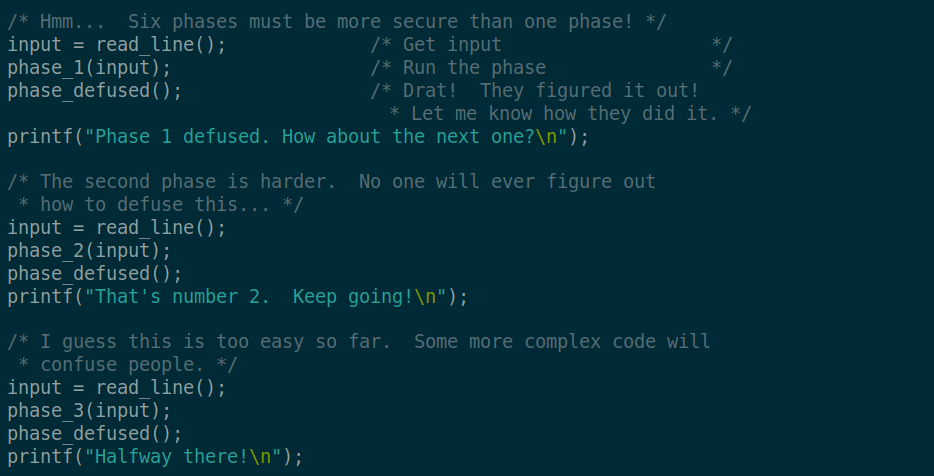
首先我们观察bomb.c文件的代码：

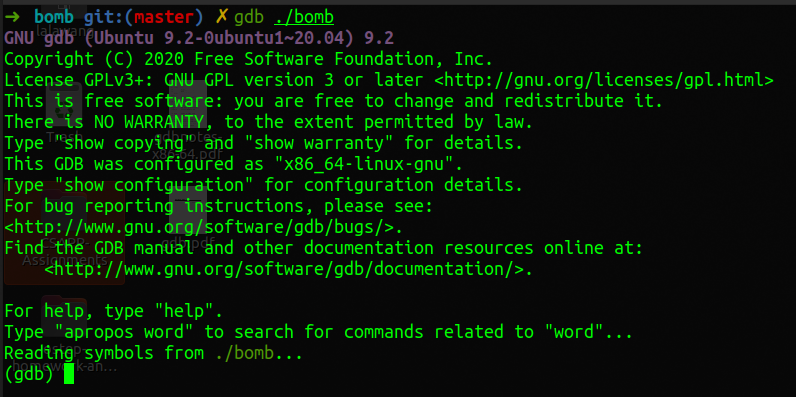


可以很明显的观察到，判断输入字符串是否是接触炸弹所需要的逻辑存在于phase\_1、phase\_2...phase\_6这几个函数中，我们只需要搞清楚这几个函数内部的逻辑，就可以获知解除各个炸弹所需要的字符串内容了。

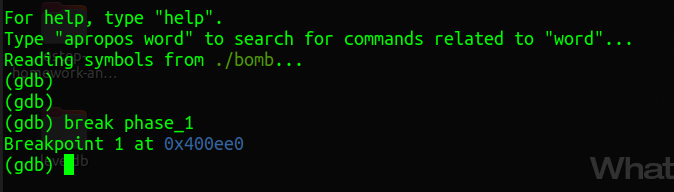
接下来分别讲述解除6个炸弹的过程：

1. phase\_1:

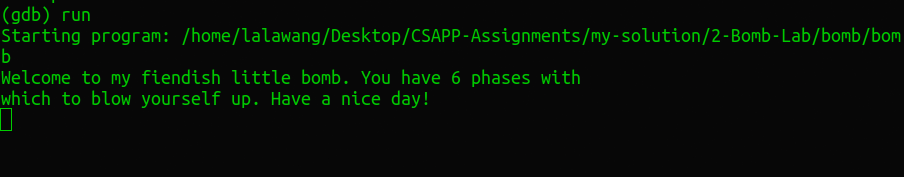
首先通过gdb调试./bomb可执行文件：



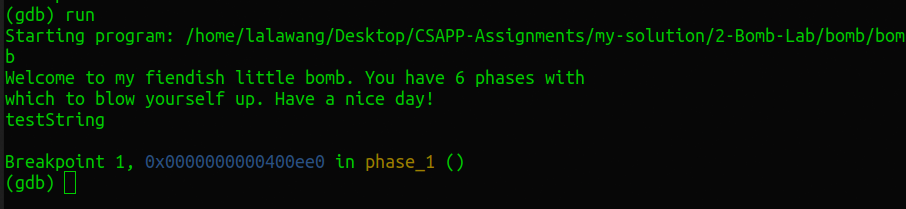
按照上面所述的逻辑，我们在phase\_1函数处打一个断点：



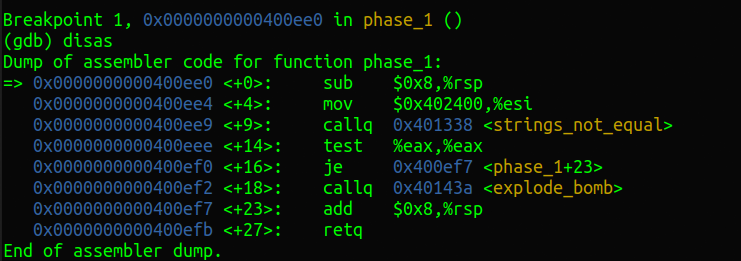
接下来输入run命令运行：



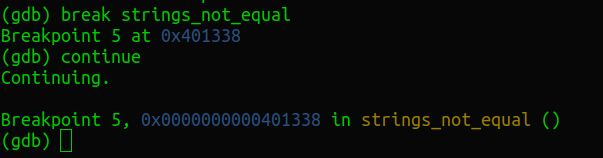
此时程序向我们索要第一个解除炸弹的字符串，这里先输入testString, 随后程序将在我们打的断点处停止：



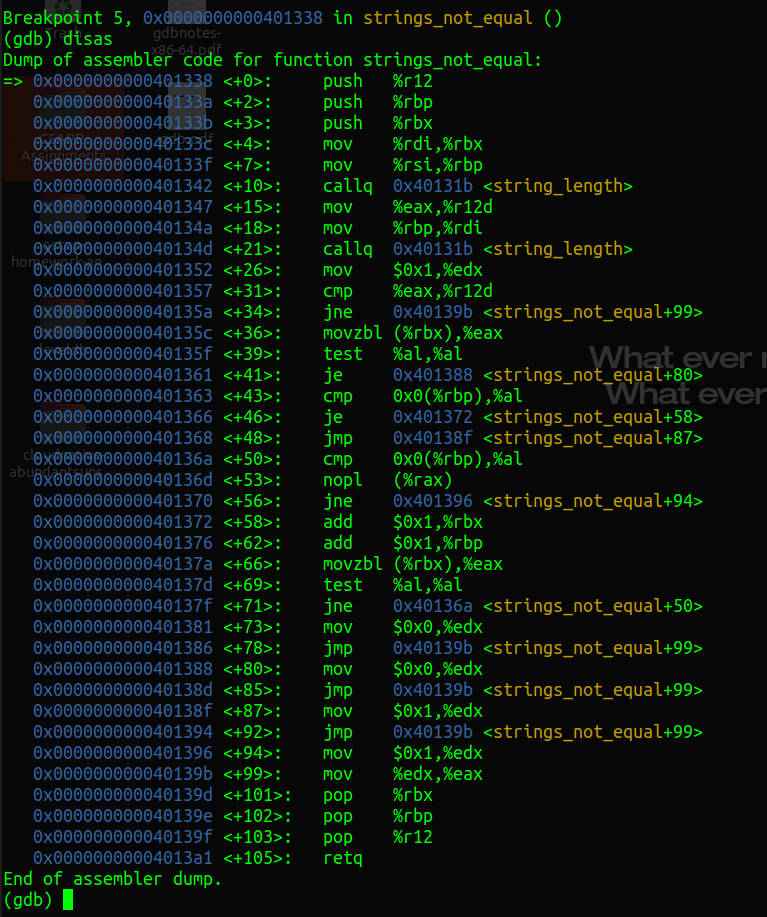
为了搞清楚phase\_1函数内部发生了什么，这里用disas命令将函数解汇编：



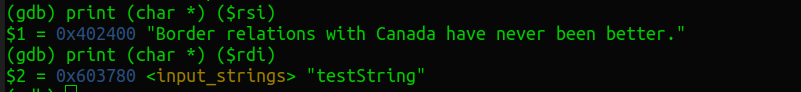
观察此处汇编代码的逻辑，实际上就是根据函数strings\_not\_equal的运行结果来决定是否调用explode\_bomb函数，如果调用该函数，则本次解除炸弹失败，因此我们需要了解函数strings\_not\_equal做了什么，在这个函数处也打一个断点, 同时让函数走到这个断点处：



再一次地，对strings\_not\_equal这个函数解汇编：



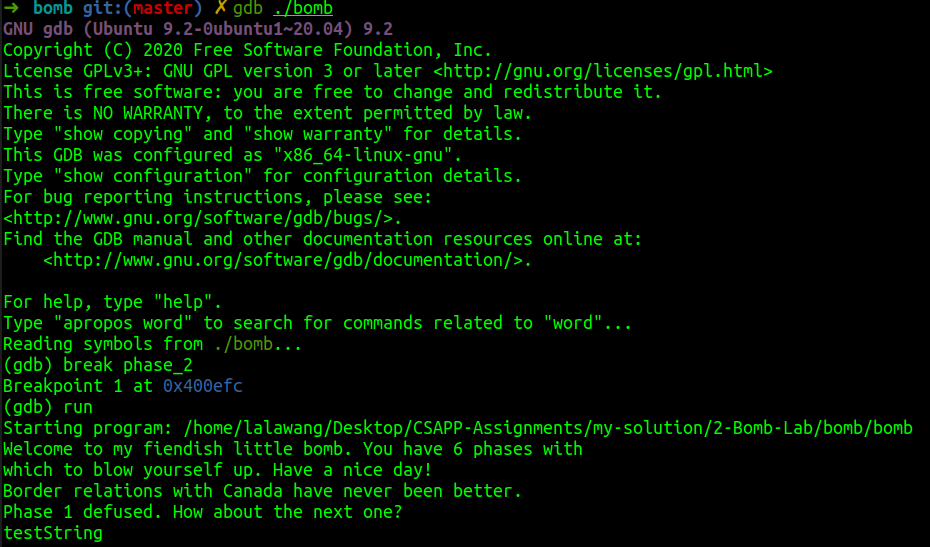
注意看<+10><+21>两处的代码，实际上是调用了两次string\_length函数来获得输入字符串和输出字符串的长度，并将结果存在%r12d和%eax寄存器里面，若果说长度都不相等，那么直接跳转至<+99>处，所做的事情就是将寄存器%eax的值置为0x1，而回过头来看phase\_1函数的汇编代码，可以发现如果%eax寄存器的值不为0，炸弹就会爆炸，因此我们可以推断两次调用string\_length的函数，是为了获得输入字符串和目标字符串的长度的，我们只需要看两次函数调用对应的参数即可，也就是参数寄存器%rsi和%rdi对应地址里存的值：

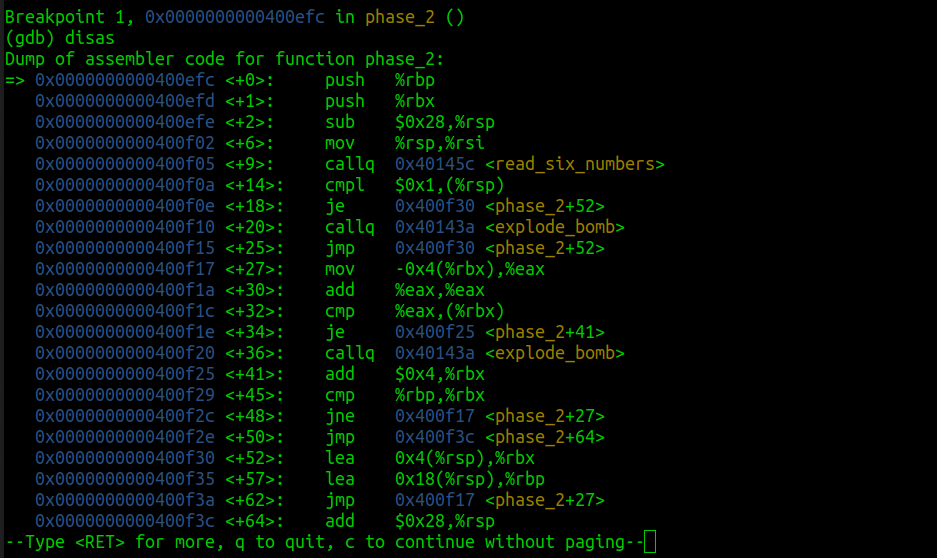


rdi是我们此前暂时输入的测试字符串testtring，这也验证了此前的猜想，那么另一个就是解除炸弹所需要的了。

1. phase\_2

和上述步骤一样，我们首先在phase\_2这个函数处打一个断点，然后用run命令开始运行，将第一个炸弹解除之后，再次输入testring作为测试用的提交的字符串，输入后程序会停止在此前我们打下的断点，类似地，在此处用disas命令解汇编：





相比于phase\_1中的strings\_not\_equal的直观，此时这里的逻辑更为复杂一些，无法一眼看出这一段代码的真实意图，我们可以暂时将不明白的部分作为一个黑箱看待，之后再去探究细节，此处先去理解各个黑箱之间的逻辑结构，

read\_six\_numbers:

