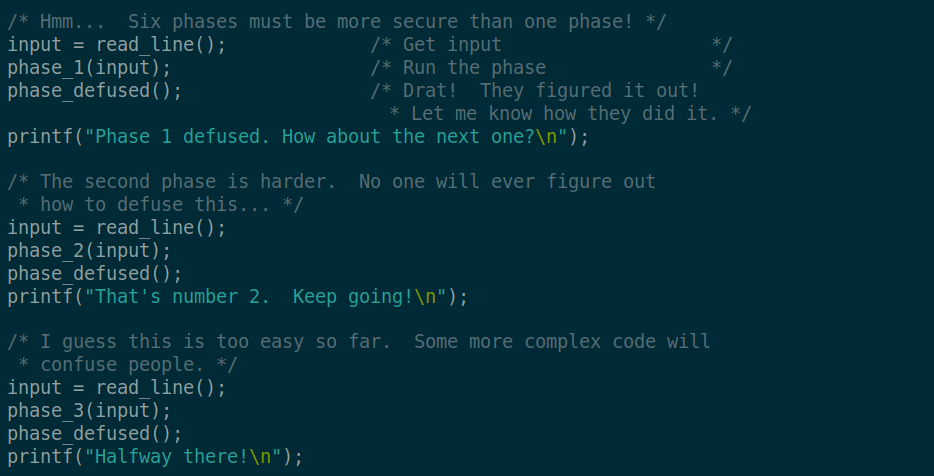
首先我们观察bomb.c文件的代码：

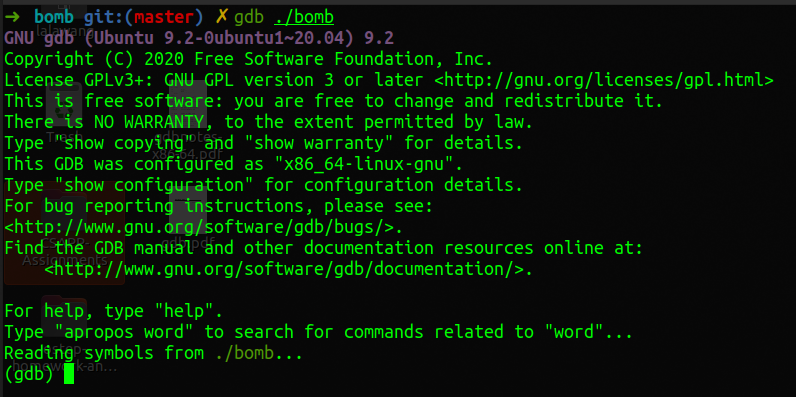


可以很明显的观察到，判断输入字符串是否是接触炸弹所需要的逻辑存在于phase\_1、phase\_2...phase\_6这几个函数中，我们只需要搞清楚这几个函数内部的逻辑，就可以获知解除各个炸弹所需要的字符串内容了。

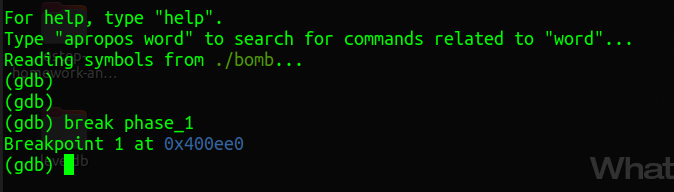
接下来分别讲述解除6个炸弹的过程：

1. phase\_1:

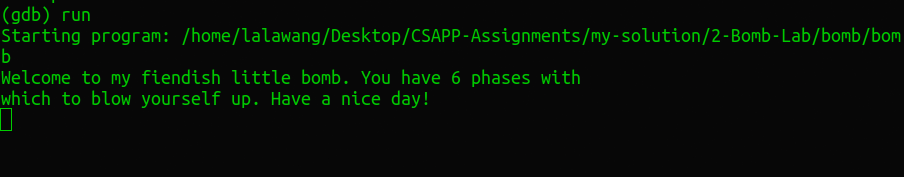
首先通过gdb调试./bomb可执行文件：



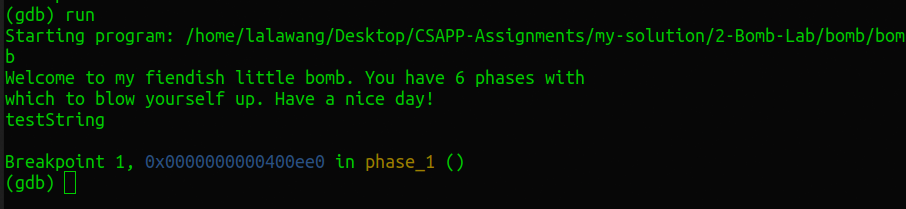
按照上面所述的逻辑，我们在phase\_1函数处打一个断点：



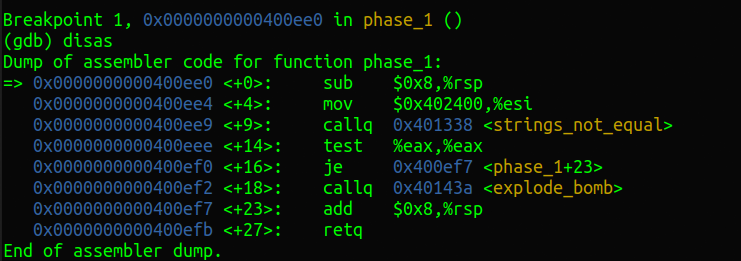
接下来输入run命令运行：



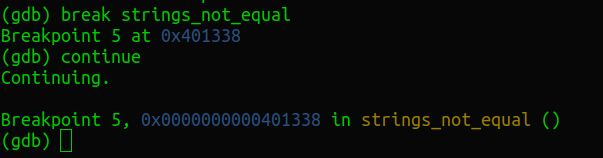
此时程序向我们索要第一个解除炸弹的字符串，这里先输入testString, 随后程序将在我们打的断点处停止：



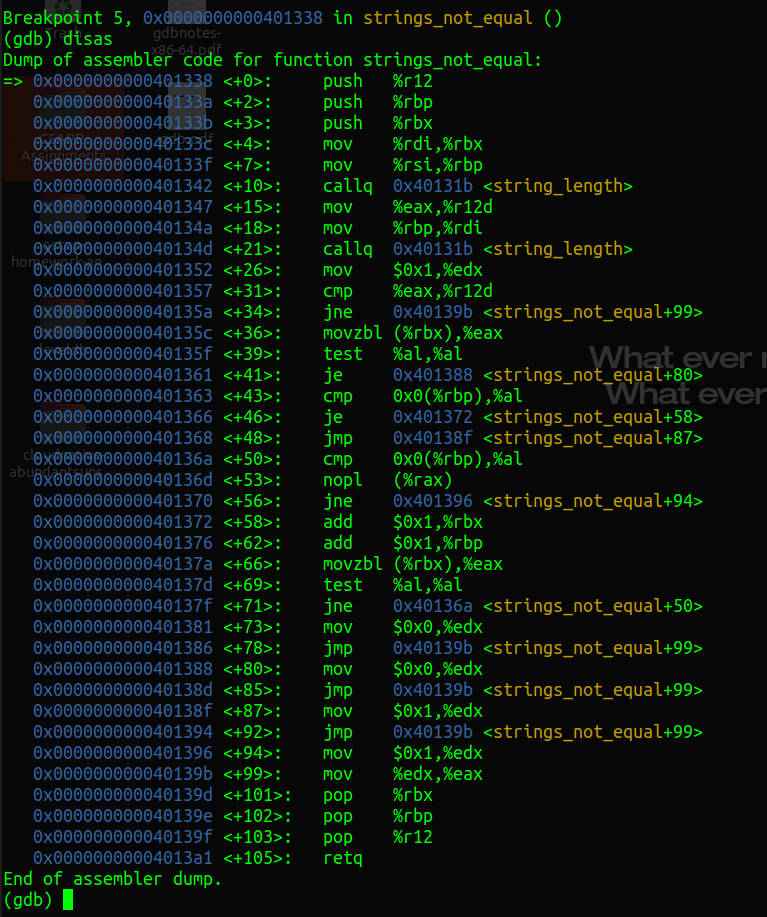
为了搞清楚phase\_1函数内部发生了什么，这里用disas命令将函数解汇编：



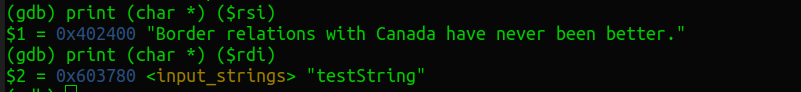
观察此处汇编代码的逻辑，实际上就是根据函数strings\_not\_equal的运行结果来决定是否调用explode\_bomb函数，如果调用该函数，则本次解除炸弹失败，因此我们需要了解函数strings\_not\_equal做了什么，在这个函数处也打一个断点, 同时让函数走到这个断点处：



再一次地，对strings\_not\_equal这个函数解汇编：



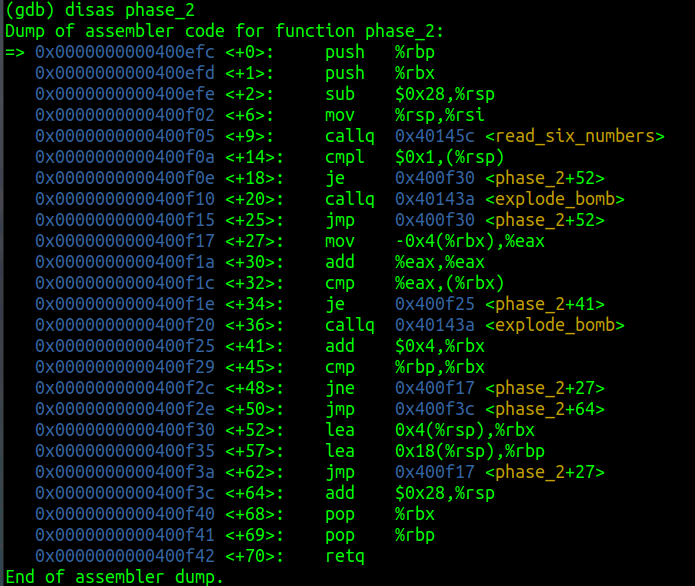
注意看<+10><+21>两处的代码，实际上是调用了两次string\_length函数来获得输入字符串和输出字符串的长度，并将结果存在%r12d和%eax寄存器里面，若果说长度都不相等，那么直接跳转至<+99>处，所做的事情就是将寄存器%eax的值置为0x1，而回过头来看phase\_1函数的汇编代码，可以发现如果%eax寄存器的值不为0，炸弹就会爆炸，因此我们可以推断两次调用string\_length的函数，是为了获得输入字符串和目标字符串的长度的，我们只需要看两次函数调用对应的参数即可，也就是参数寄存器%rsi和%rdi对应地址里存的值：



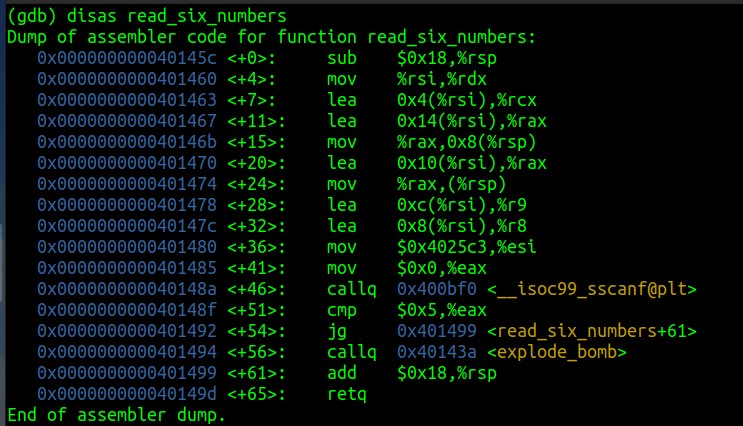
rdi是我们此前暂时输入的测试字符串testtring，这也验证了此前的猜想，那么另一个就是解除炸弹所需要的了。

1. phase\_2

和上述步骤一样，首先对phase\_2用disas命令解汇编：



阅读此处汇编代码,首先第一个关注点就在于read\_six\_numbers这个函数，从函数名上看推断，这个函数的功能应该是读入六个数字，对该函数解汇编：



可以看到，read\_six\_numbers这个函数的主体逻辑主要就是三步：

1. 给参数寄存器中传入相应参数，<+0>~<+41>
2. 调用\_\_isoc99\_sscanf@plt函数，<+46>
3. 跳转，若寄存器%eax中的值不大于五，则引爆炸弹，否则返回。<+51>~<+65>

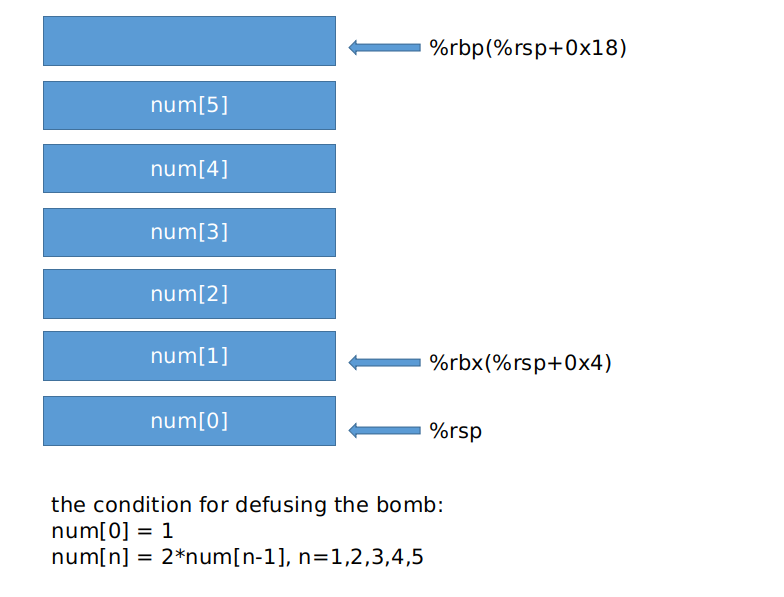
结合函数名和炸弹在此处起爆的逻辑，可以推测出此函数确实是在读取输入的数字，如果输入的数字数量小于6，那么炸弹将直接起爆，只有当数字数量大于等于6时才进行下一步的判断。

再观察phase\_2函数调用完read\_six\_numbers函数后的处理逻辑：

可以看出此处是一个循环处理的逻辑：

1. 首先判断%rsp内存的地址对应内存中数字是否为1，是则继续，否则爆炸
2. 将%rsp+0x4和%rsp+0x18分别赋值给%rbx和%rbp，刚好是20个字节，能够存储五个int类型的数据
3. 循环判断，%rbx指向当前判断的数字，%rbp指向循环的边界，每次判断%rbx指向的数字是否是(%rbp-0x4)指向的数字，也就是上一个内存单元中存储的数字的两倍，如果是，则继续循环，否则引爆炸弹。

对应的内存布局如图所示：



因此这六个数字应该是1 2 4 8 16 32