

**实验报告**

**实 验（三）**

题 目 Binary Bomb

二进制炸弹

专 业 计算机科学与技术

学　　 号 1171000410

班　　 级 1703005

学 生 强文杰

指 导 教 师 吴锐

实 验 地 点 G712

实 验 日 期 2018.10.21

**计算机科学与技术学院**

**目 录**

[第1章 实验基本信息 - 3 -](#_Toc496608242)

[1.1 实验目的 - 3 -](#_Toc496608243)

[1.2 实验环境与工具 - 3 -](#_Toc496608244)

[1.2.1 硬件环境 - 3 -](#_Toc496608245)

[1.2.2 软件环境 - 3 -](#_Toc496608246)

[1.2.3 开发工具 - 3 -](#_Toc496608247)

[1.3 实验预习 - 3 -](#_Toc496608248)

[第2章 实验环境建立 - 4 -](#_Toc496608249)

[2.1 Ubuntu下CodeBlocks反汇编（10分） - 4 -](#_Toc496608250)

[2.2 Ubuntu下EDB运行环境建立（10分） - 4 -](#_Toc496608251)

[第3章 各阶段炸弹破解与分析 - 5 -](#_Toc496608252)

[3.1 阶段1的破解与分析 - 5 -](#_Toc496608253)

[3.2 阶段2的破解与分析 - 5 -](#_Toc496608254)

[3.3 阶段3的破解与分析 - 5 -](#_Toc496608255)

[3.4 阶段4的破解与分析 - 5 -](#_Toc496608256)

[3.5 阶段5的破解与分析 - 5 -](#_Toc496608257)

[3.6 阶段6的破解与分析 - 5 -](#_Toc496608258)

[3.7 阶段7的破解与分析(隐藏阶段) - 6 -](#_Toc496608259)

[第4章 总结 - 7 -](#_Toc496608260)

[4.1 请总结本次实验的收获 - 7 -](#_Toc496608261)

[4.2 请给出对本次实验内容的建议 - 7 -](#_Toc496608262)

[参考文献 - 8 -](#_Toc496608263)

# 第1章 实验基本信息

## 实验目的

1．熟练掌握计算机系统的ISA指令系统与寻址方式

2．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

3．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

X64 CPU；2GHz；2G RAM；256GHD Disk 以上

### 1.2.2 软件环境

Windows7 64位以上；VirtualBox/Vmware 11以上；Ubuntu 16.04 LTS 64位/优麒麟 64位；

### 1.2.3 开发工具

GDB/OBJDUMP；EDB；KDD等

## 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。

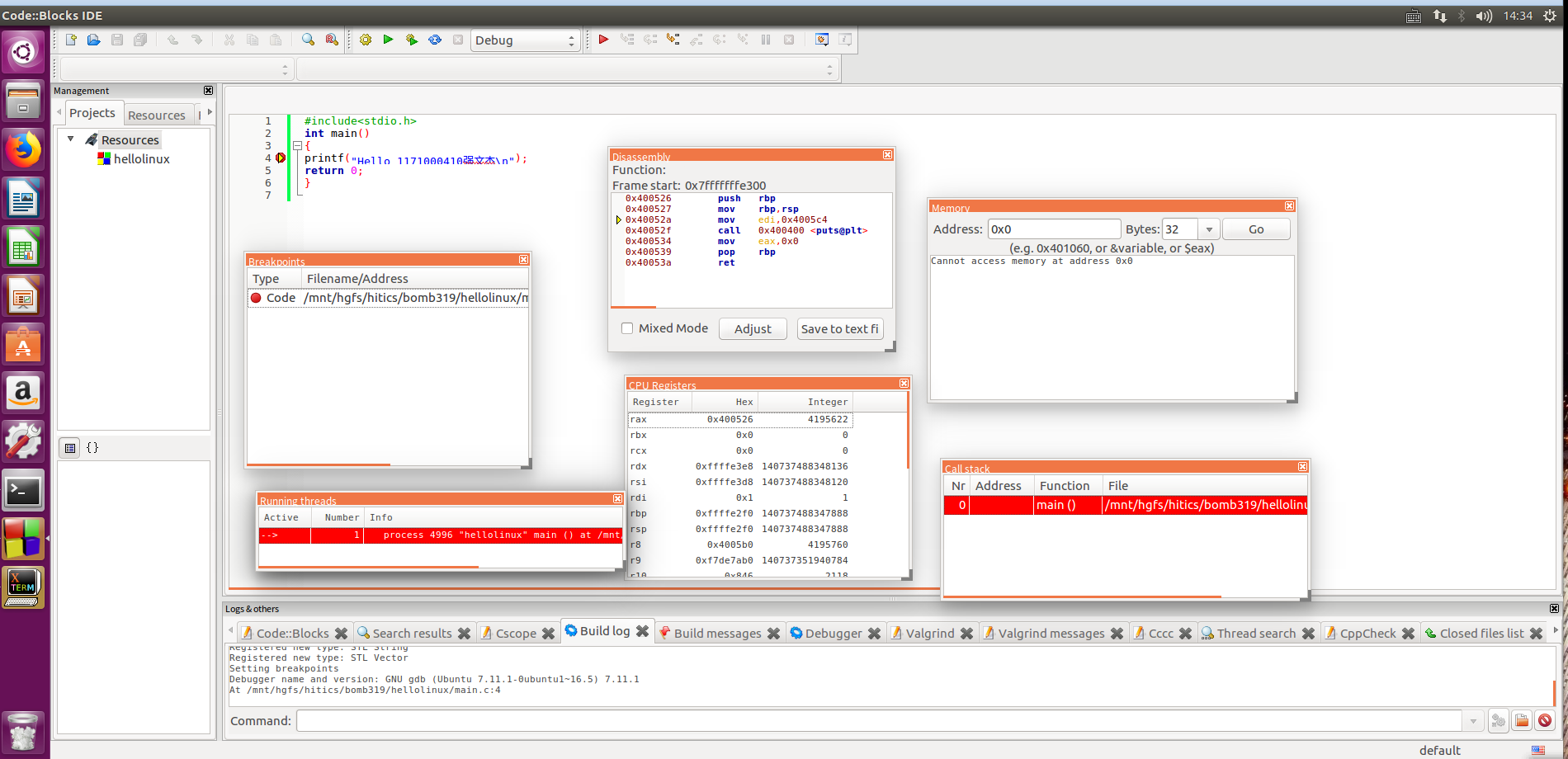


图2-1 Ubuntu下CodeBlocks反汇编截图

## 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

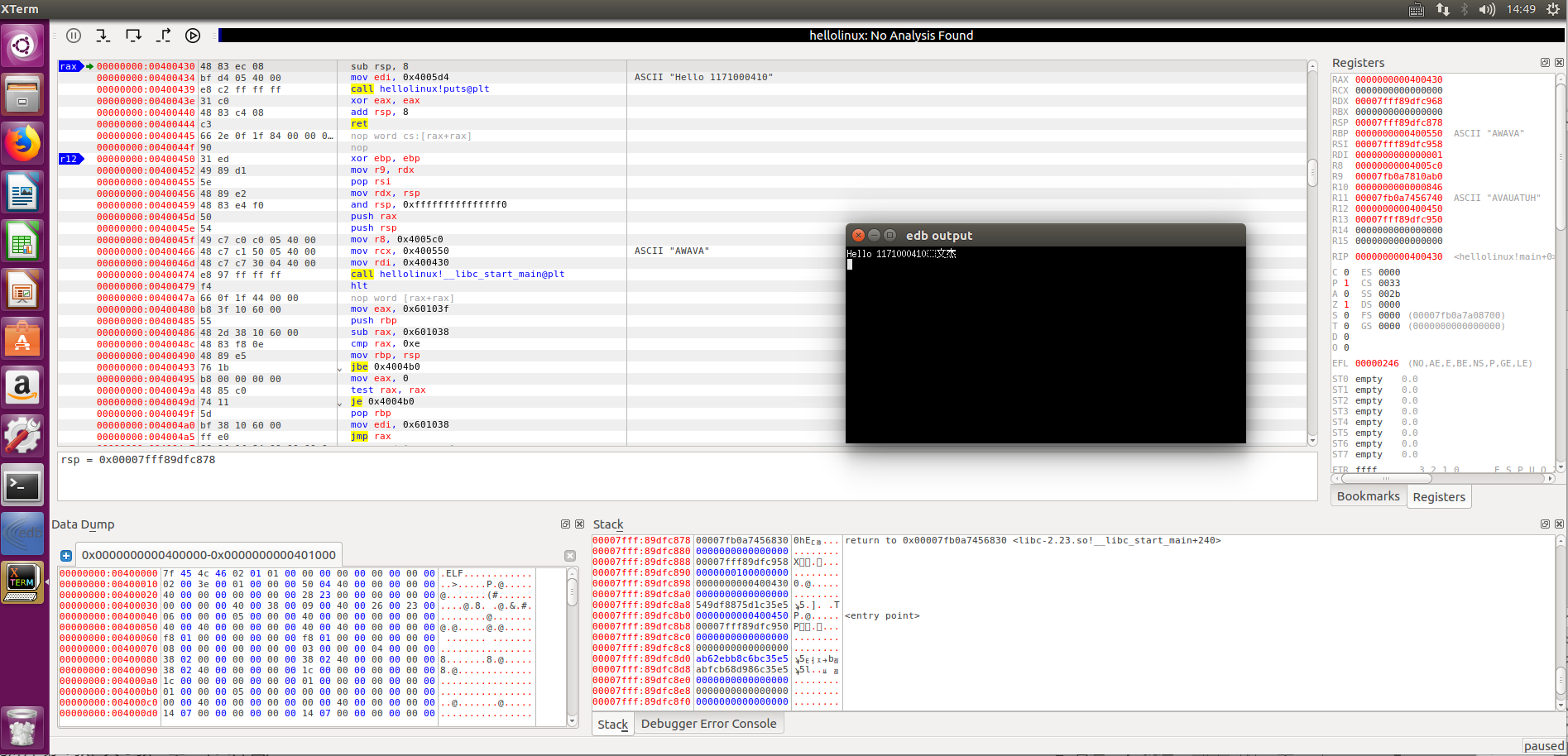


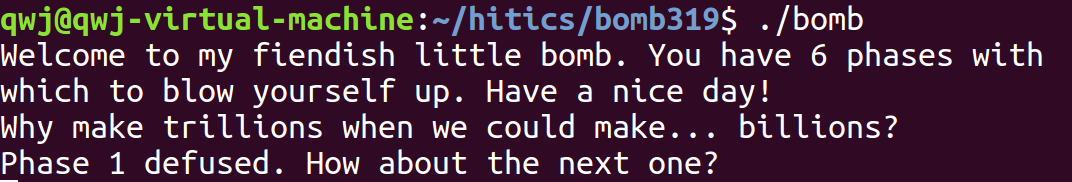
图2-2 Ubuntu下EDB截图

# 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

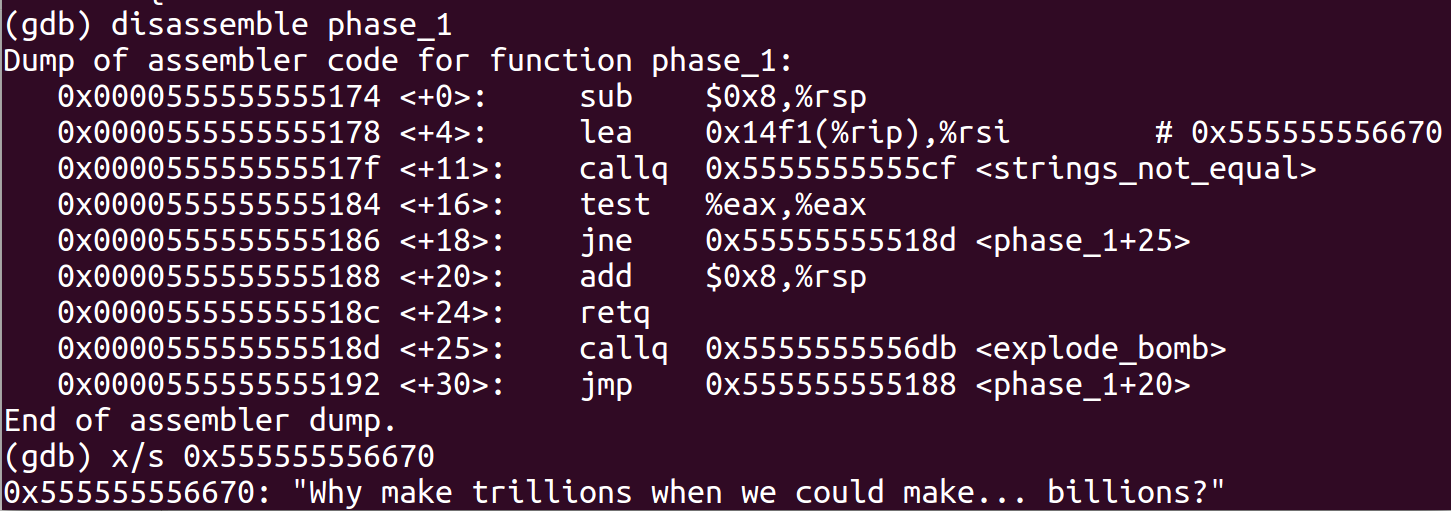
## 3.1 阶段1的破解与分析

密码如下：Why make trillions when we could make... billions?



破解过程：strings\_not\_equal函数是判断输入的字符串和%rsi中的字符串是否相等，如果相等则令%eax为0，否则为1。

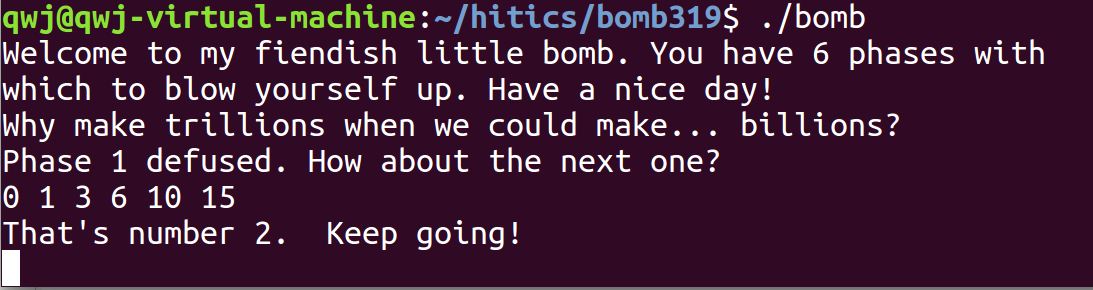
并且后面的test %eax,%eax和jne 0x55555555518d则是判断若%eax不为0，就执行explode\_bomb。因此需要输入的密码必须与%rsi中的字符串相等。



gdb通过x/s 0x555555556670查看其中的数据，即得密码。

## 3.2 阶段2的破解与分析

密码如下：其中一组密码为：0 1 3 6 10 15



破解过程：第一步：phase\_2的代码需要输入6个数字，并且当我进入read\_six\_numbers函数，通过查看%rsi时，发现其为%d %d %d %d %d %d，由此可知，函数读入的数据是6个int类型，这对栈的分析至关重要。

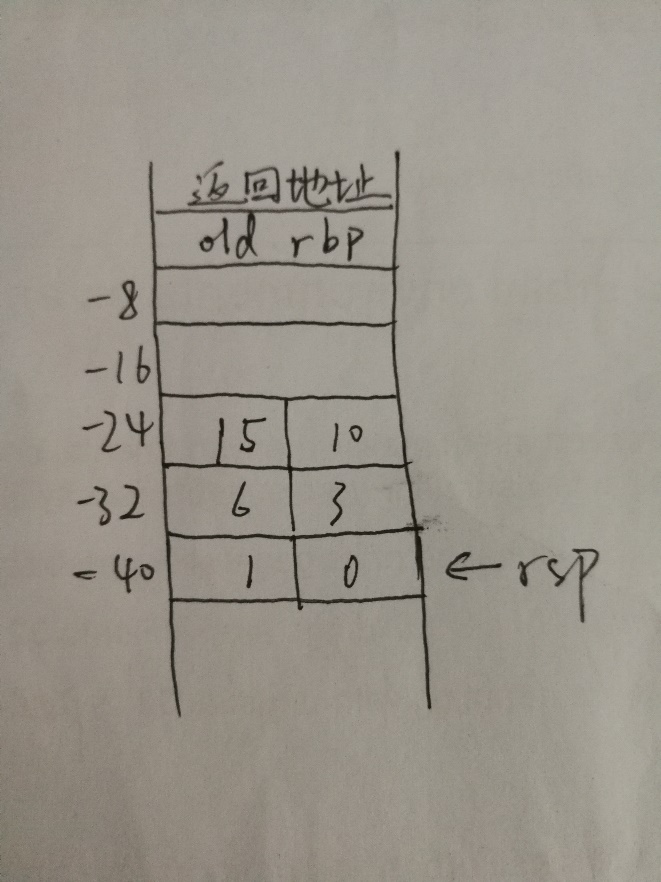
第二步：查看函数phase\_2。通过cmpl $0x0,(%rsp)和后一句js条件跳转语句可以分析第一个数大于等于0。暂且输入第一个数为0。%ebx的值由0x1执行循环，每次加0x1，当%ebx为0x6时退出循环。每次循环，将%ebx赋值给%eax。分析主要的语句可以得出每次循环内，%eax+（4%rbx-4+%rbp）与（4%rbx+%rbp）的值相等，而（4%rbx-4+%rbp）取址后为上一个输入的值，（4%rbx+%rbp）为本次输入的值。由此不难得出函数的循环大致为：

int a[6];

a[0]=0;

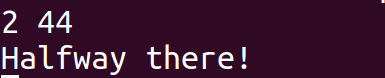
for (i=1;i<6;i++)

{ a[i]=a[i-1]+I ; }

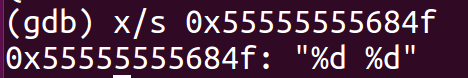
附上此时栈的示意图：

## 3.3 阶段3的破解与分析

密码如下：其中一组密码为：2 44



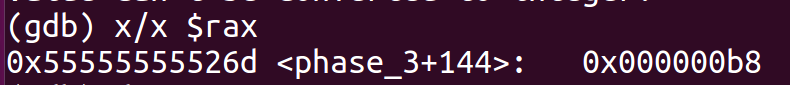
破解过程：



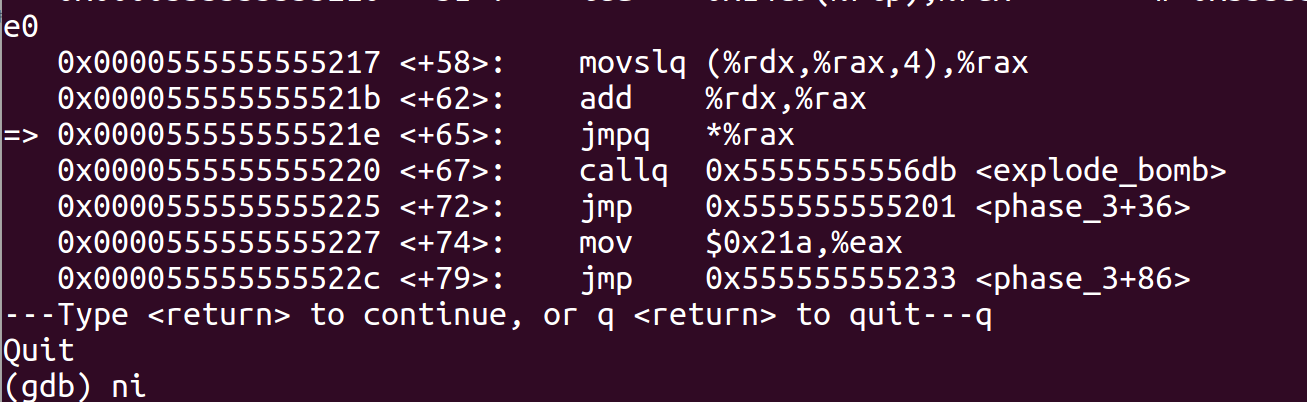
通过查看%rsi中的字符，可知输入的是两个int型数据。sscanf返回值为读入有效数据的个数，并且由$0x7,0xc(%rsp)可知，第一个数小于7。

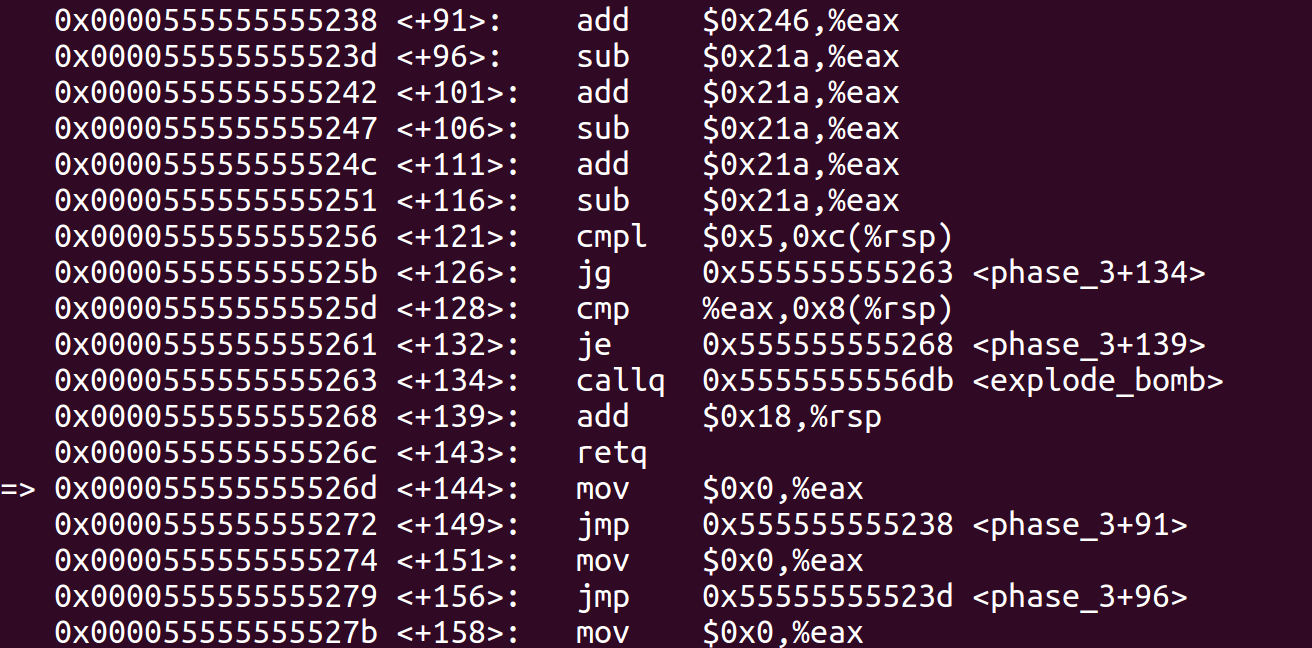
于是先输入2 2，使用gdb观察下面的执行。

观察到jmpq \*%rax语句，表示以%rax中的值作为跳转目标，此时查看%rax中十六进制的值如下：

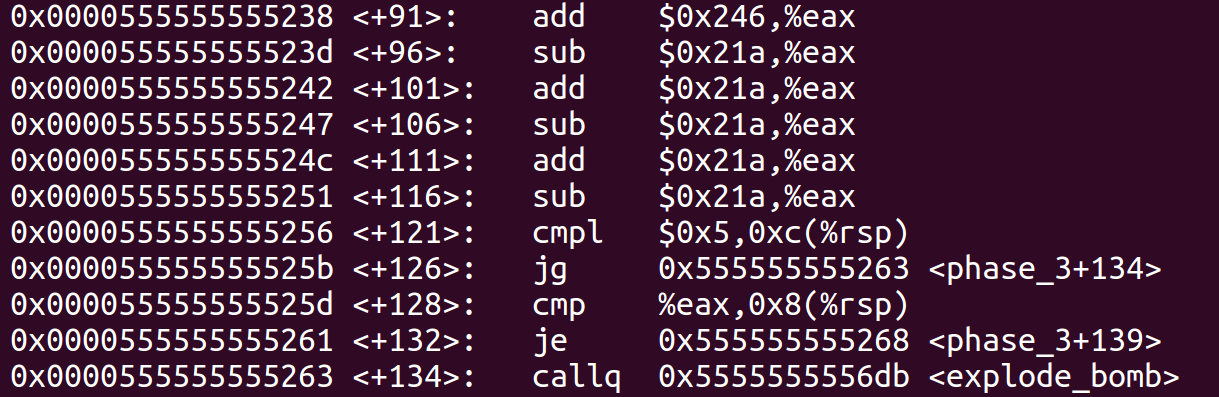


因此jmp \*rax即表示跳转到0x55555555526d，再单步执行程序。





跳转到0x55555555526d后，mov $0x0,%eax 表示将%rax赋值为0x0后再跳转到0x555555555238进行计算。



由cmpl $0x5 , 0xc(%rsp)和jg 0x555555555263可知第一个值为小于5的无符号数，而第二个数值等于计算后%eax中的值。

计算过程是从地址0x555555555238开始：

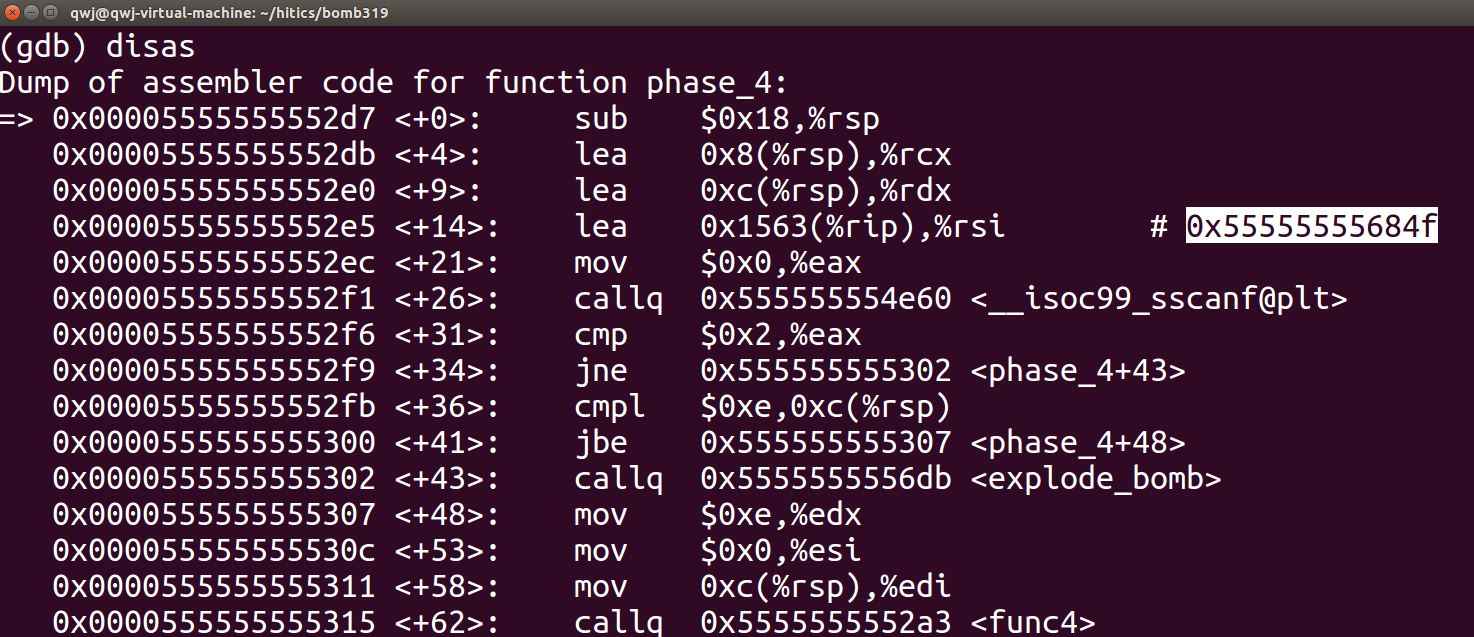
0+0x246-0x21a+0x21a-0x21a+0x21a -0x21a =44，因此第一个数输入2时，第二个数为44.

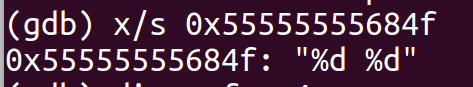
综上，此题其实是一条switch语句，根据输入小于5的第一个数值，可计算出 \*%rax，即跳转的地址，然后计算求出第二个数值，即得通关密码。

## 3.4 阶段4的破解与分析

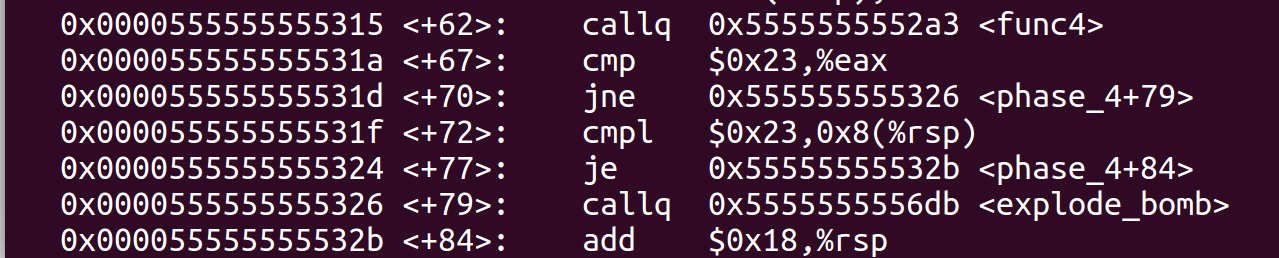
密码如下：8 35

破解过程：

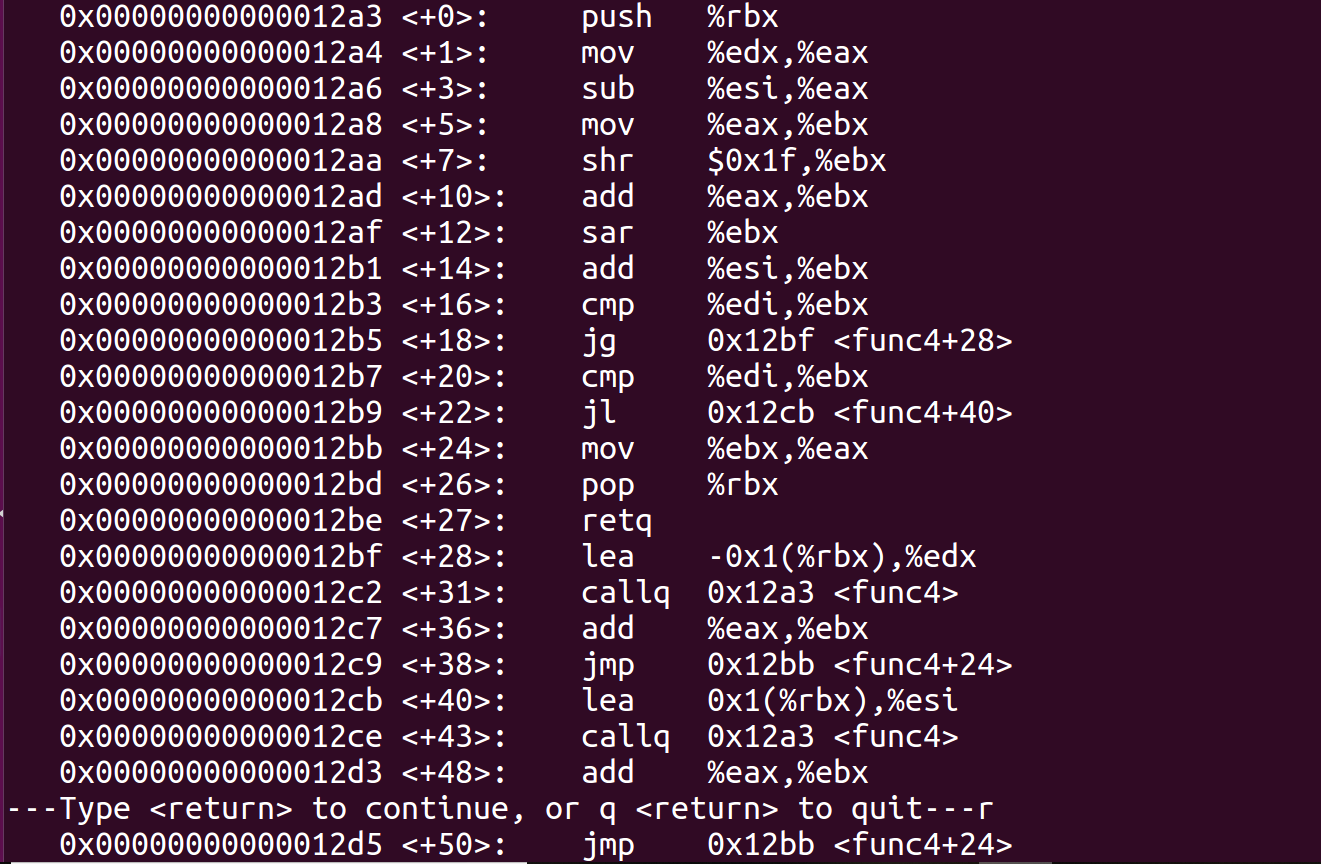




第一步，先观察调用func4之前的phase\_4，通过查看%rsi，和调用\_\_isoc99\_sscanf时返回值为2可以知道，我们输入的是两个int型数据，并且由cmpl $0xe,0xc(%rsp) 和 jbe 0x555555555307可以知道输入的第一个数小于14，进入func4之前%edx被赋值为14，%esi被赋值为0，%edi被赋值为我们输入的第一个数。



第二步，观察func4调用之后的phase\_4，可以知道输入的第二个数是35，并且fun4的返回值%eax也是35。



第三步，进入func4查看。

通过对上语句的分析，我将func简化为如下形式：

%edx初值为14，%esi初值为0,

x=%rbx, y=%eax, z=%edx, a=%esi, a1=%edi

<fun4>

y=z ;

x=(y+a)/2;

1 .if (x>a1)

z=x-1

call <func4>

y+=x;

2 .if (x<a1)

a =x+1;

call<fun4>

y+=x;

3 .if (x=a1)

y=x;

由以上简化的调用形式可知，递归的最里层是x=a1，由于每次调用<fun4>之前都要pop %rbx，因此递归返回计算时x的值是不断变化的。

返回值%rax，即设置的y，有y= a1 +x1 +x2 +x3…

进行多次计算如下：

%edi = 0 , %rax = 11

%edi = 1 , %rax = 11

%edi = 2, %rax = 13

%edi = 3 , %rax = 10

%edi = 4 , %rax = 19

%edi = 5 , %rax = 15

%edi = 6 , %rax = 21

%edi = 7 , %rax = 7

%edi = 8 , %rax = 35

%edi = 9 , %rax = 27

%edi = 10 , %rax = 37

%edi = 11 , %rax = 18

%edi = 12 , %rax = 43

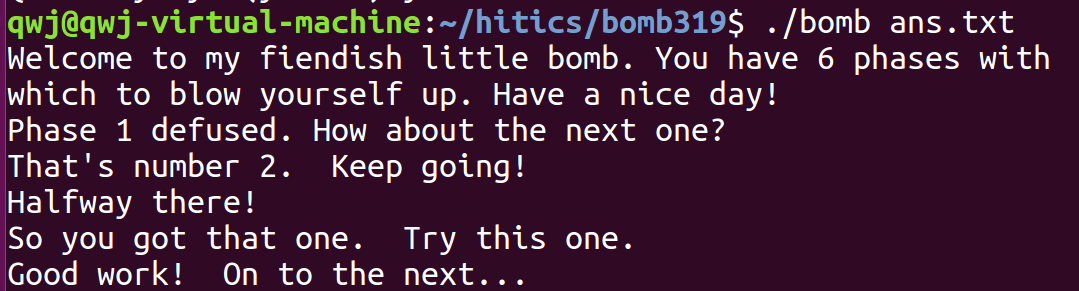
%edi = 13 , %rax = 31

%edi = 14 , %rax = 45

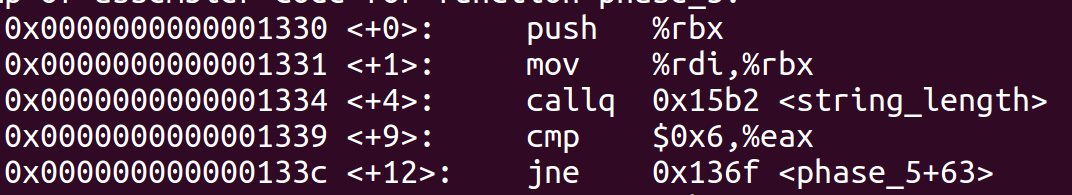
由返回值%eax为35可以知道输入第一个数为8。综上，输入8 35.

## 3.5 阶段5的破解与分析

密码如下：其中一组为444440



破解过程：

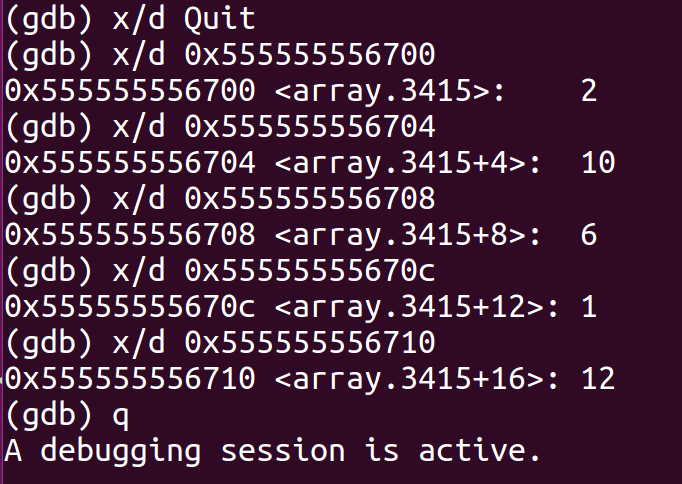


首先通过string\_length函数，及cmp $0x6,%eax语句，可知密码为长度为6的字符串。

再由movzbl (%rax),%edx及and $0xf,%edx可知，将输入字符0拓展为32为赋值给%edx再取%edx的后四位。

另外，%rax每次加一，cmp %rdi,%rax条件判断语句控制循环。

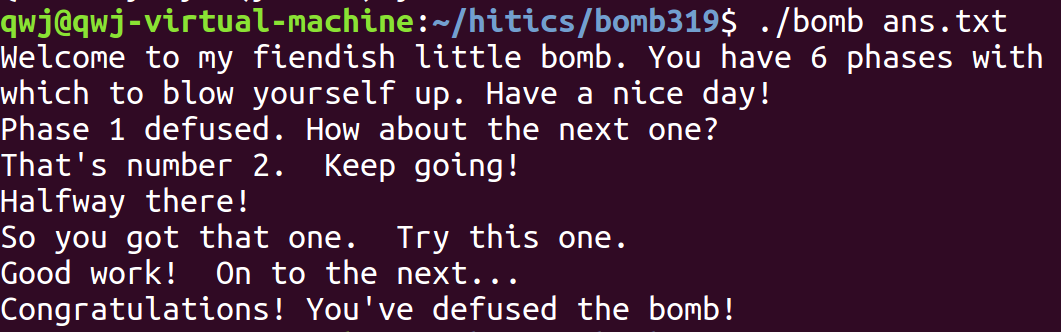
其中最重要的是add (%rsi,%rdx,4), %ecx,这条语句是指将（%rsi+4%rdx）该地址对应的值赋给%ecx，于是gdb调试出当%rdx分别为0，1，2，3，4，时，（%rsi+4%rdx）地址对应的值：



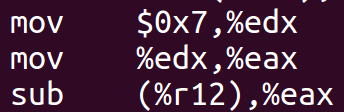
由于输出的%ecx的值为62，又根据累加出%ecx的一种计算可以为62=12+12+12+12+12+2，故逆推输入的字符串相对应为444440 。

## 3.6 阶段6的破解与分析

密码如下：1 6 4 5 2 3



破解过程：第一步：分析前面的的循环可知，read\_six\_numbers的六个参数，全部不大于6，并且互不相等。

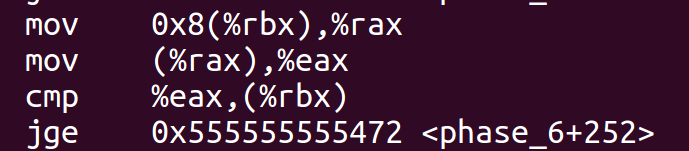
第二步：

分析后一个循环可知，输入的6个数，分别被7减，并且保存在原处。

第三步：根据7减去输入的六个数字的值，通过循环，%rsp+8n中分别存入不同的地址值。存值的方式是循环时不断对新的%rdx加0x8，其中%rdx初始值为0x555555758210

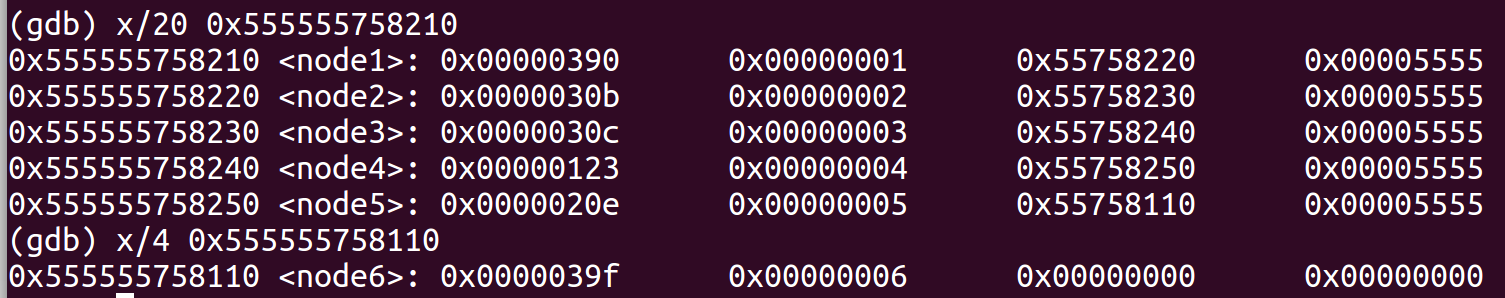
6

第四步：



由此可知，链表值由大到小排列。

第五步：



链表值由大到小排列为：0x39f 0x390 0x30c 0x30b 0x20e 0x123

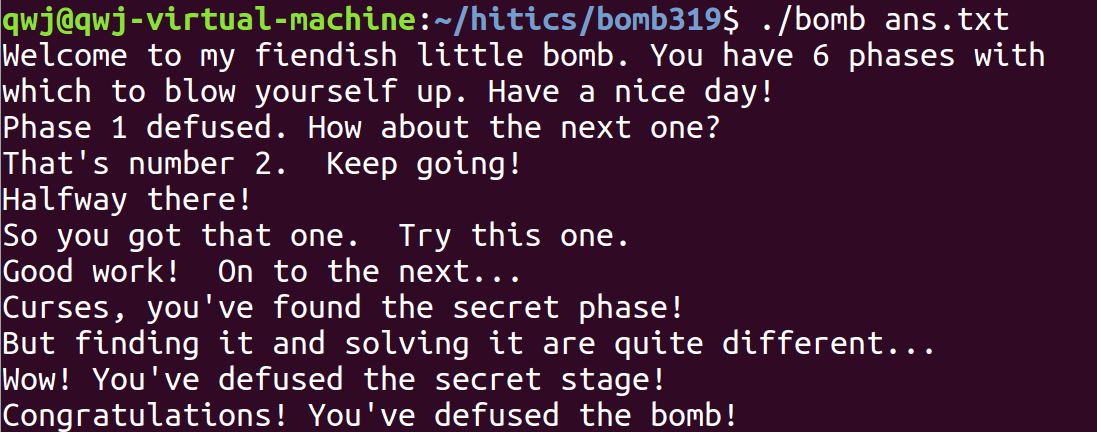
对应的值为：6 1 3 2 5 4

又因为对应的值是7减去输入数后的值。

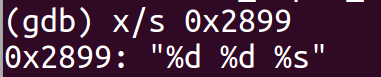
故输入的数对应为：1 6 4 5 2 3

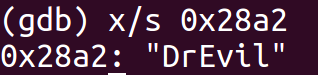
## 3.7 阶段7的破解与分析(隐藏阶段)

密码如下：1001



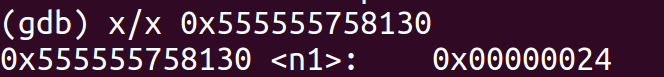
破解过程：根据phase\_defused的分析，可以知道必须解决完前六关才能进入隐藏阶段。

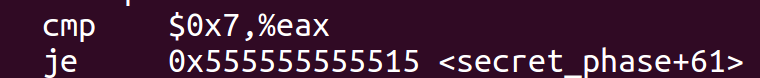




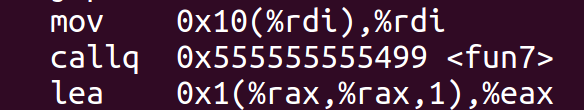
由以上截图可知，隐藏阶段是在输入两个int型数据后，再输入DrEvil，才可进入。

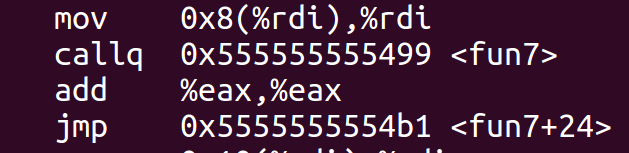
首先分析secret\_phase,由read\_line可知，输入的是一个字符串，并且strtol函数是将一个字符串转化为十进制长整数赋给%rax作为返回值，调用func7之前，%rdi被赋值为36，即第一个参数a1=0x24，a2为要输入的数。再由func7之后，可以知道返回值是0x7.

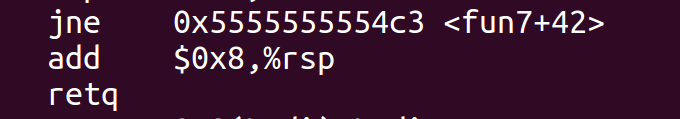




观察func7内的语句，核心部分是递归。







由以上三张截图可知，

若\*a1>a2 , a1=\*(a1+8) , call func7 , %eax=%eax\*2;

若\*a1<a2 , a1=\*(a1+16) , call fun7 , %eax=%eax\*2+1;

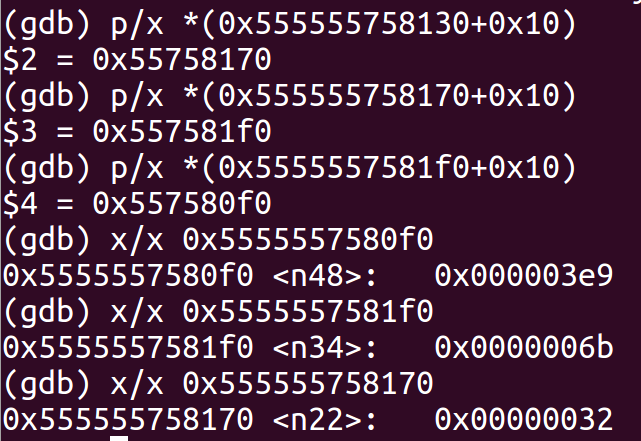
若\*a1=a2 , 跳出。

由此可知，最深层的%eax=0，并且如果\*a1=a2，则推出最里面的递归条件。

因为func7执行完后返回值是7，而逆推出7的产生过程为：

7=（（0\*2+1）\*2+1）\*2+1

则递归时a2=\*（\*（\*（a1+0x10）+0x10）+0x10）



再用gdb调试，可知a2=0x3e9，即1001。

综上可知，输入的密码是1001.

# 第4章 总结

## 4.1 请总结本次实验的收获

1．学会了gdb的调节和各种命令。

2．对C语言下字符串比较、循环、分支（含switch）、函数调用、递归、指针、结构、链表等有了更深刻的理解。

3.更加深刻地理解了汇编语言，程序机器级表示以及逆向工程。

## 4.2 请给出对本次实验内容的建议

希望对gdb调试的讲解更加细致一些。

注：本章为酌情加分项。

# 参考文献

**为完成本次实验你翻阅的书籍与网站等**

[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.