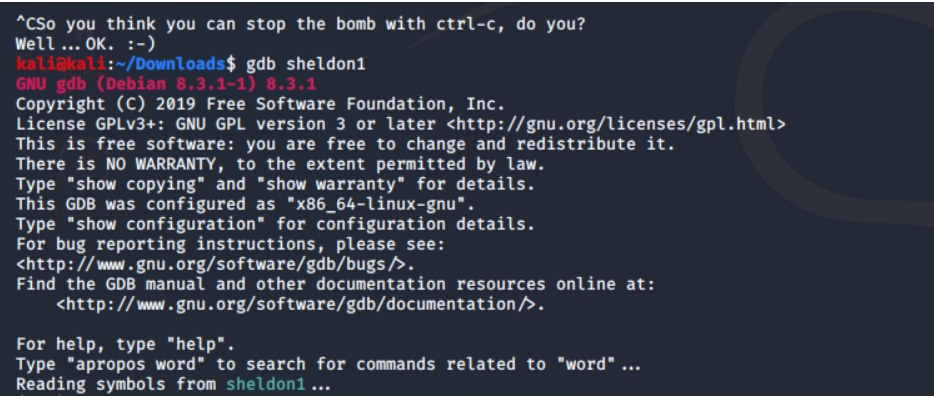
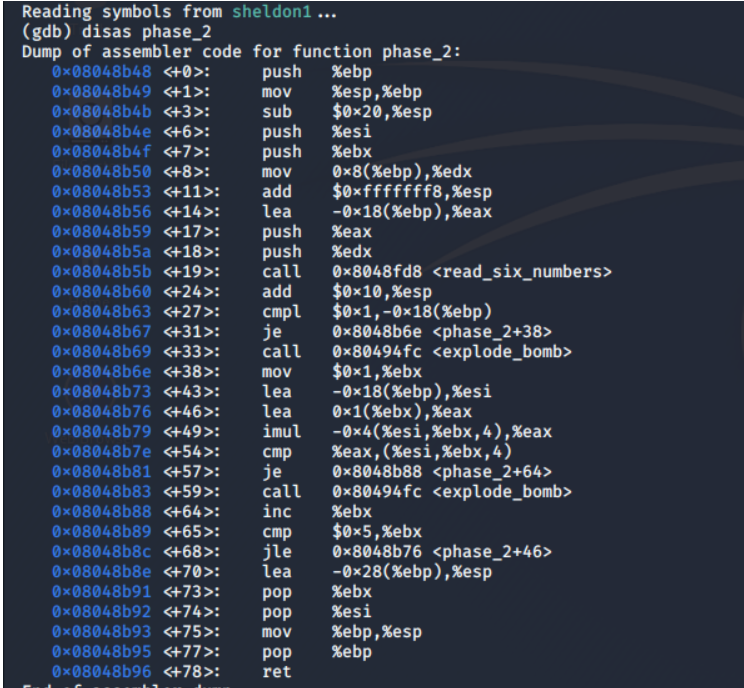
Sheldon 1\_phase\_2

Using GDB debugger we can open sheldon1

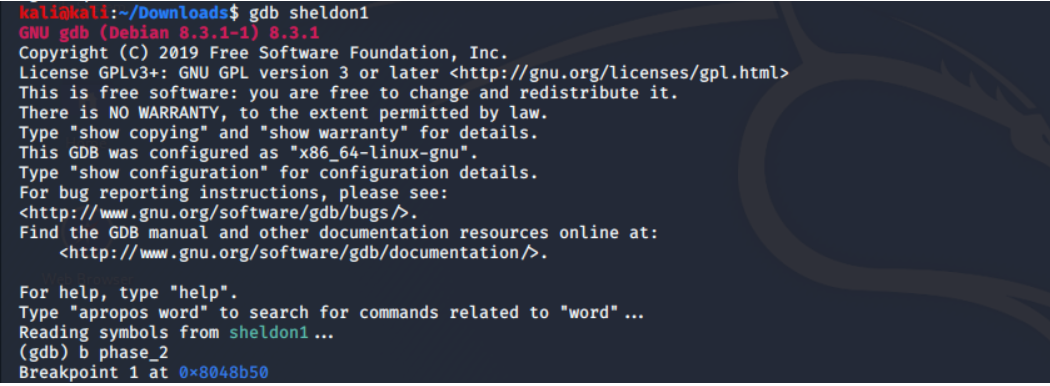


Use gdb's disassemble command to get all available assembler code for function in sheldon1. But this time we can use disas phase\_2.

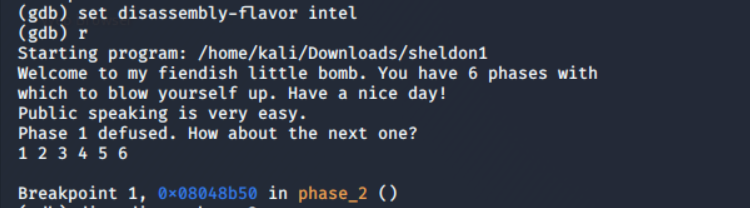


The first thing to notice here is that <phase\_2> is calling a function by the name of <read\_six\_numbers>. It looks to make sure the stdin input has six integers in it, each separated by a space. our answer takes the format a b c d e f where a-f are integers.

set a breakpoint on phase\_2.

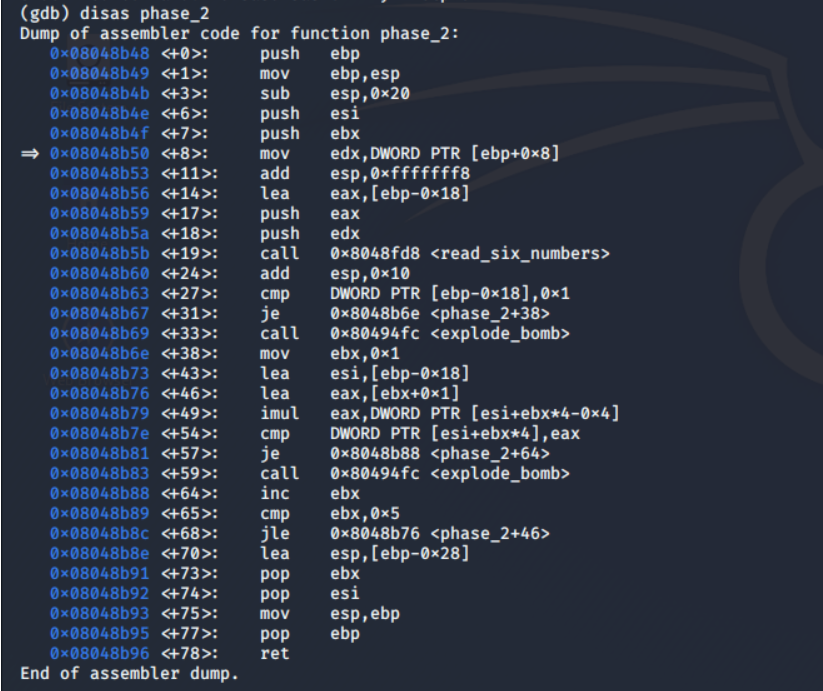


Using 'r' we can run the program in gdb, Then we can enter six random, distinct intergers before we stop through the phase.

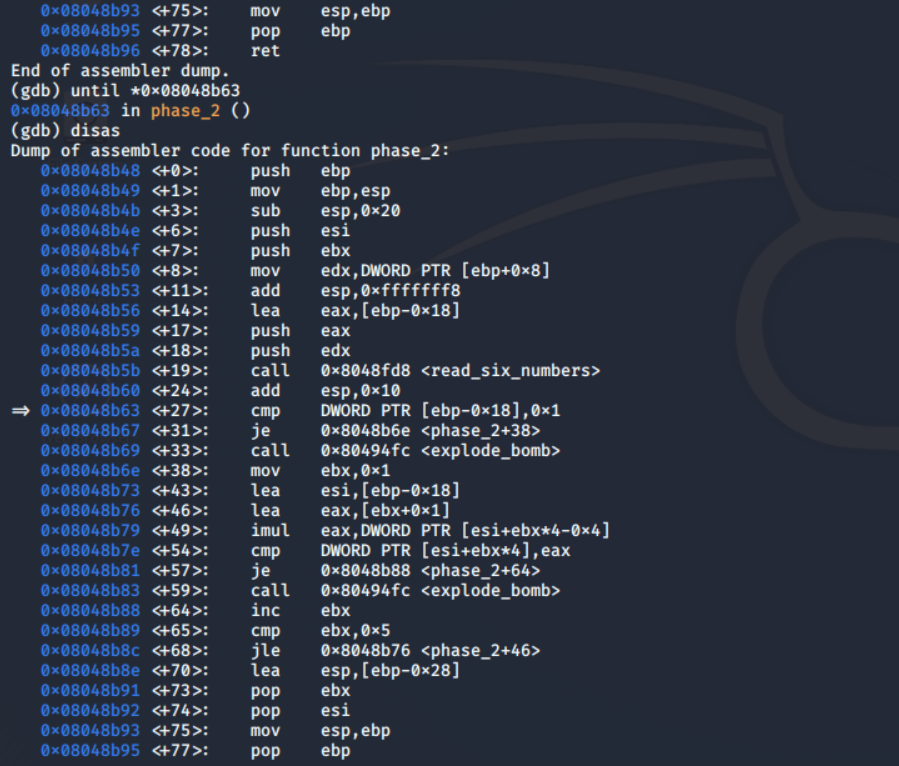


Now that we are stopped at our breakpoint.

Let’s run disas phase\_2 to take a look at the assembly code.



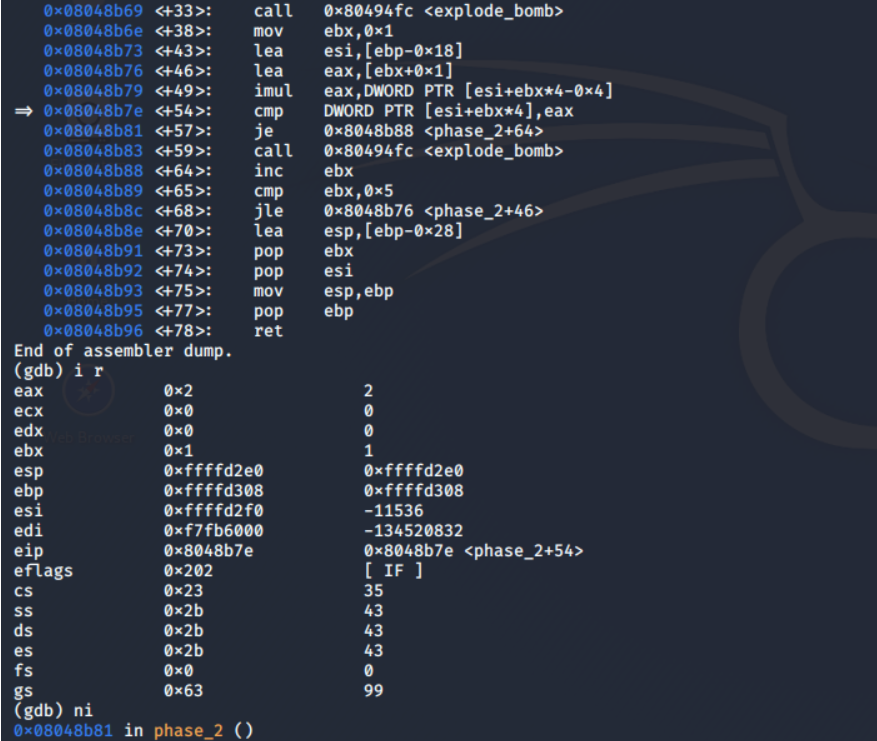
Disassemble the function using disas



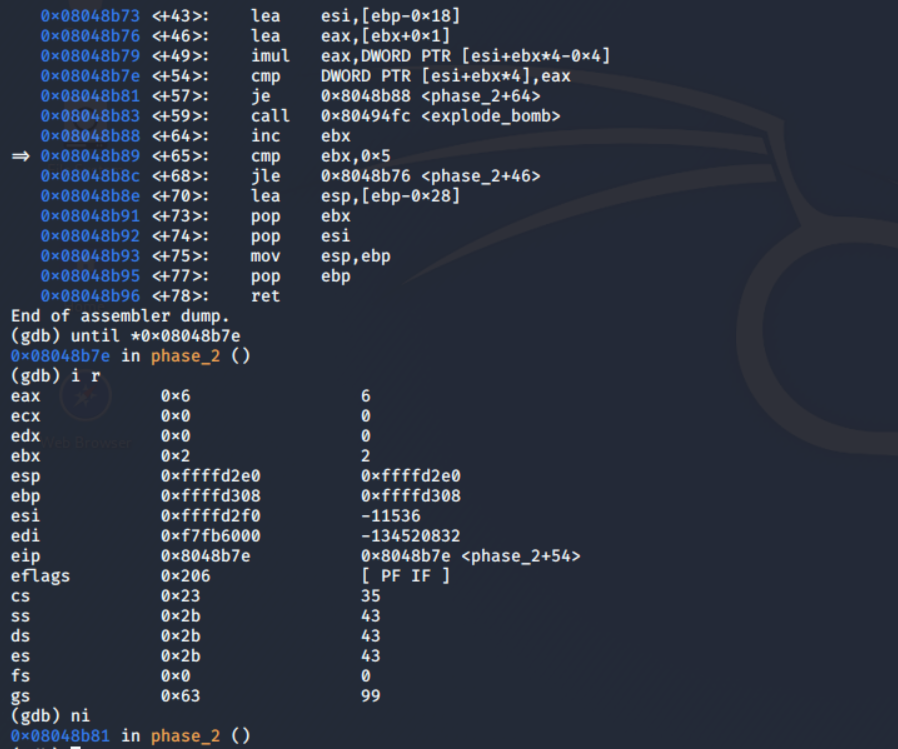
First comparison is compared with number 1 in the assembly code it represent as bellow,

cmp DWORD PTR [ebp-0x18], 0x1 =>> we can guess that the first number is " 1 ".

The second thing to note is that the function is looping over its input. After %eax is compared to (%esi, %ebx, 4) at <+54>, %ebx gets incremented and, as long as its less than 5, the function jumps back to above the comparison so that it can do it again. This means that our input integers will probably be a pattern of some sort, where <+46> is where the math that increments the pattern happens.



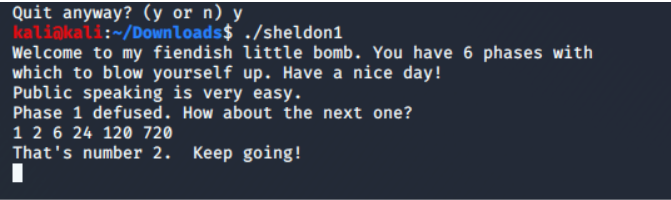
Once we’ve arrived at the comparison statement, we can use the i r command to see the contents of our registers. %eax, which is the register to which our value is being compared, is equal to 2. Therefore, the second integer in our passphrase should be 2.



The simplest way to solve this level completely is by continuing to step through the code, seeing what %eax is equal to after each iteration. Next you’ll find 6, then 24, then 120 followed by 720.

v[0] = 1  
v[i] = (i+i) \* v[i-1]

Either way, the second passphrase ends up being 1 2 6 24 120 720.



key =>> 1 2 6 24 120 720