北京科技大学实验报告

|  |  |  |
| --- | --- | --- |
| 学院：计算机与通信工程学院 | 专业：信息安全 | 班级：信安201 |
| 姓名：曾绎哲 | 学号：42024058 | 日期：2021.10.28 |

# 实**验名称：Bomblab二进制炸弹实验**

# 实验目的：

通过拆解给定的二进制炸弹程序，熟悉Linux系统的使用，掌握程序反汇编和逆向工程的基本方法，理解汇编语言，学习使用调试器的方法。

# 实验环境：

- 操作系统：ubuntu 18.04

- GDB版本：8.1.1

**实验前建议与说明：**

推荐先大致阅读王爽的《汇编语言》，对拆弹实验中的语法能有大致的了解，对其原理也能有大致的掌握。其次参考《深入理解计算机系统》，拆弹实验的原理便可以较为清晰。

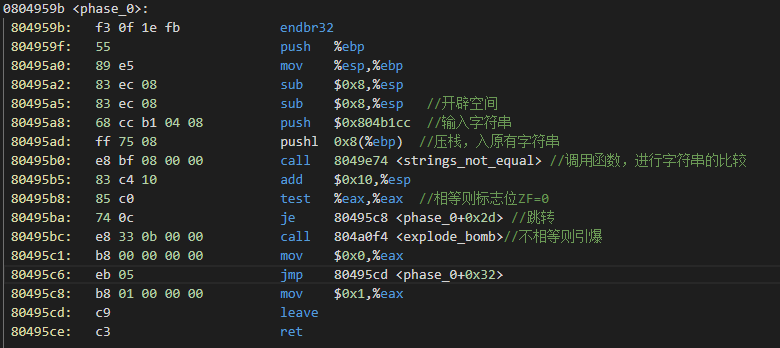
本篇实验报告侧重于讲解如何拆除炸弹，因为对于汇编语言的底层原理并未完全掌握，因此在拆除炸弹的过程中技巧与知识缺一不可，且技巧居多。

# 实验内容与步骤：

首先，使用tar -xvf 42024058.tar指令解压压缩包。

然后使用objdump -d bomb > bomb.s 对bomb文件进行反汇编，并将输出的汇编指令导入bomb.s 文件当中。通过指令gdb bomb 进入gdb调试阶段，开始拆弹！

## Phase\_0: 字符串的比较

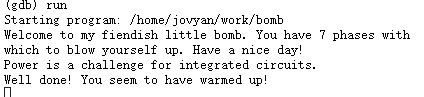


# 附录：模板使用说明

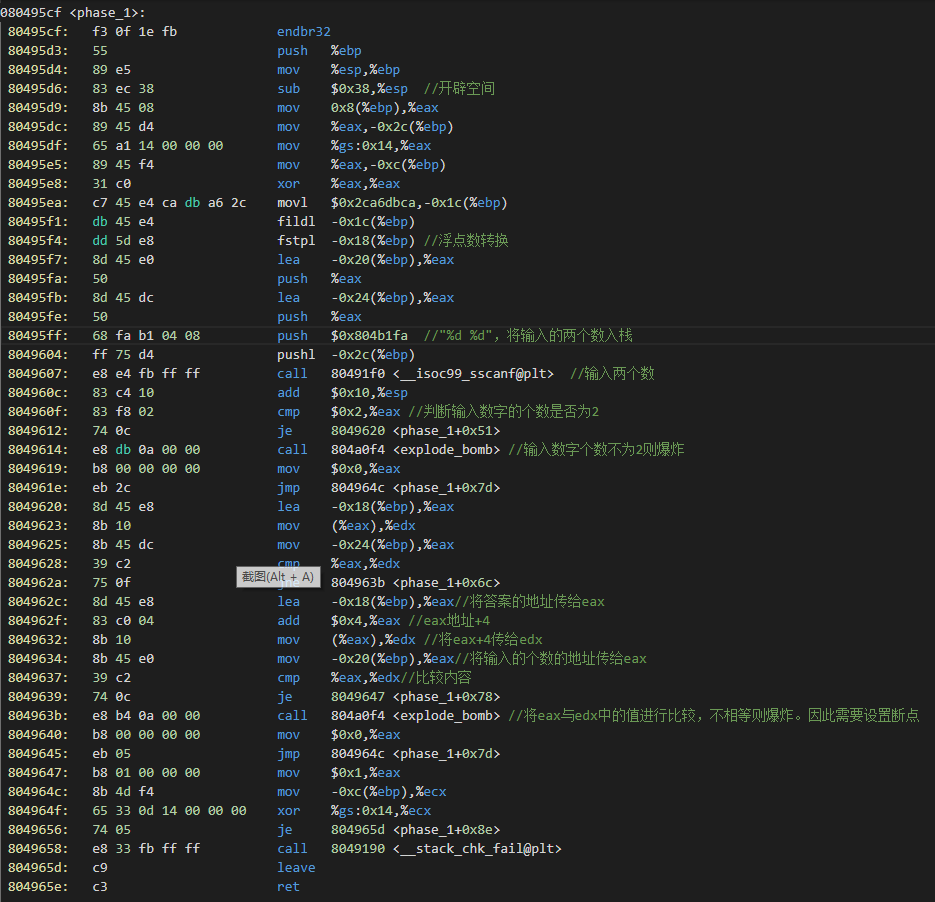
这是一个代码量极小的炸弹，快速浏览完毕后便基本可以知道这是要进行字符串的比较（由函数名称<string\_not\_equal>亦可知）。因此我们只需要输入一个与地址0x804b1cc中存储的字符串相等的字符串即可使得代码跳转从而避免炸弹爆炸。所以我们进入gdb bomb调试模式，使用指令x/s 0x804b1cc查看该地址的内容：

c4065d803896b295a95d2373a4e9ab3

随后只需要在gdb bomb中执行指令run，输入上图中的字符串内容即可，拆弹成功！



**Phase\_1:浮点数的表示**



代码量稍大，但正常浏览阅读后基本上清楚代码的目的是什么，边分析边写好注释。我们发现了push 0x804b1fa，这是一个敏感点，于是我们使用gdb调试查看一下：

7a9d7265d2793d43753ab0499a4abea

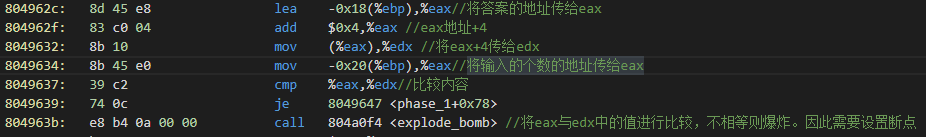
结合前两的两个push入栈以及后面的压栈和调用函数，以及函数名<sscanf@plt>基本可以判断这是一个输入的含义，需要我们输入两个数。

b8cb46b71f94aaf7dee66bd6b6707c5

这是一个判断语句，如果输入的数字个数不为 2，那么炸弹就爆炸，所以确定，我们需要输入两个数。

abb3f927c5ad9773641188aaf2d9cab

这是将数字转为浮点数，查阅资料明白其含义即可。

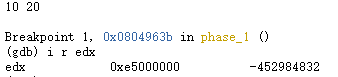


重点在这里，阅读代码我们发现，其实它就是将我们输入的数字与其原本存有的数字浮点数形式进行比较，如果都相等则跳转，否则炸弹爆炸。所以我们只需要知道原本存有的浮点数即可，而他就存在edx中，所以我们在炸弹爆炸处设置断点，用gdb模式中的i r edx指令查看即可：

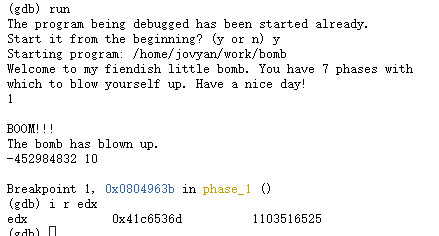
1）设置断点

1e78fd27d7c380c07ba474abeeb34ba

2）查看第一个答案



3）查看第二个答案

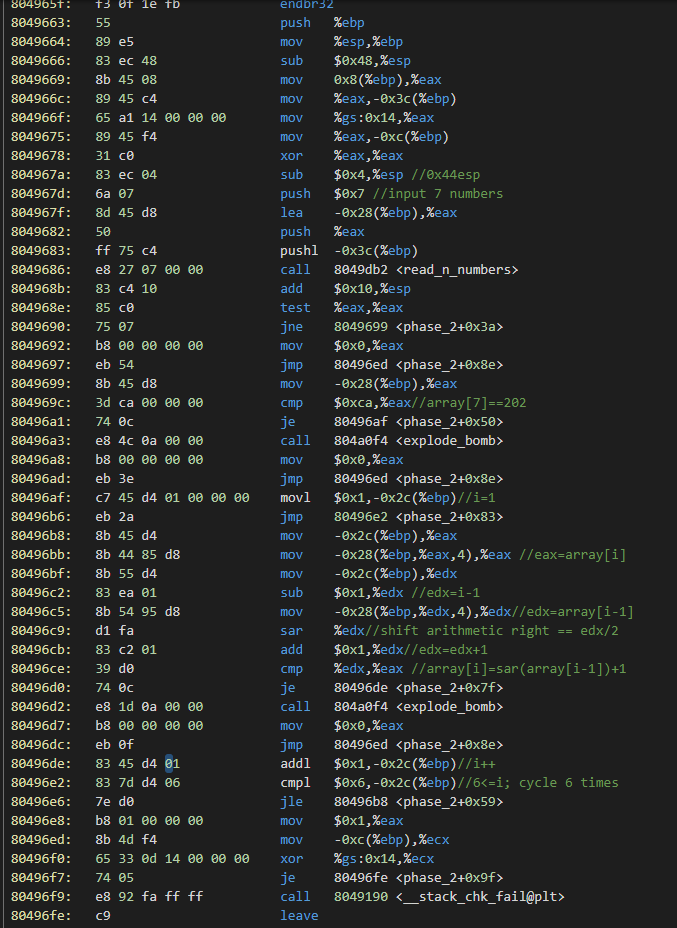


所以输入的两个数字为：-452984832 1103516525

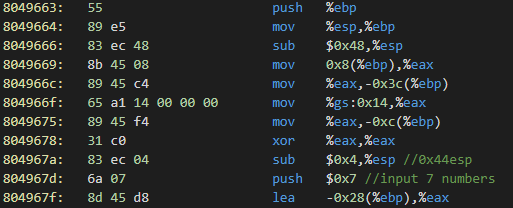
2e0679c6df95296878baa05b1d8155c

拆弹成功！

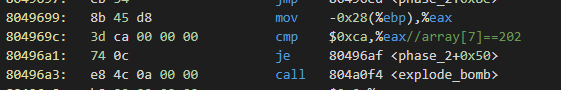
**Phase\_2：循环**



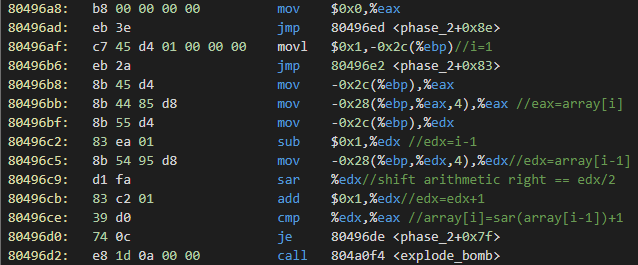
代码量稍大，但总体也不长，正常阅读浏览完成后，基本了解这是一个循环。于是我们从头开始分析代码。



首先是开辟空间，由指令push $0x7基本确定我们需要输入7个数字。



因为要输入七个数字，所以我们猜测是一个数组，由上图代码可知，输入的其中一个数要与0xca相等，而这个数的地址是栈尾，所以大胆猜测这个数组的最后一个数为0xca，即十进制的202，数组名暂且为array，否则就爆炸。



这一部分是重点，这其实就是一个较为简单的算法，类似于c++中的for循环。我们来看一下这个算法：首先，-8x2c（%ebp）存放的是for循环中i的值，即一个计数器，i初始值为1 。然后将数组array[1]的地址给edx，然后对edx进行逻辑右移操作，查阅资料得知，其实就是c++中的整型除法，edx=edx/2（注意是c++中的整型除法！！！），然后edx再加1 ，把这个值传递给eax，与array[i-1]比较，如果相等则跳转，否则爆炸。这就意味着，我们输入的数字存放在数组里，必须满足以下规律：array[i]=array[i-1]/2+1。



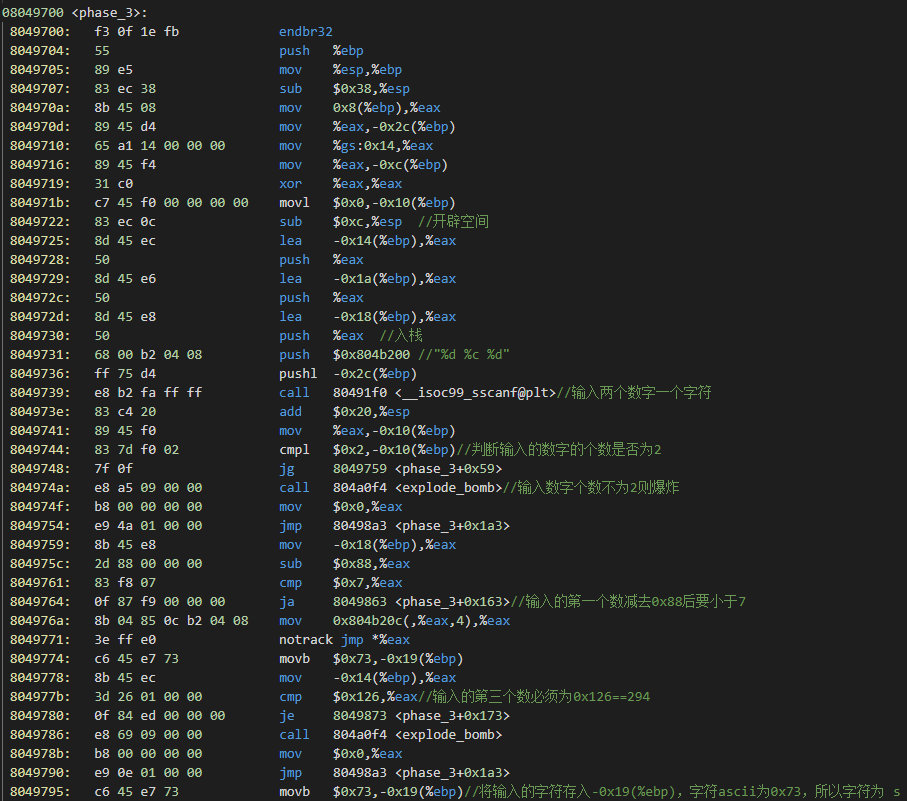
由代码知，循环一共6次。

而array[7]=202，总共有七个数字，所以我们可以得到答案：202 102 52 27 14 8 5

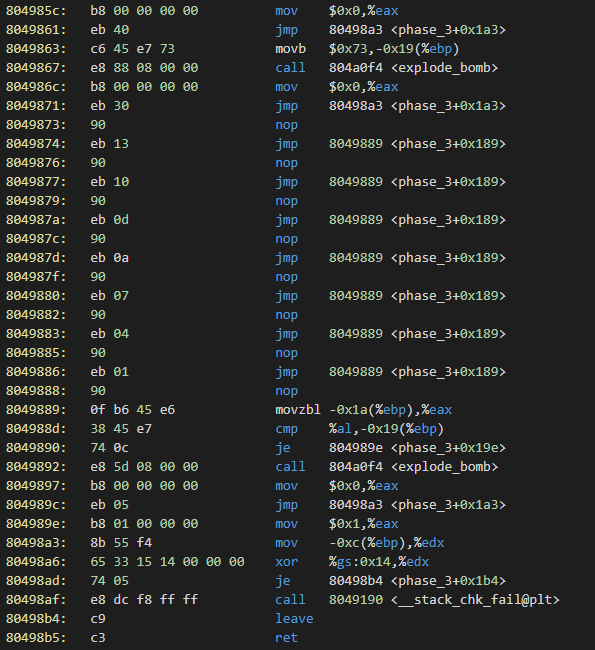


拆弹成功！

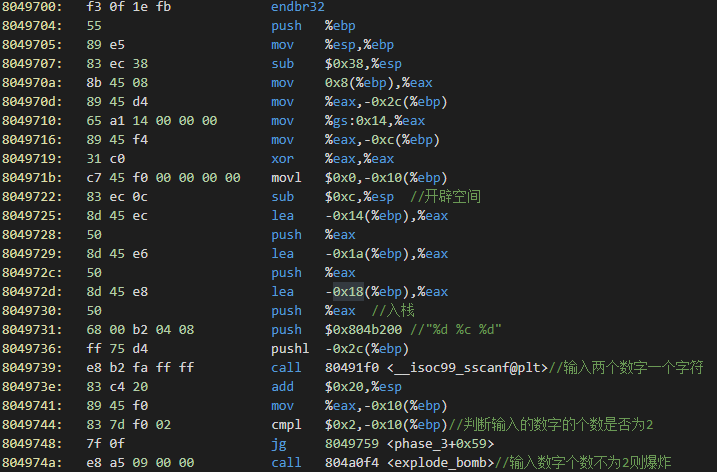
**Phase\_3:条件/分支**







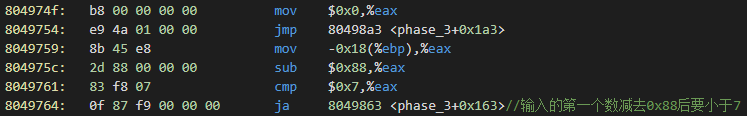
总体来看，代码很长，可能这题的任务量不小。但仔细认真研读之后会发现，其中有一半以上都是类似于c++中switch的选择语句以及if的条件判断语句。首先我们看到代码的首部分：



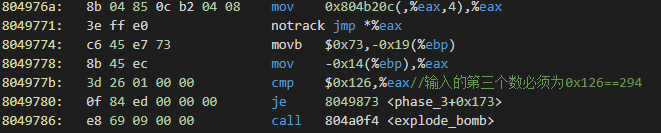
这是基础操作，开辟空间，然后查出0x804b200地址的内容，发现需要输入两个数字和一个字符，然后判断输入数字的个数是否为2，是则跳转，否则爆炸。



这是查到的0x804b200的内容。



然后再往下看，我们发现有一个判断语句，将第一个输入的数字减去0x88后要小于0x7但是大于等于0，所以我们知道第一个是输入的数字最大为为0x88+0x7，即最大为0x8f。所以我们可以选择第一个数字为0x88，即136。否则会跳转，而跳转的结果就是炸弹爆炸，所以我们不可以让他跳转。



再往下看，我们看到第二个输入的数字的地址给了eax，而eax的值必须与0x126相等，否则就爆炸，由此可知第二个输入的数字为0x126，即294 。





再往下看，发现这两段对字符的输入做了限定，输入的字符的ASCII码值必须为0x73，即字符s，所以输入的字符为s。

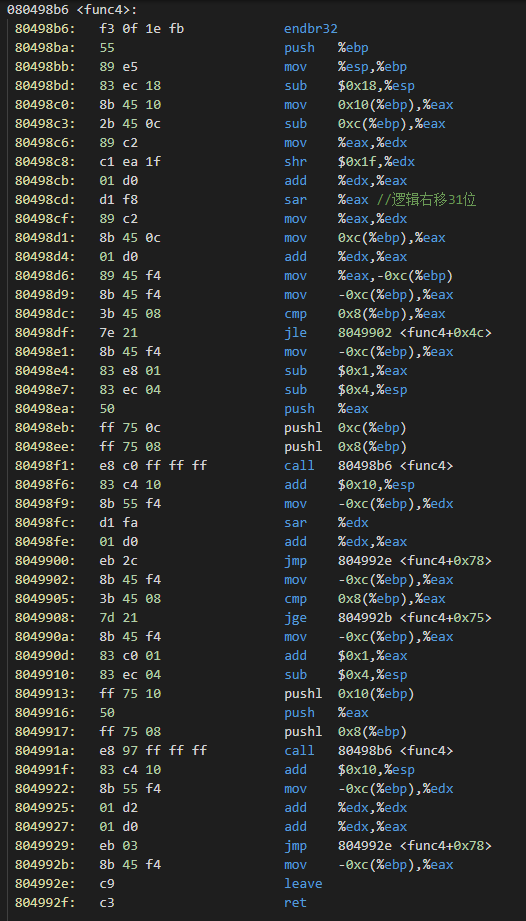
到此，我们便知道了答案为：136 s 294

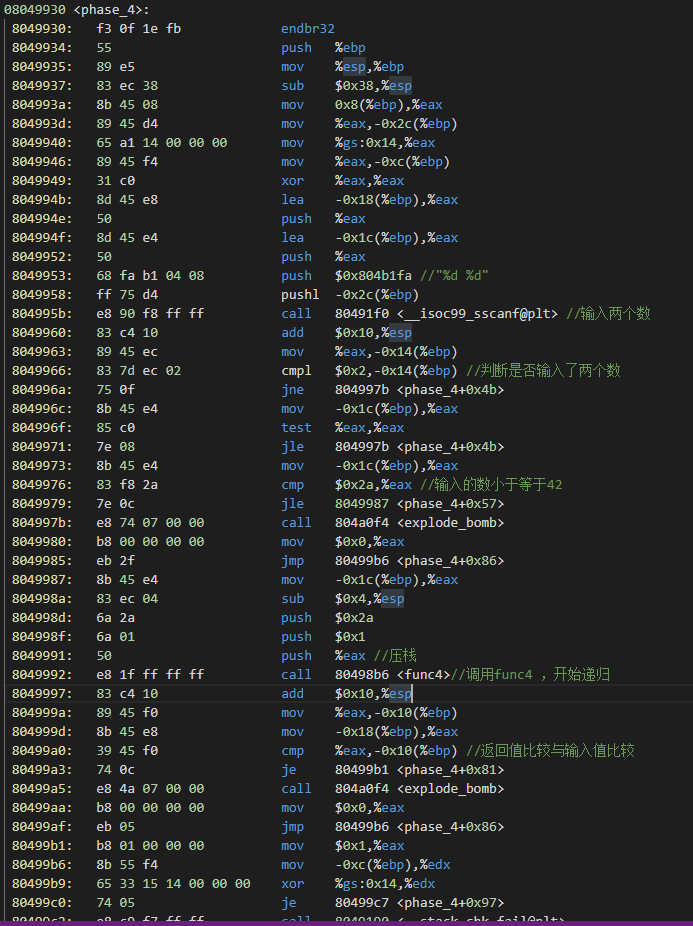
其中第一个数字不一定是136，只需满足前面的条件即可。



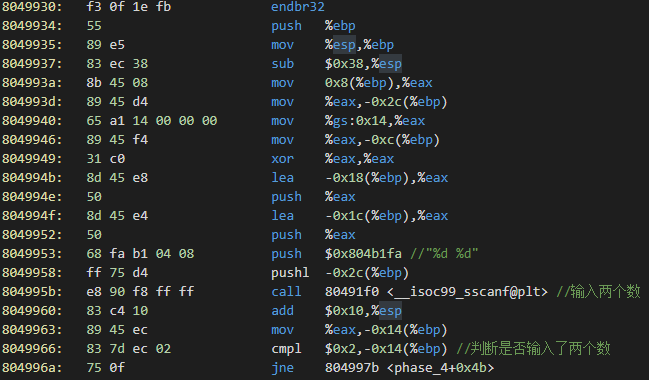
拆弹成功！

**Phase\_4：递归**

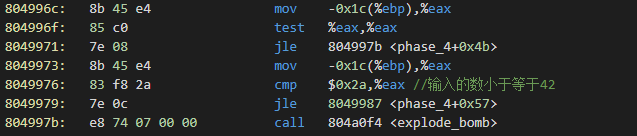




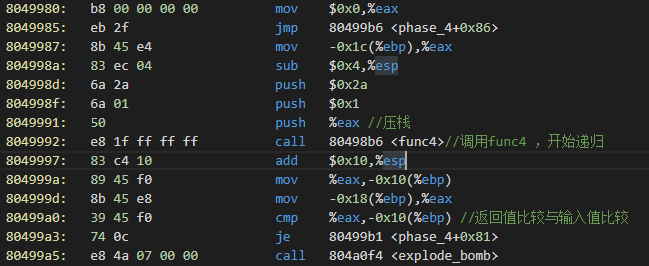
代码稍长，主要是因为有一个递归函数。先大概浏览一下递归函数，发现逻辑右移31位的算法。由于递归函数的代码不太好理解，所以我们直接看phase4中的代码。但我知道func4中一定是有一个算法的，稍后我们会提到。



前面都是基本操作，开辟空间，输入两个数，判断一下输入的是不是两个数，是则继续，否则就跳转爆炸。（注意，这里的规则跟前面不一样，这里是：输入的是两个数则继续而不跳转，否则就跳转到爆炸函数导致爆炸）



这里有一个很关键的一句话，判断输入的数是否小于等于0x2a，即42 。所以我们输入的数应该要小于等于42并且大于0



往下看，这里是重点。压栈并且将输入的第一个数字作为参数传入func4中，将最终的返回结果传给eax并且与第二个输入的数字进行比较，如果相等就跳转，否则就爆炸。所以我们有一个结论，输入的两个数字要小于等于42且第一个输入的数字经过func4的处理后与第二个输入的数字相等。接下来的步骤是搞清楚func4的内容，明白它的处理机制。根据func4中代码我们知道它是一个递归函数，所以我们就可以根据输入的第一个数字计算出第二个数字。

由于第一个数字不能大于42且不能小于0，所以我们随便取一个数字20，根据func4知第二个输入的数字应该是150，所以，答案为：20 150

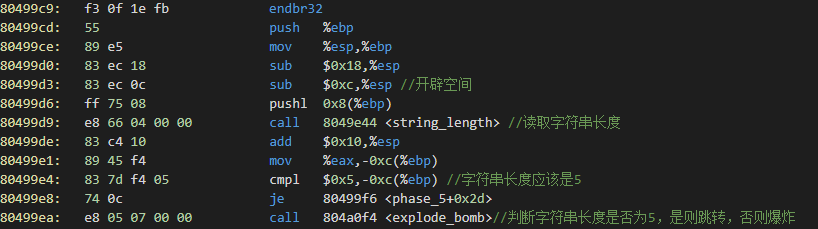


拆弹成功！

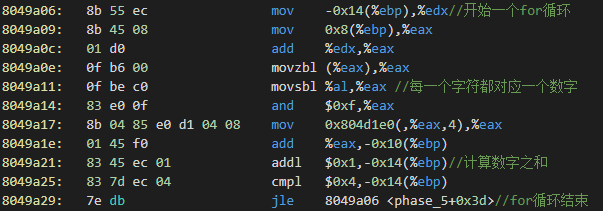
**Phase\_5:数组**



代码不长，大概浏览得知需要输入一个长度为5的字符串进行操作。因此我们从头开始看，逐步分析。



简单的操作，开辟空间，输入一个字符串并返回它的长度，长度若为5则继续，否则爆炸，因此我们需要输入一个长度为5的字符串。



开始了一个for循环，循环次数为字符串的长度。这其中说明了本题的机制。在这个循环中我们可以看到，每一个字符其实对应了一个数字，类似于ASCII码。把每一个字符的值相加，直至循环结束，相当于遍历，然后把这个值存到-0x10(%ebp)中。



在循环结束之后，我们发现他把那个相加的总和与0x26，即38进行比较，相等则跳转，否则爆炸，所以，五个字符的数值相加应该为38，但我们如何知道每一个字符对应的数字呢？有一个很简单的办法，就是逐个尝试，看似复杂，其实最为简便，因为我们可以对字符进行排列组合计算。我们首先在爆炸前设置一个断点。



然后开始调试，我们先输入5个a：



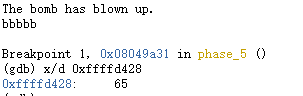
用指令i r ebp查看ebp的地址：

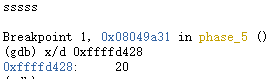


得到了ebp的地址，由于那个总和存放在-0x10(%ebp)中，所以我们需要查看-0x10(%ebp)中的内容：



得到5个a的总和为40，即字符a对应数字8。用同样的方法，以此类推，我们很快就可以知道字符b对应数字13，字符s对应数字4，所以可以有13\*2+4\*3=38 。



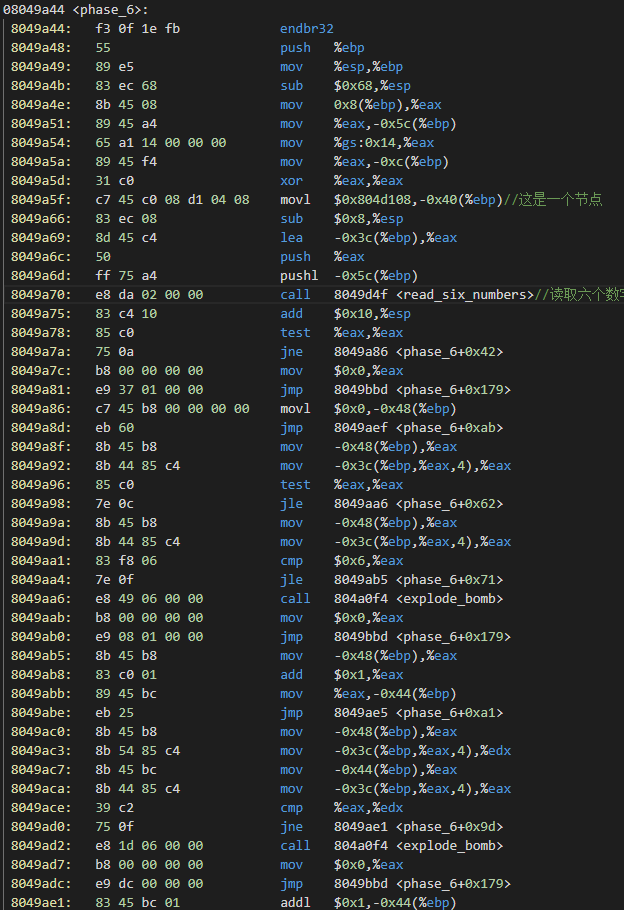


因此，答案可以为：bbsss（顺序无关紧要）

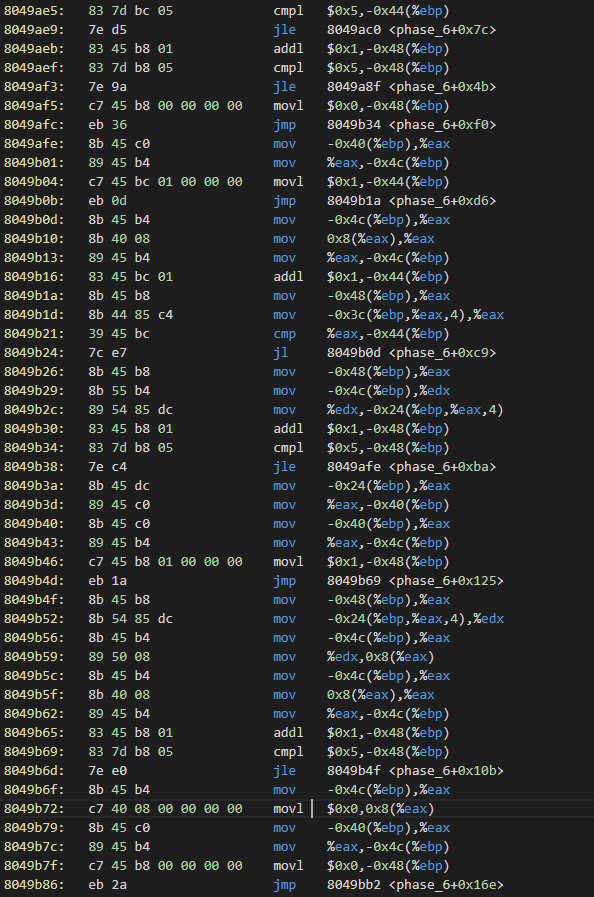


拆弹成功！

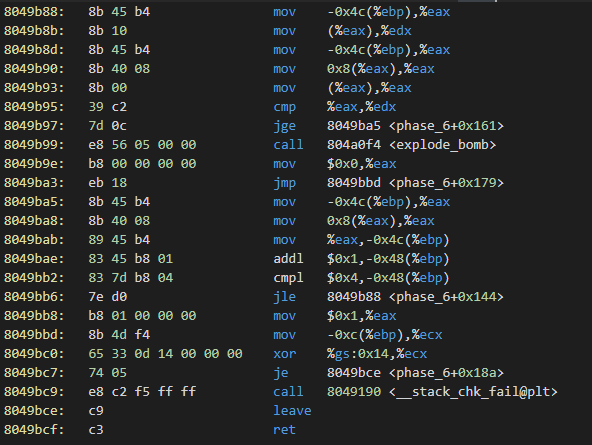
**Phase\_6:链表/二叉树**



对于每个结点内容的处理

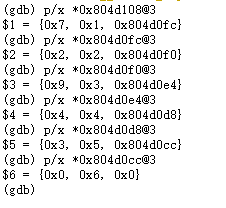


对于每个结点内容的处理



初观代码，挺长的。浏览一遍发现并有取得什么有用的信息，大概知道这是一个链表，并且知道了其中的一个结点可以查阅，然后就是我们需要输入六个数字。

从后面的长串代码中其实我们可以发现他只是实现了一个对于结点内容的大小排序所以题目就变得很简单了，我们只需要从第一个结点开始逐个查阅就可以了，运用指令p/x \*adress@3

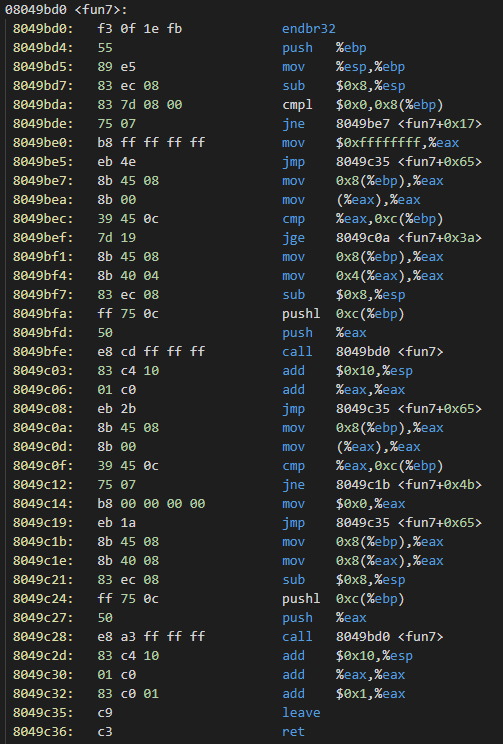


如上图所示就是我们查到的结果。其中第一个为该结点存储的值，第二个为该结点的序号，第三个为下一个结点的地址。由于代码中给出的机制是按照节点中的值从大到小进行排序，所以我们需要对结点序号按照其对应的值进行从大到小的排列，即答案为：3 1 4 5 2 6

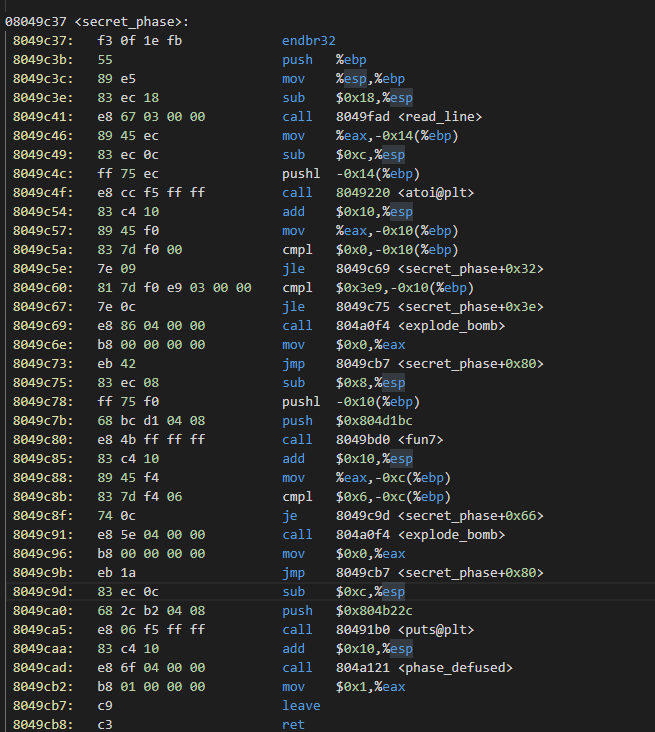


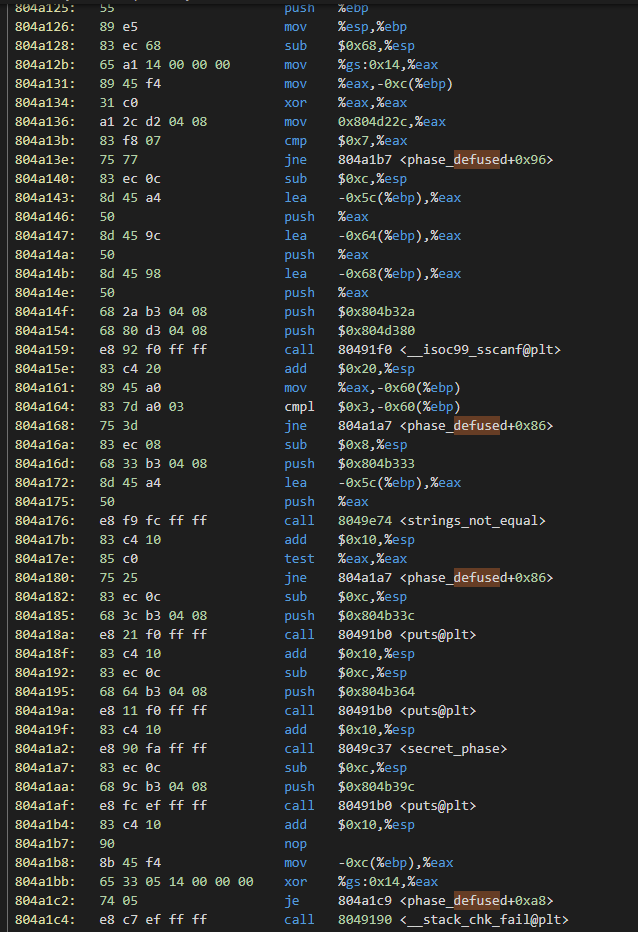
拆弹成功！

**Secret\_Phase:二叉树**



二叉树的实现



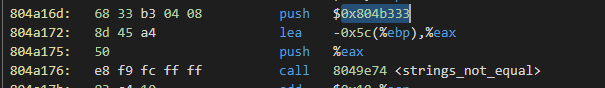


Phase\_defused

详细阐述了如何进入secret\_phase

首先我们需要明确如何才能进入secret\_phase：

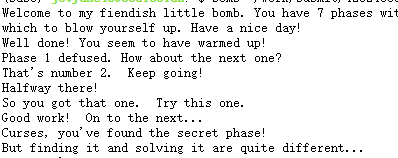
我们通过观察阅读phase\_defused我们会发现，只有当前面答对六道题，且在phase4中输入了正确的字符串，才可以进入secret\_phase。



我们在这里看到了<strings\_not\_equal>函数，所以我们判断phase4除了要输入两个数字，还应该输入一个正确的字符串，才可以进入隐藏炸弹，所以我们查看地址0x804b333：

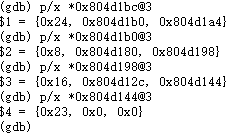


然后将之前的答案都输入，会得到：



我们成功进入了隐藏炸弹，下面分析如何拆解。

通过对于func7的阅读，我们知道这其实是一个二叉树的结构，左数为2k，右数为2k+1，而最终的答案为6，因此它的路径为：左-右-右。接下来我们就可以调试了：



因此我们需要输入的值就是最开始的结点里存储的值：0x23，即35



拆弹完成！

**至此，拆弹全部结束！！！**