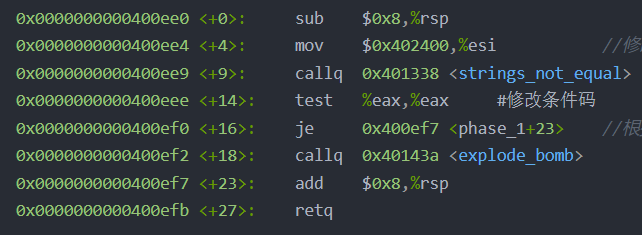
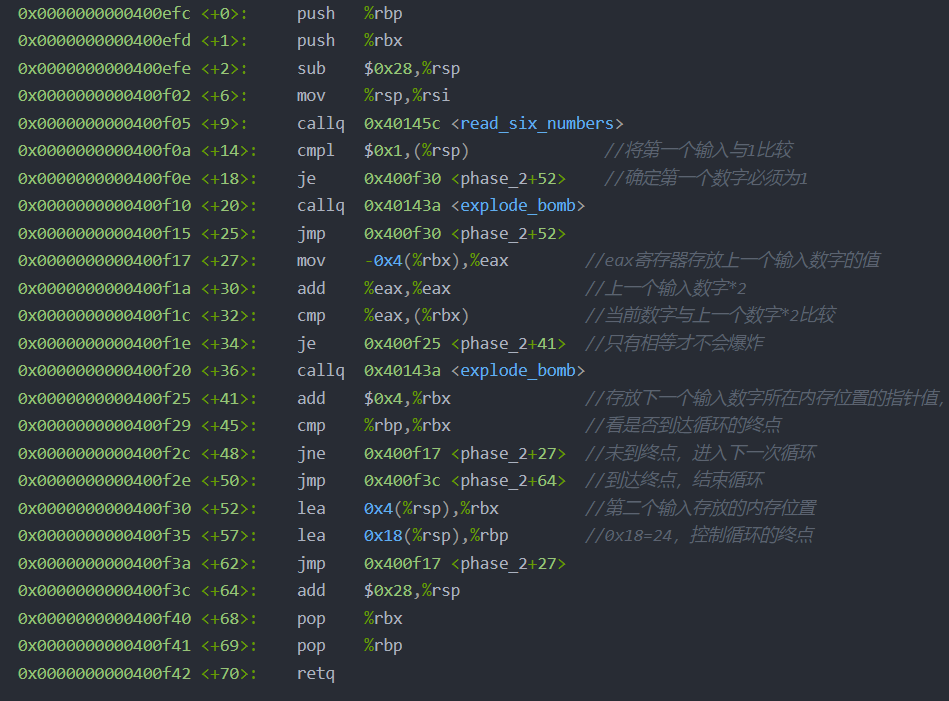
## 实验名称：bomblab

phase\_1



基本操作就是调用strings\_not\_equal函数，根据返回值来决定是否引爆炸弹。至于strings\_not\_equal函数，从名字就可以看出来是实现比较两字符串是否相等的功能。比较的两个字符串一个是用户输入的字符串，一个是地址存放在%esi寄存器里的内存中的值，也就是地址为0x402400的内存值。可以使用x/s 0x402400来查看对应的字符串，得到以下内容：“Border relations with Canada have never been better.”即为过关密码

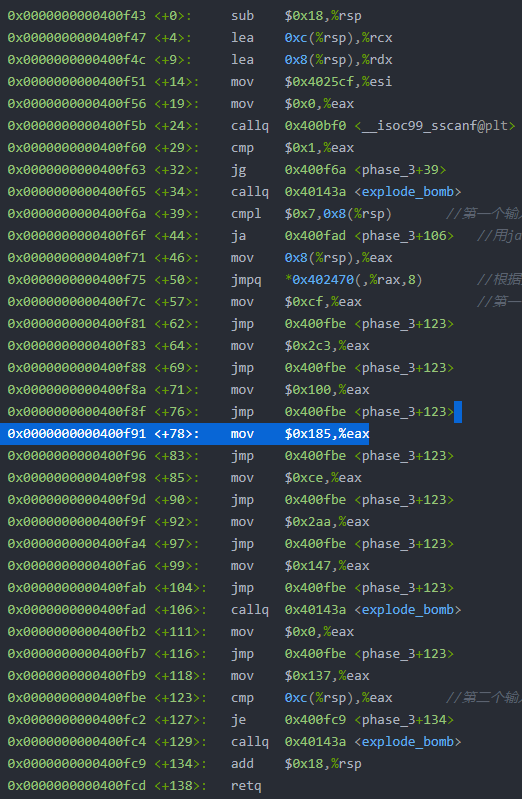
phase\_2



read\_six\_numbers函数完成的是读取用户输入的六个整数，每个数字占用四个字节大小空间，所以最终循环的终点为栈指针偏移4\*6=0x18=24的位置。

这部分代码实现的是一个循环，将输入的六个数字依次与其前一位的两倍进行比较。当且仅当第一个数字为1，后一位依次是前一位的两倍的情况下，才不会引起爆炸。所以得到第二问的答案为1 2 4 8 16 32

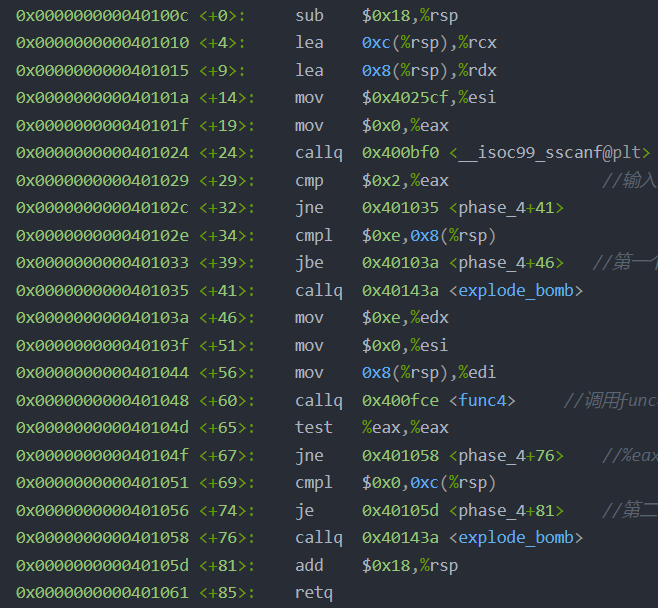
phase\_3



\_\_isoc99\_sscanf@plt函数实现的是输入，并且返回值eax是输入值的个数。从jmpq \*0x402470(,%rax,8)这条指令往下，都是和对应跳转的位置有关。跳转到哪一行与第一个输入的数值有关，所以这一部分考察的应该是switch语句的相关知识。第一个输入与7进行无符号数比较，则可以断定第一个输入只可以为0，1，2，3，4，5，6。再分别计算0x402470(,%rax,8)的地址，就可以得到对应的跳转位置。第二个输入会根据第一个输入与相应的数值比较，只有相等才可以通过。最终可以计算第三部分的答案如下

0 207（或1 311，或2 707，或3 256，或4 389，或5 206，或6 682）

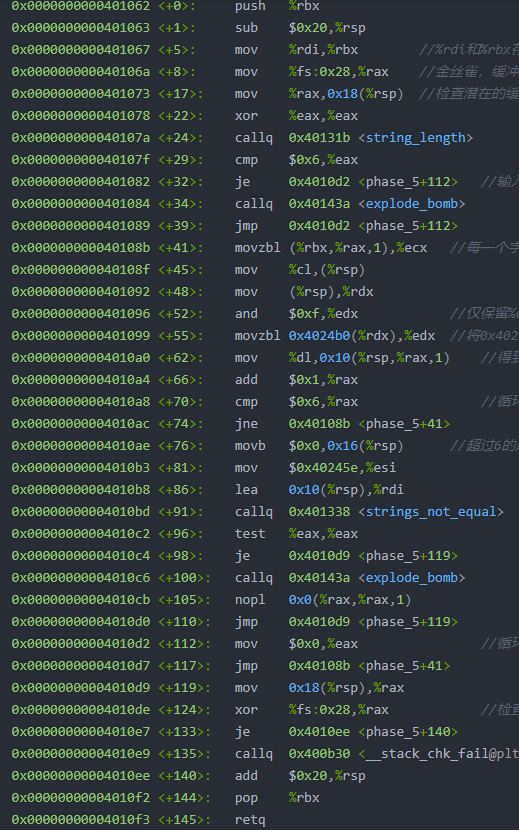
phase\_4



根据分析，phase\_4是比较第一个输入和%edx与%esi两个寄存器的值的均值，类似二分法折半查找。最直接的答案是第一次进入func4就直接出来，也就是第一个输入正好等于均值。由于在调用之前%edx=14，%esi=0,所以第一个输入为7时，可以递归一次直接返回到phase\_4，这时%eax返回值也为0，不会引爆炸弹。

如果想进一步分析的话，可以看到无论函数内部实现如何，我们只希望返回值%eax必须为0就可以了。而%eax一般都是置为0的，第20行的lea 0x1(%rax,%rax,1),%eax会对其产生影响，所以让每次递归都是上界变小就会保证%eax一定为0。这里只考虑非负数的情况，假设每次递归都是上界减小，也就是均值大于第一个输入的值，直到最终结束递归。所以答案为7 0（或3 0，或1 0，或0 0…）

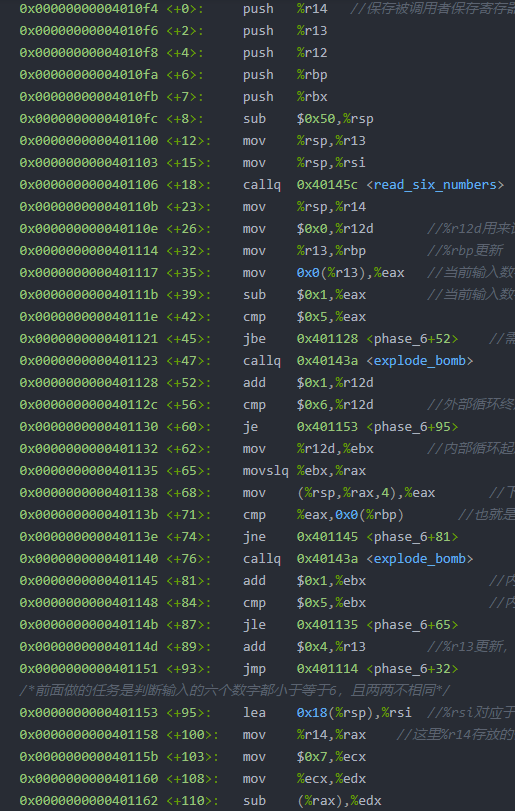
phase\_5



先找到0x40245e对应内存位置存放的字符串为“flyers"，经过分析后发现比较的两个字符串其中一个是内存地址为0x40245e的字符串，但另一个并不是输入，而是输入映射后得到的新的六个字符。也就是当你输入"flyers"，结果并不会是”flyers"。具体来说，在第一次循环时，%rbx存放的是"f"的ascii码0x66，%edx中保存的是该ascii码的最后4位，也就是0x6。再根据movzbl 0x4024b0(%rdx),%edx这句话，此时%edx得到的值为"r"，不是一开始的"f"。所以该问题还是得找到映射关系

使用x/s 0x4024b0查看得到结果如下使用x/s 0x4024b0查看得到结果如下“maduiersnfotvbylSo you think you can stop the bomb with ctrl-c, do you?”前面的那些小写字母得顺序就是对应的映射关系。依次找到“f", “l”, “y”, “e”, “r”, "s"对应的位置是“9”，“15”，“14”，“5”，“6”，“7”。所以只需要ascii码最后四位值等于这些的字符就可以解开。依次选取了“0x69”，“0x6f”，“0x6e”，“0x65”，“0x66”，“0x67”。也就是如下ionefg.

phase\_6



第一块是由两个循环嵌套组成的。外层循环依次遍历输入的第一个数字到第六个数字。外层循环首先判断输入的数字是否不超过6，紧接着内存循环遍历当前输入的数字到末尾剩下的数字，实现的功能是判断输入的六个数字两两不相等。

第二块比较简单，用一个寄存器%edx依次保存7与输入的数字的差值，再将差值覆盖原来输入的数字。

第三块是根据前两块实现后得到的新的6个数字进行操作。在内存0x6032d0到0x60332f中存放着6个大小为16字节的结构体变量。每个结构体变量的后8个字节都存放着一个指针，指向下一个结构体变量的内存位置。这部分代码是根据新的6个数字的数值来重新给这几个结构体排序，将这6个结构体变量的初始地址按照排好的顺序依次存放在栈指针偏移0x20~0x50的位置。

第四块是修改6个结构体变量中存放的指针值，按照重新排好的顺序让前一个指针指向后一个指针，所以这里基本可以确定第三第四块实现的是重新排列一个链表节点的功能。

第五块是验证新排列的六个结构体变量是否是按照从大到小的顺序来排列的，如果验证满足，则炸弹就拆除了。

node3.value>node4.value>node5.value>node6.value>node1.value>node2.value  
所以第二块得到的新的六个数字为：3，4，5，6，1，2  
所以一开始的输入的六个数字为：4 3 2 1 6 5