Bomblab实验报告

**一、实验目标**

熟悉gdb调试、反汇编等等

**二、实验方法**

使用gdb调试的各种命令，尽量在不引爆炸弹的前提下发现正确答案

**三、实验结果**

1.拆弹程度：6个phase全部解开

2.我的密码是：

Why make trillions when we could make... billions?

0 1 1 2 3 5

1 w 142

12 43

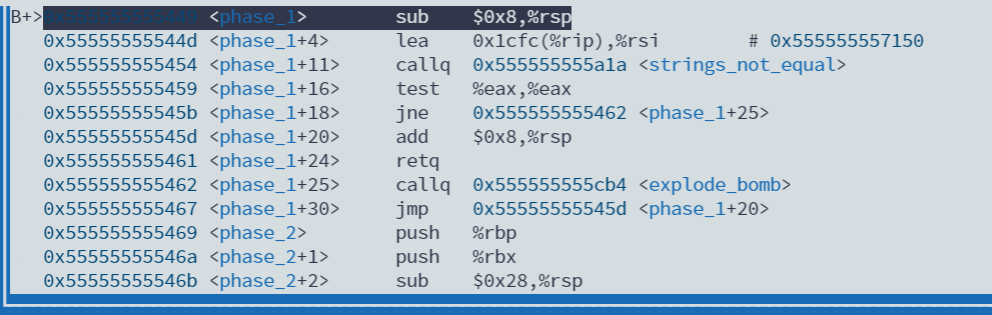
mfcdhg

4 6 5 1 2 3

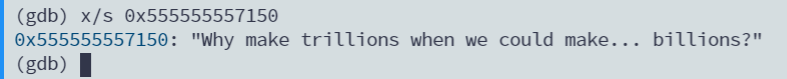
**四、详细过程**

**1.phase\_1**

* 观察这段代码，容易看出，主要是在调用strings\_not\_equal这个函数后，根据其返回结果判断是否触发爆炸，因此应该输入一个与地址0x555555557150处存放的字符串相同的字符串

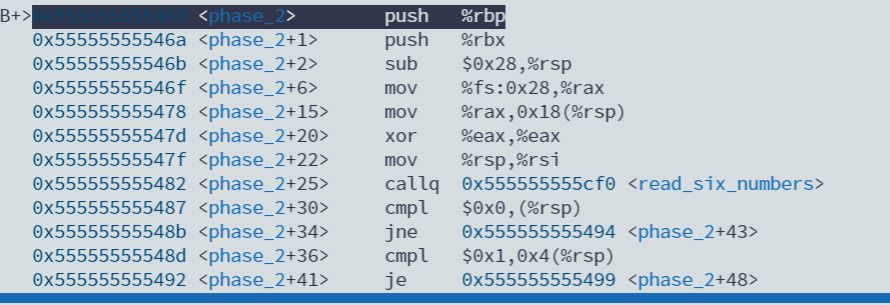


* ni指令，移动到call <strings\_not\_equal>函数处
* 运行指令 x/s 0x555555557150，得到答案



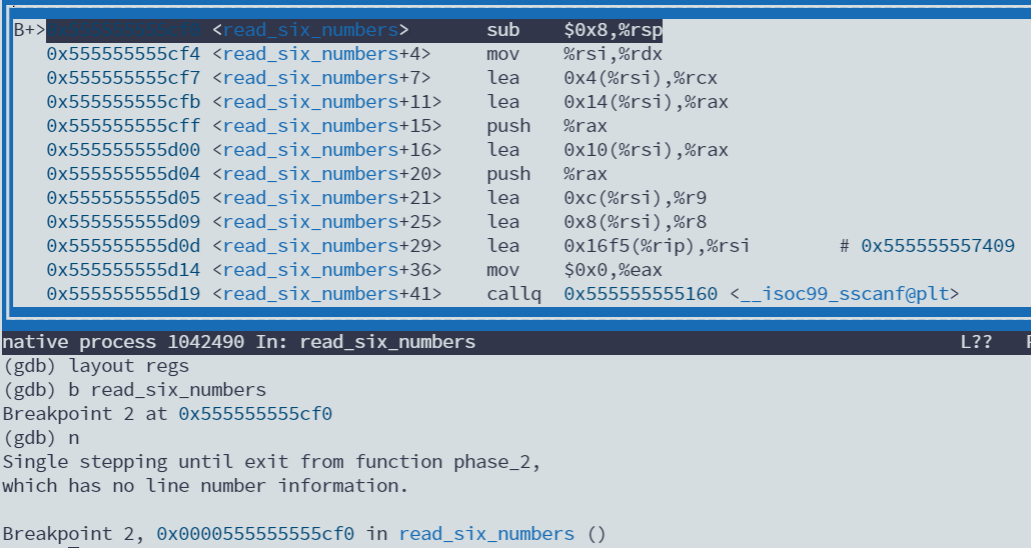
**2.phase\_2**

* 观察代码，看到有一个函数read\_six\_numbers，猜测本题要输入6个符合要求的数字



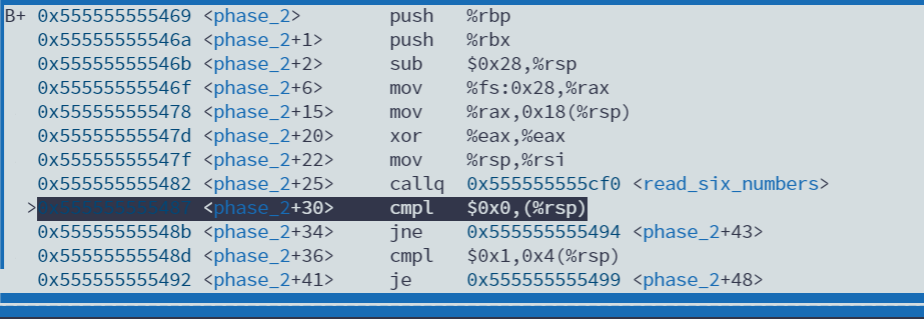
* 查看read\_six\_numbers函数代码，发现在此函数内调用了sscanf。

输入 x/s 0x555555557409查看其参数，根据结果可知sscanf从我的输入中读入6个整数。因此改变输入为6个空格分隔的整数（我在这里输入012345）

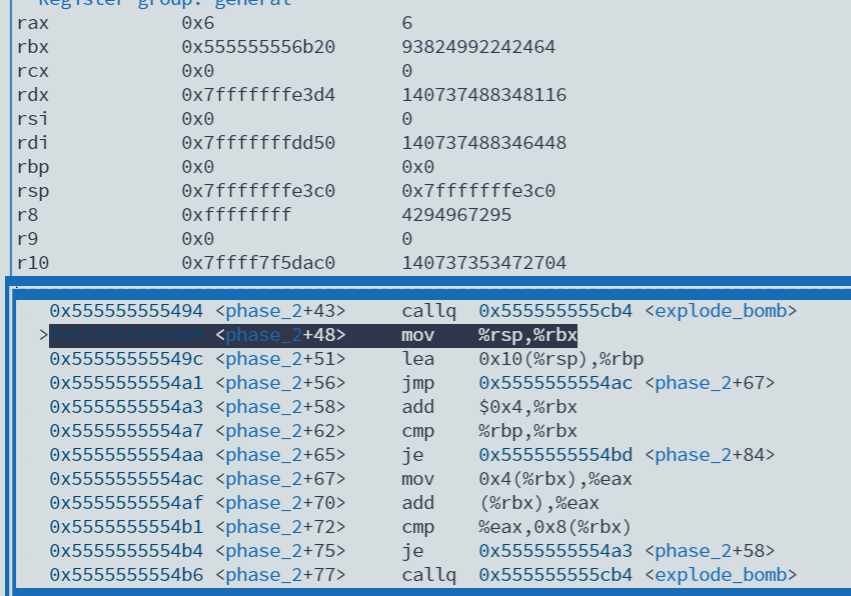




* 寄存器rsp中存放的是我输入的六个数字的首地址，根据标注的这一行，可以推出第一个数字是0。继续下一行，0x4(%rsp)是我输入的第二个数字，应该等于1。



* 跳转到标注的这一行，分析这段代码，是一个循环。
* 分析循环内容，可以发现第三个数等于前两个数相加，且之后遵循相同的规律。因此推断出答案应该是0 1 1 2 3 5



* 进行验证，确定答案就是0 1 1 2 3 5



**3.phase\_3**

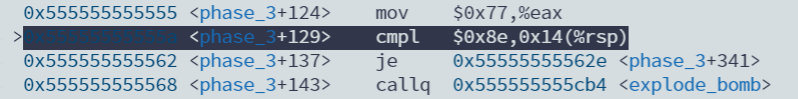
* 同上通过sscanf函数的参数，得到输入的字符串的形式应该是“%d %c %d”

并且可以看出，%rsp+0xf中存放输入的字符，%rsp+0x10中存放第一个整数，%rsp+0x14中存放第二个整数。尝试输入“1 a 2”，验证猜测正确。（代码截图不再重复贴出）

* 继续运行，看出第一个参数必须小于等于7



* 继续执行， 跳转到这里，可以看出第二个参数应该等于0x8e（即142）



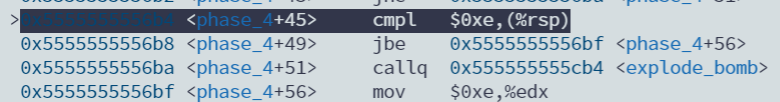
* 同理可以看出，字符参数的值应该为寄存器al中存放的值：0x77(即119)，去ASCII码表中寻找。对应的字符为w，从而确定答案为1 w 142



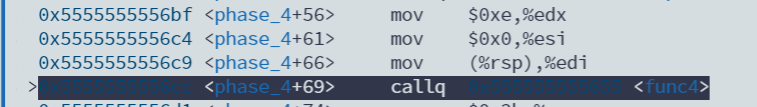
* 重新运行并输入1 w 142，phase\_3成功通过 

**4.phase\_4**

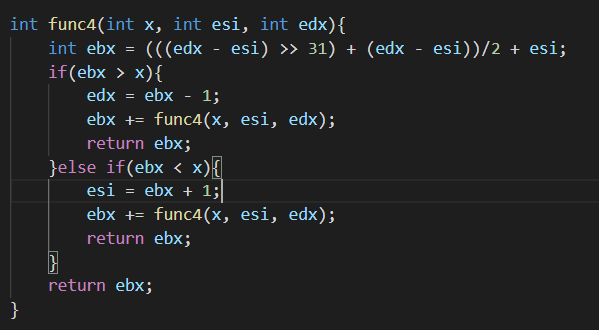
* 同样的，根据sscanf的参数，输入格式应为 “%d %d”，且两个参数分别存放在%rsp和%rsp+4的内存地址中。尝试输入“1 2”。（代码部分与之前重复，省略）
* 继续执行，可以看出第一个参数应该小于等于14（0xe）



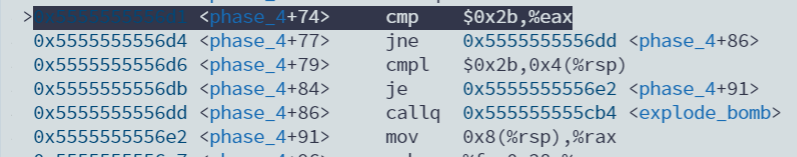
* 注意到在phase\_4中调用了函数func4，且其三个参数分别存放在%edi、%esi、%edx

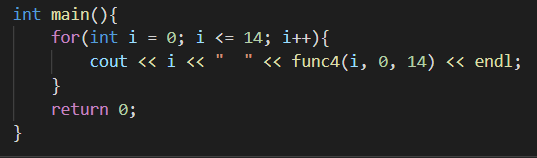
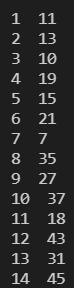


* 在func4处设置断点，根据汇编代码，发现这是一个递归函数，并据此写出c语言代码。



* 回到phase\_4函数查看剩余代码，容易看出func4函数的返回值应该等于0x2b（43），且输入的第二个参数也应该等于43。
* 还记得之前判断过第一个参数应该小于等于14，所以在c代码中从1到14循环调用func4函数（初值为x，0，14），寻找返回43的数



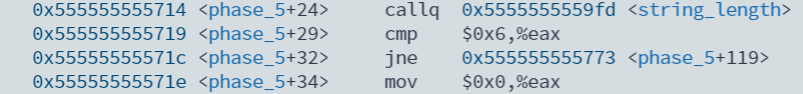
 

* 从输出结果中可以看出，12应该就是我们寻找的第一个参数，故本题答案为“12 43”

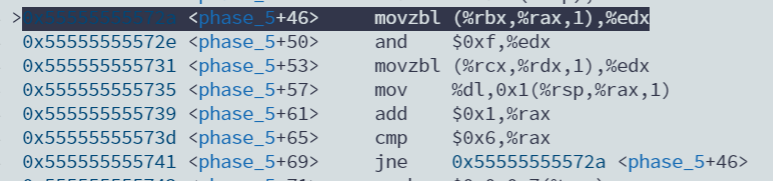


**5.phase\_5**

* 该函数中调用了一个名叫string\_length的函数，并且要求其返回值为6。所以应该输入一个长度为6的字符串，尝试输入“abcdef”，且我的输入被放在寄存器rbx内

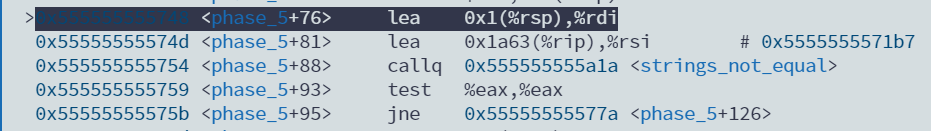


* 继续向下看，是一段循环。分析其作用，可以看出这个循环是以6个输入字符的ASCII码的低4位作为下标，找到它在%rcx，（即array.0）中对应的字符存到 %rsp+1，%rsp+2 一直到%rsp+6。





* 继续执行，见到了熟悉的strings\_not\_equal函数，根据phase\_1的分析，我们查看地址0x5555555571b1中的内容，并且应该让%rsp+1处存放的字符串与之相等





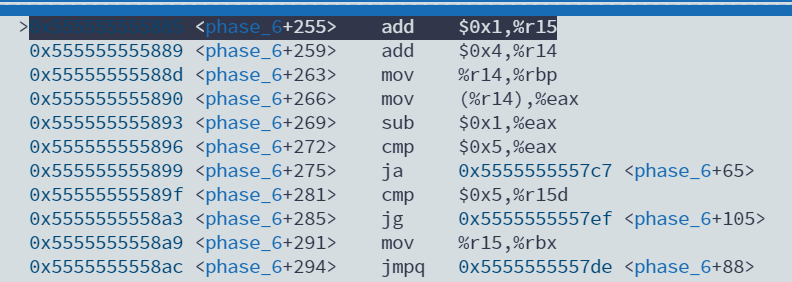
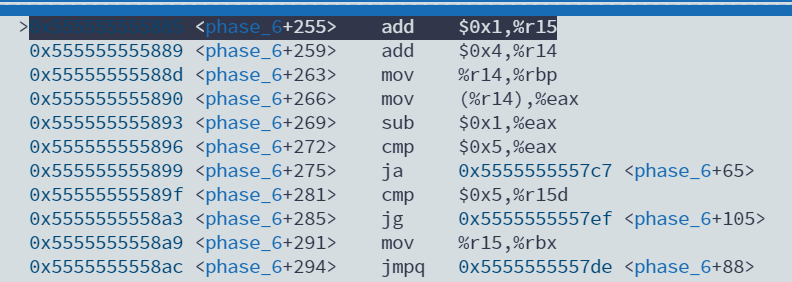
* 换言之，我们应该在array.0，即 “maduiersnfotvbyl”（对应0123456789ABCDEF） 中对应的找到 “bruins” 六个字母所对应的下标位置（从0~15），即D 6 3 4 8 7，然后去ASCII码表里找哪六个字符的ASCII码的低四位分别是 D 6 3 4 8 7的二进制形式即可。
* 最终找到对应的字符为mfcdhg 

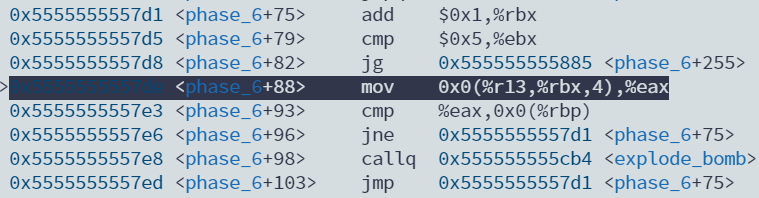
**6.phase\_6**

* Phase\_6中，根据前面分析过的read\_six\_numbers函数，猜测答案为6个整数。尝试输入1 2 3 4 5 6并发现我输入的数字分别存放在%r14，%r14+0x4……%r14+0x14内。

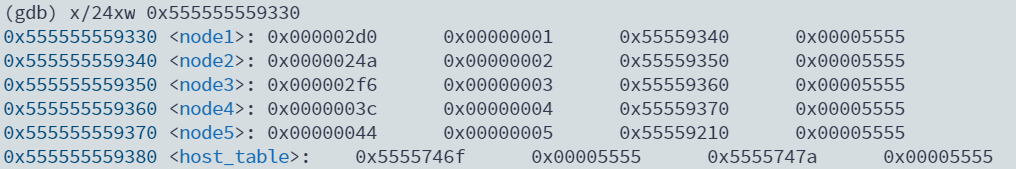


* 继续执行这段代码，是一个大循环套小循环，最终实现的效果是：对每个参数都要求其小于等于6，并调用各自的小循环，保证与后面的参数不相等，也就是说六个数互相不等，且都大于0小于等于6。大循环结束后，跳转到phase\_6+105



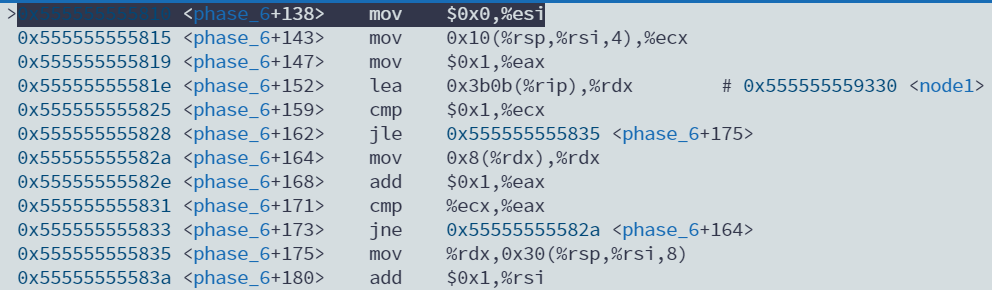


* 接下来的一段代码是用7依次减去原来的值并替换，我的输入123456变成654321，不再展示
* 继续执行，在代码中发现一个特殊的地址<node1>，将其输出，发现这应该是一个依次连接的链表。但比较特殊的是node6，它的地址在0x555555559210处。

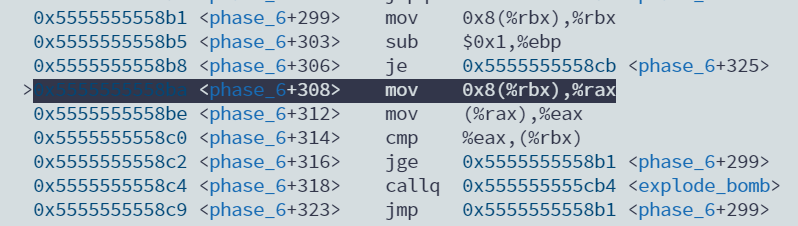




* 分析代码，发现它其实是按照我转变后的六个参数作为node的下标，按照其顺序依次链接node(参数1)->node(参数2)……->node(参数6)，即按照我目前的输入，它会链接出node6->node5->node4->node3->node2->node1，并将它们分别存放在%rsp+0x30，%rsp+0x38……%rsp+0x58，然后建立链接



* 继续执行，这段代码又是一个循环，它要求刚刚排好顺序的链表中结点的数值部分应该恰好也是降序排列，从大到小。否则就会爆炸。



* 根据刚才列出的6个node，可以降序排列出：

node3(2f6)->node1(2d0)->node2(24a)->node6(1ff)->node5(44)->node4(3c)

* 所以对应的变换后的输入应该是：3 1 2 6 5 4

所以我真正的输入，即答案应该用7减这个序列，得到4 6 5 1 2 3

验证答案正确，phase\_6解决。