Lab2 实验报告

一、实验目的

通过二进制炸弹实验，增强阅读理解汇编语言及其机器码表示的能力，熟悉 GDB 调试工具的使用。

二、实验原理

二进制炸弹是一个可执行文件，包含六个关卡。每个关卡都需要输入一个特定的字符串作为拆弹密码。如果输入正确则拆除炸弹，进入下一关；否则引爆炸弹，需要从第一关重新开始。通过反汇编和逆向工程可以确定每个关卡的字符串密码，从而拆除炸弹。

三、实验过程

(一) 反汇编

从 elearning 上下载以学号命名的可执行文件到 Lab2 文件夹下，通过反汇编指令 objdump -d 16307130293，得到该程序完整的汇编代码，因为在终端阅读非常不方便，就将其输出到一个 txt 文件中：objdump -d 16307130293>assembly.txt。

(二) 查看各文件并做好调试准备

- 文件：

  16307130293.txt：反汇编得到的代码文件，是拆除炸弹需要研究的主要文件

  16307130293：炸弹程序，可以运行程序拆除六关炸弹，也可以通过 gdb 调试器设置断点、单步运行、查看寄存器和内存状态，推测和验证程序行为

  大致浏览反汇编代码，发现整个程序共有六关，phase_1 到 phase_6，代码中还包括初始化函数、输入输出函数、main 函数、字符串比较、炸弹等其它功能函数。每一关如果输入错误，程序都会运行到 explode_bomb()函数，为避免调试时引爆炸弹，需要每次在炸弹函数之前设置断点，即 b explode_bomb。

- gdb 调试环境

  通过 gdb 16307130293 的指令进入 gdb 调试环境；

  设置断点：b phase_1；b explode_bomb；

  运行程序：run，报错：无法执行，权限不够

```
(gdb) run
Starting program: /home/qjw/Lab2/16307130293
/bin/bash: /home/qjw/Lab2/16307130293: 权限不够
/bin/bash: 第 0 行: exec: /home/qjw/Lab2/16307130293: 无法执行: 权限不够
During startup program exited with code 126.
```

```
qjw@qjw-VirtualBox:~/Lab2$ ls -l
总用量 72
-rw-rw-r-- 1 qjw qjw 17938  4月 16 00:40 16307130293
-rw-rw-r-- 1 qjw qjw 52919  4月 16 19:40 assembly.txt
qjw@qjw-VirtualBox:~/Lab2$ chmod a+x 16307130293
qjw@qjw-VirtualBox:~/Lab2$ ls -l 16307130293
-rwxrwxr-x 1 qjw qjw 17938  4月 16 00:40 16307130293
```

通过 ls -l 查看权限，果然可执行文件只有读和写的权限，没有执行的权限，使用 chomod a+x 16307130293 语句增加执行权限即可。

```
(gdb) run
Starting program: /home/qjw/Lab2/16307130293
------------------------------------------------------------------------
Welcome to fudan ics2018 binary bomb.
You have 6 phases with which to blow yourself up.
Each bomb phase tests a diffrenet aspect of machine language programs:
  Phase 1: string comparision
  Phase 2: loops
  Phase 3: conditionals/switches
  Phase 4: recursive calls and the stack discipline
  Phase 5: pointers/others
  Phase 6: linked lists/pointers/structs
Have a nice day!
------------------------------------------------------------------------
-->>> phase 1 <<<--
```

(三) 炸弹拆除

第一关

```
000000000040137f <phase_1>:
  40137f:       55                      push   %rbp
  401380:       48 89 e5                mov    %rsp,%rbp
  401383:       48 83 ec 10             sub    $0x10,%rsp
  401387:       48 89 7d f8             mov    %rdi,-0x8(%rbp)
  40138b:       48 8b 45 f8             mov    -0x8(%rbp),%rax
  40138f:       be 20 32 60 00          mov    $0x603220,%esi//0x603220中内容放入%esi，作为第2个参数
  401394:       48 89 c7                mov    %rax,%rdi//用户输入的内容放入%rdi,作为第1个参数
  401397:       e8 31 00 00 00          callq  4013cd <strings_equal>//调用字符串比较函数判等
  40139c:       85 c0                   test   %eax,%eax//test相当于与运算，判断上一步是否返回1
  40139e:       75 11                   jne    4013b1 <phase_1+0x32>//非零跳转，即如果上一步条件码为1则跳过炸弹
  4013a0:       b8 00 00 00 00          mov    $0x0,%eax
  4013a5:       e8 0e 00 00 00          callq  4013b8 <explode_bomb>//否则引爆炸弹！
  4013aa:       b8 00 00 00 00          mov    $0x0,%eax
  4013af:       eb 05                   jmp    4013b6 <phase_1+0x37>
  4013b1:       b8 01 00 00 00          mov    $0x1,%eax
  4013b6:       c9                      leaveq
  4013b7:       c3                      retq
```

程序流程：

1、取用户输入的内容

2、取内存地址 0x603220 处的内容

3、比较两者的值，如果相等则成功跳过炸弹进入下一关，否则引爆炸弹

拆弹思路：

首先看到 string_equal()函数猜测是判断两个字符串是否相等的函数，通过查看相应代码得知该函数在相等时返回 1，不等返回 0。然后向上检查传入的参数是谁，注意到立即数的存在，立刻在 gdb 中使用 x/s 0x603220 查看其内容，发现是字符串"Poipoipoi!"另外一个参数是从%rdi 中获取的，因为对程序不够熟悉，不知道%rdi 中存有什么，结合上面 main 函数中 sscanf()的调用，以及调试器的使用得知其中存有的是指向输入字符串的指针，也即字符串的地址。

```
(gdb) x/s 0x603220
0x603220 <phase1_string>:       "Poipoipoi!"
```

```
(gdb) i r rdi
rdi            0x7fffffffdd40   140737488346432
(gdb) x/s 0x7fffffffdd40
0x7fffffffdd40: "Poipoipoi!"
```

综上可知，只要输入与 0x603220 内容相同的字符串，即可匹配成功，跳过炸弹！

```
-->>> phase 1 <<<--
Poipoipoi!
Phase 1 defused, How about the next one?
-->>> phase 2 <<<--
```

第二关

```
00000000004012da <phase_2>:
  4012da:       55                      push   %rbp
  4012db:       48 89 e5                mov    %rsp,%rbp
  4012de:       48 83 ec 20             sub    $0x20,%rsp
  4012e2:       48 89 7d e8             mov    %rdi,-0x18(%rbp)
  4012e6:       c7 45 fc 01 00 00 00    movl   $0x1,-0x4(%rbp)
  4012ed:       48 8b 45 e8             mov    -0x18(%rbp),%rax
  4012f1:       8b 10                   mov    (%rax),%edx//输入的内容放入edx
  4012f3:       8b 05 1f 1f 20 00       mov    0x201f1f(%rip),%eax//将0x603218的内容（2）放入%eax   # 603218 <phase2_ini>
  4012f9:       39 c2                   cmp    %eax,%edx//比较%edx,%eax
  4012fb:       74 11                   je     40130e <phase_2+0x34>//如果相等跳过炸弹
  4012fd:       b8 00 00 00 00          mov    $0x0,%eax
  401302:       e8 b1 00 00 00          callq  4013b8 <explode_bomb>//否则引爆
  401307:       b8 00 00 00 00          mov    $0x0,%eax
  40130c:       eb 6f                   jmp    40137d <phase_2+0xa3>//爆炸后无条件跳转到最后函数返回
  40130e:       c7 45 fc 01 00 00 00    movl   $0x1,-0x4(%rbp)//将1存入(%rbp-4)指向的地址
  401315:       eb 5b                   jmp    401372 <phase_2+0x98>//无条件跳转(如果i<=4,返回)
  401317:       8b 45 fc                mov    -0x4(%rbp),%eax//将(%rbp-4)内容存入%eax(i)
  40131a:       48 98                   cltq   //%eax符号拓展到%rax
  40131c:       48 8d 14 85 00 00 00    lea    0x0(,%rax,4),%rdx//将%rdx值改为%rax*4(4i)
  401323:       00

  401324:       48 8b 45 e8             mov    -0x18(%rbp),%rax//输入的内容&a[1]放入%rax
  401328:       48 01 d0                add    %rdx,%rax//加上%rdx &a[1]+4i
  40132b:       8b 00                   mov    (%rax),%eax//eax=a[&a[1]+4i]=a[i+1]
  40132d:       8b 55 fc                mov    -0x4(%rbp),%edx//将%rbp-4的内容存入%edx(i)
  401330:       48 63 d2                movslq %edx,%rdx//将做了符号拓展的双字传送到四字
  401333:       48 83 ea 01             sub    $0x1,%rdx//%rdx内容减1(i-1)
  401337:       48 8d 0c 95 00 00 00    lea    0x0(,%rdx,4),%rcx//将%rcx值改为%rdx*4=4(i-1)
  40133e:       00

  40133f:       48 8b 55 e8             mov    -0x18(%rbp),%rdx//输入的内容&a[1]存入%rdx
  401343:       48 01 ca                add    %rcx,%rdx//加上rcx的内容&a[1]+4*(i-1)
  401346:       8b 0a                   mov    (%rdx),%ecx//*(&a[1]+4*(i-1))=a[i]
  401348:       8b 55 fc                mov    -0x4(%rbp),%edx//i
  40134b:       8d 72 01                lea    0x1(%rdx),%esi//i+1
  40134e:       8b 55 fc                mov    -0x4(%rbp),%edx
  401351:       83 c2 01                add    $0x1,%edx
  401354:       0f af d6                imul   %esi,%edx
  401357:       01 ca                   add    %ecx,%edx//(i+1)^2+a[i]
  401359:       39 d0                   cmp    %edx,%eax//比较%eax(a[i+1])和%edx即(i+1)^2+a[i]内容
  40135b:       74 11                   je     40136e <phase_2+0x94>//相等跳过炸弹，自增
  40135d:       b8 00 00 00 00          mov    $0x0,%eax
  401362:       e8 51 00 00 00          callq  4013b8 <explode_bomb>//否则引爆
  401367:       b8 00 00 00 00          mov    $0x0,%eax
  40136c:       eb 0f                   jmp    40137d <phase_2+0xa3>//引爆则退出
  40136e:       83 45 fc 01             addl   $0x1,-0x4(%rbp)//%rbp-4指向内容++(i++)
  401372:       83 7d fc 04             cmpl   $0x4,-0x4(%rbp)//将%rbp-4指向的内容与4做比较
  401376:       7e 9f                   jle    401317 <phase_2+0x3d>//M[R[rbp-4]]<=4(i<=4)时跳转回去
  401378:       b8 01 00 00 00          mov    $0x1,%eax//否则退出循环？
  40137d:       c9                      leaveq
  40137e:       c3                      retq
```

程序流程：

1. 读入用户输入的数据
2. 判断第一个数是不是等于储存在 0x603218 的数据，不等引爆，相等进入循环
3. 做 4 次循环，判断第 i 次循环是否满足 a[i+1]-a[i]= (i+1)^2（i 从 1 到 4）
   一旦不等跳出循环，引爆炸弹，否则执行完以后进入下一关

拆弹思路：

首先很容易看出这是一个循环，分为三部分：初始化、循环体和循环条件。有了第一题的经验，很容易得知，初始化是规定输入的第一个数要与储存在 0x603218 的数据相等，通过 x/s 0x603218 查看得知是 2。后面进入循环，找到循环变量 i(储存在%rbp-4)中，然后知道循环体要执行 4 次，难点在于中间代码的理解，通过阅读分析寄存器传送指令求得递归公式，得到需要输入的 5 个数：2 6 15 31 56。

```
-->>> phase 2 <<<--
2 6 15 31 56
That's number 2. Keep going!
-->>> phase 3 <<<--
```

第三关

```
00000000004011f9 <phase_3>:
  4011f9:       55                      push   %rbp
  4011fa:       48 89 e5                mov    %rsp,%rbp
  4011fd:       53                      push   %rbx
  4011fe:       48 83 ec 18             sub    $0x18,%rsp
  401202:       89 7d ec                mov    %edi,-0x14(%rbp)//一参a
  401205:       48 89 75 e0             mov    %rsi,-0x20(%rbp)//二参b
  401209:       89 55 e8                mov    %edx,-0x18(%rbp)//三参c
  40120c:       83 7d ec 08             cmpl   $0x8,-0x14(%rbp)//比较a和8
  401210:       0f 87 84 00 00 00       ja     40129a <phase_3+0xa1>//无符号大于则引爆，即要求：第一个参数a<=8
  401216:       8b 45 ec                mov    -0x14(%rbp),%eax//第一个参数a传给%eax
  401219:       48 98                   cltq   //%eax符号拓展到%rax
  40121b:       ba 01 00 00 00          mov    $0x1,%edx
  401220:       48 89 d3                mov    %rdx,%rbx
  401223:       89 c1                   mov    %eax,%ecx
  401225:       48 d3 e3                shl    %cl,%rbx//%rbx左移a位，即10...0
  401228:       48 89 d8                mov    %rbx,%rax
  40122b:       48 89 c2                mov    %rax,%rdx
  40122e:       81 e2 c0 01 00 00       and    $0x1c0,%edx//1000 0000&1 1100 0000，即取高位
  401234:       48 85 d2                test   %rdx,%rdx//%rdx是否为0/1
  401237:       75 47                   jne    401280 <phase_3+0x87>//非零跳转（a=6/7/8）
  401239:       48 89 c2                mov    %rax,%rdx
  40123c:       83 e2 38                and    $0x38,%edx//0011 1000&10...0，中三位
  40123f:       48 85 d2                test   %rdx,%rdx
  401242:       75 22                   jne    401266 <phase_3+0x6d>//非零跳转(a=3/4/5)
  401244:       83 e0 07                and    $0x7,%eax//0111&10...0，低三位
  401247:       48 85 c0                test   %rax,%rax
  40124a:       74 4e                   je     40129a <phase_3+0xa1>//相等跳转（a!=0/1/2），引爆
  40124c:       81 7d e8 09 03 00 00    cmpl   $0x309,-0x18(%rbp)//a=0/1/2跳转到此处，比较c和*0x309(777)

  401253:       74 56                   je     4012ab <phase_3+0xb2>//相等跳转
  401255:       b8 00 00 00 00          mov    $0x0,%eax
  40125a:       e8 59 01 00 00          callq  4013b8 <explode_bomb>//否则引爆
  40125f:       b8 00 00 00 00          mov    $0x0,%eax
  401264:       eb 6d                   jmp    4012d3 <phase_3+0xda>//无条件跳转
  401266:       81 7d e8 78 03 00 00    cmpl   $0x378,-0x18(%rbp)//a=3/4/5跳转到此处，比较c和*0x378(888)
  40126d:       74 3f                   je     4012ae <phase_3+0xb5>//相等跳转
  40126f:       b8 00 00 00 00          mov    $0x0,%eax
  401274:       e8 3f 01 00 00          callq  4013b8 <explode_bomb>//否则引爆
  401279:       b8 00 00 00 00          mov    $0x0,%eax
  40127e:       eb 53                   jmp    4012d3 <phase_3+0xda>//无条件跳转
  401280:       81 7d e8 e7 03 00 00    cmpl   $0x3e7,-0x18(%rbp)//a=6/7/8跳转到此处，比较c和*0x3e7(999)
  401287:       74 28                   je     4012b1 <phase_3+0xb8>//相等跳转
  401289:       b8 00 00 00 00          mov    $0x0,%eax
  40128e:       e8 25 01 00 00          callq  4013b8 <explode_bomb>//否则引爆
  401293:       b8 00 00 00 00          mov    $0x0,%eax
  401298:       eb 39                   jmp    4012d3 <phase_3+0xda>//无条件跳转
  40129a:       b8 00 00 00 00          mov    $0x0,%eax
  40129f:       e8 14 01 00 00          callq  4013b8 <explode_bomb>//引爆
  4012a4:       b8 00 00 00 00          mov    $0x0,%eax
  4012a9:       eb 28                   jmp    4012d3 <phase_3+0xda>
  4012ab:       90                      nop//不操作
  4012ac:       eb 04                   jmp    4012b2 <phase_3+0xb9>
  4012ae:       90                      nop
  4012af:       eb 01                   jmp    4012b2 <phase_3+0xb9>
  4012b1:       90                      nop
  4012b2:       48 8b 45 e0             mov    -0x20(%rbp),%rax//b
  4012b6:       0f b6 00                movzbl (%rax),%eax//将做了零扩展的字节传送到双字，把b指向的内容传给%eax
  4012b9:       3c 63                   cmp    $0x63,%al//比较%rax即b指向的内容和0x63
  4012bb:       74 11                   je     4012ce <phase_3+0xd5>//相等跳转
  4012bd:       b8 00 00 00 00          mov    $0x0,%eax
  4012c2:       e8 f1 00 00 00          callq  4013b8 <explode_bomb>//否则引爆

  4012c7:       b8 00 00 00 00          mov    $0x0,%eax
  4012cc:       eb 05                   jmp    4012d3 <phase_3+0xda>//无条件跳转
  4012ce:       b8 01 00 00 00          mov    $0x1,%eax
  4012d3:       48 83 c4 18             add    $0x18,%rsp
  4012d7:       5b                      pop    %rbx
  4012d8:       5d                      pop    %rbp
  4012d9:       c3                      retq
```

程序流程：（参数 1：a；参数 2:b；参数 3：c）

1. 调用 sscanf()函数读取输入的三个参数
2. 比较参数一与 8 的大小，大于 8 引爆
3. 根据参数一 a 的值来确定跳转地址，并进行后续的比较

如果 a=6/7，比较 c 和 999，相等跳转，比较 b 指向的内容和’c’，相等返回，否则引爆
如果 a=3/4/5，比较 c 和 888，相等跳转，比较 b 指向的内容和’c’，相等返回，否则引爆
如果 a！=0/1/2 引爆，相等比较 c 和 777，比较 b 指向的内容和’c’，相等返回否则引爆

拆弹思路：

其实一开始并没有看出来这是一个 switch 语句，可能是还没学不熟悉，就看到满屏的跳啊跳的。只能一行一行地阅读代码，结合 gdb 单步执行非常方便地知道程序运行地步骤，发现是根据第一个输入的数 a 来判断跳到哪里的，然后不同的 a 对应不同的 c，最后都要进行统一 b 的判等。发现答案不唯一，有好多种组合：

a	b	c
8	c	999
7	c	999
6	c	999
5	c	888
4	c	888
3	c	888
2	c	777
1	c	777
0	c	777

这些组合经实验都成功拆除炸弹

```
-->>> phase 3 <<<--
0 c 777
Halfway there!
-->>> phase 4 <<<--
```

第四关

```
0000000000401178 <phase_4>:
  401178:       55                      push   %rbp
  401179:       48 89 e5                mov    %rsp,%rbp
  40117c:       48 83 ec 10             sub    $0x10,%rsp
  401180:       89 7d fc                mov    %edi,-0x4(%rbp)//输入的数字x
  401183:       8b 45 fc                mov    -0x4(%rbp),%eax
  401186:       89 c7                   mov    %eax,%edi//x
  401188:       e8 28 00 00 00          callq  4011b5 <func4>
  40118d:       8b 15 4d 1f 20 00       mov    0x201f4d(%rip),%edx//1        # 6030e0 <phase4_int>
  401193:       39 d0                   cmp    %edx,%eax//比较S[x]和1
  401195:       75 06                   jne    40119d <phase_4+0x25>//不等引爆
  401197:       83 7d fc 1e             cmpl   $0x1e,-0x4(%rbp)//比较x和0x1e
  40119b:       7e 11                   jle    4011ae <phase_4+0x36>//x<=0x1e返回
  40119d:       b8 00 00 00 00          mov    $0x0,%eax
  4011a2:       e8 11 02 00 00          callq  4013b8 <explode_bomb>//否则引爆
  4011a7:       b8 00 00 00 00          mov    $0x0,%eax
  4011ac:       eb 05                   jmp    4011b3 <phase_4+0x3b>
  4011ae:       b8 01 00 00 00          mov    $0x1,%eax
  4011b3:       c9                      leaveq
  4011b4:       c3                      retq
```

```
00000000004011b5 <func4>:
  4011b5:       55                      push   %rbp
  4011b6:       48 89 e5                mov    %rsp,%rbp
  4011b9:       53                      push   %rbx
  4011ba:       48 83 ec 18             sub    $0x18,%rsp
  4011be:       89 7d ec                mov    %edi,-0x14(%rbp)//x
  4011c1:       83 7d ec 01             cmpl   $0x1,-0x14(%rbp)//比较x和1
  4011c5:       74 06                   je     4011cd <func4+0x18>//相等跳转，直接结束，返回1
  4011c7:       83 7d ec 02             cmpl   $0x2,-0x14(%rbp)//比较x和2
  4011cb:       75 07                   jne    4011d4 <func4+0x1f>//不等跳转，相等直接结束，返回1
  4011cd:       b8 01 00 00 00          mov    $0x1,%eax
  4011d2:       eb 1e                   jmp    4011f2 <func4+0x3d>//无条件跳转，结束
  4011d4:       8b 45 ec                mov    -0x14(%rbp),%eax
  4011d7:       83 e8 01                sub    $0x1,%eax
  4011da:       89 c7                   mov    %eax,%edi
  4011dc:       e8 d4 ff ff ff          callq  4011b5 <func4>//x-1调用func4函数
  4011e1:       89 c3                   mov    %eax,%ebx
  4011e3:       8b 45 ec                mov    -0x14(%rbp),%eax
  4011e6:       83 e8 02                sub    $0x2,%eax
  4011e9:       89 c7                   mov    %eax,%edi//x-2调用func4函数
  4011eb:       e8 c5 ff ff ff          callq  4011b5 <func4>
  4011f0:       01 d8                   add    %ebx,%eax//两个相加的结果压栈,即f(x)=f(x-1)+f(x-2),斐波那契数列
  4011f2:       48 83 c4 18             add    $0x18,%rsp
  4011f6:       5b                      pop    %rbx
  4011f7:       5d                      pop    %rbp
  4011f8:       c3                      retq
```

phase_4 程序流程：

1. 读取用户输入的参数 x
2. 调用函数 func4()
3. 将返回结果与存储在 0x6030e0 的值相比，相等返回退出，否则引爆

func4 程序流程：

1. 比较传入的参数是否等于 1，相等直接返回 1，否则继续
2. 比较传入的参数是否等于 2，相等直接返回 1，否则继续
3. 如果传入的参数大于等于 2，减 1 后调用本身，减 2 后调用本身，再将两个值相加，即当前项等于前两项之和，可知这是一个递归函数，并且是特殊的斐波那契数列，递归终止条件是参数小于等于 1，返回值即是参数对应序号的项

拆弹思路：

这道题目其实一眼就看到函数自己调用自己，然后 f(x-1)+f(x-2)，就联想到了斐波那契数列，经过阅读代码，在每次递归调用地地方添加断点执行，发现 fun4()函数果然是计算第 n 项，首先是读进去一个数，然后以这个数作为实参调用 fun4()，将返回的值与存储在 0x6030e0 中的值做比较，并且输入的数<=0x1e，就返回，否则就引爆。利用 x/s 0x6030e0 查看内容竟然是 1，f[1]=f[2]=1，所以答案就是 1 或者 2.

```
-->>> phase 4 <<<--
1
So you got that one. Try this one.
-->>> phase 5 <<<--
```

```
-->>> phase 4 <<<--
2
So you got that one. Try this one.
-->>> phase 5 <<<--
```

第五关

```
0000000000401098 <phase_5>:
  401098:       55                      push   %rbp
  401099:       48 89 e5                mov    %rsp,%rbp
  40109c:       48 83 ec 30             sub    $0x30,%rsp
  4010a0:       48 89 7d d8             mov    %rdi,-0x28(%rbp)//输入的数x
  4010a4:       89 75 d4                mov    %esi,-0x2c(%rbp)//7
  4010a7:       64 48 8b 04 25 28 00    mov    %fs:0x28,%rax//把fs（段寄存器）段偏移0x28处的数据y(超长的一个负数)存入%rax
  4010ae:       00 00
  4010b0:       48 89 45 f8             mov    %rax,-0x8(%rbp)//y
  4010b4:       31 c0                   xor    %eax,%eax//判断两个数是否不等（一定返回0）
  4010b6:       c7 45 ec 00 00 00 00    movl   $0x0,-0x14(%rbp)//i初始值是0
  4010bd:       eb 70                   jmp    40112f <phase_5+0x97>//无条件跳转
  4010bf:       83 7d ec 02             cmpl   $0x2,-0x14(%rbp)//比较i和2
  4010c3:       7f 4d                   jg     401112 <phase_5+0x7a>//i>2跳转
  4010c5:       8b 45 ec                mov    -0x14(%rbp),%eax//i
  4010c8:       48 63 d0                movslq %eax,%rdx//将做了符号拓展的双字传送到四字i
  4010cb:       48 8b 45 d8             mov    -0x28(%rbp),%rax//x
  4010cf:       48 01 d0                add    %rdx,%rax//x+i
  4010d2:       0f b6 00                movzbl (%rax),%eax//M[x+i]
  4010d5:       89 c2                   mov    %eax,%edx
  4010d7:       83 e2 7b                and    $0x7b,%edx//(x+i)&0111 1011
  4010da:       8b 45 ec                mov    -0x14(%rbp),%eax//i
  4010dd:       48 98                   cltq   //eax符号拓展到rax
  4010df:       88 54 05 f0             mov    %dl,-0x10(%rbp,%rax,1)//与了以后的结果传送到%rbp+i-0x10
  4010e3:       8b 45 ec                mov    -0x14(%rbp),%eax//i
  4010e6:       48 98                   cltq

  4010e8:       0f b6 54 05 f0          movzbl -0x10(%rbp,%rax,1),%edx//与了以后的结果
  4010ed:       8b 45 ec                mov    -0x14(%rbp),%eax//i
  4010f0:       48 63 c8                movslq %eax,%rcx//i
  4010f3:       48 8b 45 d8             mov    -0x28(%rbp),%rax//x
  4010f7:       48 01 c8                add    %rcx,%rax//x+i
  4010fa:       0f b6 00                movzbl (%rax),%eax//x+i
  4010fd:       38 c2                   cmp    %al,%dl//比较与了以后和原来的
  4010ff:       75 2a                   jne    40112b <phase_5+0x93>//不等跳转到i++
  401101:       b8 00 00 00 00          mov    $0x0,%eax
  401106:       e8 ad 02 00 00          callq  4013b8 <explode_bomb>//否则爆炸
  40110b:       b8 00 00 00 00          mov    $0x0,%eax
  401110:       eb 50                   jmp    401162 <phase_5+0xca>//无条件跳转
  401112:       8b 45 ec                mov    -0x14(%rbp),%eax//i
  401115:       48 63 d0                movslq %eax,%rdx//将做了符号拓展的双字传送到四字i
  401118:       48 8b 45 d8             mov    -0x28(%rbp),%rax//x[0]
  40111c:       48 01 d0                add    %rdx,%rax//x[i]
  40111f:       0f b6 10                movzbl (%rax),%edx//将做了零拓展的字节传送到双字M[x[i]]
  401122:       8b 45 ec                mov    -0x14(%rbp),%eax//i
  401125:       48 98                   cltq   //%eax符号拓展到%rax
  401127:       88 54 05 f0             mov    %dl,-0x10(%rbp,%rax,1)//M[x[i]]送到%rbp+i-0x10指向的内容
  40112b:       83 45 ec 01             addl   $0x1,-0x14(%rbp)//i++
  40112f:       8b 45 ec                mov    -0x14(%rbp),%eax//i
  401132:       3b 45 d4                cmp    -0x2c(%rbp),%eax//比较i和7
  401135:       7c 88                   jl     4010bf <phase_5+0x27>//i<7跳回去,否则继续（也就是循环i从0到6）
  401137:       48 8d 45 f0             lea    -0x10(%rbp),%rax
  40113b:       be 00 31 60 00          mov    $0x603100,%esi
  401140:       48 89 c7                mov    %rax,%rdi

  401143:       e8 85 02 00 00          callq  4013cd <strings_equal>//判断0x603100(ics8888)和%rbp-10是否相等
  401148:       85 c0                   test   %eax,%eax//判断是否是1
  40114a:       75 11                   jne    40115d <phase_5+0xc5>//非零跳转
  40114c:       b8 00 00 00 00          mov    $0x0,%eax
  401151:       e8 62 02 00 00          callq  4013b8 <explode_bomb>//否则爆炸
  401156:       b8 00 00 00 00          mov    $0x0,%eax
  40115b:       eb 05                   jmp    401162 <phase_5+0xca>//无条件跳转
  40115d:       b8 01 00 00 00          mov    $0x1,%eax
  401162:       48 8b 55 f8             mov    -0x8(%rbp),%rdx//y
  401166:       64 48 33 14 25 28 00    xor    %fs:0x28,%rdx//异或判断是否不等
  40116d:       00 00
  40116f:       74 05                   je     401176 <phase_5+0xde>//相等跳转，结束
  401171:       e8 1a f5 ff ff          callq  400690 <__stack_chk_fail@plt>//否则溢出报错
  401176:       c9                      leaveq
  401177:       c3                      retq
```

程序流程：

1. 读入一个字符串 s
2. 计数变量 i 初始为 0，执行一个 7 次的循环，（i 从 0 到 6）
3. 当 i<=2 时，将 s[i]与 0x7b 即 0111 1011 做与运算，再利用 ascii 码转换成单个字符 i>2 时不变
4. 将变化后的字符串与之前的比较，如果相等引爆炸弹，不等继续

5. 调用 string_equal()函数，将变化后的字符串与 0x603100 储存的字符串（ics8888）做比较，相等则返回通过，否则引爆

拆弹思路：

开始看到代码是比较懵的，搞不懂函数的功能是什么。然后看到了大的循环得知需要输入一个 7 位的字符串，里面又对<=2 的情况单独处理，就先看前面<=2 的情况，发现是将 0x7b 作为掩码进行与运算，结果必须要跟原来输入的不一样，重组的字符串必须要跟 ics8888 一样。采用逆向工程的方法，找到 i,c,s 对应的 ascii 码：i 对应 0x69，即 0110 1001，是 s[0]&0111 1011=0110 1001，s[0]可以等于 0110 1101(m)，或者 1110 1001，或者 1110 1101，后面两种没有 ascii 码相对应的字符，所以第一位是 m；同理 s[1]等于 0110 0111(g)，s[2]等于 0111 0111(w)。所以应该输入 mgw8888！

```
--->>> phase 5 <<<--
mgw8888
Good work! On to the next...
--->>> phase 6 <<<--
```

第六关

```
0000000000400df5 <phase_6>:
  400df5:       55                      push   %rbp
  400df6:       48 89 e5                mov    %rsp,%rbp
  400df9:       48 83 ec 30             sub    $0x30,%rsp
  400dfd:       48 89 7d d8             mov    %rdi,-0x28(%rbp)//输入x
  400e01:       c7 45 ec 01 00 00 00    movl   $0x1,-0x14(%rbp)//1（i）
  400e08:       e9 f5 00 00 00          jmpq   400f02 <phase_6+0x10d>//无条件跳转
  400e0d:       83 7d ec 01             cmpl   $0x1,-0x14(%rbp)//比较i和1
  400e11:       75 71                   jne    400e84 <phase_6+0x8f>//不等跳转(只有第一次不跳转)
  400e13:       bf 10 00 00 00          mov    $0x10,%edi//10
  400e18:       e8 e3 f8 ff ff          callq  400700 <malloc@plt>
  400e1d:       48 89 45 f0             mov    %rax,-0x10(%rbp)
  400e21:       8b 45 ec                mov    -0x14(%rbp),%eax
  400e24:       48 98                   cltq
  400e26:       48 83 e8 01             sub    $0x1,%rax//y-1
  400e2a:       48 8d 14 85 00 00 00    lea    0x0(,%rax,4),%rdx//4(y-1)
  400e31:       00
  400e32:       48 8b 45 d8             mov    -0x28(%rbp),%rax//x
  400e36:       48 01 d0                add    %rdx,%rax//x+4(y-1)
  400e39:       8b 00                   mov    (%rax),%eax//M[x+4(y-1)]
  400e3b:       83 e8 01                sub    $0x1,%eax//-1
  400e3e:       48 98                   cltq
  400e40:       8b 14 85 00 32 60 00    mov    0x603200(,%rax,4),%edx//0x603200为起始地址
  400e47:       48 8b 45 f0             mov    -0x10(%rbp),%rax//z
  400e4b:       89 10                   mov    %edx,(%rax)
  400e4d:       8b 45 ec                mov    -0x14(%rbp),%eax//y
  400e50:       48 98                   cltq
  400e52:       48 83 e8 01             sub    $0x1,%rax
  400e56:       48 8d 14 85 00 00 00    lea    0x0(,%rax,4),%rdx
  400e5d:       00
  400e5e:       48 8b 45 d8             mov    -0x28(%rbp),%rax
  400e62:       48 01 d0                add    %rdx,%rax
  400e65:       8b 10                   mov    (%rax),%edx
  400e67:       48 8b 45 f0             mov    -0x10(%rbp),%rax
  400e6b:       89 50 04                mov    %edx,0x4(%rax)
  400e6e:       48 8b 45 f0             mov    -0x10(%rbp),%rax
  400e72:       48 c7 40 08 00 00 00    movq   $0x0,0x8(%rax)
  400e79:       00
```

```
400e7a:       48 8b 45 f0             mov    -0x10(%rbp),%rax
400e7e:       48 89 45 f8             mov    %rax,-0x8(%rbp)
400e82:       eb 7a                   jmp    400efe <phase_6+0x109>
400e84:       bf 10 00 00 00          mov    $0x10,%edi//跳转到这里
400e89:       e8 72 f8 ff ff          callq  400700 <malloc@plt>
400e8e:       48 89 c2                mov    %rax,%rdx
400e91:       48 8b 45 f8             mov    -0x8(%rbp),%rax
400e95:       48 89 50 08             mov    %rdx,0x8(%rax)
400e99:       48 8b 45 f8             mov    -0x8(%rbp),%rax
400e9d:       48 8b 40 08             mov    0x8(%rax),%rax
400ea1:       48 89 45 f8             mov    %rax,-0x8(%rbp)
400ea5:       48 8b 45 f8             mov    -0x8(%rbp),%rax
400ea9:       48 c7 40 08 00 00 00    movq   $0x0,0x8(%rax)
400eb0:       00
400eb1:       8b 45 ec                mov    -0x14(%rbp),%eax
400eb4:       48 98                   cltq
400eb6:       48 83 e8 01             sub    $0x1,%rax
400eba:       48 8d 14 85 00 00 00    lea    0x0(,%rax,4),%rdx
400ec1:       00
400ec2:       48 8b 45 d8             mov    -0x28(%rbp),%rax
400ec6:       48 01 d0                add    %rdx,%rax
400ec9:       8b 00                   mov    (%rax),%eax
400ecb:       83 e8 01                sub    $0x1,%eax
400ece:       48 98                   cltq
400ed0:       8b 14 85 00 32 60 00    mov    0x603200(,%rax,4),%edx
400ed7:       48 8b 45 f8             mov    -0x8(%rbp),%rax
400edb:       89 10                   mov    %edx,(%rax)
400edd:       8b 45 ec                mov    -0x14(%rbp),%eax
400ee0:       48 98                   cltq
400ee2:       48 83 e8 01             sub    $0x1,%rax
400ee6:       48 8d 14 85 00 00 00    lea    0x0(,%rax,4),%rdx
400eed:       00
400eee:       48 8b 45 d8             mov    -0x28(%rbp),%rax
400ef2:       48 01 d0                add    %rdx,%rax
400ef5:       8b 10                   mov    (%rax),%edx
400ef7:       48 8b 45 f8             mov    -0x8(%rbp),%rax
400efb:       89 50 04                mov    %edx,0x4(%rax)
400efe:       83 45 ec 01             addl   $0x1,-0x14(%rbp)//y++
400f02:       83 7d ec 06             cmpl   $0x6,-0x14(%rbp)//比较y和0x6
400f06:       0f 8e 01 ff ff ff       jle    400e0d <phase_6+0x18>//y<=6跳转回去,上面部分创建了一个链表
400f0c:       48 8b 45 f0             mov    -0x10(%rbp),%rax//进入第二个循环
400f10:       48 89 45 f8             mov    %rax,-0x8(%rbp)
400f14:       eb 31                   jmp    400f47 <phase_6+0x152>
400f16:       48 8b 45 f8             mov    -0x8(%rbp),%rax
400f1a:       8b 10                   mov    (%rax),%edx
400f1c:       48 8b 45 f8             mov    -0x8(%rbp),%rax
400f20:       48 8b 40 08             mov    0x8(%rax),%rax
400f24:       8b 00                   mov    (%rax),%eax
400f26:       39 c2                   cmp    %eax,%edx
400f28:       7d 11                   jge    400f3b <phase_6+0x146>//%edx>=%eax跳转，前一项必须大于后一项
400f2a:       b8 00 00 00 00          mov    $0x0,%eax
400f2f:       e8 84 04 00 00          callq  4013b8 <explode_bomb>//否则引爆
400f34:       b8 00 00 00 00          mov    $0x0,%eax
400f39:       eb 1e                   jmp    400f59 <phase_6+0x164>
400f3b:       48 8b 45 f8             mov    -0x8(%rbp),%rax
400f3f:       48 8b 40 08             mov    0x8(%rax),%rax
400f43:       48 89 45 f8             mov    %rax,-0x8(%rbp)
400f47:       48 8b 45 f8             mov    -0x8(%rbp),%rax
400f4b:       48 8b 40 08             mov    0x8(%rax),%rax
400f4f:       48 85 c0                test   %rax,%rax
400f52:       75 c2                   jne    400f16 <phase_6+0x121>//非零即%rax为1时跳转,一直到最后一个数为0时跳出
400f54:       b8 01 00 00 00          mov    $0x1,%eax
400f59:       c9                      leaveq
400f5a:       c3                      retq
```

程序流程

1. 读入用户输入的数字
2. 循环一将读入的 6 个数，根据输入的数找到内存地址
3. 将相应的地址存入节点，并按顺序串联成一个链表
4. 循环二判断是否前一节点的值大于后一节点的值，满足返回，否则引爆

拆弹思路

看到这么长的代码非常不想看，事实证明的确很绕。

因为输入的数必须大于等于 1，小于等于 6，并且由后面知这 6 个数互不相同，所以就是 123456，只是需要继续阅读确定一下顺序。

发现了立即数, 并且是 4*i 作为偏移量出现的, 所以怀疑是, 查找某一数字相应 0x603200 偏移量的内存中存储的内容，后面一番操作感觉是创建了一个链表，把这些查到的内容存入作为节点存入其中。还是没有对顺序做出限制。

继续往下看，就比较熟悉了，是一个比较，回溯发现是对链表中前后两项值的比较，

a[i]>=a[i]才不会引爆炸弹，所以我们需要做的就是先查到 0x603200 开始六个节点存储的值，4 111 14 0 12 5，标号依次为 123456，将这些数按照降序排列，111 14 12 5 4 0，对应得到序号的排列 2 3 5 6 1 4 即为密码。

```
(gdb) x/6 0x603200
0x603200 <phase6_int>:  4       111     14      0
0x603210 <phase6_int+16>:       12      5
```

```
-->>> phase 6 <<<--
2 3 5 6 1 4

Breakpoint 1, 0x0000000000400df9 in phase_6 ()
(gdb) c
Continuing.
Congratulations! You've defused the bomb!
[Inferior 1 (process 5505) exited with code 01]
```

（为验证第二层循环的排序功能设置断点执行循环）

```
-->>> phase 6 <<<--
2 3 5 6 1 4

Breakpoint 1, 0x0000000000400df9 in phase_6 ()
(gdb) i r edx eax
edx            0x14     20
eax            0x0      0
(gdb) c
Continuing.

Breakpoint 2, 0x0000000000400f28 in phase_6 ()
(gdb) i r edx eax
edx            0x6f     111
eax            0xe      14
(gdb) c
Continuing.

Breakpoint 2, 0x0000000000400f28 in phase_6 ()
(gdb) i r edx eax
edx            0xe      14
eax            0xc      12
(gdb) c
Continuing.

Breakpoint 2, 0x0000000000400f28 in phase_6 ()
(gdb) i r edx eax
edx            0xc      12
eax            0x5      5
(gdb) c
Continuing.

Breakpoint 2, 0x0000000000400f28 in phase_6 ()
(gdb) i r edx eax
edx            0x5      5
eax            0x4      4
(gdb) c
Continuing.

Breakpoint 2, 0x0000000000400f28 in phase_6 ()
(gdb) i r edx eax
edx            0x4      4
eax            0x0      0
(gdb) stepi
0x0000000000400f3b in phase_6 ()
```

四、实验结论

最终完整运行程序截图（phase_3,phase_4 答案不唯一，具体见上文分析）

```
------------------------------------------------------------------------
Welcome to fudan ics2018 binary bomb.
You have 6 phases with which to blow yourself up.
Each bomb phase tests a diffrenet aspect of machine language programs:
  Phase 1: string comparision
  Phase 2: loops
  Phase 3: conditionals/switches
  Phase 4: recursive calls and the stack discipline
  Phase 5: pointers/others
  Phase 6: linked lists/pointers/structs
Have a nice day!
------------------------------------------------------------------------
-->>> phase 1 <<<--
Poipoipoi!
Phase 1 defused, How about the next one?
-->>> phase 2 <<<--
2 6 15 31 56
That's number 2. Keep going!
-->>> phase 3 <<<--
7 c 999
Halfway there!
-->>> phase 4 <<<--
1
So you got that one. Try this one.
-->>> phase 5 <<<--
mgw8888
Good work! On to the next...
-->>> phase 6 <<<--
2 3 5 6 1 4
Congratulations! You've defused the bomb!
[Inferior 1 (process 6147) exited with code 01]
```

五、实验感悟

刚开始拿到文件反汇编出汇编代码以后，完全不知道如何下手，大致浏览了框架，理解了任务。然后学习了 gdb 的一些功能，包括设置断点、单步执行、查看寄存器和内存内容等，然后就开始正式拆除炸弹的正义之旅了！

前面几个函数的解决往往是看到某一个很特殊的操作之后想到可能是问题的关键，比如第一关中的立即数，对其存储内容进行查看后问题迎刃而解。做到后面代码变得非常长，除了看代码（有的真的看不懂），还结合了 gdb 的好用的功能，像打印寄存器内容，输出内存可以随时查看程序运行带来的变化，设断点单步执行的优点尤其体现在出现很多 jump 的地方，可以方便地知道程序地运行过程。后来越做越顺手，从一道题需要花费两三个小时到半个小时（除了最后一个，呵），真心觉得阅读汇编代码也变成了一件非常有趣的事情，再加上过关的趣味性和爆炸的不确定性，真的是爱上这款游戏了啊。

最后总结收获和不足，通过这次实验，我阅读汇编代码的能力明显提高，对于汇编语言中的字符比较、循环、条件语句、过程调用、指针和链表等相关知识有了更深刻的认识；熟练掌握了 gdb 调试器中重要的功能，并用于函数的理解；但是发现自己做题目有点耽误时间，就是看到题目死扣语句，非要弄懂差不多了才去尝试，然后用 gdb 去验证，这样确实可以提高阅读理解汇编代码的能力，而且自己突然弄明白函数功能真的很爽，但是不如直接结合 gdb 来得方便直观。代码中个别看似无用的语句还不是很理解其作用，猜测可能是起到占位的作用。做完本次 Bomblab 可以说是受益匪浅！