一、实验目的

1.理解 C 语言程序的机器级表示。
2.初步掌握 GDB 调试器的用法。
3.阅读 C 编译器生成的 x86-64 机器代码，理解不同控制结构生成的基本指令模式，过程的实现。

二、实验环境

1. Windows PowerShell（10.120.11.12）
2. Linux
3. Objdump 命令反汇编
4. GDB 调试工具
5. 。。。。。

三、实验内容

登录 bupt1 服务器，在 home 目录下可以找到 Evil 博士专门为你量身定制的一个 bomb，当运行时，它会要求你输入一个字符串，如果正确，则进入下一关，继续要求你输入下一个字符串；否则，炸弹就会爆炸，输出一行提示信息并向计分服务器提交扣分信息。因此，本实验要求你必须通过反汇编和逆向工程对 bomb 执行文件进行分析，找到正确的字符串来解除这个的炸弹。

本实验通过要求使用课程所学知识拆除一个“binary bombs”来增强对程序的机器级表示、汇编语言、调试器和逆向工程等方面原理与技能的掌握。 “binary bombs”是一个 Linux 可执行程序，包含了 5 个阶段（或关卡）。炸弹运行的每个阶段要求你输入一个特定字符串，你的输入符合程序预期的输入，该阶段的炸弹就被拆除引信；否则炸弹“爆炸”，打印输出 “BOOM!!!”。炸弹的每个阶段考察了机器级程序语言的一个不同方面，难度逐级递增。

为完成二进制炸弹拆除任务，需要使用 gdb 调试器和 objdump 来反汇编 bomb 文件，可以单步跟踪调试每一阶段的机器代码，也可以阅读反汇编代码，从中理解每一汇编语言代码的行为或作用，进而设法推断拆除炸弹所需的目标字符串。实验 2 的具体内容见实验 2 说明。

四、实验步骤及实验分析

建议按照：准备工作、阶段 1、阶段 2、…等来组织内容
各阶段需要有操作步骤、运行截图、分析过程的内容

1. 准备工作

解压文件

```
2022211414@bupt1:~$ tar -xvf bomb503.tar
bomb503/README
bomb503/bomb.c
bomb503/bomb
```

进入文件夹并查看文件

```
2022211414@bupt1:~$ cd bomb503
2022211414@bupt1:~/bomb503$ ll
total 40
-rwxr-xr-x 1 2022211414 students 31144 Oct 25 09:34 bomb*
-rw-r--r-- 1 2022211414 students  4069 Oct 25 09:34 bomb.c
-rw-r--r-- 1 2022211414 students    63 Oct 25 09:34 README
```

查看 bomb.c

```
2022211414@bupt1:~/bomb503$ cat bomb.c
/*************************************
```

发现 main 函数调用了 6 个 phase 函数，都进行了读入字符串，判断

（phase_1）

```
input = read_line();             /* Get input                   */
phase_1(input);                  /* Run the phase               */
phase_defused();                 /* Drat!  They figured it out!
                                  * Let me know how they did it. */
printf("Phase 1 defused. How about the next one?\n");
```

使用 objdump 工具查看反汇编

```
2022211414@bupt1:~/bomb503$ objdump -d bomb
```

观察代码发现每个 phase 函数都有调用 explode_bomb，在之后的调试中可以将其设为断点

（以 phase_1 为例）

```
0000000000400f2d <phase_1>:
  400f2d:       48 83 ec 08             sub    $0x8,%rsp
  400f31:       be 40 27 40 00          mov    $0x402740,%esi
  400f36:       e8 64 05 00 00          callq  40149f <strings_not_equal>
  400f3b:       85 c0                   test   %eax,%eax
  400f3d:       74 05                   je     400f44 <phase_1+0x17>
  400f3f:       e8 2f 08 00 00          callq  401773 <explode_bomb>
  400f44:       48 83 c4 08             add    $0x8,%rsp
  400f48:       c3                      retq
```

2. 阶段 1

gdb bomb 进入调试，先在爆炸前设置断点

```
(gdb) break explode_bomb
Breakpoint 1 at 0x401773
```

查看 phase_1 汇编

```
(gdb) disas phase_1
Dump of assembler code for function phase_1:
   0x0000000000400f2d <+0>:     sub    $0x8,%rsp
   0x0000000000400f31 <+4>:     mov    $0x402740,%esi
   0x0000000000400f36 <+9>:     callq  0x40149f <strings_not_equal>
   0x0000000000400f3b <+14>:    test   %eax,%eax
   0x0000000000400f3d <+16>:    je     0x400f44 <phase_1+23>
   0x0000000000400f3f <+18>:    callq  0x401773 <explode_bomb>
   0x0000000000400f44 <+23>:    add    $0x8,%rsp
   0x0000000000400f48 <+27>:    retq
End of assembler dump.
```

分析：入栈，mov 指令把 0x402740 处的值放入寄存器%esi，然后调用 strings_not_equal 判断是否相同，test 进行逻辑与运算，返回结果到%eax，相同即%eax 值为 0 则跳转出栈，不同则调用 explode_bomb

查看 0x402740 处的值（以字符串形式）

```
(gdb) x/s 0x402740
0x402740:       "Border relations with Canada have never been better."
```

运行，输入结果，阶段 1 炸弹解除

```
(gdb) run
Starting program: /students/2022211414/bomb503/bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Border relations with Canada have never been better.
Phase 1 defused. How about the next one?
```

保存答案到 answer.txt

```
2022211414@bupt1:~/bomb503$ vim answer.txt
```

```
Border relations with Canada have never been better.
~
~
:wq
```

3. 阶段 2

查看 phase_2 汇编

```
(gdb) disas phase_2
Dump of assembler code for function phase_2:
   0x0000000000400f49 <+0>:     push   %rbp
   0x0000000000400f4a <+1>:     push   %rbx
   0x0000000000400f4b <+2>:     sub    $0x28,%rsp
   0x0000000000400f4f <+6>:     mov    %fs:0x28,%rax
   0x0000000000400f58 <+15>:    mov    %rax,0x18(%rsp)
   0x0000000000400f5d <+20>:    xor    %eax,%eax
   0x0000000000400f5f <+22>:    mov    %rsp,%rsi
   0x0000000000400f62 <+25>:    callq  0x4017a9 <read_six_numbers>
   0x0000000000400f67 <+30>:    cmpl   $0x0,(%rsp)
   0x0000000000400f6b <+34>:    jns    0x400f72 <phase_2+41>
   0x0000000000400f6d <+36>:    callq  0x401773 <explode_bomb>
   0x0000000000400f72 <+41>:    mov    %rsp,%rbp
   0x0000000000400f75 <+44>:    mov    $0x1,%ebx
   0x0000000000400f7a <+49>:    mov    %ebx,%eax
   0x0000000000400f7c <+51>:    add    0x0(%rbp),%eax
   0x0000000000400f7f <+54>:    cmp    %eax,0x4(%rbp)
   0x0000000000400f82 <+57>:    je     0x400f89 <phase_2+64>
   0x0000000000400f84 <+59>:    callq  0x401773 <explode_bomb>
   0x0000000000400f89 <+64>:    add    $0x1,%ebx
   0x0000000000400f8c <+67>:    add    $0x4,%rbp
   0x0000000000400f90 <+71>:    cmp    $0x6,%ebx
   0x0000000000400f93 <+74>:    jne    0x400f7a <phase_2+49>
   0x0000000000400f95 <+76>:    mov    0x18(%rsp),%rax
   0x0000000000400f9a <+81>:    xor    %fs:0x28,%rax
   0x0000000000400fa3 <+90>:    je     0x400faa <phase_2+97>
   0x0000000000400fa5 <+92>:    callq  0x400b90 <__stack_chk_fail@plt>
   0x0000000000400faa <+97>:    add    $0x28,%rsp
   0x0000000000400fae <+101>:   pop    %rbx
   0x0000000000400faf <+102>:   pop    %rbp
--Type <RET> for more, q to quit, c to continue without paging--c
   0x0000000000400fb0 <+103>:   retq
End of assembler dump.
```

分析：%rbp 和%rbx 是被调用者保存寄存器，把值压栈保存。<+15>分配空间。

<+25>调用 read_six_numbers 函数。

<+30>比较(%rsp)值和 0，SF 为 0 则跳转，否则调用 explode_bomb，可知第一个数非负。

把%rsp(第一个数地址)放入%rbp，把%ebx 和%eax 置 1。

<+49>~<+74>相当于一个循环：把(%rbp)值+%ebx 值放入%eax，与下一个数比较，不相等则调用 explode_bomb，相等则跳转把%ebx 值+1，%rbp 存下一个数地址，比较%ebx 和 6，不相等则进入下一轮循环，否则退出循环。（%ebx 相当于一个循环次数计数器）

查看 read_six_numbers 汇编

```
(gdb) disas read_six_numbers
Dump of assembler code for function read_six_numbers:
   0x00000000004017a9 <+0>:     sub    $0x8,%rsp
   0x00000000004017ad <+4>:     mov    %rsi,%rdx
   0x00000000004017b0 <+7>:     lea    0x4(%rsi),%rcx
   0x00000000004017b4 <+11>:    lea    0x14(%rsi),%rax
   0x00000000004017b8 <+15>:    push   %rax
   0x00000000004017b9 <+16>:    lea    0x10(%rsi),%rax
   0x00000000004017bd <+20>:    push   %rax
   0x00000000004017be <+21>:    lea    0xc(%rsi),%r9
   0x00000000004017c2 <+25>:    lea    0x8(%rsi),%r8
   0x00000000004017c6 <+29>:    mov    $0x402a41,%esi
   0x00000000004017cb <+34>:    mov    $0x0,%eax
   0x00000000004017d0 <+39>:    callq  0x400c40 <__isoc99_sscanf@plt>
   0x00000000004017d5 <+44>:    add    $0x10,%rsp
   0x00000000004017d9 <+48>:    cmp    $0x5,%eax
   0x00000000004017dc <+51>:    jg     0x4017e3 <read_six_numbers+58>
   0x00000000004017de <+53>:    callq  0x401773 <explode_bomb>
   0x00000000004017e3 <+58>:    add    $0x8,%rsp
   0x00000000004017e7 <+62>:    retq
End of assembler dump.
```

分析：把字符串转为 6 个数字，存储在栈上一个整数数组中。

综上，推断出 6 个数满足 a[i+1]=a[i]+i+1

令第一个数为 1 的尝试

```
(gdb) b explode_bomb
Breakpoint 1 at 0x401773
(gdb) run
Starting program: /students/2022211414/bomb503/bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Border relations with Canada have never been better.
Phase 1 defused. How about the next one?
1 2 4 7 11 16
That's number 2.  Keep going!
```

阶段 2 炸弹解除

保存答案时发现 answer 好像和计分板文件有点冲突，改名，保存答案

```
2022211414@bupt1:~/bomb503$ mv answer.txt ans.txt
```

```
2022211414@bupt1:~/bomb503$ cat ans.txt
Border relations with Canada have never been better.
1 2 4 7 11 16
```

4. 阶段 3

查看 phase_3 汇编

```
(gdb) disas phase_3
Dump of assembler code for function phase_3:
   0x0000000000400fb1 <+0>:     sub    $0x28,%rsp
   0x0000000000400fb5 <+4>:     mov    %fs:0x28,%rax
   0x0000000000400fbe <+13>:    mov    %rax,0x18(%rsp)
   0x0000000000400fc3 <+18>:    xor    %eax,%eax
   0x0000000000400fc5 <+20>:    lea    0x14(%rsp),%r8
   0x0000000000400fca <+25>:    lea    0xf(%rsp),%rcx
   0x0000000000400fcf <+30>:    lea    0x10(%rsp),%rdx
   0x0000000000400fd4 <+35>:    mov    $0x40279e,%esi
   0x0000000000400fd9 <+40>:    callq  0x400c40 <__isoc99_sscanf@plt>
   0x0000000000400fde <+45>:    cmp    $0x2,%eax
   0x0000000000400fe1 <+48>:    jg     0x400fe8 <phase_3+55>
   0x0000000000400fe3 <+50>:    callq  0x401773 <explode_bomb>
   0x0000000000400fe8 <+55>:    cmpl   $0x7,0x10(%rsp)
   0x0000000000400fed <+60>:    ja     0x4010ec <phase_3+315>
   0x0000000000400ff3 <+66>:    mov    0x10(%rsp),%eax
   0x0000000000400ff7 <+70>:    jmpq   *0x4027b0(,%rax,8)
   0x0000000000400ffe <+77>:    mov    $0x6c,%eax
   0x0000000000401003 <+82>:    cmpl   $0x3ab,0x14(%rsp)
   0x000000000040100b <+90>:    je     0x4010f6 <phase_3+325>
   0x0000000000401011 <+96>:    callq  0x401773 <explode_bomb>
   0x0000000000401016 <+101>:   mov    $0x6c,%eax
   0x000000000040101b <+106>:   jmpq   0x4010f6 <phase_3+325>
   0x0000000000401020 <+111>:   mov    $0x75,%eax
   0x0000000000401025 <+116>:   cmpl   $0x4b,0x14(%rsp)
   0x000000000040102a <+121>:   je     0x4010f6 <phase_3+325>
   0x0000000000401030 <+127>:   callq  0x401773 <explode_bomb>
   0x0000000000401035 <+132>:   mov    $0x75,%eax
   0x000000000040103a <+137>:   jmpq   0x4010f6 <phase_3+325>
   0x000000000040103f <+142>:   mov    $0x69,%eax
   0x0000000000401044 <+147>:   cmpl   $0x86,0x14(%rsp)
   0x000000000040104c <+155>:   je     0x4010f6 <phase_3+325>
   0x0000000000401052 <+161>:   callq  0x401773 <explode_bomb>
   0x0000000000401057 <+166>:   mov    $0x69,%eax
   0x000000000040105c <+171>:   jmpq   0x4010f6 <phase_3+325>
   0x0000000000401061 <+176>:   mov    $0x61,%eax
   0x0000000000401066 <+181>:   cmpl   $0x23f,0x14(%rsp)
   0x000000000040106e <+189>:   je     0x4010f6 <phase_3+325>
   0x0000000000401074 <+195>:   callq  0x401773 <explode_bomb>
   0x0000000000401079 <+200>:   mov    $0x61,%eax
   0x000000000040107e <+205>:   jmp    0x4010f6 <phase_3+325>
   0x0000000000401080 <+207>:   mov    $0x75,%eax
   0x0000000000401085 <+212>:   cmpl   $0xaa,0x14(%rsp)
   0x000000000040108d <+220>:   je     0x4010f6 <phase_3+325>
   0x000000000040108f <+222>:   callq  0x401773 <explode_bomb>
   0x0000000000401094 <+227>:   mov    $0x75,%eax
   0x0000000000401099 <+232>:   jmp    0x4010f6 <phase_3+325>
   0x000000000040109b <+234>:   mov    $0x6e,%eax
   0x00000000004010a0 <+239>:   cmpl   $0x2ee,0x14(%rsp)
   0x00000000004010a8 <+247>:   je     0x4010f6 <phase_3+325>
   0x00000000004010aa <+249>:   callq  0x401773 <explode_bomb>
   0x00000000004010af <+254>:   mov    $0x6e,%eax
   0x00000000004010b4 <+259>:   jmp    0x4010f6 <phase_3+325>
   0x00000000004010b6 <+261>:   mov    $0x62,%eax
   0x00000000004010bb <+266>:   cmpl   $0x2c3,0x14(%rsp)
   0x00000000004010c3 <+274>:   je     0x4010f6 <phase_3+325>
   0x00000000004010c5 <+276>:   callq  0x401773 <explode_bomb>
   0x00000000004010ca <+281>:   mov    $0x62,%eax
   0x00000000004010cf <+286>:   jmp    0x4010f6 <phase_3+325>
   0x00000000004010d1 <+288>:   mov    $0x62,%eax
   0x00000000004010d6 <+293>:   cmpl   $0x147,0x14(%rsp)
   0x00000000004010de <+301>:   je     0x4010f6 <phase_3+325>
   0x00000000004010e0 <+303>:   callq  0x401773 <explode_bomb>
   0x00000000004010e5 <+308>:   mov    $0x62,%eax
   0x00000000004010ea <+313>:   jmp    0x4010f6 <phase_3+325>
   0x00000000004010ec <+315>:   callq  0x401773 <explode_bomb>
   0x00000000004010f1 <+320>:   mov    $0x67,%eax
   0x00000000004010f6 <+325>:   cmp    0xf(%rsp),%al
   0x00000000004010fa <+329>:   je     0x401101 <phase_3+336>
   0x00000000004010fc <+331>:   callq  0x401773 <explode_bomb>
   0x0000000000401101 <+336>:   mov    0x18(%rsp),%rax
   0x0000000000401106 <+341>:   xor    %fs:0x28,%rax
   0x000000000040110f <+350>:   je     0x401116 <phase_3+357>
   0x0000000000401111 <+352>:   callq  0x400b90 <__stack_chk_fail@plt>
   0x0000000000401116 <+357>:   add    $0x28,%rsp
   0x000000000040111a <+361>:   retq
```

给 3 个参数分配空间，<+40>调用 scanf

查看 0x40279e 处的值（以字符串形式）

```
(gdb) x/s 0x40279e
0x40279e:       "%d %c %d"
```

可知 3 个参数分别为：整数 字符 整数（按照参数寄存器顺序可推断 3 个参数的地址为：0x10(%rsp) 0xf(%rsp) 0x14(%rsp)）

<+45>比较，%eax 值>2 即输入参数大于 2 个则跳转，否则调用 explode_bomb

<+55>比较，0x10(%rsp)值>7 则调用 explode_bomb，可知第一个整数<=7

把 0x10(%rsp)值放入%eax，<+70>开始进入 switch(第一个整数)：间接跳转到以 0x4027b0 为基址的跳转表。

查看跳转表（以字符形式显示）

```
(gdb) x/8a 0x4027b0
0x4027b0:       0x400ffe <phase_3+77>   0x401020 <phase_3+111>
0x4027c0:       0x40103f <phase_3+142>  0x401061 <phase_3+176>
0x4027d0:       0x401080 <phase_3+207>  0x40109b <phase_3+234>
0x4027e0:       0x4010b6 <phase_3+261>  0x4010d1 <phase_3+288>
(gdb)
```

因为第一个整数<=7，取整数 0 到 7，共 8 个 case

case 0:

<+77>把 0x6c 放入%eax

<+82>比较，0x14(%rsp) 值！=0x3ab 则调用 explode_bomb，可知第二个整数为 0x3ab（十进制 939）

<+325>比较，0xf(%rsp) 值！=%al（%eax 低 8 位）则调用 explode_bomb，可知字符为 0x6c（对应 ASCII 码为 l）

```
0 l 939
Halfway there!
```

阶段 3 炸弹解除

case 1:

<+111>把 0x75 放入%eax

<+116>比较，与 case0 同理可得第二个整数为 0x4b（75）

<+325>比较，与 case0 同理可得字符为 0x75（u）

case 2:

<+142>把 0x69 放入%eax

<+147>比较，与 case0 同理可得第二个整数为 0x86（134）

<+325>比较，与 case0 同理可得字符为 0x69（i）

case 3:

<+176>把 0x61 放入%eax

<+181>比较，与 case0 同理可得第二个整数为 0x23f（575）

<+325>比较，与 case0 同理可得字符为 0x61（a）

case 4:

<+207>把 0x75 放入%eax

<+212>比较，与 case0 同理可得第二个整数为 0xaa（170）

<+325>比较，与 case0 同理可得字符为 0x75（u）

case 5:

<+234>把 0x6e 放入%eax

<+239>比较，与 case0 同理可得第二个整数为 0x2ee（750）

<+325>比较，与 case0 同理可得字符为 0x6e（n）

case 6:

<+261>把 0x62 放入%eax

<+266>比较，与 case0 同理可得第二个整数为 0x2c3（707）

<+325>比较，与 case0 同理可得字符为 0x62（b）

case 7:

<+288>把 0x62 放入%eax

<+293>比较，与 case0 同理可得第二个整数为 0x147（327）

<+325>比较，与 case0 同理可得字符为 0x62（b）

关于其他指令的探索：

<+336>把 0x18(%rsp)值放入%rax，之后执行%rax 与内存地址%fs:0x28 的异或操作，如果二者相同，%rax 为 0。

经过查找资料了解到，这种操作通常用于检查线程本地存储中的数据，以确保数据的完整性和安全性。

5. 阶段 4

查看 phase_4 汇编

```
(gdb) disas phase_4
Dump of assembler code for function phase_4:
   0x0000000000401159 <+0>:     sub    $0x18,%rsp
   0x000000000040115d <+4>:     mov    %fs:0x28,%rax
   0x0000000000401166 <+13>:    mov    %rax,0x8(%rsp)
   0x000000000040116b <+18>:    xor    %eax,%eax
   0x000000000040116d <+20>:    lea    0x4(%rsp),%rcx
   0x0000000000401172 <+25>:    mov    %rsp,%rdx
   0x0000000000401175 <+28>:    mov    $0x402a4d,%esi
   0x000000000040117a <+33>:    callq  0x400c40 <__isoc99_sscanf@plt>
   0x000000000040117f <+38>:    cmp    $0x2,%eax
   0x0000000000401182 <+41>:    jne    0x40118a <phase_4+49>
   0x0000000000401184 <+43>:    cmpl   $0xe,(%rsp)
   0x0000000000401188 <+47>:    jbe    0x40118f <phase_4+54>
   0x000000000040118a <+49>:    callq  0x401773 <explode_bomb>
   0x000000000040118f <+54>:    mov    $0xe,%edx
   0x0000000000401194 <+59>:    mov    $0x0,%esi
   0x0000000000401199 <+64>:    mov    (%rsp),%edi
   0x000000000040119c <+67>:    callq  0x40111b <func4>
   0x00000000004011a1 <+72>:    cmp    $0x1,%eax
   0x00000000004011a4 <+75>:    jne    0x4011ad <phase_4+84>
   0x00000000004011a6 <+77>:    cmpl   $0x1,0x4(%rsp)
   0x00000000004011ab <+82>:    je     0x4011b2 <phase_4+89>
   0x00000000004011ad <+84>:    callq  0x401773 <explode_bomb>
   0x00000000004011b2 <+89>:    mov    0x8(%rsp),%rax
   0x00000000004011b7 <+94>:    xor    %fs:0x28,%rax
   0x00000000004011c0 <+103>:   je     0x4011c7 <phase_4+110>
   0x00000000004011c2 <+105>:   callq  0x400b90 <__stack_chk_fail@plt>
   0x00000000004011c7 <+110>:   add    $0x18,%rsp
   0x00000000004011cb <+114>:   retq
```

给 2 个参数分配空间，< phase_4+33>调用 scanf

查看$0x402a4d 处的值（以字符串形式）

```
(gdb) x/s 0x402a4d
0x402a4d:       "%d %d"
```

可知 2 个参数是 2 个整数（按照参数寄存器顺序可推断 2 个参数的地址为：(%rsp) 0x4(%rsp)）

< phase_4+38>比较，%eax 值!=2 即输入参数不等于 2 个则调用 explode_bomb

< phase_4+43>比较，(%rsp) 值<=14 则跳转，否则调用 explode_bomb，可知第一个整数<=14（因为是无符号比较，并且>=0）

把 14 放入%edx，0 放入%esi，(%rsp) 值放入%edi，调用 func4

查看 func4 汇编

```
(gdb) disas func4
Dump of assembler code for function func4:
   0x000000000040111b <+0>:     sub    $0x8,%rsp
   0x000000000040111f <+4>:     mov    %edx,%eax
   0x0000000000401121 <+6>:     sub    %esi,%eax
   0x0000000000401123 <+8>:     mov    %eax,%ecx
   0x0000000000401125 <+10>:    shr    $0x1f,%ecx
   0x0000000000401128 <+13>:    add    %ecx,%eax
   0x000000000040112a <+15>:    sar    %eax
   0x000000000040112c <+17>:    lea    (%rax,%rsi,1),%ecx
   0x000000000040112f <+20>:    cmp    %edi,%ecx
   0x0000000000401131 <+22>:    jle    0x40113f <func4+36>
   0x0000000000401133 <+24>:    lea    -0x1(%rcx),%edx
   0x0000000000401136 <+27>:    callq  0x40111b <func4>
   0x000000000040113b <+32>:    add    %eax,%eax
   0x000000000040113d <+34>:    jmp    0x401154 <func4+57>
   0x000000000040113f <+36>:    mov    $0x0,%eax
   0x0000000000401144 <+41>:    cmp    %edi,%ecx
   0x0000000000401146 <+43>:    jge    0x401154 <func4+57>
   0x0000000000401148 <+45>:    lea    0x1(%rcx),%esi
   0x000000000040114b <+48>:    callq  0x40111b <func4>
   0x0000000000401150 <+53>:    lea    0x1(%rax,%rax,1),%eax
   0x0000000000401154 <+57>:    add    $0x8,%rsp
   0x0000000000401158 <+61>:    retq
End of assembler dump.
```

发现 func4 是一个递归函数

%edx 值（b）减%esi 值（a）放入%ecx，%ecx 值逻辑右移 31 位，%eax 值（b-a）加%ecx 值，%eax 值算数右移 1 位，%eax 值为(b-a)>>31+(b-a))>>1，把(%rax,%rsi,1)放入%ecx

< func4+20>比较

若%ecx 值 <=%edi 值（x）：若值%ecx >= %edi 值，则返回 0，否则 0x1(%rcx)放入%esi，调用 func4，返回值*2+1；

否则：-0x1(%rcx)放入%edx，调用 func4，返回值*2；

< phase_4+72>比较，func4 返回值!=1 则调用 explode_bomb

< phase_4+77>比较，0x4(%rsp) 值==1 则跳转，否则调用 explode_bomb，可知第二个整数为 1

写出代码，求值

```c
#include <stdio.h>
#include <stdlib.h>
int func4(int x, int a, int b)
{
    int temp=(((b-a)>>31)+(b-a))>>1+a;
    if(temp<=x)
        if(temp==x)
            return 0;
        else
            return func4(x,temp+1,b)*2+1;
    else
        return func4(x,a,temp-1)*2;
}
int main(void)
{

    int x,y=1;
    for(x=0; x<=14; x++){
        if(func4(x,0,14)==1)
            printf("%d %d\n",x,y);
    }
    return 0;
}
```

```
8 1
9 1
11 1
```

填入其中一个答案

```
11 1
So you got that one.  Try this one.
```

阶段 4 炸弹解除

6. 阶段 5

查看 phase_5 汇编

```
(gdb) disas phase_5
Dump of assembler code for function phase_5:
   0x00000000004011cc <+0>:     push   %rbx
   0x00000000004011cd <+1>:     sub    $0x10,%rsp
   0x00000000004011d1 <+5>:     mov    %rdi,%rbx
   0x00000000004011d4 <+8>:     mov    %fs:0x28,%rax
   0x00000000004011dd <+17>:    mov    %rax,0x8(%rsp)
   0x00000000004011e2 <+22>:    xor    %eax,%eax
   0x00000000004011e4 <+24>:    callq  0x401481 <string_length>
   0x00000000004011e9 <+29>:    cmp    $0x6,%eax
   0x00000000004011ec <+32>:    je     0x4011f3 <phase_5+39>
   0x00000000004011ee <+34>:    callq  0x401773 <explode_bomb>
   0x00000000004011f3 <+39>:    mov    $0x0,%eax
   0x00000000004011f8 <+44>:    movzbl (%rbx,%rax,1),%edx
   0x00000000004011fc <+48>:    and    $0xf,%edx
   0x00000000004011ff <+51>:    movzbl 0x4027f0(%rdx),%edx
   0x0000000000401206 <+58>:    mov    %dl,(%rsp,%rax,1)
   0x0000000000401209 <+61>:    add    $0x1,%rax
   0x000000000040120d <+65>:    cmp    $0x6,%rax
   0x0000000000401211 <+69>:    jne    0x4011f8 <phase_5+44>
   0x0000000000401213 <+71>:    movb   $0x0,0x6(%rsp)
   0x0000000000401218 <+76>:    mov    $0x4027a7,%esi
   0x000000000040121d <+81>:    mov    %rsp,%rdi
   0x0000000000401220 <+84>:    callq  0x40149f <strings_not_equal>
   0x0000000000401225 <+89>:    test   %eax,%eax
   0x0000000000401227 <+91>:    je     0x40122e <phase_5+98>
   0x0000000000401229 <+93>:    callq  0x401773 <explode_bomb>
   0x000000000040122e <+98>:    mov    0x8(%rsp),%rax
   0x0000000000401233 <+103>:   xor    %fs:0x28,%rax
   0x000000000040123c <+112>:   je     0x401243 <phase_5+119>
--Type <RET> for more, q to quit, c to continue without paging--c
   0x000000000040123e <+114>:   callq  0x400b90 <__stack_chk_fail@plt>
   0x0000000000401243 <+119>:   add    $0x10,%rsp
   0x0000000000401247 <+123>:   pop    %rbx
   0x0000000000401248 <+124>:   retq
End of assembler dump.
```

<+29>比较，string_length==6 则跳转，否则调用 explode_bomb，可知输入字符串长度为 6

<+39>把%eax 置 0，<+44>~<+69>相当于一个循环：每次(%rbx+%rax)值和 0xf 进行与运算，即取(%rbx+%rax)值低 4 位放到%edx，再取 0x4027f0 (%rdx)值低 4 位放到%edx，把%edx 低 8 位放到地址%rsp+%rax。（%rbx 为输入字符串首地址，%rax 相当于一个循环计数器，每次循环+1，和 6 比较，不相等则进入下一轮循环，否则退出循环。）

查看 0x4027f0 处的值（以字符串形式）

```
(gdb) x/s 0x4027f0
0x4027f0 <array.3602>:  "maduiersnfotvbylSo you think you can stop the bomb with ctrl-c, do you?"
```

循环结束，0x6(%rsp)值赋为 ‘\0’后，构成了新字符串。

和 phase_1 类似，把新字符串与 0x4027a7 处的值比较，不相等则调用 explode_bomb。
查看 0x4027a7 处的值（以字符串形式）

```
(gdb) x/s 0x4027a7
0x4027a7:       "devils"
```

分析：使新字符串第 1 个字符 ‘d’，对应“maduiersnfotvbyl”的‘d’下标为 2，需要输入字符串第 1 个字符低 4 位为 0x2
同理可得：输入字符串第 2 个字符低 4 位为 0x5，第 3 个字符低 4 位为 0xc，第 4 个字符低 4 位为 0x4，第 5 个字符低 4 位为 0xf，第 6 个字符低 4 位为 0x7。
对应 ASCII 码有多种答案
填入其中一种

```
25<4?7
Good work!  On to the next...
```

阶段 5 炸弹解除

7. 阶段 6
查看 phase_6 汇编

```
(gdb) disas phase_6
Dump of assembler code for function phase_6:
   0x0000000000401249 <+0>:     push   %r14
   0x000000000040124b <+2>:     push   %r13
   0x000000000040124d <+4>:     push   %r12
   0x000000000040124f <+6>:     push   %rbp
   0x0000000000401250 <+7>:     push   %rbx
   0x0000000000401251 <+8>:     sub    $0x60,%rsp
   0x0000000000401255 <+12>:    mov    %fs:0x28,%rax
   0x000000000040125e <+21>:    mov    %rax,0x58(%rsp)
   0x0000000000401263 <+26>:    xor    %eax,%eax
   0x0000000000401265 <+28>:    mov    %rsp,%rsi
   0x0000000000401268 <+31>:    callq  0x4017a9 <read_six_numbers>
   0x000000000040126d <+36>:    mov    %rsp,%r12
   0x0000000000401270 <+39>:    mov    %rsp,%r13
   0x0000000000401273 <+42>:    mov    $0x0,%r14d
   0x0000000000401279 <+48>:    mov    %r13,%rbp
   0x000000000040127c <+51>:    mov    0x0(%r13),%eax
   0x0000000000401280 <+55>:    sub    $0x1,%eax
   0x0000000000401283 <+58>:    cmp    $0x5,%eax
   0x0000000000401286 <+61>:    jbe    0x40128d <phase_6+68>
   0x0000000000401288 <+63>:    callq  0x401773 <explode_bomb>
   0x000000000040128d <+68>:    add    $0x1,%r14d
   0x0000000000401291 <+72>:    cmp    $0x6,%r14d
   0x0000000000401295 <+76>:    je     0x4012b8 <phase_6+111>
   0x0000000000401297 <+78>:    mov    %r14d,%ebx
   0x000000000040129a <+81>:    movslq %ebx,%rax
   0x000000000040129d <+84>:    mov    (%rsp,%rax,4),%eax
   0x00000000004012a0 <+87>:    cmp    %eax,0x0(%rbp)
   0x00000000004012a3 <+90>:    jne    0x4012aa <phase_6+97>
```

```
   0x00000000004012a5 <+92>:    callq  0x401773 <explode_bomb>
   0x00000000004012aa <+97>:    add    $0x1,%ebx
   0x00000000004012ad <+100>:   cmp    $0x5,%ebx
   0x00000000004012b0 <+103>:   jle    0x40129a <phase_6+81>
   0x00000000004012b2 <+105>:   add    $0x4,%r13
   0x00000000004012b6 <+109>:   jmp    0x401279 <phase_6+48>
   0x00000000004012b8 <+111>:   lea    0x18(%rsp),%rcx
   0x00000000004012bd <+116>:   mov    $0x7,%edx
   0x00000000004012c2 <+121>:   mov    %edx,%eax
   0x00000000004012c4 <+123>:   sub    (%r12),%eax
   0x00000000004012c8 <+127>:   mov    %eax,(%r12)
   0x00000000004012cc <+131>:   add    $0x4,%r12
   0x00000000004012d0 <+135>:   cmp    %r12,%rcx
   0x00000000004012d3 <+138>:   jne    0x4012c2 <phase_6+121>
   0x00000000004012d5 <+140>:   mov    $0x0,%esi
   0x00000000004012da <+145>:   jmp    0x4012f6 <phase_6+173>
   0x00000000004012dc <+147>:   mov    0x8(%rdx),%rdx
   0x00000000004012e0 <+151>:   add    $0x1,%eax
   0x00000000004012e3 <+154>:   cmp    %ecx,%eax
   0x00000000004012e5 <+156>:   jne    0x4012dc <phase_6+147>
   0x00000000004012e7 <+158>:   mov    %rdx,0x20(%rsp,%rsi,2)
   0x00000000004012ec <+163>:   add    $0x4,%rsi
   0x00000000004012f0 <+167>:   cmp    $0x18,%rsi
   0x00000000004012f4 <+171>:   je     0x40130a <phase_6+193>
   0x00000000004012f6 <+173>:   mov    (%rsp,%rsi,1),%ecx
   0x00000000004012f9 <+176>:   mov    $0x1,%eax
   0x00000000004012fe <+181>:   mov    $0x6042f0,%edx
   0x0000000000401303 <+186>:   cmp    $0x1,%ecx
   0x0000000000401306 <+189>:   jg     0x4012dc <phase_6+147>
   0x0000000000401308 <+191>:   jmp    0x4012e7 <phase_6+158>
   0x000000000040130a <+193>:   mov    0x20(%rsp),%rbx
   0x000000000040130f <+198>:   lea    0x20(%rsp),%rax
   0x0000000000401314 <+203>:   lea    0x48(%rsp),%rsi
   0x0000000000401319 <+208>:   mov    %rbx,%rcx
   0x000000000040131c <+211>:   mov    0x8(%rax),%rdx
   0x0000000000401320 <+215>:   mov    %rdx,0x8(%rcx)
   0x0000000000401324 <+219>:   add    $0x8,%rax
   0x0000000000401328 <+223>:   mov    %rdx,%rcx
   0x000000000040132b <+226>:   cmp    %rax,%rsi
   0x000000000040132e <+229>:   jne    0x40131c <phase_6+211>
   0x0000000000401330 <+231>:   movq   $0x0,0x8(%rdx)
   0x0000000000401338 <+239>:   mov    $0x5,%ebp
   0x000000000040133d <+244>:   mov    0x8(%rbx),%rax
   0x0000000000401341 <+248>:   mov    (%rax),%eax
   0x0000000000401343 <+250>:   cmp    %eax,(%rbx)
   0x0000000000401345 <+252>:   jge    0x40134c <phase_6+259>
   0x0000000000401347 <+254>:   callq  0x401773 <explode_bomb>
```

```
   0x000000000040134c <+259>:   mov    0x8(%rbx),%rbx
   0x0000000000401350 <+263>:   sub    $0x1,%ebp
   0x0000000000401353 <+266>:   jne    0x40133d <phase_6+244>
   0x0000000000401355 <+268>:   mov    0x58(%rsp),%rax
   0x000000000040135a <+273>:   xor    %fs:0x28,%rax
   0x0000000000401363 <+282>:   je     0x40136a <phase_6+289>
   0x0000000000401365 <+284>:   callq  0x400b90 <__stack_chk_fail@plt>
   0x000000000040136a <+289>:   add    $0x60,%rsp
   0x000000000040136e <+293>:   pop    %rbx
   0x000000000040136f <+294>:   pop    %rbp
   0x0000000000401370 <+295>:   pop    %r12
   0x0000000000401372 <+297>:   pop    %r13
   0x0000000000401374 <+299>:   pop    %r14
   0x0000000000401376 <+301>:   retq
End of assembler dump.
```

分析：<+31>调用 read_six_numbers 函数，输入是 6 个整数。<+48>~<+109>是一个循环，栈顶元素-1 放入%eax 和 5 进行无符号数比较，大于 5 则调用 explode_bomb，<+87>比较后一个数和前一个数，相等则调用 explode_bomb，推测出 6 个整数为 1 到 6，且互不相等。<+111>~<+191>处理 6 个数，<+181>把 0x6042f0 处的值放入%edx

查看

```
(gdb) x/4x 0x6042f0
0x6042f0 <node1>:       0x00000371      0x00000001      0x00604300      0x00000000
```

因为显示<node1>所以推测是链表，最后一个地址和本地址连续，继续查看

```
(gdb) x/4x 0x604300
0x604300 <node2>:       0x000002d3      0x00000002      0x00604310      0x00000000
(gdb) x/4x 0x604310
0x604310 <node3>:       0x00000372      0x00000003      0x00604320      0x00000000
(gdb) x/4x 0x604320
0x604320 <node4>:       0x00000353      0x00000004      0x00604330      0x00000000
(gdb) x/4x 0x604330
0x604330 <node5>:       0x0000025c      0x00000005      0x00604340      0x00000000
(gdb) x/4x 0x604340
0x604340 <node6>:       0x00000237      0x00000006      0x00000000      0x00000000
```

五、总结体会