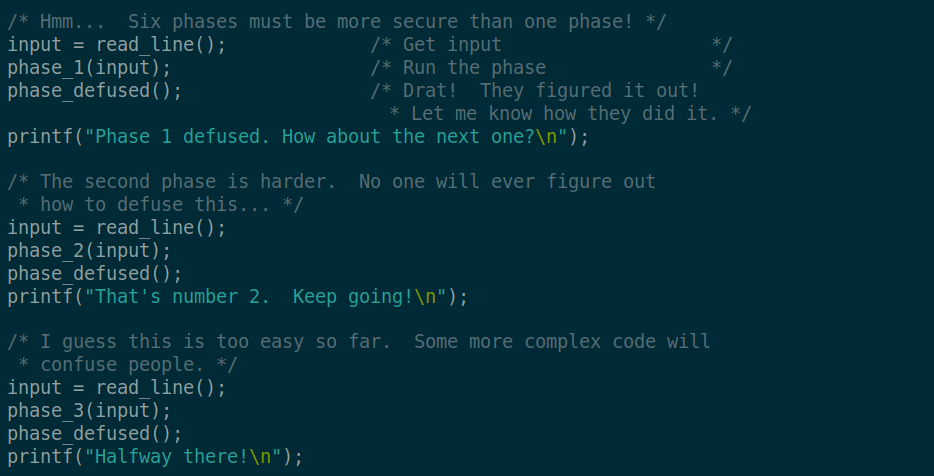
First we observe the code of the bomb.c file：

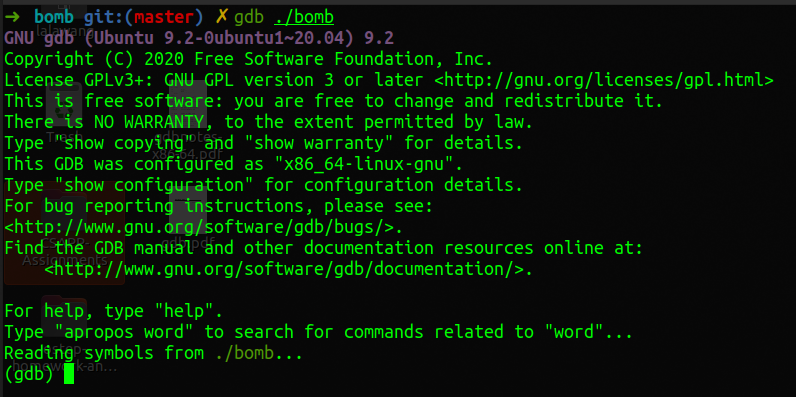


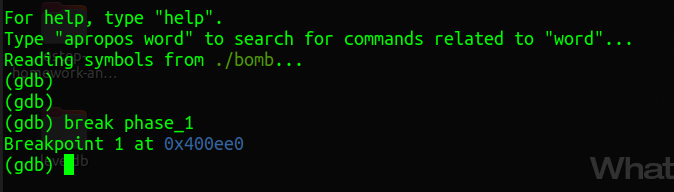
It can be clearly observed that the logic required to determine whether the input string is required to disarm the bomb exists in the phase\_1, phase\_2...phase\_6 functions. We only need to figure out the internal logic of these functions, then we can know the string required to disarm each bomb.

Next, here will describe the process of disarming the 6 bombs:

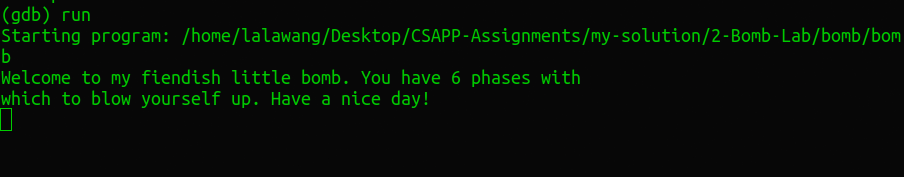
1. phase\_1:

First debug the ./bomb executable file through gdb:

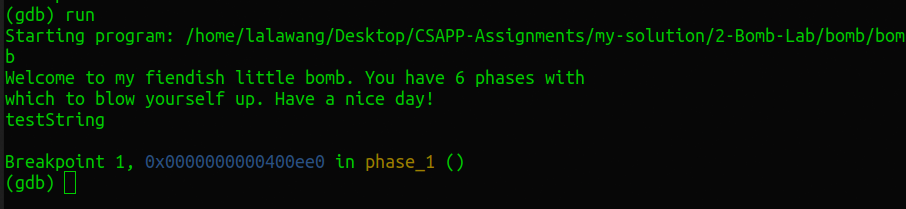


According to the logic described above, we set a breakpoint at the phase\_1 function:

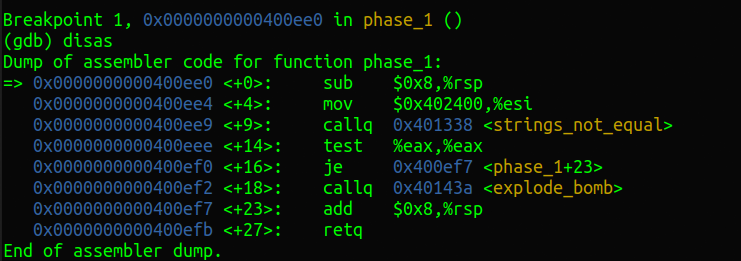
Then enter the **run** command to run:



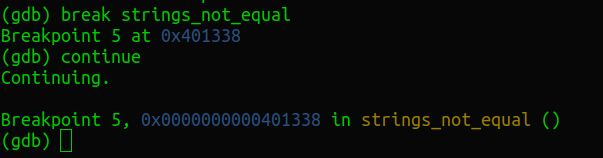
At this point, the program asks us for the first string to disarm the bomb. Here, enter “testString”, and then the program will stop at the breakpoint we hit:



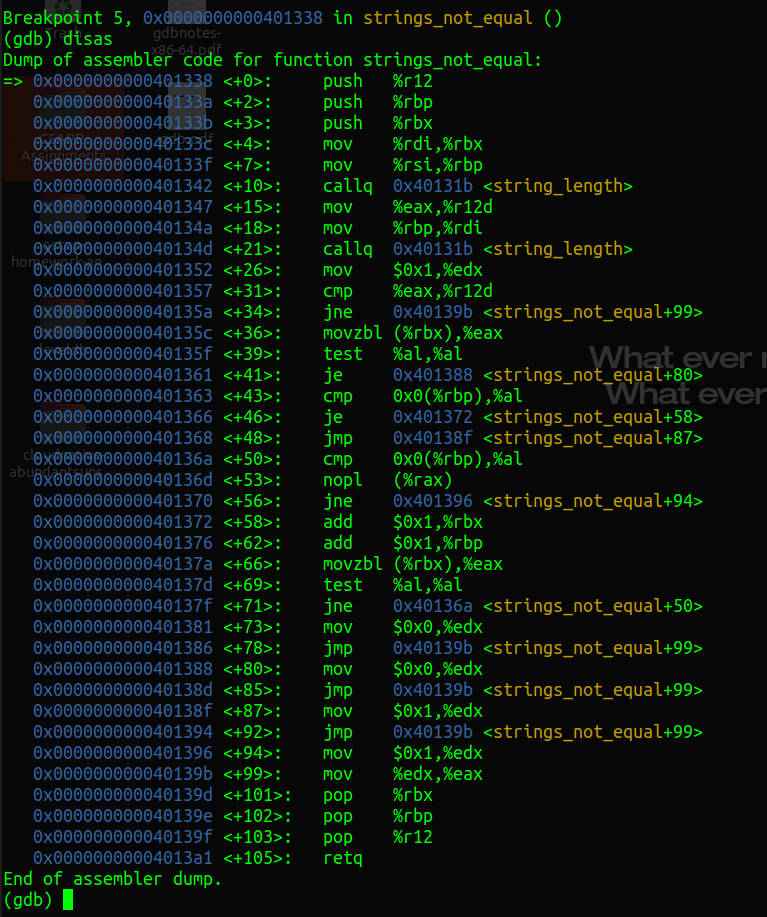
In order to figure out what is happening inside the phase\_1 function, use the **disas** command to disassemble the function:



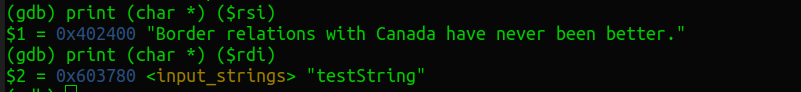
Observing the logic of the assembly code here, it is actually based on the running result of the function ***strings\_not\_equal*** to decide whether to call the explode\_bomb function. If this function is called, we will fail this time, so we need to understand what the function ***strings\_not\_equal*** does. Also hit a breakpoint, and let the function go to this breakpoint at the same time:



Once again, disassemble the ***strings\_not\_equal*** function:



Pay attention to the code in the two places <+10><+21>. In fact, the ***string\_length*** function is called twice to obtain the length of the input string and the target string, and the result is stored in the %r12d and %eax registers. Say that the lengths are not equal, then jump directly to <+99>, what it do is set the value of register %eax to 0x1, and looking back at the assembly code of the ***phase\_1*** function, you can find that if the %eax register‘s value is not 0, the bomb will explode. Therefore, we can infer that calling the ***string\_length*** function twice is to obtain the length of the input string and the target string. We only need to look at the parameters corresponding to the two function calls. It is the value stored in the corresponding address of the parameter register %rsi and %rdi:

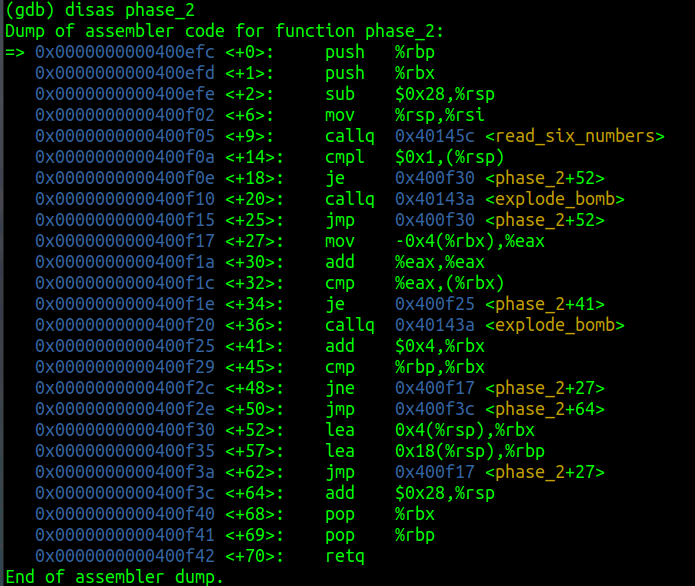


The %rdi register corresponds to the test string “testString” that we temporarily input before, which also verifies the previous conjecture, so the other is what is needed to disarm the bomb:

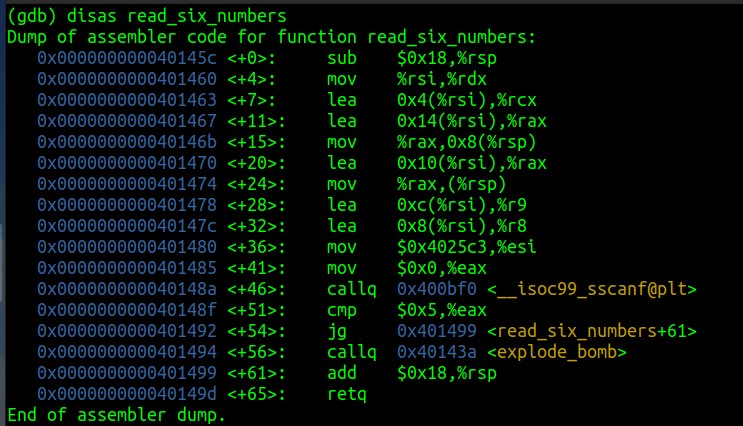
**Border relations with Canada have never been better.**

1. phase\_2

First, disassemble the function *phsae\_2*:



Reading the assembly code here, the first point to concern is the function read\_six\_numbers. From the function name, it can be inferred that the function of this function should be to read six numbers. disassemble the function:

As you can see, the main logic of the read\_six\_numbers function is mainly three steps:

1. Pass the corresponding parameters into the parameter register，<+0>~<+41>
2. Call \_\_isoc99\_sscanf@plt function，<+46>
3. If the value in register %eax is not greater than five, detonate the bomb, otherwise return.<+51>~<+65>

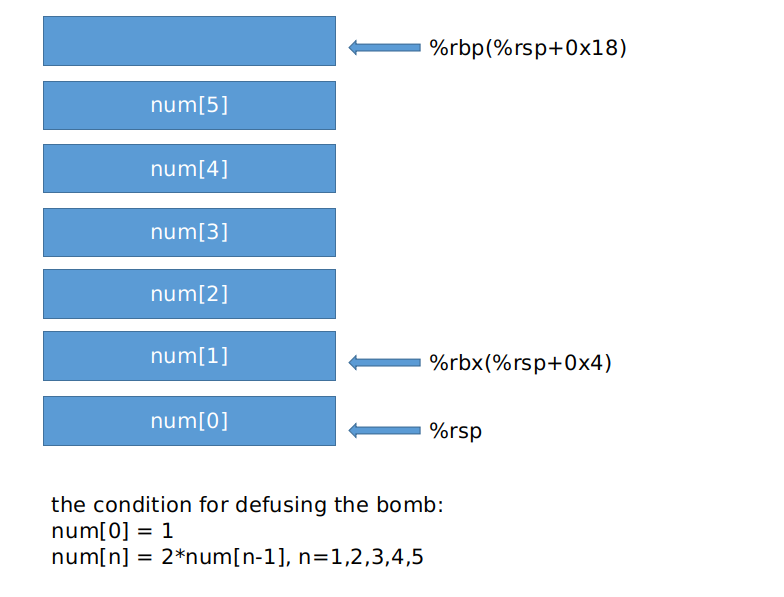
Combining the function name and the logic of the bomb detonation here, it can be inferred that this function is indeed reading the input number. If the number of input numbers is less than 6, then the bomb will directly detonate. Only when the number of numbers is greater than or equal to 6 the function will continue.

Then observe the processing logic after the phase\_2 function calls the read\_six\_numbers function:

It can be seen that there is a loop processing logic:

1. First judge whether the number in the memory pointed to by %rsp is 1, if yes, continue, otherwise it will explode.
2. Assign %rsp+0x4 and %rsp+0x18 to %rbx and %rbp respectively, which are exactly 20 bytes, which can store five int type data
3. Cycle judgment, %rbx points to the number currently judged, %rbp points to the boundary of the cycle, each time it is judged whether the number pointed to by %rbx is the number pointed to by (%rbp-0x4), which is the number stored in the previous memory unit. If it is, continue the cycle, otherwise detonate the bomb.

The corresponding memory layout is shown in the figure:



So these six numbers should be

**1 2 4 8 16 32**