華中科技大學

课 程 实 验 报 告

课程名称： 计算机系统基础

实验名称： 二进制程序分析

院　　系：计算机科学与技术

专业班级：计算机 2202 班

学　　号：U202215378

姓　　名：冯瑞琦

指导教师：王多强

2024 年 **4** 月 **12** 日

一、实验目的与要求

通过逆向分析一个二进制程序（称为“二进制炸弹”）的构成和运行逻辑，加深对理论课中关于程序的机器级表示各方面知识点的理解，增强反汇编、跟踪、分析、调试等能力。

实验环境：Ubuntu，GCC，GDB 等

二、实验内容

作为实验目标的二进制炸弹（binary bombs）可执行程序由多个“关”组成。每一个“关”（阶段）要求输入一个特定字符串，如果输入满足程序代码的要求，该阶段即通过，否则程序输出失败。实验的目标是设法得到得出解除尽可能多阶段的字符串。

为了完成二进制炸弹的拆除任务，需要通过反汇编和分析跟踪程序每一阶段的机器代码，从中定位和理解程序的主要执行逻辑，包括关键指令、控制结构和相关数据变量等等，进而推断拆除炸弹所需要的目标字符串。

实验源程序及相关文件　bomb.rar

bomb.c　　主程序

phases.o　各个阶段的目标程序

support.c 完成辅助功能的目标程序

support.h　公共头文件

阶段 1：　串比较 phase_1(char *input);

要求输出的字符串(input) 与程序中内置的某一特定字符串相同。提示：找到与 input 串相比较的特定串的地址，查看相应单元中的内容，从而确定 input 应输入的串。

阶段 2：循环 phase_2(char *input);

要求在一行上输入 6 个整数数据，与程序自动产生的 6 个数据进行比较，若一致，则过关。提示：将输入串 input 拆分成 6 个数据由函数 read_six_numbers(input, numbers) 完成。之后是各个数据与自动产生的数据的比较，在比较中使用了循环语句。

阶段 3：条件分支 phase_3(char *input);

要求输入一个整数数据，该数据与程序自动生成的 一个数据比较，相等则过关。提示：在自动生成数据时，使用了 switch … case 语句。

阶段 4：递归调用和栈 phase_4(char *input);

要求在一行中输入两个数，第一个数表示在一个有序的数组（或者 binary search tree）中需要搜索到的数，该数是在一定范围之内的；第二个数表示找到搜索数的路径（在树的左边搜索编码为二进制位 0，在树的有边搜索编码为二进制位 1）。

阶段 5：指针和数组访问 phase_5(char *input);

要求在一行中输入一个串，该串与程序自动生成的串相同。在生成串和比较串时，使用了数组和指针。

阶段 6：链表、结构、指针的访问 phase_6(char *input);

要求在一行中输入 6 个数，这 6 个数是一个链表中结点的顺序号（从 1 到 6）。按照输入的顺序号，将对应链表结点中的值形成一个数组。若该数组是按照降序排列的，则过关。

三、实验记录及问题回答

（1） 阶段 1： 串比较 phase_1(char *input);

使用 gdb bomb 进入 gdb，在调用 phase_1 处设置断点设置断点 break 102，执行 run，照操作提示依次输入学号，程序在断点处停下。

```
qr@qr-virtual-machine:~$ gdb bomb
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from bomb...
(gdb) b 102
Breakpoint 1 at 0x80493e1: file bomb.c, line 102.
(gdb) r
Starting program: /home/qr/bomb
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Input your Student ID :
U202215378
welcome  U202215378
You have 6 bombs to defuse!
Gate 1 :  input a string that meets the requirements.
c

Breakpoint 1, main (argc=1, argv=0xffffd1d4) at bomb.c:102
102         phase_1(input);                  /* 若一关炸弹拆除失败，会报错  */
(gdb)
```

反汇编 phase_1 函数：disass phase_1，得到下图。阅读该函数的反汇编程序。可见在 0x08049862 处调用了 strings_not_equal 函数。

```
(gdb) disass phase_1
Dump of assembler code for function phase_1:
   0x08049836 <+0>:     push   %ebp
   0x08049837 <+1>:     mov    %esp,%ebp
   0x08049839 <+3>:     sub    $0x18,%esp
   0x0804983c <+6>:     movzbl 0x804c375,%eax
   0x08049843 <+13>:    sub    $0x30,%eax
   0x08049846 <+16>:    mov    %al,-0xd(%ebp)
   0x08049849 <+19>:    movsbl -0xd(%ebp),%eax
```

```
   0x08049849 <+19>:    movsbl -0xd(%ebp),%eax
   0x0804984d <+23>:    mov    %eax,-0xc(%ebp)
   0x08049850 <+26>:    mov    -0xc(%ebp),%eax
   0x08049853 <+29>:    imul   $0x32,%eax,%eax
   0x08049856 <+32>:    add    $0x804c060,%eax
   0x0804985b <+37>:    sub    $0x8,%esp
   0x0804985e <+40>:    push   %eax
   0x0804985f <+41>:    push   0x8(%ebp)
   0x08049862 <+44>:    call   0x80495cf <strings_not_equal>
   0x08049867 <+49>:    add    $0x10,%esp
   0x0804986a <+52>:    test   %eax,%eax
   0x0804986c <+54>:    je     0x8049873 <phase_1+61>
   0x0804986e <+56>:    call   0x8049816 <explode_bomb>
   0x08049873 <+61>:    nop
   0x08049874 <+62>:    leave
   0x08049875 <+63>:    ret
End of assembler dump.
```

使用 layout asm 命令打开反汇编窗口，在调用 strings_not_equal 处设置断点，继续执行。我们可以看到，在 call strings_not_equal 前，设置了进栈参数，打开查看这个参数，查看 eax 所指向的字符串地址。

```
   0x0804985b <phase_1+37>  sub    $0x8,%esp
   0x0804985e <phase_1+40>  push   %eax
   0x0804985f <phase_1+41>  push   0x8(%ebp)
B+> 0x8049862 <phase_1+44>  call   0x80495cf <strings_not_equal>
   0x8049867 <phase_1+49>   add    $0x10,%esp
   0x804986a <phase_1+52>   test   %eax,%eax
   0x804986c <phase_1+54>   je     0x8049873 <phase_1+61>
   0x804986e <phase_1+56>   call   0x8049816 <explode_bomb>
   0x8049873 <phase_1+61>   nop
   0x8049874 <phase_1+62>   leave
   0x8049875 <phase_1+63>   ret
   0x8049876 <phase_2>      push   %ebp
   0x8049877 <phase_2+1>    mov    %esp,%ebp
multi-thre Thread 0xf7fbf540 ( In: phase_1                          L??   PC: 0x8049862
(gdb) b *0x8049862
Breakpoint 2 at 0x8049862
(gdb) c
Continuing.

Breakpoint 2, 0x08049862 in phase_1 ()
(gdb) i r eax
eax            0x804c1f0           134529520
(gdb) x/s 0x804c1f0
0x804c1f0 <special+400>:        "Assemble Language"
```

由此可破解 phase_1 为输入字符串应为“Assemble Language”。经验证结果正确。

```
qr@qr-virtual-machine:~$ ./bomb
Input your Student ID :
U202215378
welcome  U202215378
You have 6 bombs to defuse!
Gate 1 :  input a string that meets the requirements.
Assemble Language
Phase 1 passed!
```

（2） 阶段 2：循环 phase_2(char *input)；

使用 gdb bomb 进入 gdb，在调用 phase_2 处设置断点设置断点 break 107，执行 run，照操作提示依次输入学号以及一个字符串，程序在断点处停下。

```
Gate 2 :  input six intergers that meets the requirements.
c

Breakpoint 1, main (argc=1, argv=0xffffd1d4) at bomb.c:107
107         phase_2(input);
```

反汇编 phase_2 函数：disass phase_2， 阅读该函数的反汇编程序。

```
(gdb) disass phase_2
Dump of assembler code for function phase_2:
   0x08049876 <+0>:     push   %ebp
   0x08049877 <+1>:     mov    %esp,%ebp
   0x08049879 <+3>:     sub    $0x38,%esp
   0x0804987c <+6>:     mov    0x8(%ebp),%eax
   0x0804987f <+9>:     mov    %eax,-0x2c(%ebp)
   0x08049882 <+12>:    mov    %gs:0x14,%eax
   0x08049888 <+18>:    mov    %eax,-0xc(%ebp)
   0x0804988b <+21>:    xor    %eax,%eax
   0x0804988d <+23>:    sub    $0x8,%esp
   0x08049890 <+26>:    lea    -0x24(%ebp),%eax
   0x08049893 <+29>:    push   %eax
   0x08049894 <+30>:    push   -0x2c(%ebp)
   0x08049897 <+33>:    call   0x804954f <read_six_numbers>
   0x0804989c <+38>:    add    $0x10,%esp
   0x0804989f <+41>:    mov    -0x24(%ebp),%eax
   0x080498a2 <+44>:    test   %eax,%eax
   0x080498a4 <+46>:    jns    0x80498ab <phase_2+53>
   0x080498a6 <+48>:    call   0x8049816 <explode_bomb>
   0x080498ab <+53>:    mov    -0x24(%ebp),%eax
   0x080498ae <+56>:    movzbl 0x804c375,%edx
   0x080498b5 <+63>:    movsbl %dl,%edx
   0x080498b8 <+66>:    sub    $0x30,%edx
   0x080498bb <+69>:    cmp    %edx,%eax
   0x080498bd <+71>:    je     0x80498c4 <phase_2+78>
   0x080498bf <+73>:    call   0x8049816 <explode_bomb>
   0x080498c4 <+78>:    mov    -0x20(%ebp),%eax
   0x080498c7 <+81>:    movzbl 0x804c374,%edx
   0x080498ce <+88>:    movsbl %dl,%edx
   0x080498d1 <+91>:    sub    $0x30,%edx
   0x080498d4 <+94>:    cmp    %edx,%eax
   0x080498d6 <+96>:    je     0x80498dd <phase_2+103>
   0x080498d8 <+98>:    call   0x8049816 <explode_bomb>
   0x080498dd <+103>:   movl   $0x2,-0x28(%ebp)
   0x080498e4 <+110>:   jmp    0x8049910 <phase_2+154>
   0x080498e6 <+112>:   mov    -0x28(%ebp),%eax
   0x080498e9 <+115>:   mov    -0x24(%ebp,%eax,4),%eax
   0x080498ed <+119>:   mov    -0x28(%ebp),%edx
   0x080498f0 <+122>:   sub    $0x1,%edx
   0x080498f3 <+125>:   mov    -0x24(%ebp,%edx,4),%ecx
   0x080498f7 <+129>:   mov    -0x28(%ebp),%edx
   0x080498fa <+132>:   sub    $0x2,%edx
   0x080498fd <+135>:   mov    -0x24(%ebp,%edx,4),%edx
   0x08049901 <+139>:   add    %ecx,%edx
   0x08049903 <+141>:   cmp    %edx,%eax
   0x08049905 <+143>:   je     0x804990c <phase_2+150>
   0x08049907 <+145>:   call   0x8049816 <explode_bomb>
   0x0804990c <+150>:   addl   $0x1,-0x28(%ebp)
```

从反汇编程序 0x080498ae <+56>: movzbl 0x804c375,%edx 和 0x080498c7 <+81>: movzbl 0x804c374,%edx 及其后面数行代码可知，先从 0x804c375 和 0x804c374 加载两个特定值，然后将 %dl 中的字节符号扩展到 %edx，将 %edx 中的值减去 '0' 的 ASCII 值，以获取实际数字值，最后分别将输入的第一个和第二个整数的值与上述两个特定值进行比较，如果相等才能继续。查看 0x804c375 和 0x804c374 中的值，发现是开始输入学号（U202215378）的倒数一、二位，即 8 和 7，所以要输入的前两个数字就是 8 和 7。

```
(gdb) x/s 0x804c375-9
0x804c36c <studentid>:  "U202215378"
(gdb) x/c 0x804c375
0x804c375 <studentid+9>:        56 '8'
(gdb) x/c 0x804c374
0x804c374 <studentid+8>:        55 '7'
```

继续分析反汇编程序后半段，发现一个循环结构：

```
0x080498dd <+103>: movl   $0x2,-0x28(%ebp)    ; 初始化计数器 %ebp 为 2
0x08049901 <+139>: add    %ecx,%edx           ;不断将前两个整数相加得到第三个整数
0x0804990c <+150>: addl   $0x1,-0x28(%ebp)    ; 计数器 %ebp 加 1
0x08049910 <+154>: cmpl   $0x5,-0x28(%ebp)    ; 检查计数器 %ebp 的值是否小于或等于 5
```

，即重复 4 次。

根据这个规律得到后面的四个数分别为 8+7=15，7+15=22，15+22=37，22+37=59。

由此破解 phase_2 输入的 6 个整数应为 8 7 15 22 37 59，经验证结果正确。

```
qr@qr-virtual-machine:~$ ./bomb
Input your Student ID :
U202215378
welcome  U202215378
You have 6 bombs to defuse!
Gate 1 :  input a string that meets the requirements.
Assemble Language
Phase 1 passed!
Gate 2 :  input six intergers that meets the requirements.
8 7 15 22 37 59
Phase 2 passed!
```

（3） 阶段 3：条件分支 phase_3(char *input)；

使用 gdb bomb 进入 gdb，在调用 phase_3 处设置断点设置断点 break 112，执行 run，照操作提示依次输入内容，程序在断点处停下。反汇编 phase_3 函数：disass phase_3，阅读该函数的反汇编程序。

```
Gate 3 :  input 2 intergers.
c

Breakpoint 1, main (argc=1, argv=0xffffd1d4) at bomb.c:112
112         phase_3(input);
(gdb) disass phase_3
Dump of assembler code for function phase_3:
   0x0804992a <+0>:     push   %ebp
   0x0804992b <+1>:     mov    %esp,%ebp
   0x0804992d <+3>:     sub    $0x38,%esp
   0x08049930 <+6>:     mov    0x8(%ebp),%eax
```

```
0x08049933 <+9>:     mov    %eax,-0x2c(%ebp)
0x08049936 <+12>:    mov    %gs:0x14,%eax
0x0804993c <+18>:    mov    %eax,-0xc(%ebp)
0x0804993f <+21>:    xor    %eax,%eax
0x08049941 <+23>:    movl   $0x0,-0x14(%ebp)
0x08049948 <+30>:    movl   $0x0,-0x10(%ebp)
0x0804994f <+37>:    lea    -0x18(%ebp),%eax
0x08049952 <+40>:    push   %eax
0x08049953 <+41>:    lea    -0x1c(%ebp),%eax
0x08049956 <+44>:    push   %eax
0x08049957 <+45>:    push   $0x804a360
0x0804995c <+50>:    push   -0x2c(%ebp)
0x0804995f <+53>:    call   0x80490d0 <__isoc99_sscanf@plt>
0x08049964 <+58>:    add    $0x10,%esp
0x08049967 <+61>:    mov    %eax,-0x10(%ebp)
0x0804996a <+64>:    cmpl   $0x1,-0x10(%ebp)
0x0804996e <+68>:    jg     0x8049975 <phase_3+75>
0x08049970 <+70>:    call   0x8049816 <explode_bomb>
0x08049975 <+75>:    movzbl 0x804c373,%eax
0x0804997c <+82>:    movsbl %al,%eax
0x0804997f <+85>:    lea    -0x30(%eax),%edx
0x08049982 <+88>:    mov    -0x1c(%ebp),%eax
0x08049985 <+91>:    cmp    %eax,%edx
0x08049987 <+93>:    je     0x804998e <phase_3+100>
0x08049989 <+95>:    call   0x8049816 <explode_bomb>
0x0804998e <+100>:   mov    -0x1c(%ebp),%eax
0x08049991 <+103>:   cmp    $0x9,%eax
0x08049994 <+106>:   ja     0x80499f9 <phase_3+207>
0x08049996 <+108>:   mov    0x804a368(,%eax,4),%eax
0x0804999d <+115>:   jmp    *%eax
0x0804999f <+117>:   movl   $0x32f,-0x14(%ebp)
0x080499a6 <+124>:   jmp    0x80499fe <phase_3+212>
0x080499a8 <+126>:   movl   $0x130,-0x14(%ebp)
0x080499af <+133>:   jmp    0x80499fe <phase_3+212>
0x080499b1 <+135>:   movl   $0x184,-0x14(%ebp)
0x080499b8 <+142>:   jmp    0x80499fe <phase_3+212>
0x080499ba <+144>:   movl   $0x28e,-0x14(%ebp)
0x080499c1 <+151>:   jmp    0x80499fe <phase_3+212>
0x080499c3 <+153>:   movl   $0x11c,-0x14(%ebp)
0x080499ca <+160>:   jmp    0x80499fe <phase_3+212>
0x080499cc <+162>:   movl   $0x201,-0x14(%ebp)
0x080499d3 <+169>:   jmp    0x80499fe <phase_3+212>
0x080499d5 <+171>:   movl   $0x1a9,-0x14(%ebp)
0x080499dc <+178>:   jmp    0x80499fe <phase_3+212>
0x080499de <+180>:   movl   $0x374,-0x14(%ebp)
0x080499e5 <+187>:   jmp    0x80499fe <phase_3+212>
0x080499e7 <+189>:   movl   $0x7b,-0x14(%ebp)
0x080499ee <+196>:   jmp    0x80499fe <phase_3+212>
0x080499f0 <+198>:   movl   $0x141,-0x14(%ebp)
0x080499f7 <+205>:   jmp    0x80499fe <phase_3+212>
0x080499f9 <+207>:   call   0x8049816 <explode_bomb>
0x080499fe <+212>:   mov    -0x18(%ebp),%eax
0x08049a01 <+215>:   cmp    %eax,-0x14(%ebp)
0x08049a04 <+218>:   je     0x8049a0b <phase_3+225>
0x08049a06 <+220>:   call   0x8049816 <explode_bomb>
0x08049a0b <+225>:   nop
0x08049a0c <+226>:   mov    -0xc(%ebp),%eax
0x08049a0f <+229>:   sub    %gs:0x14,%eax
0x08049a16 <+236>:   je     0x8049a1d <phase_3+243>
0x08049a18 <+238>:   call   0x8049070 <__stack_chk_fail@plt>
0x08049a1d <+243>:   leave
0x08049a1e <+244>:   ret
nd of assembler dump.
```

分析代码得要输入的第一个数由 0x08049975 <+75>:　movzbl 0x804c373,%eax 在 0x804c373 处获得，查询地址可知为学号倒数第三位数字“3”。

```
(gdb) x/c 0x804c373
0x804c373 <studentid+7>:        51 '3'
```

又由下列代码：

0x08049996 <+108>:　mov　0x804a368(,%eax,4),%eax

；根据第 1 个的值获取对应的操作地址

0x0804999d <+115>:　jmp　*%eax　；跳转到操作地址处执行相应的操作

以及 phase_3 中用红色方框框出的内容可知，根据第一个数字 3 自动生成第二个数字，在自动生成过程中使用了 switch … case 语句。解读 switch … case 语句内容可知：

如果第 1 个整数为 0，将密码设为 0x32f

如果第 1 个整数为 1，将密码设为 0x130

如果第 1 个整数为 2，将密码设为 0x184

如果第 1 个整数为 3，将密码设为 0x28e

如果第 1 个整数为 4，将密码设为 0x11c

如果第 1 个整数为 5，将密码设为 0x201

如果第 1 个整数为 6，将密码设为 0x1a9

如果第 1 个整数为 7，将密码设为 0x374

如果第 1 个整数为 8，将密码设为 0x7b

如果第 1 个整数为 9，将密码设为 0x141

而已知第一个数为 3，对应的第二个数为 0x28e，用十进制表示为 654.

由此破解 phase_3 输入的 3 个整数应为 3 654，经验证结果正确。

```
qr@qr-virtual-machine:~$ ./bomb
Input your Student ID :
U202215378
welcome  U202215378
You have 6 bombs to defuse!
Gate 1 :  input a string that meets the requirements.
Assemble Language
Phase 1 passed!
Gate 2 :  input six intergers that meets the requirements.
8 7 15 22 37 59
Phase 2 passed!
Gate 3 :  input 2 intergers.
3 654
Phase 3 passed!
```

（4） 阶段 4：递归调用和栈 phase_4(char *input)；

使用 gdb bomb 进入 gdb，在调用 phase_4 处

设置断点设置断点 break 117，执行 run，照操作提示依次输入内容，程序在断点处停下。

反汇编 phase_4 函数：disass phase_4， 阅读该函数的反汇编程序。

```
Breakpoint 1, main (argc=1, argv=0xffffd1d4) at bomb.c:117
117         phase_4(input);
(gdb) disass phase_4
Dump of assembler code for function phase_4:
   0x08049a90 <+0>:     push   %ebp
   0x08049a91 <+1>:     mov    %esp,%ebp
   0x08049a93 <+3>:     sub    $0x38,%esp
   0x08049a96 <+6>:     mov    0x8(%ebp),%eax
   0x08049a99 <+9>:     mov    %eax,-0x2c(%ebp)
   0x08049a9c <+12>:    mov    %gs:0x14,%eax
   0x08049aa2 <+18>:    mov    %eax,-0xc(%ebp)
   0x08049aa5 <+21>:    xor    %eax,%eax
   0x08049aa7 <+23>:    lea    -0x1c(%ebp),%eax
   0x08049aaa <+26>:    push   %eax
   0x08049aab <+27>:    lea    -0x20(%ebp),%eax
   0x08049aae <+30>:    push   %eax
   0x08049aaf <+31>:    push   $0x804a360
   0x08049ab4 <+36>:    push   -0x2c(%ebp)
   0x08049ab7 <+39>:    call   0x80490d0 <__isoc99_sscanf@plt>
   0x08049abc <+44>:    add    $0x10,%esp
   0x08049abf <+47>:    mov    %eax,-0x18(%ebp)
   0x08049ac2 <+50>:    cmpl   $0x2,-0x18(%ebp)
   0x08049ac6 <+54>:    jne    0x8049ad7 <phase_4+71>
   0x08049ac8 <+56>:    mov    -0x20(%ebp),%eax
   0x08049acb <+59>:    test   %eax,%eax
   0x08049acd <+61>:    js     0x8049ad7 <phase_4+71>
   0x08049acf <+63>:    mov    -0x20(%ebp),%eax
   0x08049ad2 <+66>:    cmp    $0xe,%eax
   0x08049ad5 <+69>:    jle    0x8049adc <phase_4+76>
   0x08049ad7 <+71>:    call   0x8049816 <explode_bomb>
   0x08049adc <+76>:    movl   $0x7,-0x14(%ebp)
   0x08049ae3 <+83>:    mov    -0x20(%ebp),%eax
   0x08049ae6 <+86>:    sub    $0x4,%esp
   0x08049ae9 <+89>:    push   $0xe
   0x08049aeb <+91>:    push   $0x0
   0x08049aed <+93>:    push   %eax
   0x08049aee <+94>:    call   0x8049a1f <func4>
   0x08049af3 <+99>:    add    $0x10,%esp
   0x08049af6 <+102>:   mov    %eax,-0x10(%ebp)
   0x08049af9 <+105>:   mov    -0x10(%ebp),%eax
   0x08049afc <+108>:   cmp    -0x14(%ebp),%eax
   0x08049aff <+111>:   jne    0x8049b09 <phase_4+121>
   0x08049b01 <+113>:   mov    -0x1c(%ebp),%eax
   0x08049b04 <+116>:   cmp    %eax,-0x14(%ebp)
   0x08049b07 <+119>:   je     0x8049b0e <phase_4+126>
   0x08049b09 <+121>:   call   0x8049816 <explode_bomb>
   0x08049b0e <+126>:   nop
   0x08049b0f <+127>:   mov    -0xc(%ebp),%eax
--Type <RET> for more, q to quit, c to continue without paging--c
   0x08049b12 <+130>:   sub    %gs:0x14,%eax
   0x08049b19 <+137>:   je     0x8049b20 <phase_4+144>
   0x08049b1b <+139>:   call   0x8049070 <__stack_chk_fail@plt>
   0x08049b20 <+144>:   leave
   0x08049b21 <+145>:   ret
End of assembler dump.
```

```
0x08049ac2 <+50>:cmpl   $0x2,-0x18(%ebp)        ; 比较解析的参数个数和 2
0x08049ac6 <+54>:jne    0x8049ad7 <phase_4+71> ; 若不为 2，则引爆炸弹
```

根据以上两行代码可得输入的数必须为 2 个。

```
0x08049ac8 <+56>:mov    -0x20(%ebp),%eax
0x08049acb <+59>:test   %eax,%eax
0x08049acd <+61>:js     0x8049ad7 <phase_4+71>
0x08049acf <+63>:mov    -0x20(%ebp),%eax
0x08049ad2 <+66>:cmp    $0xe,%eax
0x08049ad5 <+69>:jle    0x8049adc <phase_4+76>
0x08049ad7 <+71>:call   0x8049816 <explode_bomb>
```

这段代码说明输入的数字如果为负数或者大于 14，都会引爆炸弹，故输入数字在 0-14。根据将 0 7 14 送入相应寄存器的操作，猜测要在 0-14 的二叉搜索树中找到输入的数字。

又由：

```
0x08049adc <+76>:movl   $0x7,-0x14(%ebp)        ; 将 7 存入局部变量
0x08049b04 <+116>:   cmp    %eax,-0x14(%ebp)    ; 比较第二个整数值和 7
```

可以看到最后要 eax 与-0x14(%ebp)处的值比较，这个值为 7，编码为 111，逆推得到第一个数应该为 14　第二个数为 7，字符串为 111 ，由此 phase_4 的炸弹破解完毕，经验证结果正确。

```
qr@qr-virtual-machine:~$ ./bomb
Input your Student ID :
U202215378
welcome  U202215378
You have 6 bombs to defuse!
Gate 1 :  input a string that meets the requirements.
Assemble Language
Phase 1 passed!
Gate 2 :  input six intergers that meets the requirements.
8 7 15 22 37 59
Phase 2 passed!
Gate 3 :  input 2 intergers.
3 654
Phase 3 passed!
Gate 4 :  input 2 intergers and a string.
14 7 111
```

（5） 阶段 5：指针和数组访问 phase_5(char *input)；

使用 gdb bomb 进入 gdb，在调用 phase_5 处设置断点设置断点 break 122，执行 run，照操作提示依次输入内容，程序在断点处停下。

```
Breakpoint 1, main (argc=1, argv=0xffffd1d4) at bomb.c:122
122         phase_5(input);
```

反汇编 phase_5 函数：disass phase_5， 阅读该函数的反汇编程序。

```
(gdb) disass phase_5
Dump of assembler code for function phase_5:
   0x08049b22 <+0>:     push   %ebp
   0x08049b23 <+1>:     mov    %esp,%ebp
   0x08049b25 <+3>:     sub    $0x38,%esp
   0x08049b28 <+6>:     mov    0x8(%ebp),%eax
   0x08049b2b <+9>:     mov    %eax,-0x2c(%ebp)
   0x08049b2e <+12>:    mov    %gs:0x14,%eax
   0x08049b34 <+18>:    mov    %eax,-0xc(%ebp)
   0x08049b37 <+21>:    xor    %eax,%eax
   0x08049b39 <+23>:    sub    $0xc,%esp
   0x08049b3c <+26>:    push   -0x2c(%ebp)
   0x08049b3f <+29>:    call   0x80495a3 <string_length>
   0x08049b44 <+34>:    add    $0x10,%esp
   0x08049b47 <+37>:    mov    %eax,-0x18(%ebp)
   0x08049b4a <+40>:    cmpl   $0x6,-0x18(%ebp)
   0x08049b4e <+44>:    je     0x8049b55 <phase_5+51>
   0x08049b50 <+46>:    call   0x8049816 <explode_bomb>
   0x08049b55 <+51>:    movl   $0x0,-0x1c(%ebp)
   0x08049b5c <+58>:    jmp    0x8049b84 <phase_5+98>
   0x08049b5e <+60>:    mov    -0x1c(%ebp),%edx
   0x08049b61 <+63>:    mov    -0x2c(%ebp),%eax
   0x08049b64 <+66>:    add    %edx,%eax
   0x08049b66 <+68>:    movzbl (%eax),%eax
   0x08049b69 <+71>:    movsbl %al,%eax
   0x08049b6c <+74>:    and    $0xf,%eax
   0x08049b6f <+77>:    movzbl 0x804c350(%eax),%eax
   0x08049b76 <+84>:    lea    -0x13(%ebp),%ecx
   0x08049b79 <+87>:    mov    -0x1c(%ebp),%edx
   0x08049b7c <+90>:    add    %ecx,%edx
   0x08049b7e <+92>:    mov    %al,(%edx)
   0x08049b80 <+94>:    addl   $0x1,-0x1c(%ebp)
   0x08049b84 <+98>:    cmpl   $0x5,-0x1c(%ebp)
   0x08049b88 <+102>:   jle    0x8049b5e <phase_5+60>
   0x08049b8a <+104>:   movb   $0x0,-0xd(%ebp)
   0x08049b8e <+108>:   sub    $0x8,%esp
   0x08049b91 <+111>:   push   $0x804a390
   0x08049b96 <+116>:   lea    -0x13(%ebp),%eax
   0x08049b99 <+119>:   push   %eax
   0x08049b9a <+120>:   call   0x80495cf <strings_not_equal>
   0x08049b9f <+125>:   add    $0x10,%esp
   0x08049ba2 <+128>:   test   %eax,%eax
   0x08049ba4 <+130>:   je     0x8049bab <phase_5+137>
   0x08049ba6 <+132>:   call   0x8049816 <explode_bomb>
   0x08049bab <+137>:   nop
   0x08049bac <+138>:   mov    -0xc(%ebp),%eax
   0x08049baf <+141>:   sub    %gs:0x14,%eax
   0x08049bb6 <+148>:   je     0x8049bbd <phase_5+155>
   0x08049bb8 <+150>:   call   0x8049070 <__stack_chk_fail@plt>
   0x08049bbd <+155>:   leave
   0x08049bbe <+156>:   ret
End of assembler dump.
```

```
0x08049b3c <+26>:push   -0x2c(%ebp)          ; 将参数推入栈中
0x08049b3f <+29>:call   0x80495a3 <string_length>  ; 调用字符串长度函数
0x08049b44 <+34>:add    $0x10,%esp           ; 栈指针回收
0x08049b47 <+37>:mov    %eax,-0x18(%ebp)     ; 将字符串长度保存到栈中
0x08049b4a <+40>:cmpl   $0x6,-0x18(%ebp)     ; 比较字符串长度是否为 6
0x08049b4e <+44>:je     0x8049b55 <phase_5+51>       ; 如果是则跳转
0x08049b50 <+46>:call   0x8049816 <explode_bomb>     ; 否则引爆炸弹
```

由上述代码可见可知在调用函数前，将-0x2c(%ebp) 的值送入堆栈，因此可以推断出输入的字符串的首地址存储在寄存器-0x2c(%ebp) 中。在执行完函数调用后，将-0x18(%ebp)的值，也即函数的返回值，与立即数 0x6 进行比较，若相等，则跳转到地址 0x8049b55，否则 顺序执行调用函数的指令，引爆炸弹。因此，输入的字符串的长 度必须为 6。

```
0x08049b5e <+60>: mov    -0x1c(%ebp),%edx
0x08049b61 <+63>: mov    -0x2c(%ebp),%eax
0x08049b64 <+66>: add    %edx,%eax
0x08049b66 <+68>: movzbl (%eax),%eax
0x08049b69 <+71>: movsbl %al,%eax
0x08049b6c <+74>: and    $0xf,%eax
0x08049b6f <+77>: movzbl 0x804c350(%eax),%eax
0x08049b76 <+84>: lea    -0x13(%ebp),%ecx
0x08049b79 <+87>: mov    -0x1c(%ebp),%edx
0x08049b7c <+90>: add    %ecx,%edx
0x08049b7e <+92>: mov    %al,(%edx)
0x08049b80 <+94>: addl   $0x1,-0x1c(%ebp)
0x08049b84 <+98>: cmpl   $0x5,-0x1c(%ebp)
0x08049b88 <+102>:jle    0x8049b5e <phase_5+60>
```

分析上述代码可知这是一个循环体。循环功能为将输入的字符串的每一位字符取出，并将其作为偏移量，在给定的数组中选择对应下标的字符，并将其存储起来。循环过程每次将%edx+%eax 的值赋给%eax，然后将%eax 按位与上 0xf，即可得到所需的拓展为字长度的对应字符的低四位值；再将%eax 的值加上 0x804c350 的值再赋给%eax，取出低位字节存储到地址(%edx)，再将,-0x1c(%ebp)的值加一，继续上述循环，直到,-0x1c(%ebp) 的值等于 0x5,退出循环。其中 0x804c350 的值的观察结果如下图。

```
(gdb) x/16x 0x804c350
0x804c350 <array.0>:    0x6d    0x61    0x64    0x75    0x69    0x65    0x72    0x73
0x804c358 <array.0+8>:  0x6e    0x66    0x6f    0x74    0x76    0x62    0x79    0x6c
(gdb) x/16c 0x804c350
0x804c350 <array.0>:    109 'm' 97 'a'  100 'd' 117 'u' 105 'i' 101 'e' 114 'r' 115 's'
0x804c358 <array.0+8>:  110 'n' 102 'f' 111 'o' 116 't' 118 'v' 98 'b'  121 'y' 108 'l'
```

上图可见 0x804c350 中存储的 16 个字节的 ASCII 码值和数组中存储的 16 个字节的内容。

```
0x08049b91 <+111>:push   $0x804a390
0x08049b96 <+116>:lea    -0x13(%ebp),%eax
0x08049b99 <+119>:push   %eax
0x08049b9a <+120>:call   0x80495cf <strings_not_equal>
```

观察上述汇编代码，可以观察到在执行<string_not_equal>函数前，将立即数 0x804a390 送入堆栈，观察 0x804a390 的值，观察结果如下图。

```
(gdb) x/s 0x804a390
0x804a390:      "bruins"
```

综合上述内容，可以得出所需的偏移值为 13、6、3、4、8、7（在图中标红的字母对应的序号）。因此输入的 字符串的每一个字符的低四位分别为 D、6、3、4、8、7。查看 ASCII 码表。

ASCII值			控制字符	ASCII值			控制字符	ASCII值			控制字符
二	十	十六		二	十	十六		二	十	十六	
0010 0000	32	20	SPACE(空格)	0100 0000	64	40	@	0110 0000	96	60	、
0010 0001	33	21	!	0100 0001	65	41	A	0110 0001	97	61	a
0010 0010	34	22	"	0100 0010	66	42	B	0110 0010	98	62	b
0010 0011	35	23	#	0100 0011	67	43	C	0110 0011	99	63	c
0010 0100	36	24	$	0100 0100	68	44	D	0110 0100	100	64	d
0010 0101	37	25	%	0100 0101	69	45	E	0110 0101	101	65	e
0010 0110	38	26	&	0100 0110	70	46	F	0110 0110	102	66	f
0010 0111	39	27	,	0100 0111	71	47	G	0110 0111	103	67	g
0010 1000	40	28	(	0100 1000	72	48	H	0110 1000	104	68	h
0010 1001	41	29	)	0100 1001	73	49	I	0110 1001	105	69	i
0010 1010	42	2A	*	0100 1010	74	4A	J	0110 1010	106	6A	j
0010 1011	43	2B	+	0100 1011	75	4B	K	0110 1011	107	6B	k
0010 1100	44	2C	,	0100 1100	76	4C	L	0110 1100	108	6C	l
0010 1101	45	2D	-	0100 1101	77	4D	M	0110 1101	109	6D	m
0010 1110	46	2E	.	0100 1110	78	4E	N	0110 1110	110	6E	n
0010 1111	47	2F	/	0100 1111	79	4F	O	0110 1111	111	6F	o
0011 0000	48	30	0	0101 0000	80	50	P	0111 0000	112	70	p
0011 0001	49	31	1	0101 0001	81	51	Q	0111 0001	113	71	q
0011 0010	50	32	2	0101 0010	82	52	R	0111 0010	114	72	r
0011 0011	51	33	3	0101 0011	83	53	X	0111 0011	115	73	s
0011 0100	52	34	4	0101 0100	84	54	T	0111 0100	116	74	t
0011 0101	53	35	5	0101 0101	85	55	U	0111 0101	117	75	u
0011 0110	54	36	6	0101 0110	86	56	V	0111 0110	118	76	v
0011 0111	55	37	7	0101 0111	87	57	W	0111 0111	119	77	w
0011 1000	56	38	8	0101 1000	88	58	X	0111 1000	120	78	x
0011 1001	57	39	9	0101 1001	89	59	Y	0111 1001	121	79	y
0011 1010	58	3A	:	0101 1010	90	5A	Z	0111 1010	122	7A	z
0011 1011	59	3B	;	0101 1011	91	5B	[	0111 1011	123	7B	{
0011 1100	60	3C	<	0101 1100	92	5C	/	0111 1100	124	7C	|
0011 1101	61	3D	=	0101 1101	93	5D	]	0111 1101	125	7D	}
0011 1110	62	3E	>	0101 1110	94	5E	^	0111 1110	126	7E	~
0011 1111	63	3F	?	0101 1111	95	5F	—	0111 1111	127	7F	DEL(删除)

查询可知对应的字符分别为“-” “6” “3” “4” “8” “7”。

由此，破解 phase_5 输入的字符串为-63487，经验证结果正确。

```
Gate 5 :  input  a string.
-63487
Phase 5 passed!
```

（6） 阶段 6：链表、结构、指针的访问 phase_6(char *input)；

使用 gdb bomb 进入 gdb，在调用 phase_6 处设置断点设置断点 break 127，执行 run，照操作提示依次输入内容，程序在断点处停下。反汇编 phase_6 函数：disass phase_6， 阅读该函数的反汇编程序。程序较长，不全部展示，此处为部分截图。

```
Breakpoint 1, main (argc=1, argv=0xffffd1d4) at bomb.c:127
127         phase_6(input);
(gdb) disass phase_6
Dump of assembler code for function phase_6:
   0x08049bbf <+0>:     push   %ebp
   0x08049bc0 <+1>:     mov    %esp,%ebp
   0x08049bc2 <+3>:     sub    $0x68,%esp
   0x08049bc5 <+6>:     mov    0x8(%ebp),%eax
   0x08049bc8 <+9>:     mov    %eax,-0x5c(%ebp)
   0x08049bcb <+12>:    mov    %gs:0x14,%eax
   0x08049bd1 <+18>:    mov    %eax,-0xc(%ebp)
   0x08049bd4 <+21>:    xor    %eax,%eax
   0x08049bd6 <+23>:    movl   $0x804c290,-0x40(%ebp)
   0x08049bdd <+30>:    sub    $0x8,%esp
   0x08049be0 <+33>:    lea    -0x3c(%ebp),%eax
   0x08049be3 <+36>:    push   %eax
   0x08049be4 <+37>:    push   -0x5c(%ebp)
   0x08049be7 <+40>:    call   0x804954f <read_six_numbers>
```

在上述截图中调用了<read_six_numbers>函数，可以看出所需要输入的字符串为 6 个数字。

```
8049bf8 <+57>:     mov    -0x48(%ebp),%eax
8049bfb <+60>:     mov    -0x3c(%ebp,%eax,4),%eax
8049bff <+64>:     test   %eax,%eax
8049c01 <+66>:     jle    0x8049c0f <phase_6+80>
8049c03 <+68>:     mov    -0x48(%ebp),%eax
8049c06 <+71>:     mov    -0x3c(%ebp,%eax,4),%eax
8049c0a <+75>:     cmp    $0x6,%eax
8049c0d <+78>:     jle    0x8049c14 <phase_6+85>
8049c0f <+80>:     call   0x8049816 <explode_bomb>
```

```
0x08049c14 <+85>:    mov    -0x48(%ebp),%eax
0x08049c17 <+88>:    add    $0x1,%eax
0x08049c1a <+91>:    mov    %eax,-0x44(%ebp)
0x08049c1d <+94>:    jmp    0x8049c3a <phase_6+123>
0x08049c1f <+96>:    mov    -0x48(%ebp),%eax
0x08049c22 <+99>:    mov    -0x3c(%ebp,%eax,4),%edx
0x08049c26 <+103>:   mov    -0x44(%ebp),%eax
0x08049c29 <+106>:   mov    -0x3c(%ebp,%eax,4),%eax
0x08049c2d <+110>:   cmp    %eax,%edx
0x08049c2f <+112>:   jne    0x8049c36 <phase_6+119>
0x08049c31 <+114>:   call   0x8049816 <explode_bomb>
0x08049c36 <+119>:   addl   $0x1,-0x44(%ebp)
0x08049c3a <+123>:   cmpl   $0x5,-0x44(%ebp)
0x08049c3e <+127>:   jle    0x8049c1f <phase_6+96>
0x08049c40 <+129>:   addl   $0x1,-0x48(%ebp)
0x08049c44 <+133>:   cmpl   $0x5,-0x48(%ebp)
0x08049c48 <+137>:   jle    0x8049bf8 <phase_6+57>
0x08049c4a <+139>:   movl   $0x0,-0x48(%ebp)
0x08049c51 <+146>:   jmp    0x8049c89 <phase_6+202>
0x08049c53 <+148>:   mov    -0x40(%ebp),%eax
0x08049c56 <+151>:   mov    %eax,-0x4c(%ebp)
0x08049c59 <+154>:   movl   $0x1,-0x44(%ebp)
0x08049c60 <+161>:   jmp    0x8049c6f <phase_6+176>
0x08049c62 <+163>:   mov    -0x4c(%ebp),%eax
0x08049c65 <+166>:   mov    0x8(%eax),%eax
0x08049c68 <+169>:   mov    %eax,-0x4c(%ebp)
0x08049c6b <+172>:   addl   $0x1,-0x44(%ebp)
Type <RET> for more, q to quit, c to continue without paging--c
```

从地址 0x08049bf8 到地址 0x08049c0f 的汇编指令可以看出，在外层循环中，输入的每一个数字必须大于 0x0 且小于 0x6，否则会顺序执行 调用函数的指令，引爆炸弹。

在内层循环中，当前遍历到的每一个输入的数字与其后面输入的数字不能相同，否则会会顺序执行调用函数的指令，引爆炸弹。因此输入的六个数字任意两个都不能相等。综合以上分析，输入的六个数只能为 0x1、0x2、0x3、0x4、 0x5 和 0x6，并且两两不能相同。由此推测这 6 个数是一个链表中结点的顺序号（从 1 到 6）。按照输入的顺序号，将对应链表结点中的值形成一个数组。若该数组是按照降序排列的，则过关。

从 phase_6 反汇编代码中可知，该部分代码为双重循环。在每次外循环中，首先将立即数 x804c290 赋给-0x40(%ebp)，每一轮外循环中取出一个输入的数字。地址 0x08049c14 到地址 0x08049c48 的汇编代码为内循环，内循环的退出条件为循环次数达到取出的输入的数字-0x1，每次内循环就将-0x4c(%ebp)作为地址，赋给%eax。退出内层循环后，就将寄存器%eax 的值按取得的次序存放在一 起，然后判断外循环是否结束，若未结束，则跳转到地址 0x8048e11，否则跳转到地址 0x8049070。在 gdb 中，观察所有取到并保存到的数，如下图所示。

```
(gdb) x/x 0x804c290
0x804c290 <node1>:      0x00000119
(gdb) x/x *(0x804c290+0x8)
0x804c284 <node2>:      0x0000038b
(gdb) x/x *(*(0x804c290+0x8)+0x8)
0x804c278 <node3>:      0x00000142
(gdb) x/x *(*(*(0x804c290+0x8)+0x8)+0x8)
0x804c26c <node4>:      0x00000079
(gdb) x/x *(*(*(*(0x804c290+0x8)+0x8)+0x8)+0x8)
0x804c260 <node5>:      0x00000338
(gdb) x/x *(*(*(*(*(0x804c290+0x8)+0x8)+0x8)+0x8)+0x8)
0x804c254 <node6>:      0x00000210
(gdb)
```

据分析，六个结点中的数要降序排列，故顺序应为 2 5 6 3 1 4，由此破解 phase_6 输入的六个数为 2 5 6 3 1 4，经验证结果正确。至此成功拆除所有炸弹。

```
qr@qr-virtual-machine:~$ ./bomb
Input your Student ID :
U202215378
welcome  U202215378
You have 6 bombs to defuse!
Gate 1 :  input a string that meets the requirements.
Assemble Language
Phase 1 passed!
Gate 2 :  input six intergers that meets the requirements.
8 7 15 22 37 59
Phase 2 passed!
Gate 3 :  input 2 intergers.
3 654
Phase 3 passed!
Gate 4 :  input 2 intergers and a string.
14 7 111
Phase 4 passed!
Gate 5 :  input  a string.
-63487
Phase 5 passed!
Gate 6 :  input 6 intergers.
2 5 6 3 1 4
Phase 6 passed!
Congratulations! You have passed all the tests!
```

四、体会

在本次拆除二进制炸弹的实验中，我深刻体会到了逆向工程和调试技能的重要性。通过使用课程中学到的知识，我成功地拆除了一个由多个阶段组成的二进制炸弹。在实验过程中，我不仅使用了 disass 指令生成反汇编文件，还通过 gdb 调试器对程序进行了断点设置、跟踪和单步运行，同时查看了内存单元的内容和寄存器的状态，这些都为我理解程序的运行逻辑和解决问题提供了重要的帮助。

在分析反汇编文件的过程中，我结合使用了 C 语言对程序功能的模拟，并绘制了流程图，这有助于我更清晰地理解程序的执行过程和逻辑。特别是在破解密码字符串的过程中，我通过模拟程序的行为，成功地推导出了正确的密码。这个过程不仅增强了我对汇编语言原理和机器级表示的理解，也锻炼了我的逆向工程技能。

除了技术上的收获，我认为这次任务的形式比之前的任务更为新颖有趣，完成任务的过程有种逐步通关游戏的成就感。刚开始拿到任务我很无措，最简单的 phase_1 几乎用了我第一次课的所有时间才破解，但是第一次突破之后就掌握了一些固定的方法与技巧，让我面对后面的关卡更加有信心。即使面对困难和挑战，我也不会轻易放弃，而是努力追踪问题的根源，掌握调试器的使用方法，并逐步解决问题。这种探索的过程让我收获了更多的知识和经验，也增强了我解决问题的能力。

总的来说，本次实验不仅加深了我对计算机系统基础知识的理解，也培养了我解决问题和探索未知领域的能力。通过拆除二进制炸弹，我不仅获得了技术上的收获，也感受到了解决问题的乐趣和成就感，这将对我的学习和职业发展产生积极的影响。