

**课 程 实 验 报 告**

**课程名称： 计算机系统基础**

**专业班级： ACM1601**

**学 号： U201614831**

**姓 名： 苏墨馨**

**指导教师： 黄宏 石宣化**

**报告日期： 2018年 6 月 11日**

**计算机科学与技术学院**

**目录**

[实验2： 1](#_Toc518326390)

[2.1 实验概述 1](#_Toc518326391)

[2.2 实验内容 1](#_Toc518326392)

[2.3 实验小结 16](#_Toc518326393)

实验2： Binary Bomb

### 2.1 实验概述

本实验中， 你要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。一个“binary bombs”（二进制炸弹，下文将简称为炸弹）是一个 Linux 可执行 C 程序，包含了 6 个阶段（phase1~phase6）。炸弹运行的每个阶段要求你输入一个特定的字符串， 若你的输入符合程序预期的输入，该阶段的炸弹就被“拆除”，否则炸弹“爆炸” 并打印输出 "BOOM!!!"字样。

实验的目标是拆除尽可能多的炸弹层次。

每个炸弹阶段考察了机器级语言程序的一个不同方面，难度逐级递增：

\* 阶段 1：字符串比较

\* 阶段 2：循环

\* 阶段 3：条件/分支

\* 阶段 4：递归调用和栈

\* 阶段 5：指针

\* 阶段 6：链表/指针/结构

另外还有一个隐藏阶段， 但只有当你在第 4 阶段的解之后附加一特定字符串后才会出现。

为了完成二进制炸弹拆除任务，你需要使用 gdb 调试器和 objdump 来反汇编炸弹的可执行文件，并单步跟踪调试每一阶段的机器代码，从中理解每一汇编语言代码的行为或作用，进而设法“推断”出拆除炸弹所需的目标字符串。 这可能需要你在每一阶段的开始代码前和引爆炸弹的函数前设置断点，以便于调试。

实验语言： C 语言

实验环境： linux

### 2.2 实验内容

用objdump反汇编bomb执行程序，将汇编源代码输出到“tobomb.txt”文本文件中。观察bomb.c的代码框架，找到不同阶段调用的函数，分阶段破解bomb。

#### 2.2.1 阶段1 字符串比较

1.任务描述：根据phase\_1的代码输入字符串，使它和计算机中储存的字符串一样，炸弹不爆炸。

2.实验设计：用gdb设置断点进行调试，根据反汇编代码读取内存中的数据，找到字符串。

3.实验过程：

观察bomb.c代码，找到阶段一调用的函数为phase\_1，在汇编源代码中找到phase\_1函数如图1.1所示。

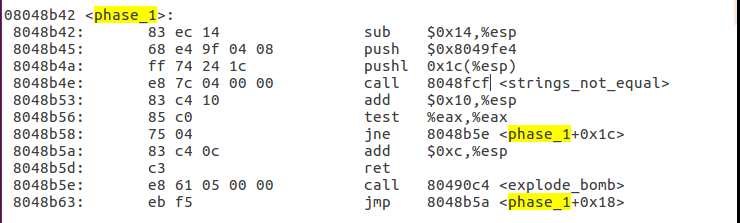


图1.1 phase\_1函数的汇编代码调用比较字符串

在调用strings\_not\_equal函数比较字符串之前，phase\_1入栈了$0x8049fe4和0x1c(%esp)，用gdb在phase\_1设置断点，观察0x8049fe4的内容如图1.2所示

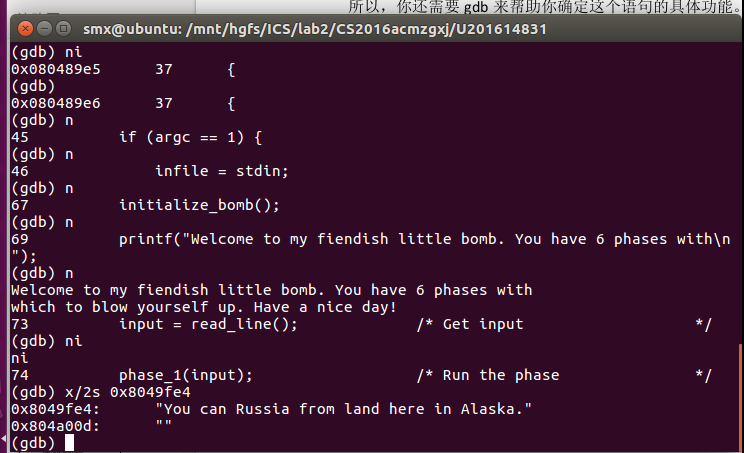


图1.2 字符串的内容

这个字符串即为计算机储存的用于比较的字符串。

4.实验结果：

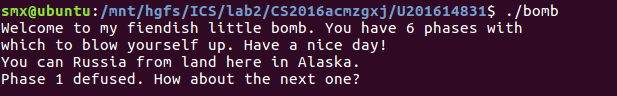


图1.3 破解phase\_1

#### 2.2.2 阶段2 循环

1.任务描述：根据phase\_2的汇编代码，输入6个数字，使炸弹不会爆炸。

2.实验设计：读汇编代码，找到循环的初始条件和循环体，再用gdb设置断点进行调试。

3.实验过程：

观察bomb.c代码，找到阶段一调用的函数为phase\_2，在汇编源代码中找到phase\_2函数。

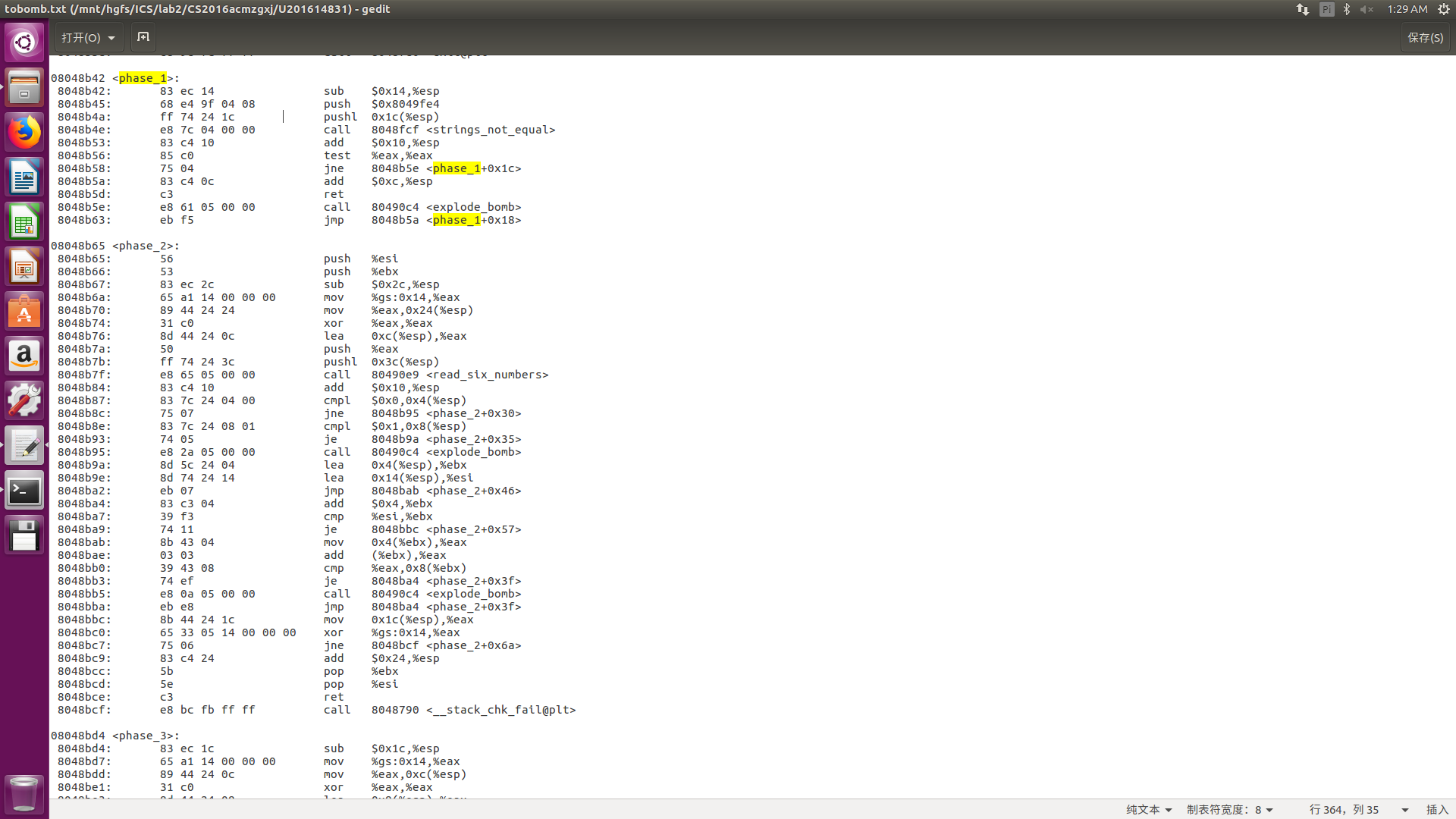


图2.1 phase\_2函数

在汇编代码中看到read\_six\_numbers，得知需要输入六个数字。后面两个比较要求0x4(%esp)和0x8(%esp)分别为0和1，否则炸弹爆炸。用gdb观察0x4(%esp)和0x8(%esp)的值，发现是输入的前两个数。

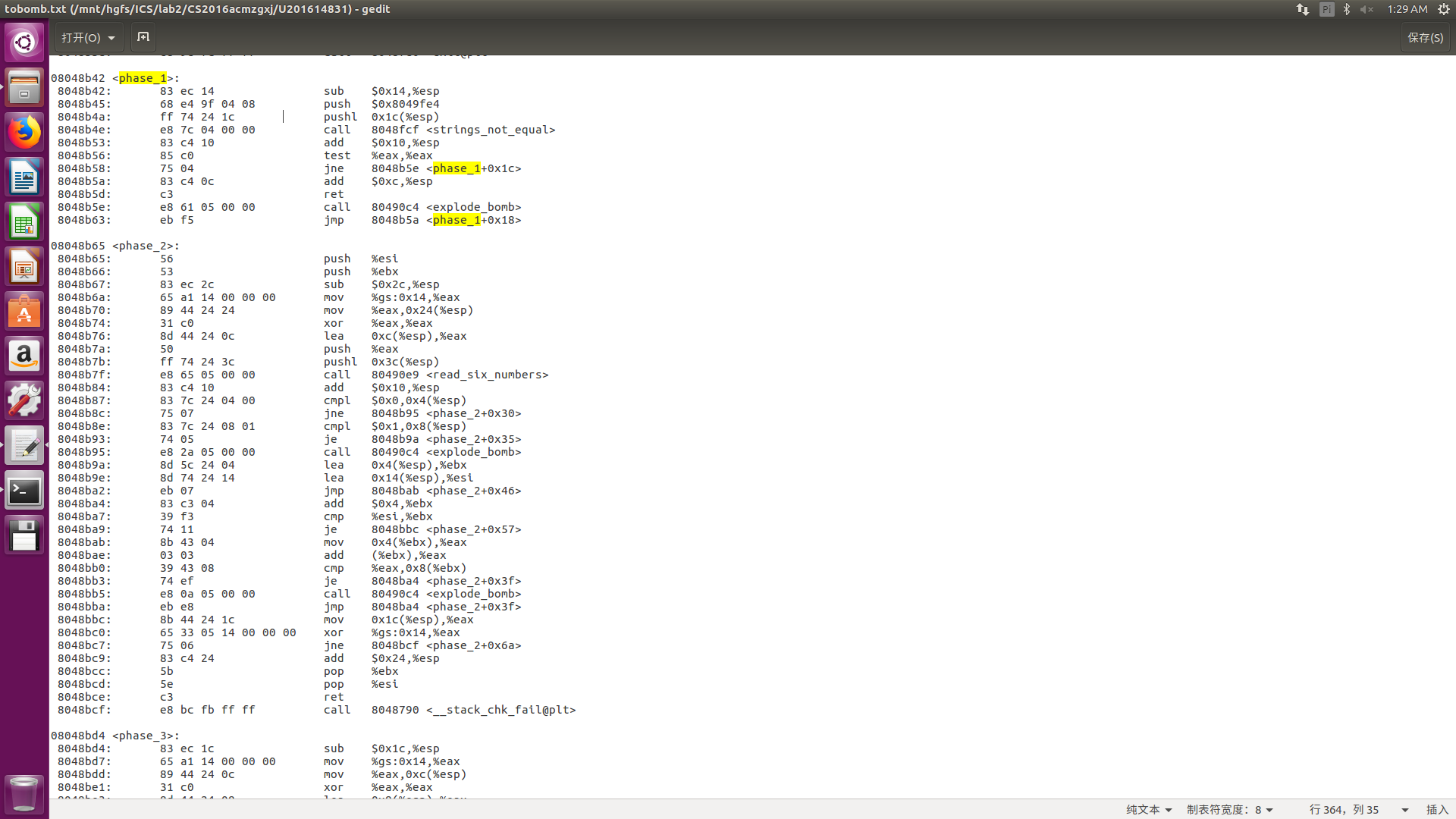


图2.2 第一个数和第二个数

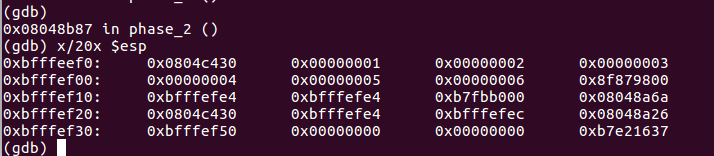


图2.3 输入存放的位置

比较后跟着的代码对循环进行初始化，ebx为第一个数的地址，esi为第三个数的地址，当ebx = esi时循环退出。每次循环ebx后移四位（下一个数）。循环体可以表示为input[i+1]+input[i] = input[i+2]。

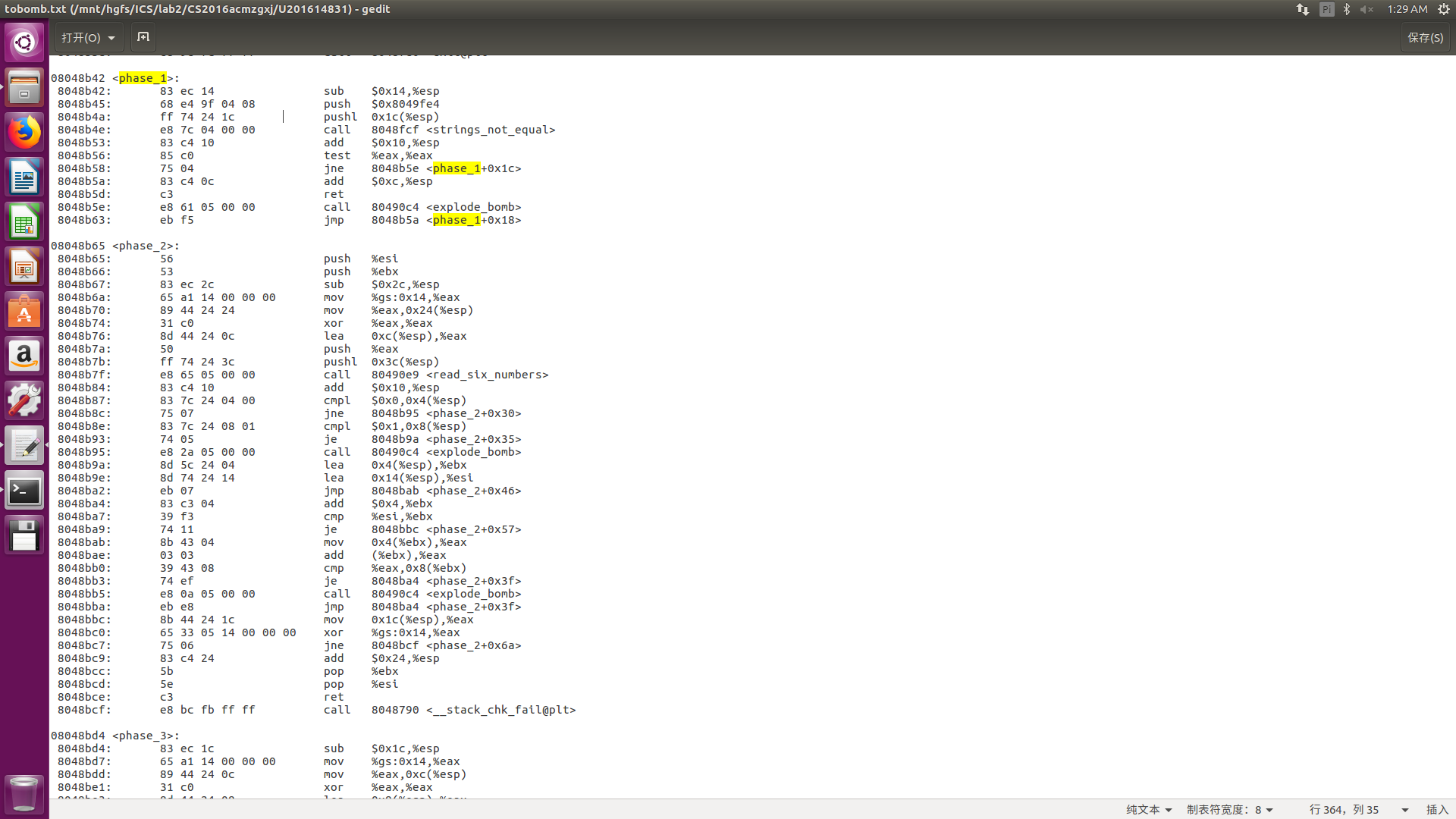


图2.4 循环

所以最后结果应该为斐波拉契数列0，1，1，2，3，5。

4.实验结果：

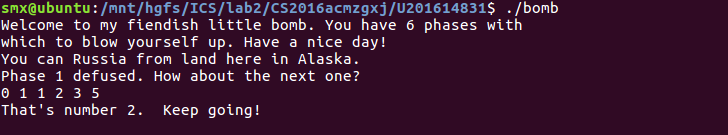


图2.5 破解phase\_2

#### 2.2.3 阶段3 条件/分支

1.任务描述：根据phase\_3的汇编代码，给出一组合适的输入，使炸弹不会爆炸。

2.实验设计：读汇编代码，找到判断语句，用gdb调试找到合适的分支判断语句和解，找到一组结果。

3.实验过程：

在汇编代码中找到phase\_3函数的汇编代码如图3.1所示

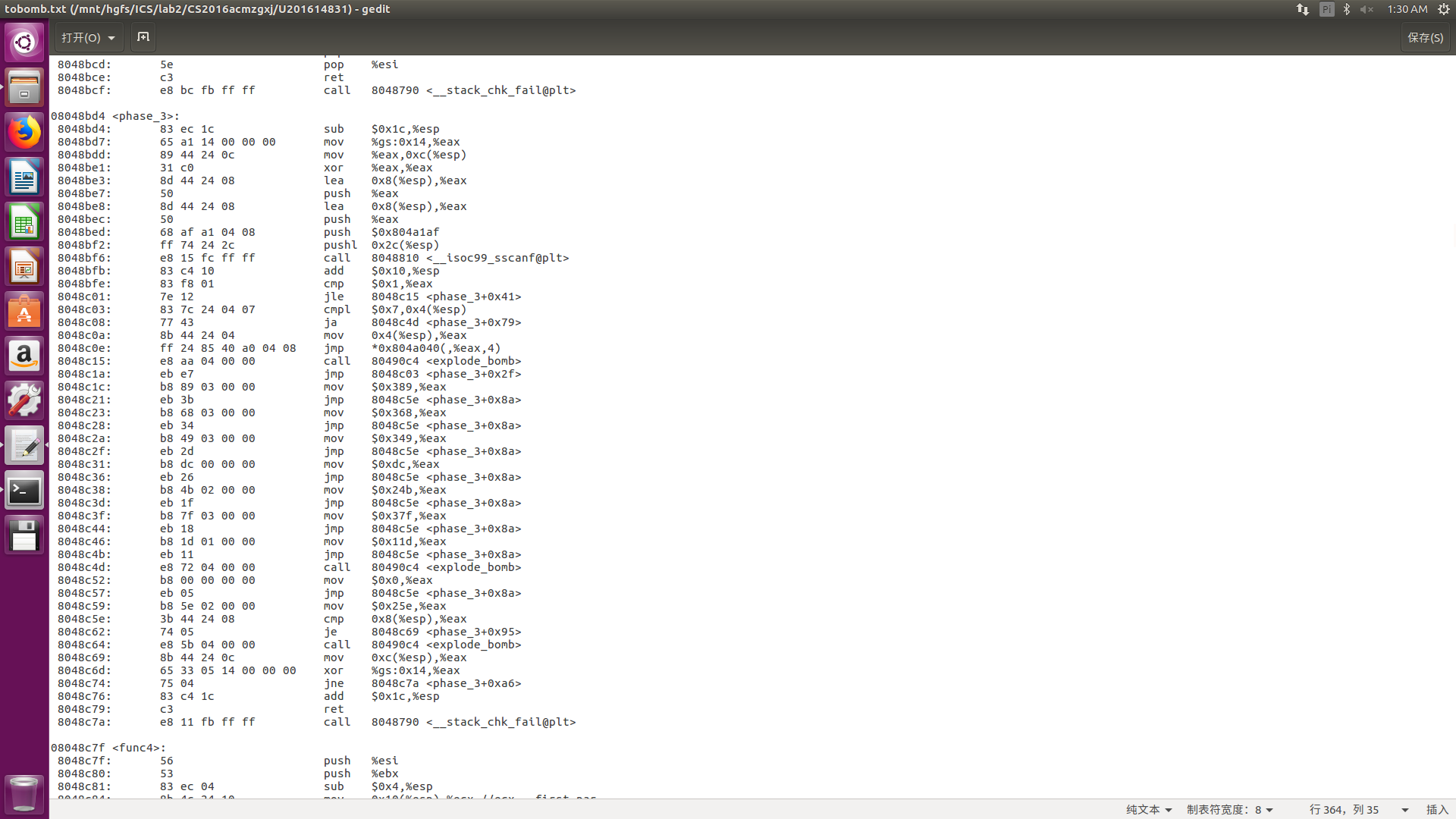


图3.1 phase\_3函数

观察sscanf的参数里有一个地址$0x804a1af，打印观察如图3.2，发现输入需要两个数字。

C:\Users\smx\Documents\Tencent Files\392558193\FileRecv\MobileFile\Image\~PIMGC{QQGB@ZS~0QBM{C$F.png

图3.2 输入两个数字

调用完输入函数后eax返回输入的参数个数，再和1比较，但是前面已经知道输入为两个数字，因此这一句话可以跳过。后面一句说明输入的第一个数小于等于7。

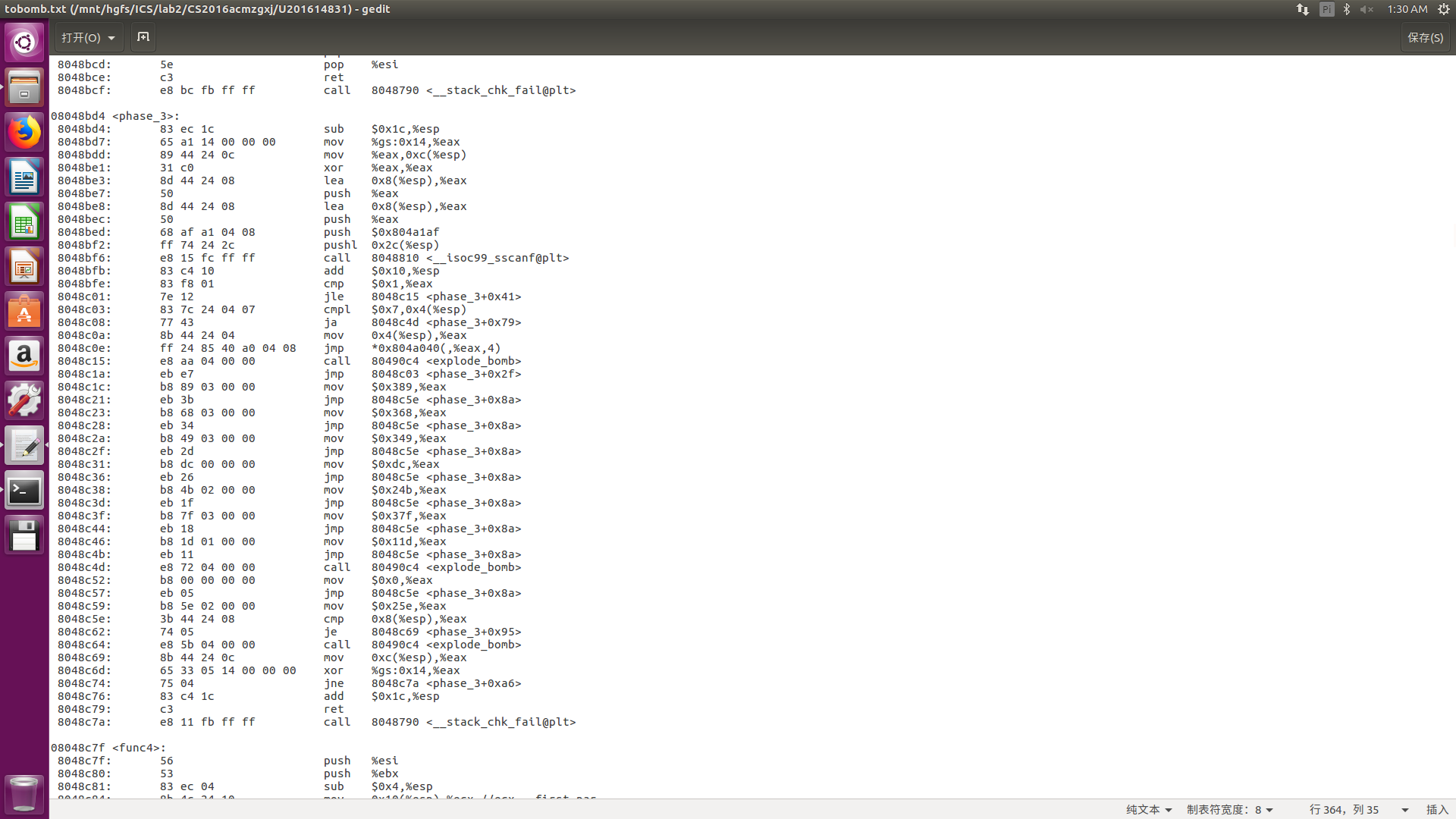


图3.3 输入的第一个数小于等于7

将输入的第一个数传给eax，然后跳转到[0x804a040+4\*eax]处存的地址执行，是一个地址表，根据输入的第一个数的值可以找到它对应的地址。用gdb观察0x804a040存的地址如图3.4所示。这些地址对应一个mov语句和一个cmp语句。所以mov里的立即数即为第二个输入的数。可以得到一组解为1，0x389（905）。此外还有0，0x25e；2，0x368；3，0x349；4，0xdc；5，0x24b；6，0x37f；7，0x11d。

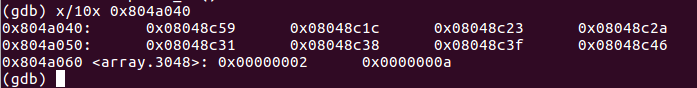


图3.4 地址表

4.实验结果：

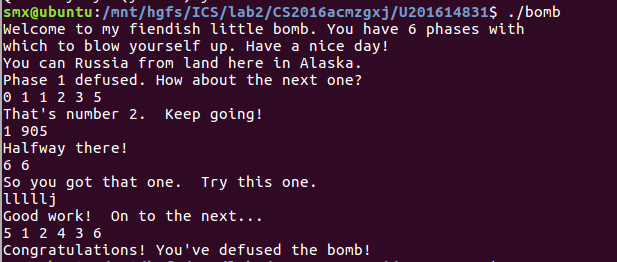


图3.5 phase\_4破解

#### 2.2.4 阶段4 递归调用和栈

1.任务描述：根据phase\_4的汇编代码，输入两个数，使炸弹不会爆炸。

2.实验设计：读汇编代码，写出递归表达式，用gdb调试找到合法的输入。

3.实验过程：

找到phase\_4的汇编代码，如图4.1所示，

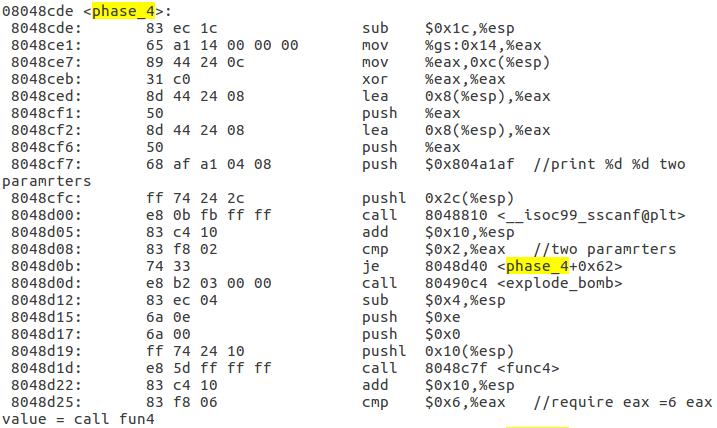


图4.1 phase\_4函数

在sscanf调用之前入栈了$0x804a1af打印如图4.2所示，可以知道输入是两个数字。返回的eax是输入的参数数目，如果不是2，则爆炸。

C:\Users\smx\AppData\Local\Temp\1530513822(1).png

图4.2 输入的参数形式

观察输入后的代码部分如图4.3，要求函数func4调用后返回值eax为6。func4函数比较复杂，先看其他代码部分。根据比较后的跳转，0x8048d40处要求输入的第一个参数小于等于15，0x8048d49处要求输入的第二个参数为6，如果不是则爆炸，如果是，退出函数。

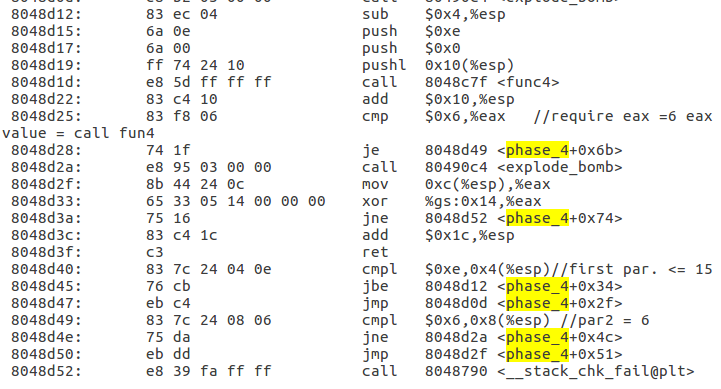


图4.3 参数要求及函数调用

phase\_4在后面调用的函数func4是递归函数，函数汇编代码如图4.4所示

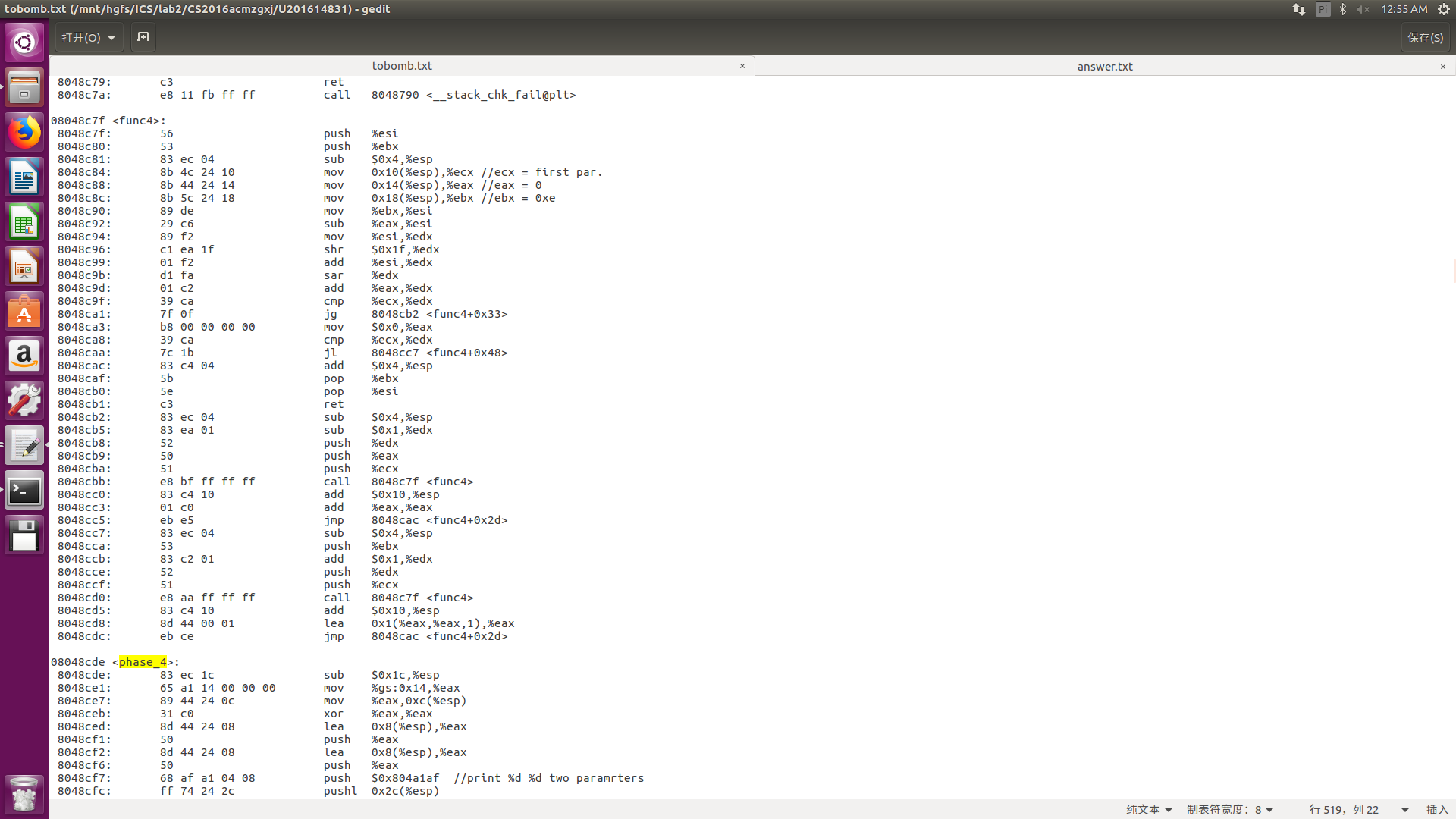


图4.4 func4的汇编代码

观察这部分代码，是有两个分支的递归代码，可以用c语言表示如图4.5所示，func(输入的第一个参数,0,14)作为递归的函数。前面要求函数返回的结果为6，可以计算出(或者暴力)第一个参数为6。

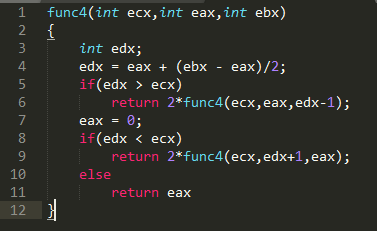


图4.5 func4的C语言表示

4.实验结果：

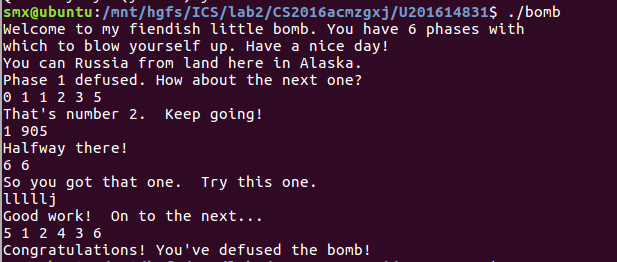


图4.6 破解phase\_4

#### 2.2.5 阶段5 指针

1.任务描述：根据phase\_5的汇编代码，输入字符串，使炸弹不会爆炸。

2.实验设计：读phase\_5的代码，找到对输入的字符串的约束条件，找到字符串的指针所在的循环以及循环做的操作，找到结束条件并解出合法输入。

3.实验过程：

找到phase\_5的汇编代码，8048d60处的代码调用了string\_length函数，这个函数返回字符串的长度到eax，后面比较eax和6如果不等则爆炸。所以要求输入的字符串长度为6。

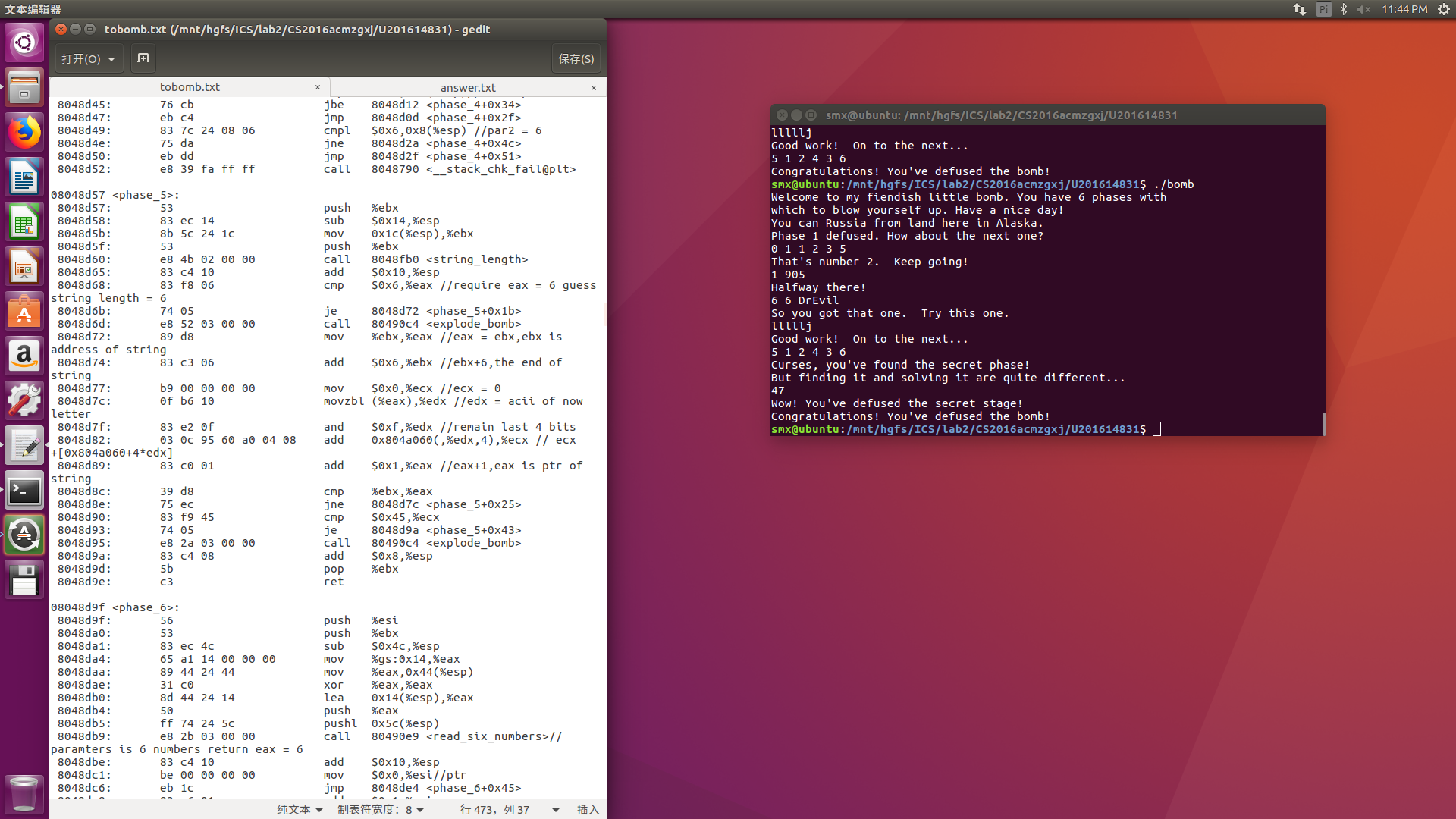


图5.1 phase\_5函数

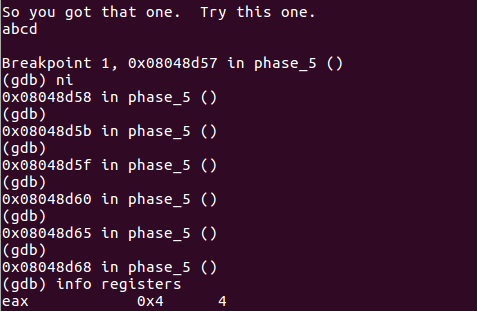


图5.2 eax返回输入字符串abcd的长度

后面用到了ebx寄存器，观察ebx的内容，发现ebx存的是输入的字符串的地址，如图5.3。add $0x6,%ebx将ebx移到字符串尾部。eax是字符串的首址。

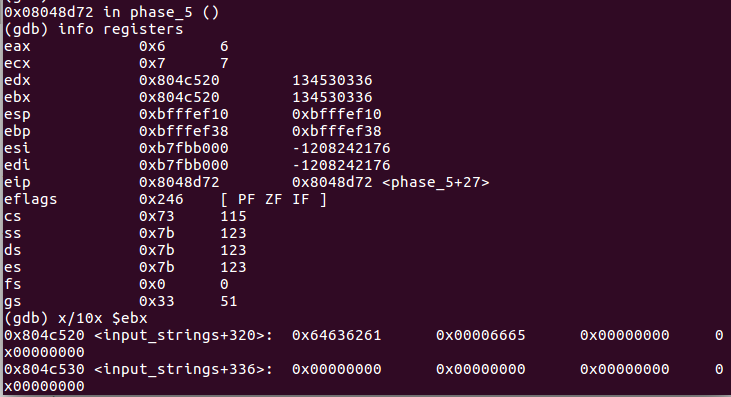


图5.3 ebx存输入的字符串的地址

如图5.4所示的8048d7c到8048d8e代码段是一个循环段，在循环开始之前给ecx赋初值0，在循环结束的地方要求eax = ebx即指针eax遍历完字符串。循环内的and语句对输入的字符串的每一个字符进行操作，保留输入的字符的acii码的后四位存在edx中。add语句对ecx和内存[0x804a060+4\*edx]里的内容进行求和，8048d90处可以看到这个和要求为0x45，内存里的内容如图5.5所示。

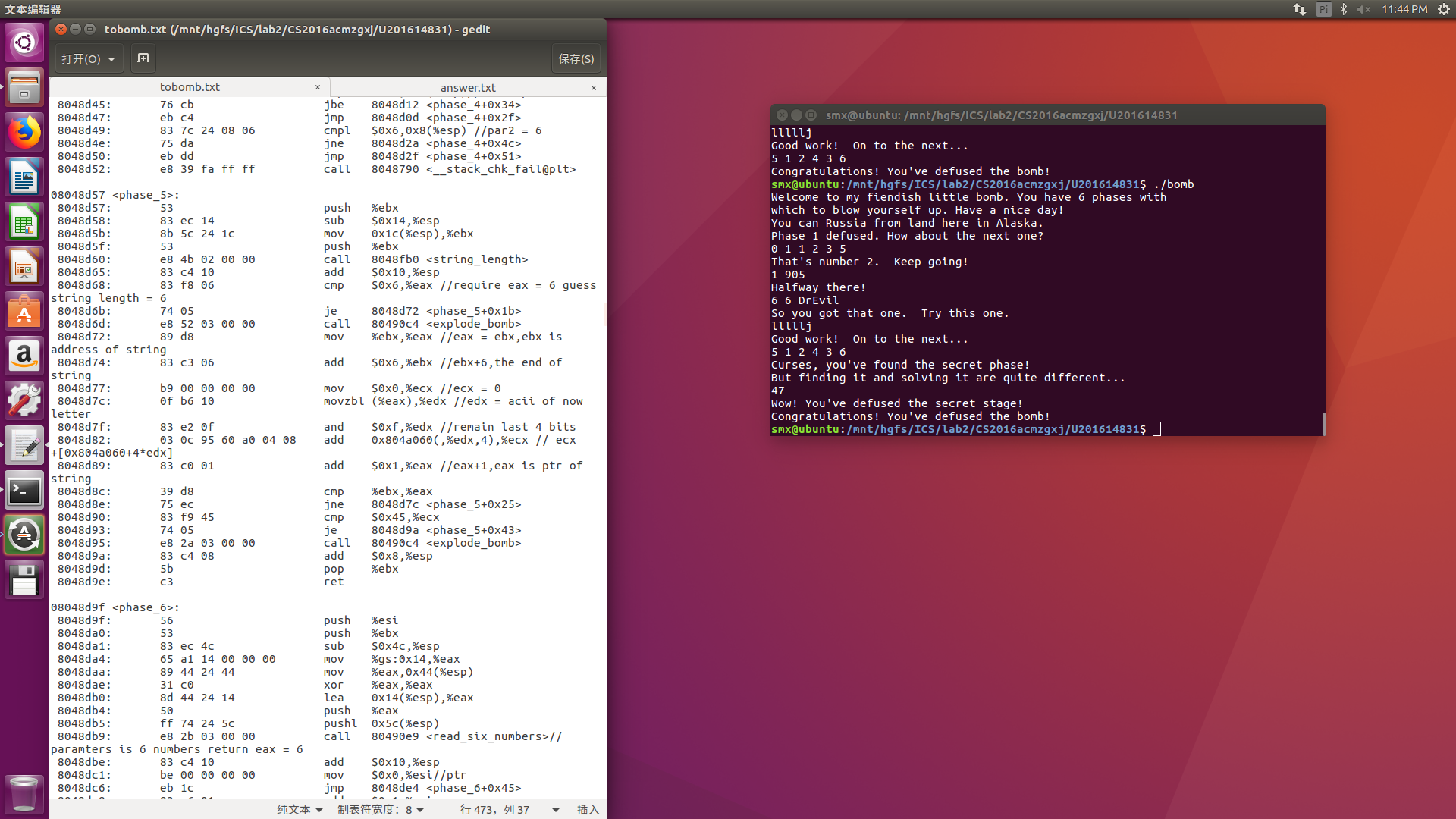


图5.4 循环段

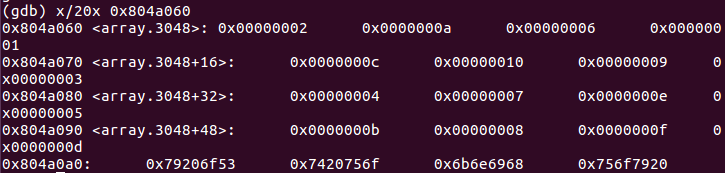


图5.5 内存0x804a060里的内容

根据acii码表求出各个字母的后四位(edx)，再找到它们对应的内存里的内容，可以求出一组解lllllj使和为0x45。

4.实验结果：

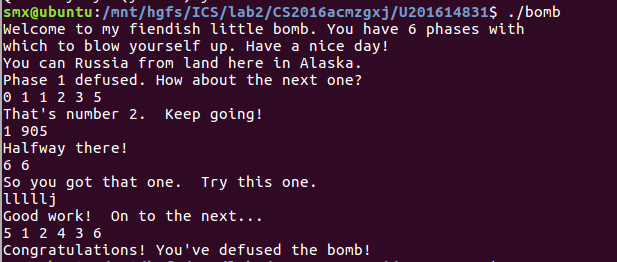


图5.6 phase\_5破解

#### 2.2.6 阶段6 链表/指针/结构

1.任务描述：根据phase\_6的汇编代码，输入六个数，使炸弹不会爆炸。

2.实验设计：

3.实验过程：

找到phase\_6的汇编代码,因为调用了函数read\_six\_numbers可以看出输入为六个数字。

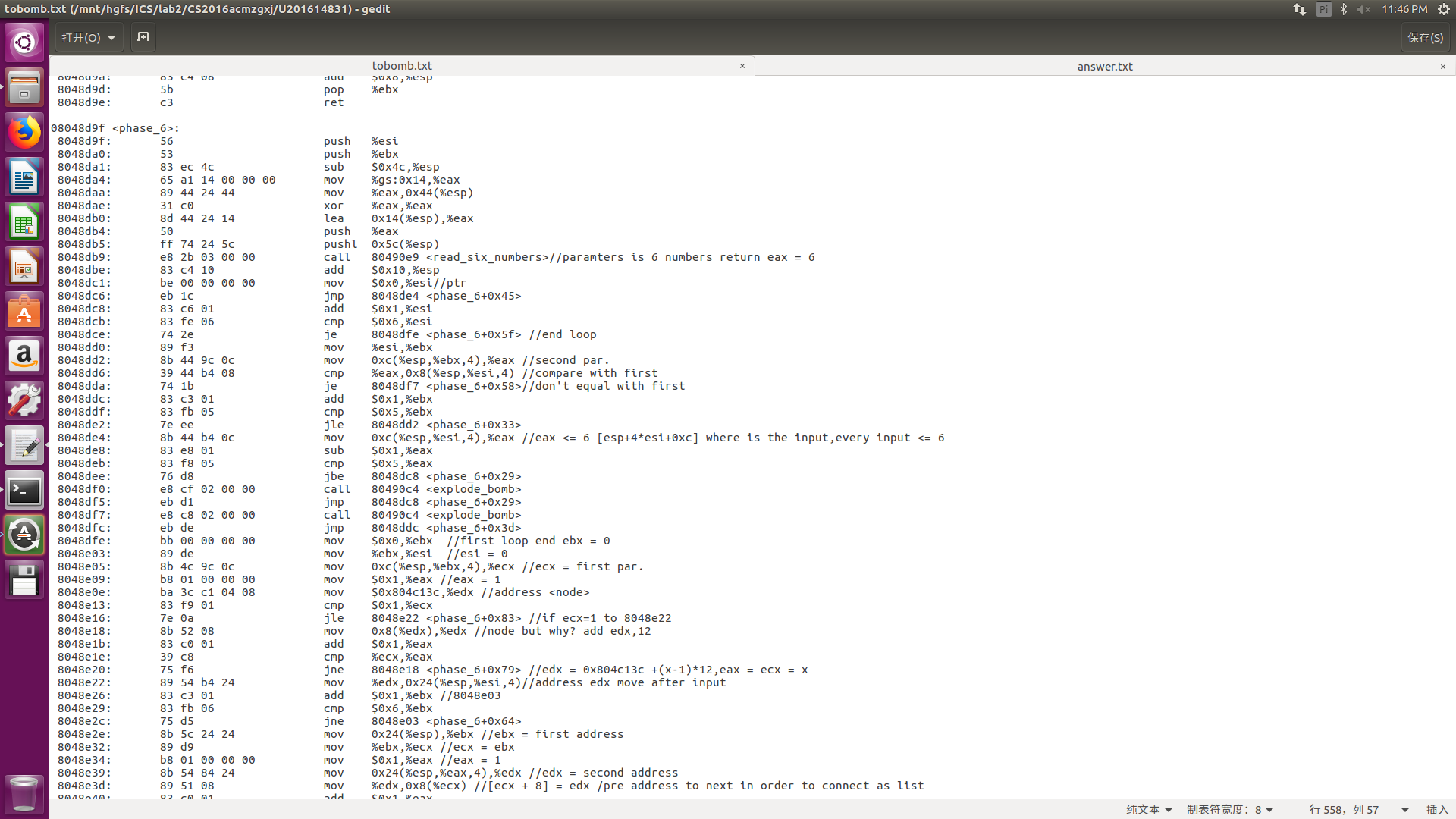


图6.1 phase\_6函数

8048dc8到8048dee是一个双重循环，内层循环为8048dd2到8048de2。这个双重循环判断输入的每一个数都不相等并且都小于等于6。因此输入应该为1到6的某个排列。

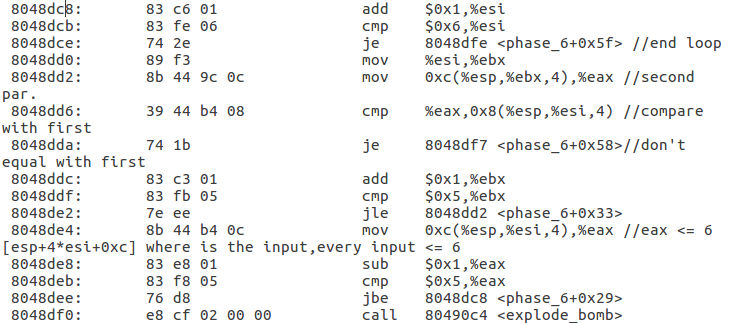


图6.2 第一个双重循环

8048e03到8048e56是一个双重循环，8048e18到8048e20是内层循环。外层循环对ebx初始化为0，内层循环对esi，ecx和eax初始化。

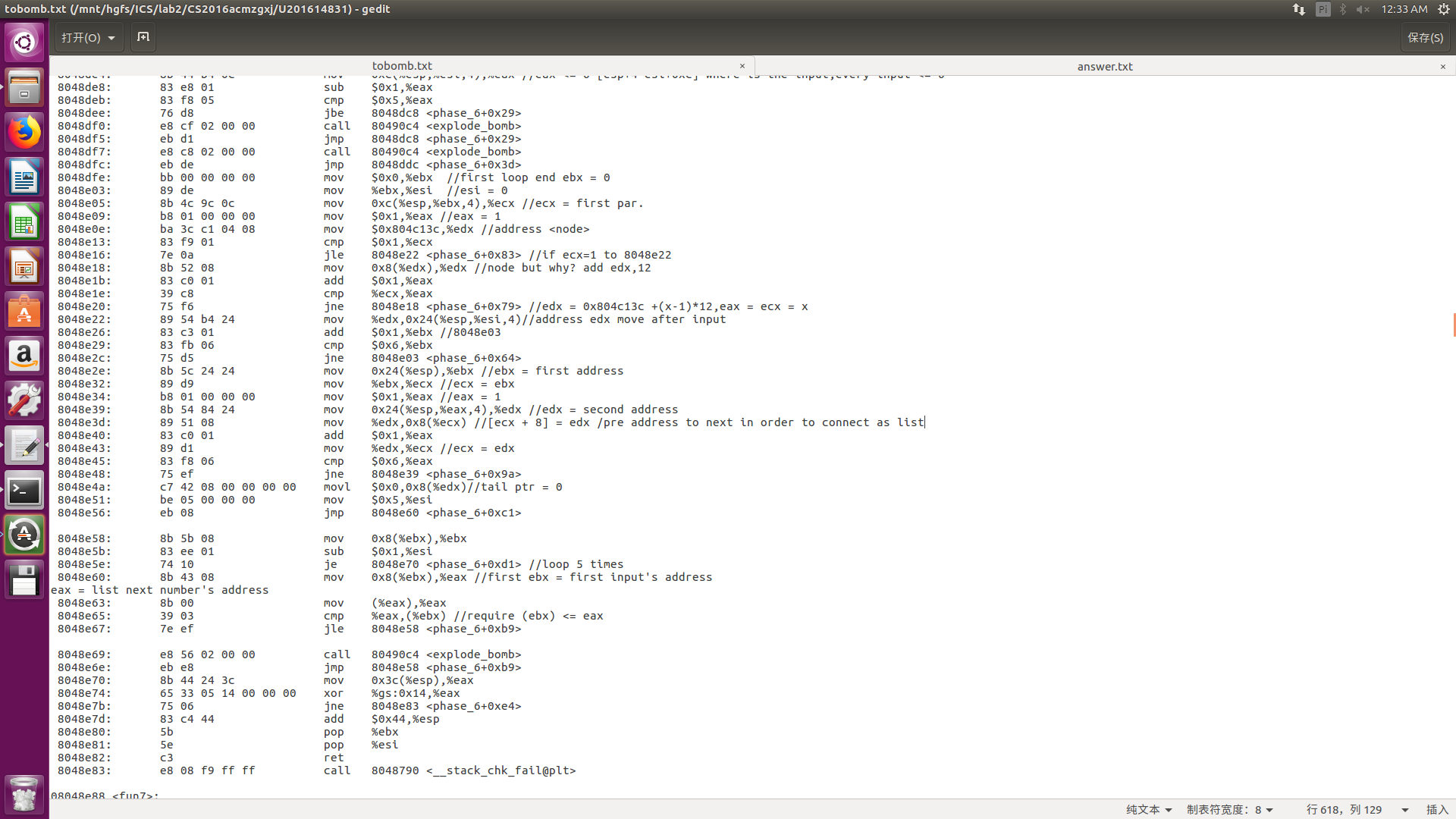


图6.3 第二个双重循环

观察到8048e05处的位置是一个赋值操作，对ecx进行初始化，查看内存[esp+0xc+4\*ebx]的值如图6.4所示对应的是输入的数字，因此ecx是输入的参数的值。

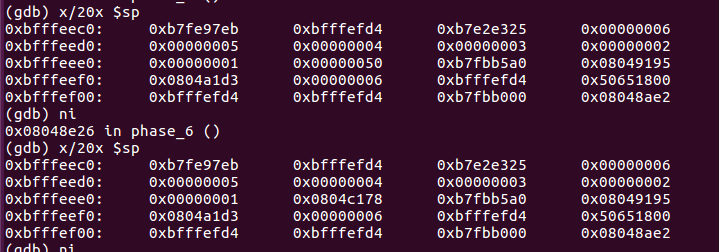


图6.4 输入在内存的位置

观察0x804c13c处的内容为一个链表，如图6.4所示。每一个node由三部分组成，value，key(1到6)，地址。地址是下一个node的位置。

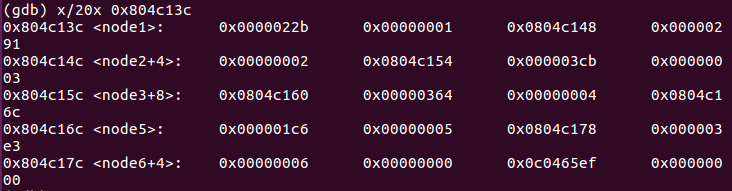
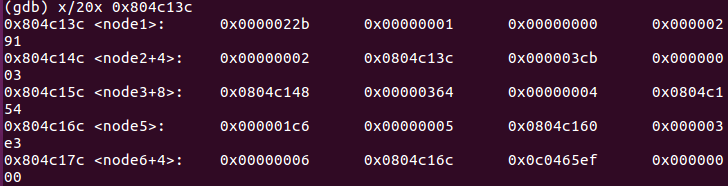
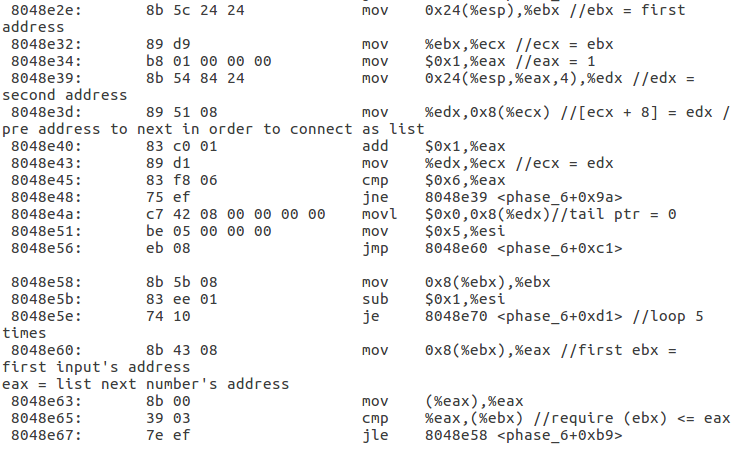


图6.5 初始的链表

这一部分的循环是根据输入修改地址对链表进行连接，最后一个结点的指针指向0x0。输入为6 5 4 3 2 1时连接结果如图6.5所示

图6.6 连接后的链表

8048e58到8048e67是一个循环，这个循环要求链表按照从小到大的顺序连接，所以只需要比较链表的value就可以得到合适的输入顺序使链表连接正确。



跟据前面找到的链表的value的值解出排序为5 1 2 4 3 6

4.实验结果：

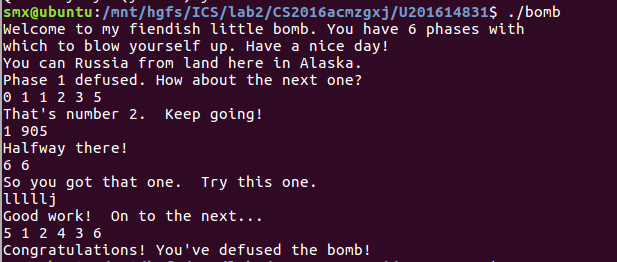


图6.7 phase\_6破解

#### 2.2.7 阶段7 隐藏阶段

1.任务描述：找到隐藏阶段的入口，在第四阶段后面加一串字符串，在第六阶段结束后输入一个数，破解隐藏炸弹。

2.实验设计：因为每个阶段调用的函数都是phase\_i()和phase\_defused()，所以隐藏阶段的入口应该在phase\_defused()内，观察汇编代码并用gdb调试找到合适的输入开启隐藏阶段。隐藏阶段的函数为secret\_phase()和fun7()，观察代码，用gdb调试找到结果。

3.实验过程：

找到phase\_defused()的代码，如图7.1所示

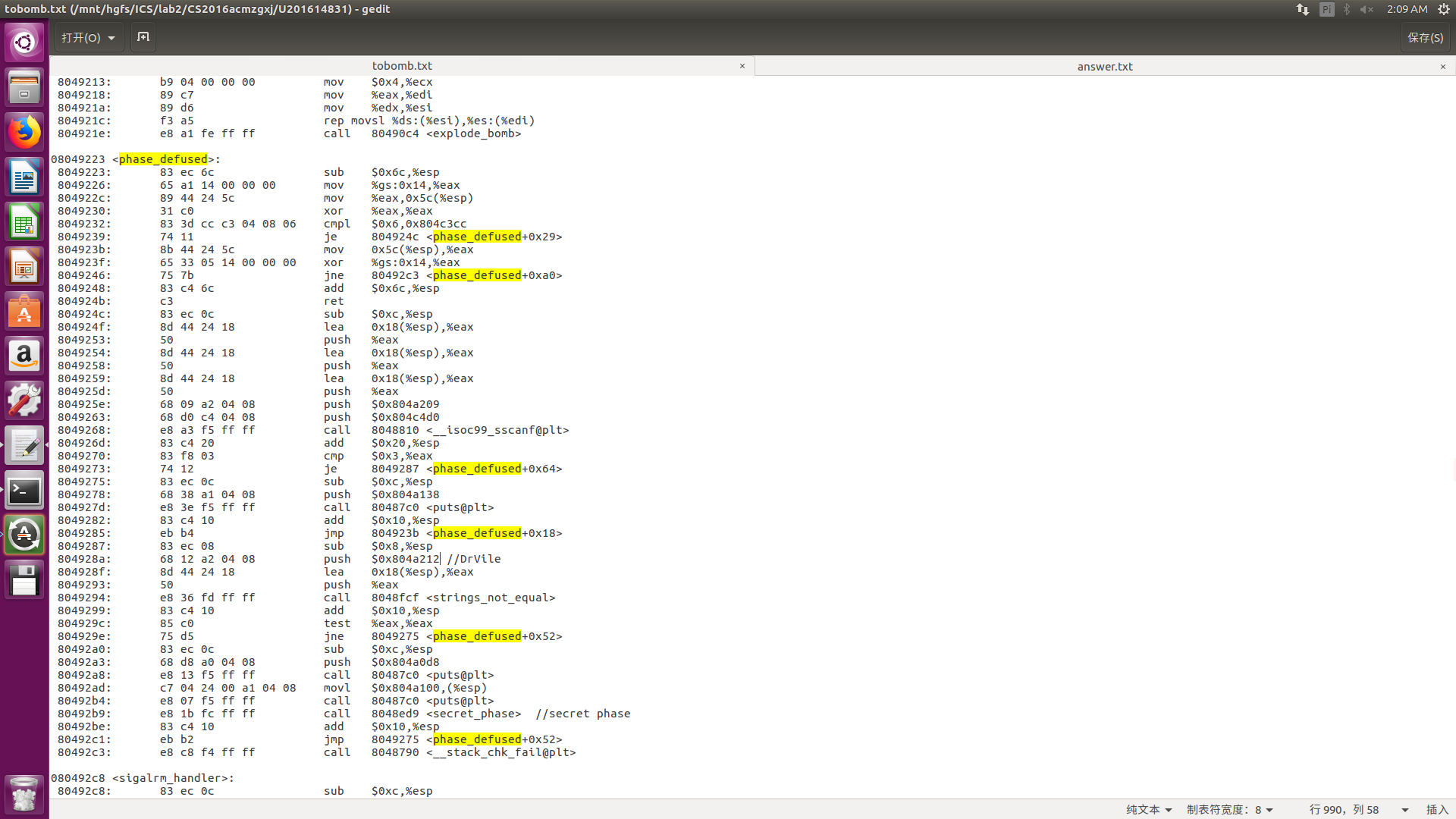


图7.1 phase\_defused()函数

可以看到在8049294处调用了一个string\_not\_equal，推测这个字符串比较是针对隐藏入口的参数。前面入栈了一个参数0x804a212，打印这个参数的内容得到入口的字符串为“DrEvil”。

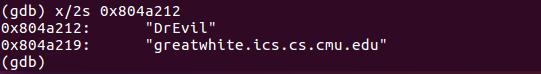


图7.2 隐藏入口字符串

找到secret\_phase的汇编代码，它在8048eea调用的函数将输入从字符串转换为long int型，后面的比较说明输入的字符串转换为数字后是一个小于0x3e8的数。然后调用fun7。要求fun7返回的eax = 5。

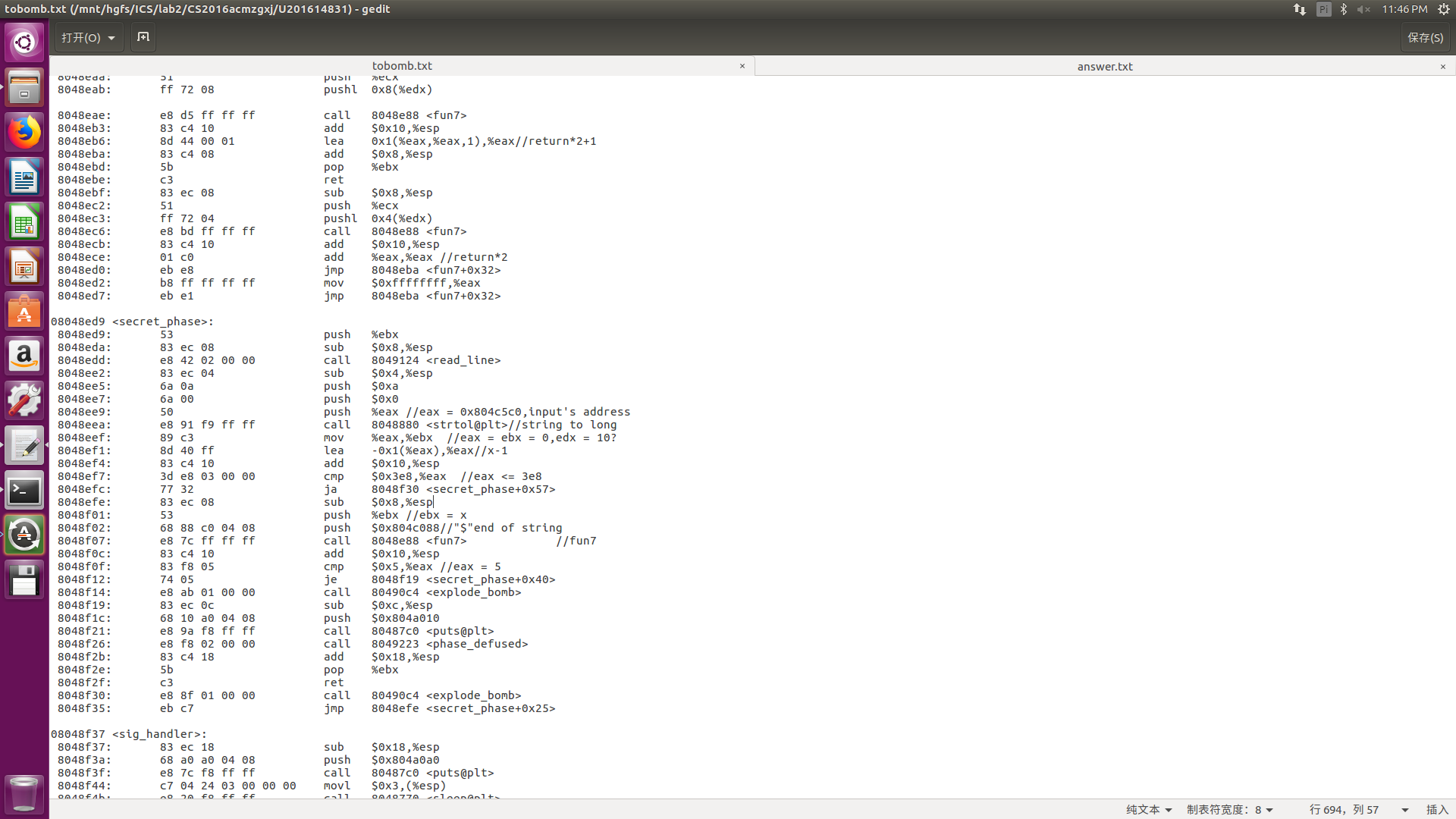


图7.3 secret\_phase函数

fun7是一个递归函数，它有两个参数，一个参数对应的是地址，另一个参数是数fun7(address,num)。如果对应的地址的值为0，函数返回-1。如果num >[address]，fun(addr+4,num),eax内的返回值为eax\*2+1；如果 num < [address]，fun7(addr+8,num),eax内的返回值为eax\*2；如果 num = address，返回。后两种情况在递归开始之前eax = 0。

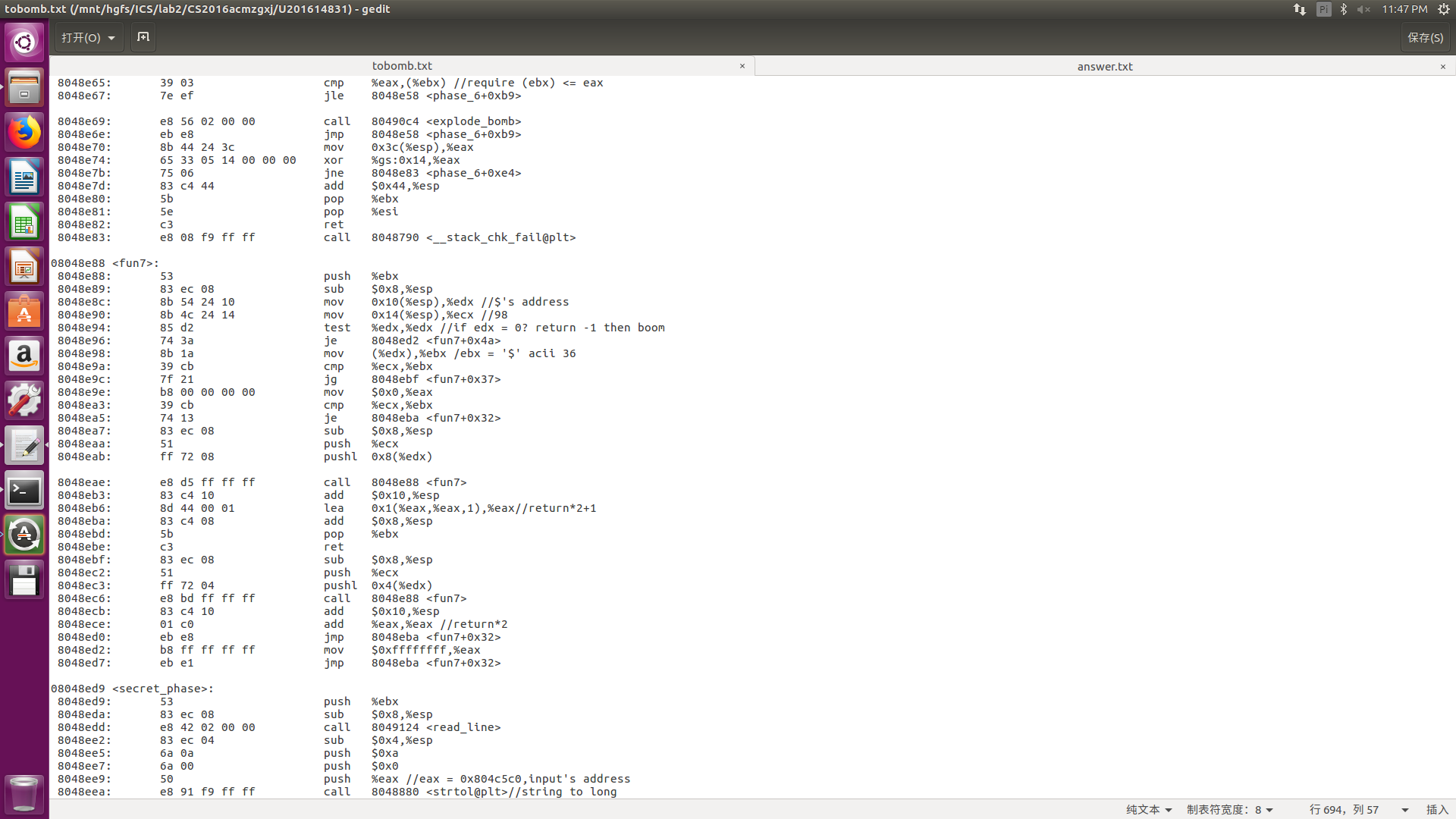
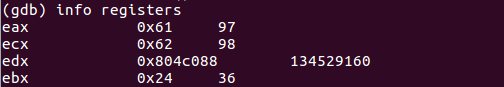
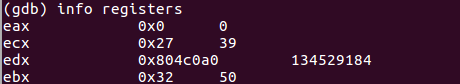
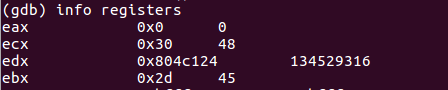


图7.4 fun7函数

5 = ((0\*2+1)\*2)\*2+1，因此可以知道每次递归的num的范围。ebx里为[address]的值，ecx为输入的数，观察可以知道ebx内的值为36，50，45，47。所以最后退出要求输入为47。







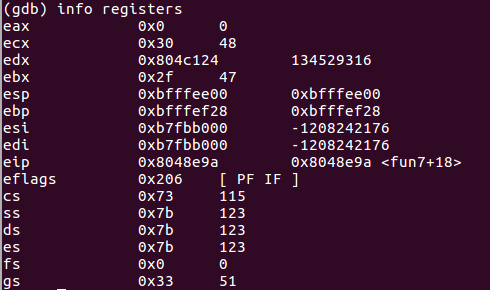


图7.5 递归路径上用于比较的值(36->50->45->47)

4.实验结果：

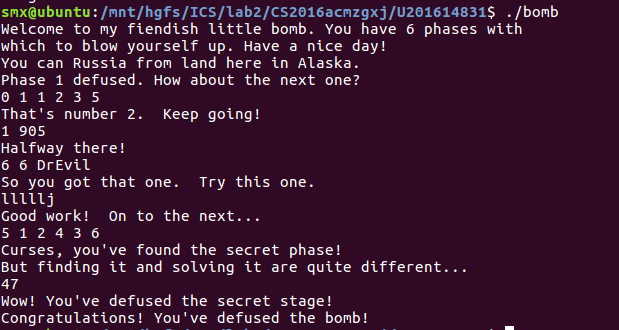


图7.6 secret\_phase破解

### 2.3 实验小结

本次实验通过gdb和objdump对可执行文件的汇编代码进行破译。汇编代码的理解因为在汇编课和计算机系统课都有涉及，所以读汇编代码比较简单。比较复杂的是gdb的使用，数据在内存中存放的理解，以及对函数调用时的参数的传递。gdb可以读取内存中的数据，单步执行代码，读取寄存器的信息等。根据汇编代码，打印内存中的某些数据，找到需要的部分。第四个任务和第七个任务涉及到递归，因为读取内存不是很熟练，所以第七个任务选择了暴力找到合适的参数，第七个任务实际上是二叉树的查找，树的结点的值可以用读取内存的方法找到。第四个函数是递归，要求返回值是一个特定的数，因为输入的数据范围比较小，所以可以推也可以直接暴力尝试，我用c语言的程序计算了一下f(x)的值，找到了符合要求的输入。其他的任务涉及到了c语言中的条件判断(if)，循环(for)，分支跳转(switch)等语句翻译后的汇编语句。本次实验让我对于各种结构在内存中的存放，可执行程序反汇编为汇编代码，读汇编代码，逆向工程有了一定的了解。对于程序的调用，执行过程有了更深入的理解。