哈爾濱工業大學

# 实验报告

## 实 验（三）

题　　目 Binary Bomb

二进制炸弹

专　　业 计算机系

学　　号 1180300811

班　　级 1803008

学　　生 孙骁

指 导 教 师 吴锐

实 验 地 点 G712

实 验 日 期 2019/10/19

## 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

熟练掌握计算机系统的 ISA 指令系统与寻址方式

熟练掌握 Linux 下调试器的反汇编调试跟踪分析机器语言的方法

增强对程序机器级表示、汇编语言、调试器和逆向工程等的理解

## 1.2 实验环境与工具

### 1.2.1 硬件环境

i7-8550U X64 CPU；1.80GHz；2G RAM；256GHD Disk

### 1.2.2 软件环境

Windows10 64 位；Vmware 15.1.0；Ubuntu 18.04 LTS

### 1.2.3 开发工具

Visual Studio 2019 ；CodeBlocks；gcc

## 1.3 实验预习

上实验课前，必须认真预习实验指导书（PPT 或 PDF）

了解实验的目的、实验环境与软硬件工具、实验操作步骤，复习与实验有关的理论知识。

请写出 C 语言下包含字符串比较、循环、分支（含 switch）、函数调用、递归、指针、结构、链表等的例子程序 sample.c。

生成执行程序 sample.out。

用 gcc –S 或 CodeBlocks 或 GDB 或 OBJDUMP 等，反汇编，比较。

列出每一部分的 C 语言对应的汇编语言。

修改编译选项-O (缺省 2)、O0、O1、O2、O3，-m32/m64。再次查看生成的汇

编语言与原来的区别。

注意 O1 之后无栈帧，EBP 做别的用途。-fno-omit-frame-pointer 加上栈指针。

GDB 命令详解 –tui 模式 ^XA 切换 layout 改变等等

有目的地学习: 看 VS 的功能 GDB 命令用什么？

# 第 2 章 实验环境建立

## 2.1 Ubuntu 下 CodeBlocks 反汇编（10 分）

CodeBlocks 运行 hellolinux.c。反汇编查看 printf 函数的实现。

要求：C、ASM、内存(显示 hello 等内容)、堆栈（call printf 前）、寄存器同时在一个窗口。

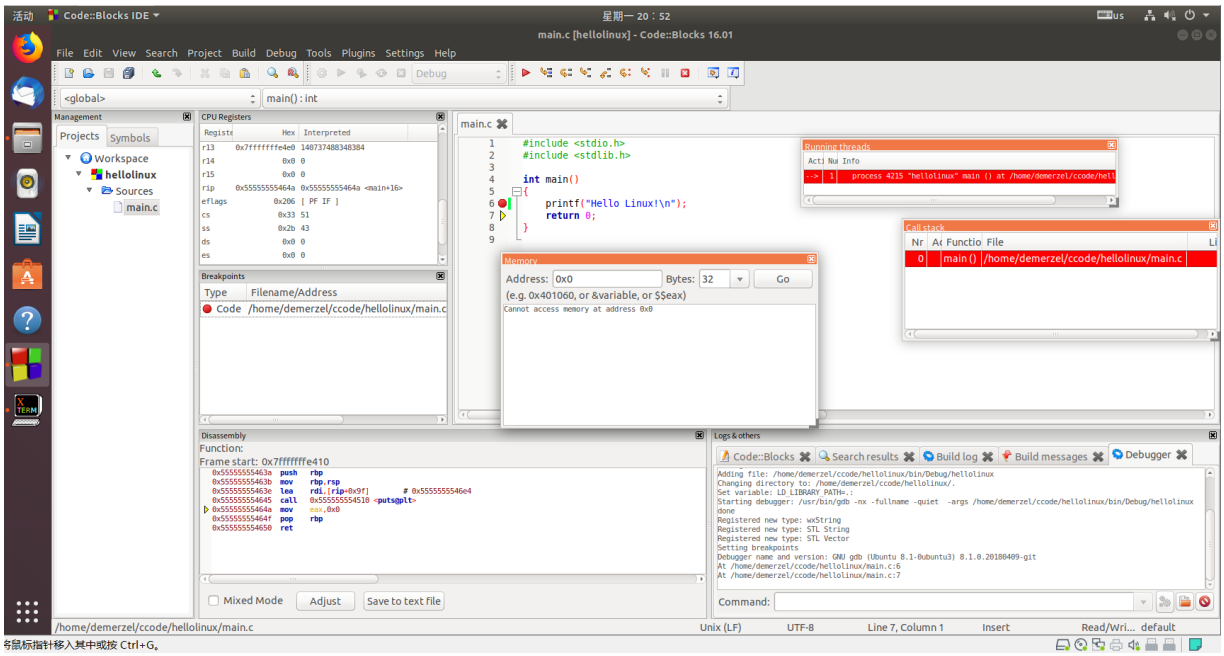



图 2-1　Ubuntu 下 CodeBlocks 反汇编截图

## 2.2 Ubuntu 下 EDB 运行环境建立（10 分）

用 EDB 调试 hellolinux.c 的执行文件，截图，要求同 2.1

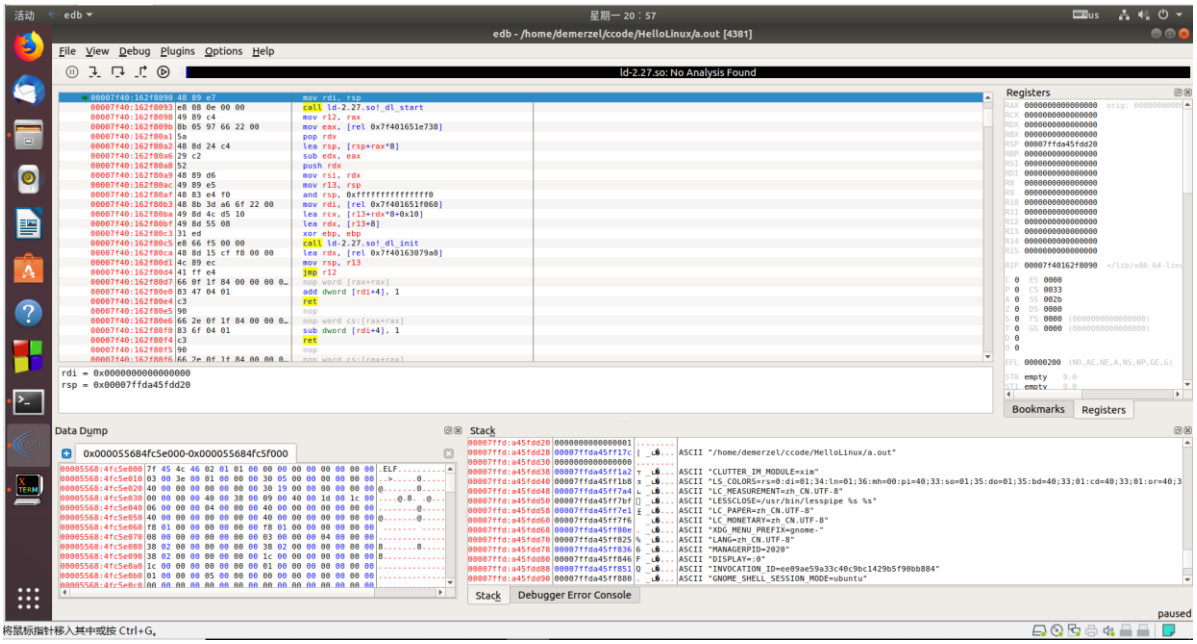



图 2-2　Ubuntu 下 EDB 截图

# 第 3 章 各阶段炸弹破解与分析

每阶段 15 分（密码 10 分，分析 5 分），总分不超过 80 分

## 3.1 阶段 1 的破解与分析

密码如下：I am not part of the problem. I am a Republican.

破解过程：首先通过 objdimp -D 得到 bomb 文件的反汇编文件，定位到<main>段，发现需要连续调用 6 个<phase>，即判断六次密码，首先定位到第一个<phase_1>函数的反汇编区。

```
00000000004012a2 <main>:
  4012a2:       55                      push   %rbp
  4012a3:       48 89 e5                mov    %rsp,%rbp
  4012a6:       53                      push   %rbx
  4012a7:       48 83 ec 08             sub    $0x8,%rsp
  4012ab:       83 ff 01                cmp    $0x1,%edi
  4012ae:       0f 84 ed 00 00 00       je     4013a1 <main+0xff>
  4012b4:       48 89 f3                mov    %rsi,%rbx
  4012b7:       83 ff 02                cmp    $0x2,%edi
  4012ba:       0f 85 14 01 00 00       jne    4013d4 <main+0x132>
  4012c0:       48 8b 7e 08             mov    0x8(%rsi),%rdi
  4012c4:       be 04 30 40 00          mov    $0x403004,%esi
  4012c9:       e8 72 fe ff ff          callq  401140 <fopen@plt>
  4012ce:       48 89 05 9b 44 00 00    mov    %rax,0x449b(%rip)        # 405770 <infile>
  4012d5:       48 85 c0                test   %rax,%rax
  4012d8:       0f 84 d6 00 00 00       je     4013b4 <main+0x112>
  4012de:       e8 32 06 00 00          callq  401915 <initialize_bomb>
  4012e3:       bf 88 30 40 00          mov    $0x403088,%edi
  4012e8:       e8 73 fd ff ff          callq  401060 <puts@plt>
  4012ed:       bf c8 30 40 00          mov    $0x4030c8,%edi
  4012f2:       e8 69 fd ff ff          callq  401060 <puts@plt>
  4012f7:       e8 19 07 00 00          callq  401a15 <read_line>
  4012fc:       48 89 c7                mov    %rax,%rdi
  4012ff:       e8 f1 00 00 00          callq  4013f5 <phase_1>
  401304:       e8 3d 08 00 00          callq  401b46 <phase_defused>
  401309:       bf f8 30 40 00          mov    $0x4030f8,%edi
  40130e:       e8 4d fd ff ff          callq  401060 <puts@plt>
  401313:       e8 fd 06 00 00          callq  401a15 <read_line>
  401318:       48 89 c7                mov    %rax,%rdi
  40131b:       e8 f0 00 00 00          callq  401410 <phase_2>
  401320:       e8 21 08 00 00          callq  401b46 <phase_defused>
  401325:       bf 3d 30 40 00          mov    $0x40303d,%edi
  40132a:       e8 31 fd ff ff          callq  401060 <puts@plt>
  40132f:       e8 e1 06 00 00          callq  401a15 <read_line>
  401334:       48 89 c7                mov    %rax,%rdi
  401337:       e8 24 01 00 00          callq  401460 <phase_3>
  40133c:       e8 05 08 00 00          callq  401b46 <phase_defused>
  401341:       bf 5b 30 40 00          mov    $0x40305b,%edi
  401346:       e8 15 fd ff ff          callq  401060 <puts@plt>
  40134b:       e8 c5 06 00 00          callq  401a15 <read_line>
  401350:       48 89 c7                mov    %rax,%rdi
  401353:       e8 92 02 00 00          callq  4015ea <phase_4>
  401358:       e8 e9 07 00 00          callq  401b46 <phase_defused>
```

```
00000000004013f5 <phase_1>:
  4013f5:       55                      push   %rbp
  4013f6:       48 89 e5                mov    %rsp,%rbp
  4013f9:       be 50 31 40 00          mov    $0x403150,%esi
  4013fe:       e8 b5 04 00 00          callq  4018b8 <strings_not_equal>
  401403:       85 c0                   test   %eax,%eax
  401405:       75 02                   jne    401409 <phase_1+0x14>
  401407:       5d                      pop    %rbp
  401408:       c3                      retq
  401409:       e8 a9 05 00 00          callq  4019b7 <explode_bomb>
  40140e:       eb f7                   jmp    401407 <phase_1+0x12>
```

<phase_1>函数首先一个地址中的数（间接寻址）mov 进%esi 中，调用字符串比较函数判断输入字符串与内存中的是否相等。若不等，跳到 401409 行，执行引爆函数，即破解失败；若相等，跳到 401407 行，弹出栈，返回主函数继续检查第二个密码。即得知，0x401350 处存放的是第一个密码。通过 gdb 调试如下：

```
demerzel@demerzel-virtual-machine: ~/ccode/lab3/bomb273
文件(F)  编辑(E)  查看(V)  搜索(S)  终端(T)  帮助(H)
demerzel@demerzel-virtual-machine:~/ccode/lab3/bomb273$ gdb bomb
GNU gdb (Ubuntu 8.1-0ubuntu3) 8.1.0.20180409-git
Copyright (C) 2018 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:
<http://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...done.
(gdb) x/s 0x403150
0x403150:       "I am not part of the problem. I am a Republican."
(gdb)
```

打印 0x401350 处的字符串，得知第一个密码为 I am not part of the problem. I am a Republican.

输入后如下：

```
(gdb) r
Starting program: /home/demerzel/ccode/lab3/bomb273/bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
I am not part of the problem. I am a Republican.
Phase 1 defused. How about the next one?
```

即第一个密码破解完毕。

## 3.2 阶段 2 的破解与分析

密码如下：1 2 4 8 16 32

破解过程：

```
0000000000401410 <phase_2>:
  401410:       55                      push   %rbp
  401411:       48 89 e5                mov    %rsp,%rbp
  401414:       53                      push   %rbx
  401415:       48 83 ec 28             sub    $0x28,%rsp
  401419:       48 8d 75 d0             lea    -0x30(%rbp),%rsi
  40141d:       e8 b7 05 00 00          callq  4019d9 <read_six_numbers>
  401422:       83 7d d0 01             cmpl   $0x1,-0x30(%rbp)
  401426:       75 07                   jne    40142f <phase_2+0x1f>
  401428:       bb 01 00 00 00          mov    $0x1,%ebx
  40142d:       eb 0f                   jmp    40143e <phase_2+0x2e>
  40142f:       e8 83 05 00 00          callq  4019b7 <explode_bomb>
  401434:       eb f2                   jmp    401428 <phase_2+0x18>
  401436:       e8 7c 05 00 00          callq  4019b7 <explode_bomb>
  40143b:       83 c3 01                add    $0x1,%ebx
  40143e:       83 fb 05                cmp    $0x5,%ebx
  401441:       7f 16                   jg     401459 <phase_2+0x49>
  401443:       48 63 d3                movslq %ebx,%rdx
  401446:       8d 43 ff                lea    -0x1(%rbx),%eax
  401449:       48 98                   cltq
  40144b:       8b 44 85 d0             mov    -0x30(%rbp,%rax,4),%eax
  40144f:       01 c0                   add    %eax,%eax
  401451:       39 44 95 d0             cmp    %eax,-0x30(%rbp,%rdx,4)
  401455:       74 e4                   je     40143b <phase_2+0x2b>
  401457:       eb dd                   jmp    401436 <phase_2+0x26>
  401459:       48 83 c4 28             add    $0x28,%rsp
  40145d:       5b                      pop    %rbx
  40145e:       5d                      pop    %rbp
  40145f:       c3                      retq
```

首先根据函数的名字<read_six_numbers>可以判断这是读入 6 个值,而且保存至从%rsi 开始的地址。找到<read_six_numbers>函数，反汇编代码如下

```
00000000004019d9 <read_six_numbers>:
  4019d9:       55                      push   %rbp
  4019da:       48 89 e5                mov    %rsp,%rbp
  4019dd:       48 89 f2                mov    %rsi,%rdx
  4019e0:       48 8d 4e 04             lea    0x4(%rsi),%rcx
  4019e4:       48 8d 46 14             lea    0x14(%rsi),%rax
  4019e8:       50                      push   %rax
  4019e9:       48 8d 46 10             lea    0x10(%rsi),%rax
  4019ed:       50                      push   %rax
  4019ee:       4c 8d 4e 0c             lea    0xc(%rsi),%r9
  4019f2:       4c 8d 46 08             lea    0x8(%rsi),%r8
  4019f6:       be 43 33 40 00          mov    $0x403343,%esi
  4019fb:       b8 00 00 00 00          mov    $0x0,%eax
  401a00:       e8 1b f7 ff ff          callq  401120 <__isoc99_sscanf@plt>
  401a05:       48 83 c4 10             add    $0x10,%rsp
  401a09:       83 f8 05                cmp    $0x5,%eax
  401a0c:       7e 02                   jle    401a10 <read_six_numbers+0x37>
  401a0e:       c9                      leaveq
  401a0f:       c3                      retq
  401a10:       e8 a2 ff ff ff          callq  4019b7 <explode_bomb>
```

即在栈上开辟了 24 个字节的空间，接收 sscnaf 读取的六个整型数值，存入寄存器%rbp 中。当 sscanf 返回值<=5 时，引爆炸弹，否则返回 phase_2，设六个数构成数组 a[6]。

phrase_2 核心代码如下：

```
401422:       83 7d d0 01             cmpl   $0x1,-0x30(%rbp)
401426:       75 07                   jne    40142f <phase_2+0x1f>
401428:       bb 01 00 00 00          mov    $0x1,%ebx
40142d:       eb 0f                   jmp    40143e <phase_2+0x2e>
40142f:       e8 83 05 00 00          callq  4019b7 <explode_bomb>
401434:       eb f2                   jmp    401428 <phase_2+0x18>
401436:       e8 7c 05 00 00          callq  4019b7 <explode_bomb>
40143b:       83 c3 01                add    $0x1,%ebx
40143e:       83 fb 05                cmp    $0x5,%ebx
401441:       7f 16                   jg     401459 <phase_2+0x49>
401443:       48 63 d3                movslq %ebx,%rdx
401446:       8d 43 ff                lea    -0x1(%rbx),%eax
401449:       48 98                   cltq
40144b:       8b 44 85 d0             mov    -0x30(%rbp,%rax,4),%eax
40144f:       01 c0                   add    %eax,%eax
401451:       39 44 95 d0             cmp    %eax,-0x30(%rbp,%rdx,4)
401455:       74 e4                   je     40143b <phase_2+0x2b>
401457:       eb dd                   jmp    401436 <phase_2+0x26>
```

首先看这段指令

```
cmpl  $0x1, -0x30(%rbp)
jne       40143e<phrase_2+0x1f>
```

即比较 1 与(%rbp)-0x30 内的值，若不相等则爆炸，故得知第一个数为 1，且由

```
add       $0x1,%ebx
cmp    $0x5,%ebx
```

得知%ebx 内存储循环控制变量。

且由此行指令

```
mov    -0x30(%rbp,%rax,4) ,%eax
```

得知数组首地址为(%rbp)-0x30，每次更新%eax 的值为 4*%rax+%rbp-0x30，由于数组是 int 型，所以每次更新结果为 a[i]=2*a[i-1]，故第二个密码为 1 2 4 8 16 32。

输入后如下：

```
demerzel@demerzel-virtual-machine:~/ccode/lab3/bomb273$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
I am not part of the problem. I am a Republican.
Phase 1 defused. How about the next one?
1 2 4 8 16 32
That's number 2.  Keep going!
```

## 3.3 阶段 3 的破解与分析

密码如下：0 s 456

破解过程：

```
0000000000401460 <phase_3>:
  401460:       55                      push   %rbp
  401461:       48 89 e5                mov    %rsp,%rbp
  401464:       48 83 ec 10             sub    $0x10,%rsp
  401468:       4c 8d 45 f8             lea    -0x8(%rbp),%r8
  40146c:       48 8d 4d f7             lea    -0x9(%rbp),%rcx
  401470:       48 8d 55 fc             lea    -0x4(%rbp),%rdx
  401474:       be ae 31 40 00          mov    $0x4031ae,%esi
  401479:       b8 00 00 00 00          mov    $0x0,%eax
  40147e:       e8 9d fc ff ff          callq  401120 <__isoc99_sscanf@plt>
  401483:       83 f8 02                cmp    $0x2,%eax
  401486:       7e 14                   jle    40149c <phase_3+0x3c>
  401488:       83 7d fc 07             cmpl   $0x7,-0x4(%rbp)
  40148c:       0f 87 06 01 00 00       ja     401598 <phase_3+0x138>
  401492:       8b 45 fc                mov    -0x4(%rbp),%eax
  401495:       ff 24 c5 c0 31 40 00    jmpq   *0x4031c0(,%rax,8)
```

通过 gdb 得知输入格式是%d,%c,%d,因此调用 sscanf(“%d %c %d”,&a,&b,&c);同时比较返回值，若返回值大于 7 则爆炸。

注意到最后一行执行了无条件跳转地址 0x4031c0+8*%rax，在 gdb 中单步调试查看此地址如下：

```
(gdb) si
0x0000000000401495      73      in phases.c
(gdb) si
76      in phases.c
(gdb) si
0x00000000004014aa      76      in phases.c
(gdb) si
0x00000000004014ac      76      in phases.c
(gdb) c
```

查看 0x4014a3 处代码如下：

```
4014a3:       81 7d f8 c8 01 00 00    cmpl   $0x1c8,-0x8(%rbp)
4014aa:       75 0a                   jne    4014b6 <phase_3+0x56>
4014ac:       b8 73 00 00 00          mov    $0x73,%eax
4014b1:       e9 ec 00 00 00          jmpq   4015a2 <phase_3+0x142>
```

首先比较了$0x1c8，如果不相等则跳转到 0x4014b6 引爆。然后将%eax 赋值立即数$0x73，跳转到 0x4015a2，这里是比较%eax 的最后一个字节（即字符型），若成功则返回主函数。

故第一个数取 0，查找 ASCII 表得知，0x73 为字符 s，第三个数字是 0x1c8，即十进制 456，输入后如下

```
demerzel@demerzel-virtual-machine:~/ccode/lab3/bomb273$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
I am not part of the problem. I am a Republican.
Phase 1 defused. How about the next one?
1 2 4 8 16 32
That's number 2.  Keep going!
0 s 456
Halfway there!
```

## 3.4 阶段 4 的破解与分析

密码如下：14 7

破解过程：

```
00000000004015ea <phase_4>:
  4015ea:       55                      push   %rbp
  4015eb:       48 89 e5                mov    %rsp,%rbp
  4015ee:       48 83 ec 10             sub    $0x10,%rsp
  4015f2:       48 8d 4d f8             lea    -0x8(%rbp),%rcx
  4015f6:       48 8d 55 fc             lea    -0x4(%rbp),%rdx
  4015fa:       be 4f 33 40 00          mov    $0x40334f,%esi
  4015ff:       b8 00 00 00 00          mov    $0x0,%eax
  401604:       e8 17 fb ff ff          callq  401120 <__isoc99_sscanf@plt>
  401609:       83 f8 02                cmp    $0x2,%eax
  40160c:       75 0c                   jne    40161a <phase_4+0x30>
  40160e:       8b 45 fc                mov    -0x4(%rbp),%eax
  401611:       85 c0                   test   %eax,%eax
  401613:       78 05                   js     40161a <phase_4+0x30>
  401615:       83 f8 0e                cmp    $0xe,%eax
  401618:       7e 05                   jle    40161f <phase_4+0x35>
  40161a:       e8 98 03 00 00          callq  4019b7 <explode_bomb>
  40161f:       ba 0e 00 00 00          mov    $0xe,%edx
  401624:       be 00 00 00 00          mov    $0x0,%esi
  401629:       8b 7d fc                mov    -0x4(%rbp),%edi
  40162c:       e8 7f ff ff ff          callq  4015b0 <func4>
  401631:       83 f8 07                cmp    $0x7,%eax
  401634:       75 06                   jne    40163c <phase_4+0x52>
  401636:       83 7d f8 07             cmpl   $0x7,-0x8(%rbp)
  40163a:       74 05                   je     401641 <phase_4+0x57>
  40163c:       e8 76 03 00 00          callq  4019b7 <explode_bomb>
  401641:       c9                      leaveq
  401642:       c3                      retq
```

首先发现程序用地址 0x40334f 接收输入，gdb 查看后，发现是输入两个整型变量

```
(gdb) x /s 0x40334f
0x40334f:       "%d %d"
```

由：

```
mov    -0x4(%rbp),%eax
test   %eax,%eax
js     40161a <phase_4+0x30>
cmp    $0xe,%eax
jle    40161f <phase_4+0x35>
callq  4019b7 <explode_bomb>
```

得知，第一个数据为不大于 14 的正数；

由：

```
cmpl   $0x7,-0x8(%rbp)
je     401641 <phase_4+0x57>
```

```
leaveq
```

得知第二个数据是 7。

下面分析调用的函数 func4:

```
 401618:       7e 05                   jle    40161f <phase_4+0x35>
 40161a:       e8 98 03 00 00          callq  4019b7 <explode_bomb>
 40161f:       ba 0e 00 00 00          mov    $0xe,%edx
 401624:       be 00 00 00 00          mov    $0x0,%esi
 401629:       8b 7d fc                mov    -0x4(%rbp),%edi
 40162c:       e8 7f ff ff ff          callq  4015b0 <func4>

00000000004015b0 <func4>:
  4015b0:       55                      push   %rbp
  4015b1:       48 89 e5                mov    %rsp,%rbp
  4015b4:       89 d1                   mov    %edx,%ecx
  4015b6:       29 f1                   sub    %esi,%ecx
  4015b8:       89 c8                   mov    %ecx,%eax
  4015ba:       c1 e8 1f                shr    $0x1f,%eax
  4015bd:       01 c8                   add    %ecx,%eax
  4015bf:       d1 f8                   sar    %eax
  4015c1:       01 f0                   add    %esi,%eax
  4015c3:       39 f8                   cmp    %edi,%eax
  4015c5:       7f 09                   jg     4015d0 <func4+0x20>
  4015c7:       7c 13                   jl     4015dc <func4+0x2c>
  4015c9:       b8 00 00 00 00          mov    $0x0,%eax
  4015ce:       5d                      pop    %rbp
  4015cf:       c3                      retq
  4015d0:       8d 50 ff                lea    -0x1(%rax),%edx
  4015d3:       e8 d8 ff ff ff          callq  4015b0 <func4>
  4015d8:       01 c0                   add    %eax,%eax
  4015da:       eb f2                   jmp    4015ce <func4+0x1e>
  4015dc:       8d 70 01                lea    0x1(%rax),%esi
  4015df:       e8 cc ff ff ff          callq  4015b0 <func4>
  4015e4:       8d 44 00 01             lea    0x1(%rax,%rax,1),%eax
  4015e8:       eb e4                   jmp    4015ce <func4+0x1e>
```

在 phase_4 中，构造参数 high,参数值为 0xe，保存在%edx；构造参数 low,参数值为 0x0,保存在%esi;构造参数 val,参数值为输入第一个参数值,保存在%edi;执行 func4。

func4 内部是一个递归调用，

转化为相应的 c 语言代码，为

```
int func4(int val, int low, int high)

{

  int v3; // eax
```

```
  v3 = low + (high - low) / 2;
  if ( v3 > val )
    return 2 * func4(val, low, v3 - 1);
  if ( v3 < val )
    return 2 * func4(val, v3 + 1, high) + 1;
  return 0;
}
```

得出 14 是正确答案。

输入后如下：

```
demerzel@demerzel-virtual-machine:~/ccode/lab3/bomb273$ ./bomb ans.txt
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Phase 1 defused. How about the next one?
That's number 2.  Keep going!
Halfway there!
So you got that one.  Try this one.
```

## 3.5 阶段 5 的破解与分析

密码如下：450367

破解过程：

```
0000000000401643 <phase_5>:
  401643:       55                      push   %rbp
  401644:       48 89 e5                mov    %rsp,%rbp
  401647:       53                      push   %rbx
  401648:       48 83 ec 08             sub    $0x8,%rsp
  40164c:       48 89 fb                mov    %rdi,%rbx
  40164f:       e8 50 02 00 00          callq  4018a4 <string_length>
  401654:       83 f8 06                cmp    $0x6,%eax
  401657:       75 25                   jne    40167e <phase_5+0x3b>
  401659:       b9 00 00 00 00          mov    $0x0,%ecx
  40165e:       b8 00 00 00 00          mov    $0x0,%eax
  401663:       83 f8 05                cmp    $0x5,%eax
  401666:       7f 1d                   jg     401685 <phase_5+0x42>
  401668:       48 63 d0                movslq %eax,%rdx
  40166b:       0f b6 14 13             movzbl (%rbx,%rdx,1),%edx
  40166f:       83 e2 0f                and    $0xf,%edx
  401672:       03 0c 95 00 32 40 00    add    0x403200(,%rdx,4),%ecx
  401679:       83 c0 01                add    $0x1,%eax
  40167c:       eb e5                   jmp    401663 <phase_5+0x20>
  40167e:       e8 34 03 00 00          callq  4019b7 <explode_bomb>
  401683:       eb d4                   jmp    401659 <phase_5+0x16>
  401685:       83 f9 2b                cmp    $0x2b,%ecx
  401688:       75 07                   jne    401691 <phase_5+0x4e>
  40168a:       48 83 c4 08             add    $0x8,%rsp
  40168e:       5b                      pop    %rbx
  40168f:       5d                      pop    %rbp
  401690:       c3                      retq
  401691:       e8 21 03 00 00          callq  4019b7 <explode_bomb>
  401696:       eb f2                   jmp    40168a <phase_5+0x47>
```

第四行为 rsp 开辟 0x8 个字节的栈，第 5 行把输入的地址 rdi 给 rbx，第六行调用了 string_length，第七行把 eax 的值与 6 比较，不同则爆炸。想到之前的把输入放入 rbx 这个动作，可以推测这个函数是为了统计输入字符的个数，并存放在了 eax 中，且密码为 6 个字符。

```
  401659:       b9 00 00 00 00          mov    $0x0,%ecx
  40165e:       b8 00 00 00 00          mov    $0x0,%eax
  401663:       83 f8 05                cmp    $0x5,%eax
  401666:       7f 1d                   jg     401685 <phase_5+0x42>
  401668:       48 63 d0                movslq %eax,%rdx
  40166b:       0f b6 14 13             movzbl (%rbx,%rdx,1),%edx
  40166f:       83 e2 0f                and    $0xf,%edx
  401672:       03 0c 95 00 32 40 00    add    0x403200(,%rdx,4),%ecx
  401679:       83 c0 01                add    $0x1,%eax
  40167c:       eb e5                   jmp    401663 <phase_5+0x20>
```

这一段是一个 while 循环，首先循环变量 eax 置零，eax 符号扩展送往 rdx，rdx*2 零扩展送往 edx，然后将 edx 逐位与 1 相与，相加到 ecx。然后循环，直到处理完六个字符。最后比较 ecx 在数值与 0x2b，即十进制的 43。

结合 ASCII 的知识，针对字符 0-9，0-9 的 ASCII 码值为 0x30，0x31，0x32 0x33 ...0x39，因此与 0xf 按位与后只保留个位 0x0，0x1...0x9，故是以字符串形式读入六位数字。

通过 gdb 调试，得知输入为 1234567890 时，分别引起 ecx 的增加量为 10 6 1 12 16 9 3 4 7 2，由于中间并未检查输入次序，因此选择相加为 43 的数字即可。

所以，得出密码为 450367（数字选取不唯一，排序不唯一）

输入后如下：

```
demerzel@demerzel-virtual-machine:~/ccode/lab3/bomb273$ ./bomb ans.txt
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Phase 1 defused. How about the next one?
That's number 2.  Keep going!
Halfway there!
So you got that one.  Try this one.
Good work!  On to the next...
```

## 3.6 阶段 6 的破解与分析

密码如下：5 2 1 3 6 4

破解过程：

```
0000000000401698 <phase_6>:
  401698:       55                      push   %rbp
  401699:       48 89 e5                mov    %rsp,%rbp
  40169c:       41 55                   push   %r13
  40169e:       41 54                   push   %r12
  4016a0:       53                      push   %rbx
  4016a1:       48 83 ec 58             sub    $0x58,%rsp
  4016a5:       48 8d 75 c0             lea    -0x40(%rbp),%rsi
  4016a9:       e8 2b 03 00 00          callq  4019d9 <read_six_numbers>
  4016ae:       41 bc 00 00 00 00       mov    $0x0,%r12d
  4016b4:       eb 29                   jmp    4016df <phase_6+0x47>
  4016b6:       e8 fc 02 00 00          callq  4019b7 <explode_bomb>
  4016bb:       eb 37                   jmp    4016f4 <phase_6+0x5c>
  4016bd:       e8 f5 02 00 00          callq  4019b7 <explode_bomb>
  4016c2:       83 c3 01                add    $0x1,%ebx
  4016c5:       83 fb 05                cmp    $0x5,%ebx
  4016c8:       7f 12                   jg     4016dc <phase_6+0x44>
  4016ca:       49 63 c4                movslq %r12d,%rax
  4016cd:       48 63 d3                movslq %ebx,%rdx
  4016d0:       8b 7c 95 c0             mov    -0x40(%rbp,%rdx,4),%edi
  4016d4:       39 7c 85 c0             cmp    %edi,-0x40(%rbp,%rax,4)
  4016d8:       75 e8                   jne    4016c2 <phase_6+0x2a>
  4016da:       eb e1                   jmp    4016bd <phase_6+0x25>
  4016dc:       45 89 ec                mov    %r13d,%r12d
  4016df:       41 83 fc 05             cmp    $0x5,%r12d
  4016e3:       7f 19                   jg     4016fe <phase_6+0x66>
  4016e5:       49 63 c4                movslq %r12d,%rax
  4016e8:       8b 44 85 c0             mov    -0x40(%rbp,%rax,4),%eax
  4016ec:       83 e8 01                sub    $0x1,%eax
  4016ef:       83 f8 05                cmp    $0x5,%eax
  4016f2:       77 c2                   ja     4016b6 <phase_6+0x1e>
  4016f4:       45 8d 6c 24 01          lea    0x1(%r12),%r13d
  4016f9:       44 89 eb                mov    %r13d,%ebx
  4016fc:       eb c7                   jmp    4016c5 <phase_6+0x2d>
  4016fe:       be 00 00 00 00          mov    $0x0,%esi
  401703:       eb 18                   jmp    40171d <phase_6+0x85>
  401705:       48 8b 52 08             mov    0x8(%rdx),%rdx
  401709:       83 c0 01                add    $0x1,%eax
  40170c:       48 63 ce                movslq %esi,%rcx
  40170f:       39 44 8d c0             cmp    %eax,-0x40(%rbp,%rcx,4)
  401713:       7f f0                   jg     401705 <phase_6+0x6d>
  401715:       48 89 54 cd 90          mov    %rdx,-0x70(%rbp,%rcx,8)
  40171a:       83 c6 01                add    $0x1,%esi
```

由 14 和 15 行比较得知，若输入的个数比 5 大，则爆炸。故需要输入六个数据。通过 gdb 查看-0x40(%rbp)的地址，得知为输入数组的地址。

紧接着此段代码：

```
4016c2:       83 c3 01                add    $0x1,%ebx
4016c5:       83 fb 05                cmp    $0x5,%ebx
4016c8:       7f 12                   jg     4016dc <phase_6+0x44>
4016ca:       49 63 c4                movslq %r12d,%rax
4016cd:       48 63 d3                movslq %ebx,%rdx
4016d0:       8b 7c 95 c0             mov    -0x40(%rbp,%rdx,4),%edi
4016d4:       39 7c 85 c0             cmp    %edi,-0x40(%rbp,%rax,4)
4016d8:       75 e8                   jne    4016c2 <phase_6+0x2a>
```

对栈中的数组元素进行比较，当元素减一比 5 大时爆炸

```
4016dc:       45 89 ec                mov    %r13d,%r12d
4016df:       41 83 fc 05             cmp    $0x5,%r12d
4016e3:       7f 19                   jg     4016fe <phase_6+0x66>
4016e5:       49 63 c4                movslq %r12d,%rax
4016e8:       8b 44 85 c0             mov    -0x40(%rbp,%rax,4),%eax
4016ec:       83 e8 01                sub    $0x1,%eax
4016ef:       83 f8 05                cmp    $0x5,%eax
4016f2:       77 c2                   ja     4016b6 <phase_6+0x1e>
4016f4:       45 8d 6c 24 01          lea    0x1(%r12),%r13d
4016f9:       44 89 eb                mov    %r13d,%ebx
4016fc:       eb c7                   jmp    4016c5 <phase_6+0x2d>
```

此段比较数组中前后两个元素，当两个元素相等时爆炸，即数组中为 0123456 中不重复的 6 个数，由 `sub $0x1, eax` 与 `cmp $0x5, %eax` 与 `ja 4016b6` 得知，进行无符号数比较前进行了减一操作，故不能输入 0，为 123456 这六个数字，具体输入顺序由以下破解。

```
4016fe:       be 00 00 00 00          mov    $0x0,%esi
401703:       eb 18                   jmp    40171d <phase_6+0x85>
401705:       48 8b 52 08             mov    0x8(%rdx),%rdx
401709:       83 c0 01                add    $0x1,%eax
40170c:       48 63 ce                movslq %esi,%rcx
40170f:       39 44 8d c0             cmp    %eax,-0x40(%rbp,%rcx,4)
401713:       7f f0                   jg     401705 <phase_6+0x6d>
401715:       48 89 54 cd 90          mov    %rdx,-0x70(%rbp,%rcx,8)
40171a:       83 c6 01                add    $0x1,%esi
40171d:       83 fe 05                cmp    $0x5,%esi
401720:       7f 0c                   jg     40172e <phase_6+0x96>
401722:       b8 01 00 00 00          mov    $0x1,%eax
401727:       ba d0 52 40 00          mov    $0x4052d0,%edx
40172c:       eb de                   jmp    40170c <phase_6+0x74>
40172e:       48 8b 5d 90             mov    -0x70(%rbp),%rbx
401732:       48 89 d9                mov    %rbx,%rcx
401735:       b8 01 00 00 00          mov    $0x1,%eax
40173a:       eb 12                   jmp    40174e <phase_6+0xb6>
40173c:       48 63 d0                movslq %eax,%rdx
40173f:       48 8b 54 d5 90          mov    -0x70(%rbp,%rdx,8),%rdx
401744:       48 89 51 08             mov    %rdx,0x8(%rcx)
401748:       83 c0 01                add    $0x1,%eax
40174b:       48 89 d1                mov    %rdx,%rcx
40174e:       83 f8 05                cmp    $0x5,%eax
401751:       7e e9                   jle    40173c <phase_6+0xa4>
```

一开始先将 esi 归零，然后跳到第 40171d 行处执行。第 40171d 行中把 esi 与 5 比较，然后将 eax 置一，edx 赋地址 0x4052d0，之后跳回循环，对数组中下一元素进行比较，后跳转到 40172e 处，后循环对 rdx 进行赋地址，每次为 -0x70(%rbp,%rdx,8)，故猜测这是一个有 8 个字节的链表，共两个元素，第一个是 4 个字节的 int 型，第二个四字节是指向下一节点的指针，打印 0x4052d0 处的地址：

```
(gdb) x /12xg0x4052d0
0x4052d0 <node1>:       0x00000001000001fa      0x00000000004052e0
0x4052e0 <node2>:       0x0000000200000110      0x00000000004052f0
0x4052f0 <node3>:       0x00000003000002ee      0x0000000000405300
0x405300 <node4>:       0x00000004000003d7      0x0000000000405310
0x405310 <node5>:       0x0000000500000063      0x0000000000405320
0x405320 <node6>:       0x00000006000003ce      0x0000000000000000
```

123456 分别对应六个节点，地址为第一列的倒数三位。

注意到此段代码：

```
  40175b:       41 bc 00 00 00 00       mov    $0x0,%r12d
  401761:       eb 08                   jmp    40176b <phase_6+0xd3>
  401763:       48 8b 5b 08             mov    0x8(%rbx),%rbx
  401767:       41 83 c4 01             add    $0x1,%r12d
  40176b:       41 83 fc 04             cmp    $0x4,%r12d
  40176f:       7f 11                   jg     401782 <phase_6+0xea>
  401771:       48 8b 43 08             mov    0x8(%rbx),%rax
  401775:       8b 00                   mov    (%rax),%eax
  401777:       39 03                   cmp    %eax,(%rbx)
  401779:       7e e8                   jle    401763 <phase_6+0xcb>
  40177b:       e8 37 02 00 00          callq  4019b7 <explode_bomb>
  401780:       eb e1                   jmp    401763 <phase_6+0xcb>
  401782:       48 83 c4 58             add    $0x58,%rsp
```

最后对链表的元素进行比较大小，任何时候，如果前面一个节点的值比后一个节点的值大，则爆炸，故需要 123456 的地址按升序排列，所以顺序为 5 2 1 3 6 4

输入后如下：

```
demerzel@demerzel-virtual-machine:~/ccode/lab3/bomb273$ ./bomb ans.txt
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Phase 1 defused. How about the next one?
That's number 2.  Keep going!
Halfway there!
So you got that one.  Try this one.
Good work!  On to the next...
Congratulations! You've defused the bomb!
```

## 3.7 阶段 7 的破解与分析(隐藏阶段)

密码如下：107

破解过程：

在汇编的过程中已经注意到了有<secret_phase>以及<fun7>，在反汇编文件里直接搜索<secret_phase>的进入地址，发现在每个阶段的 phase_x 之后都有一行 phase_defused，就在这个函数里面存在 callq secret_phase 的代码。分析这个 phase_defused：

```
0000000000401b46 <phase_defused>:
  401b46:       83 3d 1f 3c 00 00 06    cmpl   $0x6,0x3c1f(%rip)        # 40576c <num_input_strings>
  401b4d:       74 01                   je     401b50 <phase_defused+0xa>
  401b4f:       c3                      retq
  401b50:       55                      push   %rbp
  401b51:       48 89 e5                mov    %rsp,%rbp
  401b54:       48 83 ec 60             sub    $0x60,%rsp
  401b58:       4c 8d 45 b0             lea    -0x50(%rbp),%r8
  401b5c:       48 8d 4d a8             lea    -0x58(%rbp),%rcx
  401b60:       48 8d 55 ac             lea    -0x54(%rbp),%rdx
  401b64:       be 99 33 40 00          mov    $0x403399,%esi
  401b69:       bf 70 58 40 00          mov    $0x405870,%edi
  401b6e:       b8 00 00 00 00          mov    $0x0,%eax
  401b73:       e8 a8 f5 ff ff          callq  401120 <__isoc99_sscanf@plt>
  401b78:       83 f8 03                cmp    $0x3,%eax
  401b7b:       74 0c                   je     401b89 <phase_defused+0x43>
  401b7d:       bf d8 32 40 00          mov    $0x4032d8,%edi
  401b82:       e8 d9 f4 ff ff          callq  401060 <puts@plt>
  401b87:       c9                      leaveq
  401b88:       c3                      retq
  401b89:       be a2 33 40 00          mov    $0x4033a2,%esi
  401b8e:       48 8d 7d b0             lea    -0x50(%rbp),%rdi
  401b92:       e8 21 fd ff ff          callq  4018b8 <strings_not_equal>
  401b97:       85 c0                   test   %eax,%eax
  401b99:       75 e2                   jne    401b7d <phase_defused+0x37>
  401b9b:       bf 78 32 40 00          mov    $0x403278,%edi
  401ba0:       e8 bb f4 ff ff          callq  401060 <puts@plt>
  401ba5:       bf a0 32 40 00          mov    $0x4032a0,%edi
  401baa:       e8 b1 f4 ff ff          callq  401060 <puts@plt>
  401baf:       b8 00 00 00 00          mov    $0x0,%eax
  401bb4:       e8 0e fc ff ff          callq  4017c7 <secret_phase>
  401bb9:       eb c2                   jmp    401b7d <phase_defused+0x37>
```

可以看到第 7 行将函数__isoc99_sscanf@plt 的返回值与 3 进行比较，如果不等于 3 则的直接跳过中间代码到达最后的结束部分。从函数名我们可以推测这个函数的作用的是检测读取的字符串的数量，当读取了 3 个字符串时，就不会跳过中间的代码。继续看中间的代码：第 9 到 14 行又是熟悉的 sscanf 调用过程，已经知道 esi 指向的是格式化字符串的首地址，先查看它的内容：

```
(gdb) x/s 0x403399
0x403399:       "%d %d %s"
```

读取两个整数和一个字符串。有所不同的是之后又有一行给 edi 赋上了一个地址值，之前所有阶段中 edi 的值都是来自于 read_line 的地址，想到 sscanf 参数中确实存在一个输入，可以推测这个 edi 中存放的是读取位置的首地址。

esi 被赋上了一个地址值，edi 被赋上了 esp+0x10，可以推测出 edi 的地址就是指向读入的第三个字符串的，用 gdb 查看内存的内容：

```
(gdb) x/s 0x405870
0x405870 <input_strings+240>:    "14 7 DrEvil"
```

即 phase_4 的值与隐藏进入的入口。

查看 secret_phase 代码：

```
00000000004017c7 <secret_phase>:
  4017c7:       55                      push   %rbp
  4017c8:       48 89 e5                mov    %rsp,%rbp
  4017cb:       53                      push   %rbx
  4017cc:       48 83 ec 08             sub    $0x8,%rsp
  4017d0:       e8 40 02 00 00          callq  401a15 <read_line>
  4017d5:       ba 0a 00 00 00          mov    $0xa,%edx
  4017da:       be 00 00 00 00          mov    $0x0,%esi
  4017df:       48 89 c7                mov    %rax,%rdi
  4017e2:       e8 09 f9 ff ff          callq  4010f0 <strtol@plt>
  4017e7:       48 89 c3                mov    %rax,%rbx
  4017ea:       8d 40 ff                lea    -0x1(%rax),%eax
  4017ed:       3d e8 03 00 00          cmp    $0x3e8,%eax
  4017f2:       77 27                   ja     40181b <secret_phase+0x54>
  4017f4:       89 de                   mov    %ebx,%esi
  4017f6:       bf f0 50 40 00          mov    $0x4050f0,%edi
  4017fb:       e8 8d ff ff ff          callq  40178d <fun7>
  401800:       83 f8 03                cmp    $0x3,%eax
  401803:       75 1d                   jne    401822 <secret_phase+0x5b>
  401805:       bf 88 31 40 00          mov    $0x403188,%edi
  40180a:       e8 51 f8 ff ff          callq  401060 <puts@plt>
  40180f:       e8 32 03 00 00          callq  401b46 <phase_defused>
  401814:       48 83 c4 08             add    $0x8,%rsp
  401818:       5b                      pop    %rbx
  401819:       5d                      pop    %rbp
  40181a:       c3                      retq
  40181b:       e8 97 01 00 00          callq  4019b7 <explode_bomb>
  401820:       eb d2                   jmp    4017f4 <secret_phase+0x2d>
  401822:       e8 90 01 00 00          callq  4019b7 <explode_bomb>
  401827:       eb dc                   jmp    401805 <secret_phase+0x3e>
```

第 5 行调用了 read_line 函数，接着把 read_line 的返回值赋给了 rdi，并调用了 strtol 函数，这个标准库函数的作用是把一个字符串转换成对应的长整型数值。返回值还是存放在 rax 中，第 10 行将 rax 复制给了 rbx，第 11 行将 rax 减 1 赋给 eax，第 12 行与 0x3e8 进行比较，如果这个值小于等于 0x3e8 就跳过引爆代码。看到这里我们可以知道我们需要再加入一行数据，它应该是一个小于等于 1001 的数值。接下来将 ebx 赋给了 esi，也就是一开始输入的 rax 值。第 15 行将一个地址值赋给了 edi，16 行调用了 fun7 函数。此函数返回后令返回值 eax 与 0x3 做了一个比较，如果相等则跳过引爆代码。所以我们需要返回 3。查看 fun7 的代码：

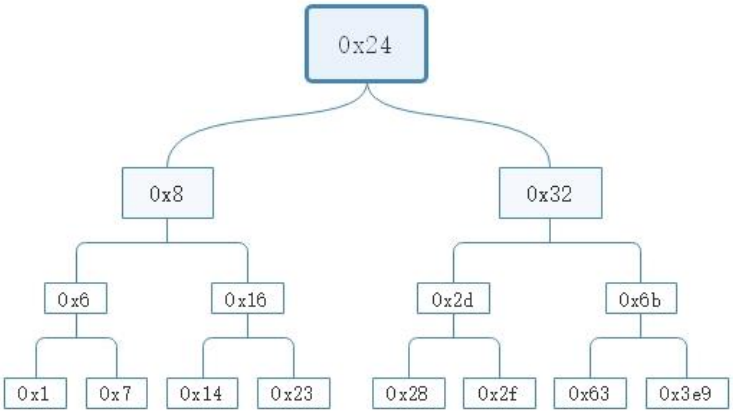
```
000000000040178d <fun7>:
  40178d:       48 85 ff                test   %rdi,%rdi
  401790:       74 2f                   je     4017c1 <fun7+0x34>
  401792:       55                      push   %rbp
  401793:       48 89 e5                mov    %rsp,%rbp
  401796:       8b 07                   mov    (%rdi),%eax
  401798:       39 f0                   cmp    %esi,%eax
  40179a:       7f 09                   jg     4017a5 <fun7+0x18>
  40179c:       75 14                   jne    4017b2 <fun7+0x25>
  40179e:       b8 00 00 00 00          mov    $0x0,%eax
  4017a3:       5d                      pop    %rbp
  4017a4:       c3                      retq
  4017a5:       48 8b 7f 08             mov    0x8(%rdi),%rdi
  4017a9:       e8 df ff ff ff          callq  40178d <fun7>
  4017ae:       01 c0                   add    %eax,%eax
  4017b0:       eb f1                   jmp    4017a3 <fun7+0x16>
  4017b2:       48 8b 7f 10             mov    0x10(%rdi),%rdi
  4017b6:       e8 d2 ff ff ff          callq  40178d <fun7>
  4017bb:       8d 44 00 01             lea    0x1(%rax,%rax,1),%eax
  4017bf:       eb e2                   jmp    4017a3 <fun7+0x16>
  4017c1:       b8 ff ff ff ff          mov    $0xffffffff,%eax
  4017c6:       c3                      retq
```

第 1、2 两行先对我们输入的这个数作一个判断，如果等于 0 直接跳到第 20 行，返回-1。第 5 行将 rdi 的值读入到了 eax 中，第 6 行则将这个数与我们读入的数进行比较，如果这个数小于等于我们读入的数就跳至第 12 行，第 12 行将 rdi 加 8，再进行一次调用 fun7；如果不相等则跳至第 16 行，将 rdi 加 10，再次调用 fun7。这几处代码有些与之前的链表跳至下一个节点类似，需要查看一下 rdi 这个地址里存放的是怎样一种数据结构：

```
(gdb) x/128xg 0x4050f0
0x4050f0 <n1>:  0x0000000000000024      0x0000000000405110
0x405100 <n1+16>:       0x0000000000405130      0x0000000000000000
0x405110 <n21>: 0x0000000000000008      0x0000000000405190
0x405120 <n21+16>:      0x0000000000405150      0x0000000000000000
0x405130 <n22>: 0x0000000000000032      0x0000000000405170
0x405140 <n22+16>:      0x00000000004051b0      0x0000000000000000
0x405150 <n32>: 0x0000000000000016      0x0000000000405270
0x405160 <n32+16>:      0x0000000000405230      0x0000000000000000
0x405170 <n33>: 0x000000000000002d      0x00000000004051d0
0x405180 <n33+16>:      0x0000000000405290      0x0000000000000000
0x405190 <n31>: 0x0000000000000006      0x00000000004051f0
0x4051a0 <n31+16>:      0x0000000000405250      0x0000000000000000
0x4051b0 <n34>: 0x000000000000006b      0x0000000000405210
0x4051c0 <n34+16>:      0x00000000004052b0      0x0000000000000000
0x4051d0 <n45>: 0x0000000000000028      0x0000000000000000
0x4051e0 <n45+16>:      0x0000000000000000      0x0000000000000000
0x4051f0 <n41>: 0x0000000000000001      0x0000000000000000
0x405200 <n41+16>:      0x0000000000000000      0x0000000000000000
0x405210 <n47>: 0x0000000000000063      0x0000000000000000
0x405220 <n47+16>:      0x0000000000000000      0x0000000000000000
0x405230 <n44>: 0x0000000000000023      0x0000000000000000
0x405240 <n44+16>:      0x0000000000000000      0x0000000000000000
0x405250 <n42>: 0x0000000000000007      0x0000000000000000
---Type <return> to continue, or q <return> to quit---
0x405260 <n42+16>:      0x0000000000000000      0x0000000000000000
0x405270 <n43>: 0x0000000000000014      0x0000000000000000
0x405280 <n43+16>:      0x0000000000000000      0x0000000000000000
0x405290 <n46>: 0x000000000000002f      0x0000000000000000
0x4052a0 <n46+16>:      0x0000000000000000      0x0000000000000000
0x4052b0 <n48>: 0x00000000000003e9      0x0000000000000000
0x4052c0 <n48+16>:      0x0000000000000000      0x0000000000000000
```

仔细观察可以发现这是一个二叉树的结构，每个节点第 1 个 8 字节存放数据，第 2 个 8 字节存放左子树地址，第 3 个 8 字节存放右子树位置。并且命令也有规律，nab，a 代表层数，b 代表从左至右第 b 个节点。

用图表表示如下：




回到代码，得知第 16 行代码的作用是，如果节点值小于读入的数，将 rdi 移

到它的右子树的位置，接着调用 fun7，在返回后令 eax = 2 * rax + 1。

如果节点值大于读入的数，代码会进行到第 12 行，令 rdi 移到它的左子树的位置，接下来调用 fun7 在返回后令 eax = 2 * eax。下面跳至返回处。

总结上面的过程：edi 指向一个树的节点，令 edi 节点的值与我们读入的值进行比较。如果两者相等：返回-1。

如果前者大于后者：rdi 移至左子树，返回 2 * rax

如果后者大于前者：rdi 移至右子树，返回 2 * rax + 1

那么我们需要返回 3，应该在最后一次调用返回 1，倒数第二次调用返回 0 ，第一次调用返回 0。换句话说，这个数应该在第三层，比父节点大且比根结节大。观察上图，唯一的答案是：0x6b （107），输入后如下：

```
demerzel@demerzel-virtual-machine:~/ccode/lab3/bomb273$ ./bomb ans.txt
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Phase 1 defused. How about the next one?
That's number 2.  Keep going!
Halfway there!
So you got that one.  Try this one.
Good work!  On to the next...
Curses, you've found the secret phase!
But finding it and solving it are quite different...
Wow! You've defused the secret stage!
Congratulations! You've defused the bomb!
```

至此破解所有密码！

# 第 4 章 总结

## 4.1 请总结本次实验的收获

本次实验详细的训练应用了各种反汇编代码，在逐级递增的难度中训练了循环、递归、指针、链表与二叉树。十分全面的复习了反汇编的代码与具体操作办法。

## 4.2 请给出对本次实验内容的建议

希望可以继续沿用此项实验，整体非常好！

注：本章为酌情加分项。

# 参考文献

[1] 林来兴. 空间控制技术[M]. 北京：中国宇航出版社，1992：25-42.

[2] 辛希孟. 信息技术与信息服务国际研讨会论文集：A 集[C]. 北京：中国科学出版社，1999.

[3] 赵耀东. 新时代的工业工程师[M/OL]. 台北：天下文化出版社，1998 [1998-09-26]. http://www.ie.nthu.edu.tw/info/ie.newie.htm（Big5）.

[4] 谌颖. 空间交会控制理论与方法研究[D]. 哈尔滨：哈尔滨工业大学，1992：8-13.

[5] KANAMORI H. Shaking Without Quaking[J]. Science，1998，279（5359）：2063-2064.

[6] CHRISTINE M. Plant Physiology: Plant Biology in the Genome Era[J/OL]. Science，1998，281：331-332[1998-09-23]. http://www.sciencemag.org/cgi/collection/anatmorp.