中保存的地址对应的字符串与 0x40207c 处的字符串相比，故将%rdi 所对应的字符串修改为 1190200708.

故将%rdi 赋值为 0x40207c 即可，且发现在 HPZrkXfZ 函数中存在对%rdi,%rax,%rsi 的调用，故先将其保存。

将 phase2_assist.s 修改为如下：

phase2.o U | linkbomb2.s U | phase2_assist.s U

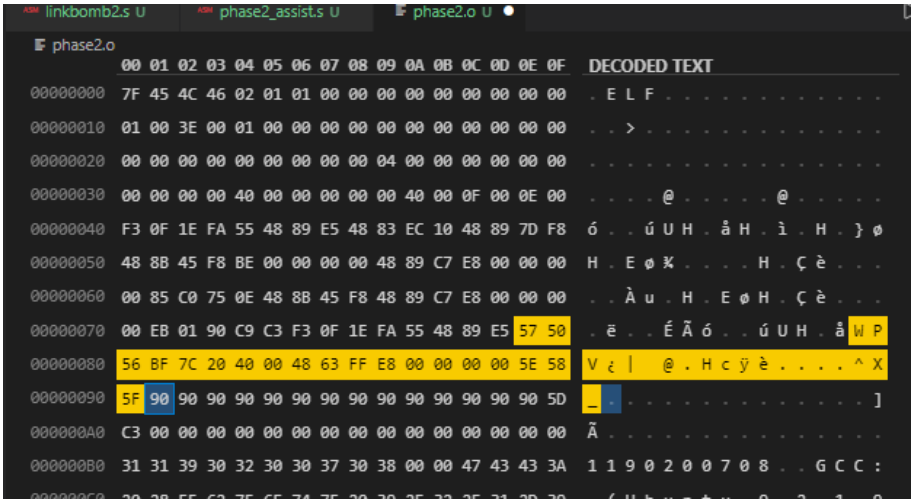
phase2_assist.s

```
1   push   %rdi
2   push   %rax
3   push   %rsi
4   mov    $0x40207c,%edi
5   movslq %edi,%rdi
6   callq 0x40207c
7   pop  %rsi
8   pop  %rax
9   pop  %rdi
10
```

通过 gcc -c phase2_assist.s，将汇编代码编译为机器指令。

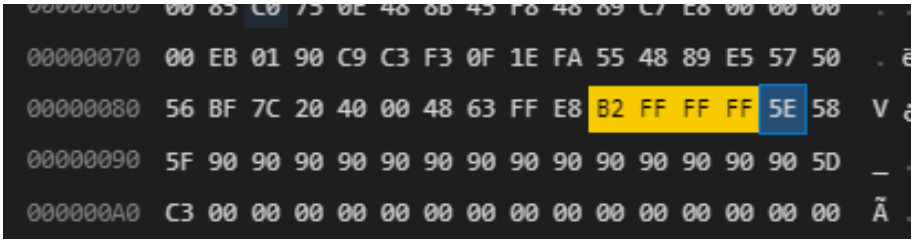
```
baileys@xf1190200708:/mnt/c/Users/Alienware/Desktop/share/Lab5/64$ gcc -c phase2_assist.s
baileys@xf1190200708:/mnt/c/Users/Alienware/Desktop/share/Lab5/64$ objdump -d phase2_assist.o

phase2_assist.o:     file format elf64-x86-64


Disassembly of section .text:

0000000000000000 <.text>:
   0:   57                      push   %rdi
   1:   50                      push   %rax
   2:   56                      push   %rsi
   3:   bf 7c 20 40 00          mov    $0x40207c,%edi
   8:   48 63 ff                movslq %edi,%rdi
   b:   e8 00 00 00 00          callq  0x10
  10:   5e                      pop    %rsi
  11:   58                      pop    %rax
  12:   5f                      pop    %rdi
```

将 phase2.o 中的 0x76 处后的位置修改为上图所示指令。



通过链接，得到下一条指令的地址为 0x4011e4，而 HPZrkXfZ 函数的地址为 0x401196，故通过计算，0x401196 - 0x4011e4 = - 0x4e.且-0x4e 的补码为 ff ff ff b2，由于机器为小端法表示机器，故将 call 后修改为 b2 ff ff ff。

```
00000000004011cc <do_phase>:
  4011cc:       f3 0f 1e fa             endbr64
  4011d0:       55                      push   %rbp
  4011d1:       48 89 e5                mov    %rsp,%rbp
  4011d4:       57                      push   %rdi
  4011d5:       50                      push   %rax
  4011d6:       56                      push   %rsi
  4011d7:       bf 7c 20 40 00          mov    $0x40207c,%edi
  4011dc:       48 63 ff                movslq %edi,%rdi
  4011df:       e8 00 00 00 00          callq  4011e4 <do_phase+0x18>
  4011e4:       5e                      pop    %rsi
  4011e5:       58                      pop    %rax
  4011e6:       5f                      pop    %rdi
  4011e7:       90                      nop
  4011e8:       90                      nop
```



保存后重新编译，并运行程序，程序成功输出 1190200708.

```
baileys@xf1190200708:/mnt/c/Users/Alienware/Desktop/share/Lab5/64$ gcc -o linkbomb2 main.o
phase2.o -no-pie
baileys@xf1190200708:/mnt/c/Users/Alienware/Desktop/share/Lab5/64$ ./linkbomb2
1190200708
baileys@xf1190200708:/mnt/c/Users/Alienware/Desktop/share/Lab5/64$
```

3.3 阶段 3 的分析

程序运行结果截图：

```
baileys@xf1190200708:/mnt/c/Users/Alienware/Desktop/share/Lab5/64$ gcc -c phase3_patch.c
baileys@xf1190200708:/mnt/c/Users/Alienware/Desktop/share/Lab5/64$ gcc -o linkbomb3 main.o phase3.o phase3_patch.o -no-p
ie
baileys@xf1190200708:/mnt/c/Users/Alienware/Desktop/share/Lab5/64$ ./linkbomb3
1190200708
baileys@xf1190200708:/mnt/c/Users/Alienware/Desktop/share/Lab5/64$
```

分析与设计的过程：

```
66: 0000000000404048     0 OBJECT  GLOBAL HIDDEN    25 __TMC_END__
67: 0000000000401000     0 FUNC    GLOBAL HIDDEN    12 _init
68: 0000000000404080   256 OBJECT  GLOBAL DEFAULT   26 cvLJDHqIcl
```

首先，通过 gcc -o linkbomb3 main.o phase3.o -no-pie，将 main.o 与 phase3.o 链接起来，通过阅读.symtab 段，可以发现在存在字符串名为 cvLJDHqIcl 的字符串，且大小为 256 字节。

```
[26] .bss              NOBITS           0000000000404060  00003048
     0000000000000120  0000000000000000  WA       0     0     32
```

发现其位于.bss 段，即未初始化。满足题设条件中的 PHASE3_CODEBOOK 数组的要求。

接下来解读 do_phase 函数。

```
0x00000000004011b6 <+0>:     endbr64
0x00000000004011ba <+4>:     push   %rbp
0x00000000004011bb <+5>:     mov    %rsp,%rbp
0x00000000004011be <+8>:     sub    $0x20,%rsp
0x00000000004011c2 <+12>:    mov    %fs:0x28,%rax
0x00000000004011cb <+21>:    mov    %rax,-0x8(%rbp)
0x00000000004011cf <+25>:    xor    %eax,%eax
0x00000000004011d1 <+27>:    movabs $0x61706b6574736376,%rax
0x00000000004011db <+37>:    mov    %rax,-0x13(%rbp)
0x00000000004011df <+41>:    movw   $0x696a,-0xb(%rbp)
0x00000000004011e5 <+47>:    movb   $0x0,-0x9(%rbp)
0x00000000004011e9 <+51>:    movl   $0x0,-0x18(%rbp)
0x00000000004011f0 <+58>:    jmp    0x401216 <do_phase+96>
0x00000000004011f2 <+60>:    mov    -0x18(%rbp),%eax
0x00000000004011f5 <+63>:    cltq
0x00000000004011f7 <+65>:    movzbl -0x13(%rbp,%rax,1),%eax
0x00000000004011fc <+70>:    movzbl %al,%eax
0x00000000004011ff <+73>:    cltq
0x0000000000401201 <+75>:    movzbl 0x404060(%rax),%eax
0x0000000000401208 <+82>:    movsbl %al,%eax
0x000000000040120b <+85>:    mov    %eax,%edi
0x000000000040120d <+87>:    callq  0x401060 <putchar@plt>
0x0000000000401212 <+92>:    addl   $0x1,-0x18(%rbp)
0x0000000000401216 <+96>:    mov    -0x18(%rbp),%eax
0x0000000000401219 <+99>:    cmp    $0x9,%eax
0x000000000040121c <+102>:   jbe    0x4011f2 <do_phase+60>
0x000000000040121e <+104>:   mov    $0xa,%edi
0x0000000000401223 <+109>:   callq  0x401060 <putchar@plt>
0x0000000000401228 <+114>:   nop
0x0000000000401229 <+115>:   mov    -0x8(%rbp),%rax
0x000000000040122d <+119>:   xor    %fs:0x28,%rax
0x0000000000401236 <+128>:   je     0x40123d <do_phase+135>
0x0000000000401238 <+130>:   callq  0x401080 <__stack_chk_fail@plt>
0x000000000040123d <+135>:   leaveq
0x000000000040123e <+136>:   retq
```

在 0x401219 处控制循环的出口，即需要循环十次。在 0x40120d 控制传入的参数，0x404060 处为 PHASE3_CODEBOOK 数组，此时%rax 为偏移量，因此只需要将十次循环中的偏移量求出即可。

```
End of assembler d
(gdb) p/x $rax
$1 = 0x76
```

```
End of assembler dump.
(gdb) p/x $rax
$2 = 0x63
```

```
End of assembler dump.
(gdb) p/x $rax
$3 = 0x73
```

```
End of assembler dump.
(gdb) p/x $rax
$4 = 0x74
```

```
End of assembler dump.
(gdb) p/x $rax
$5 = 0x65
```

```
End of assembler dump.
(gdb) p/x $rax
$6 = 0x6b
```

```
End of assembler dump.
(gdb) p/x $rax
$7 = 0x70
```

```
End of assembler dump.
(gdb) p/x $rax
$8 = 0x61
```

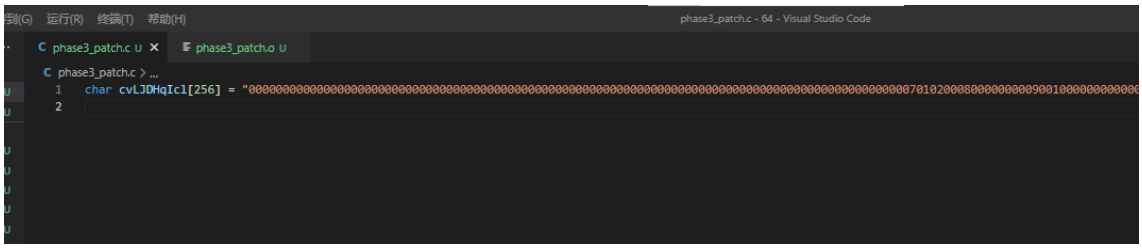
```
End of assembler dump.
(gdb) p/x $rax
$9 = 0x6a
```

```
End of assembler dump.
(gdb) p/x $rax
$10 = 0x69
```

计算出十个元素的偏移量分别为 0x76、0x63、0x73、0x74、0x65、0x6b、0x70、0x61、0x6a、0x69。其分别对应学号的每一位，因此

cvLJDHqIcl[0x76]=1; cvLJDHqIcl[0x63]=1; cvLJDHqIcl[0x73]=9;
cvLJDHqIcl[0x74]=0; cvLJDHqIcl[0x65]=2; cvLJDHqIcl[0x6b]=0;
cvLJDHqIcl[0x70]=0; cvLJDHqIcl[0x61]=7; cvLJDHqIcl[0x6a]=0;
cvLJDHqIcl[0x69]=8;

故将 cvLJDHqIcl 数组中第 0x60 至 0x80 处的元素修改为 0701 0200 0800 0000 0009 0010 0000 0000 即可。



按分析编写程序，将 cvLJDHqIcl 定义为全局变量，并按上述分析初始化。将 phase3_patch.c 编译，并与 main.o、phase3.o 链接，并运行链接后所得程序，程序输出学号。

```
baileys@xf1190200708:/mnt/c/Users/Alienware/Desktop/share/Lab5/64$ gcc -c phase3_patch.c
baileys@xf1190200708:/mnt/c/Users/Alienware/Desktop/share/Lab5/64$ gcc -o linkbomb3 main.o phase3.o phase3_patch.o -no-p
ie
baileys@xf1190200708:/mnt/c/Users/Alienware/Desktop/share/Lab5/64$ ./linkbomb3
1190200708
baileys@xf1190200708:/mnt/c/Users/Alienware/Desktop/share/Lab5/64$
```

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

通过本次实验，我对链接的了解更加深刻；
对 readelf、objdump 等指令的掌握更加牢固；
对可重定位文件的节头表等有了更深刻的了解；
同时对各个节所存放的变量等了解的更加深刻；
同时对强弱符号的应用有了更深层次的了解。

4.2 请给出对本次实验内容的建议

希望 ppt 上的内容更加充实，使我们可以尽快掌握所需掌握的方法。

注：本章为酌情加分项。

参考文献

[1] RANDALE.BRYANT, DAVIDR.O‘HALLARON. 深入理解计算机系统[M]. 机械工业出版社, 2011.