COMPUTER ARCHITECHTURE
BOMB LAB -1 REPORT

DOLA RAJ SEKAR S20220010065

Frequently Used GDB Commands :

- objdump -d bomb
- gdb bomb
- break <location>
- run <args>
- disas
- stepi / nexti
- info registers
- info breakpoints
- print /x $register
- x/s 0xAddress
- x/d 0xAddress

Phase-1

- Use command “gdb bomb” to start gdb debugger in the terminal.
- Then add break point at disas phase_1 using “break” command.

```
raj@raj-HP-Laptop-15s-fq5xxx:~/Downloads/terminal files/bomb-lab/student-bombs-sec-2/bomb22$ gdb bomb
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from bomb...
(gdb) disas phase_1
Dump of assembler code for function phase_1:
   0x0000000000400e8d <+0>:     sub    $0x8,%rsp
   0x0000000000400e91 <+4>:     mov    $0x4023b0,%esi
   0x0000000000400e96 <+9>:     call   0x40131b <strings_not_equal>
   0x0000000000400e9b <+14>:    test   %eax,%eax
   0x0000000000400e9d <+16>:    je     0x400ea4 <phase_1+23>
   0x0000000000400e9f <+18>:    call   0x40141a <explode_bomb>
   0x0000000000400ea4 <+23>:    add    $0x8,%rsp
   0x0000000000400ea8 <+27>:    ret
End of assembler dump.
(gdb) x/s 0x4023b0
0x4023b0:       "Why make trillions when we could make... billions?"
(gdb)
```

1[i]

iCheck memory $0x402390 (x/s 0x402390),it has a string which is being moved into $esi register.And passes onto string_not_equal.

```
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from bomb...
(gdb) disas phase_1
Dump of assembler code for function phase_1:
   0x0000000000400e8d <+0>:     sub    $0x8,%rsp
   0x0000000000400e91 <+4>:     mov    $0x4023b0,%esi
   0x0000000000400e96 <+9>:     call   0x40131b <strings_not_equal>
   0x0000000000400e9b <+14>:    test   %eax,%eax
   0x0000000000400e9d <+16>:    je     0x400ea4 <phase_1+23>
   0x0000000000400e9f <+18>:    call   0x40141a <explode_bomb>
   0x0000000000400ea4 <+23>:    add    $0x8,%rsp
   0x0000000000400ea8 <+27>:    ret
End of assembler dump.
(gdb) x/s 0x4023b0
0x4023b0:       "Why make trillions when we could make... billions?"
(gdb) r
Starting program: /home/raj/Downloads/terminal files/bomb-lab/student-bombs-sec-2/bomb22/bomb
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
```

----------DONE WITH THE PHASE-1----------

- Here it is checking user input and the string in $esi registers are same,if
- not explode_bomb will be called in phase1.
-
- • So, the solution key for phase1 is the string that moved into $esi
- register.
- • In my case string is “Why make trillions when we could make... billions?”.
- • Save this solution keys for phases in a text file .

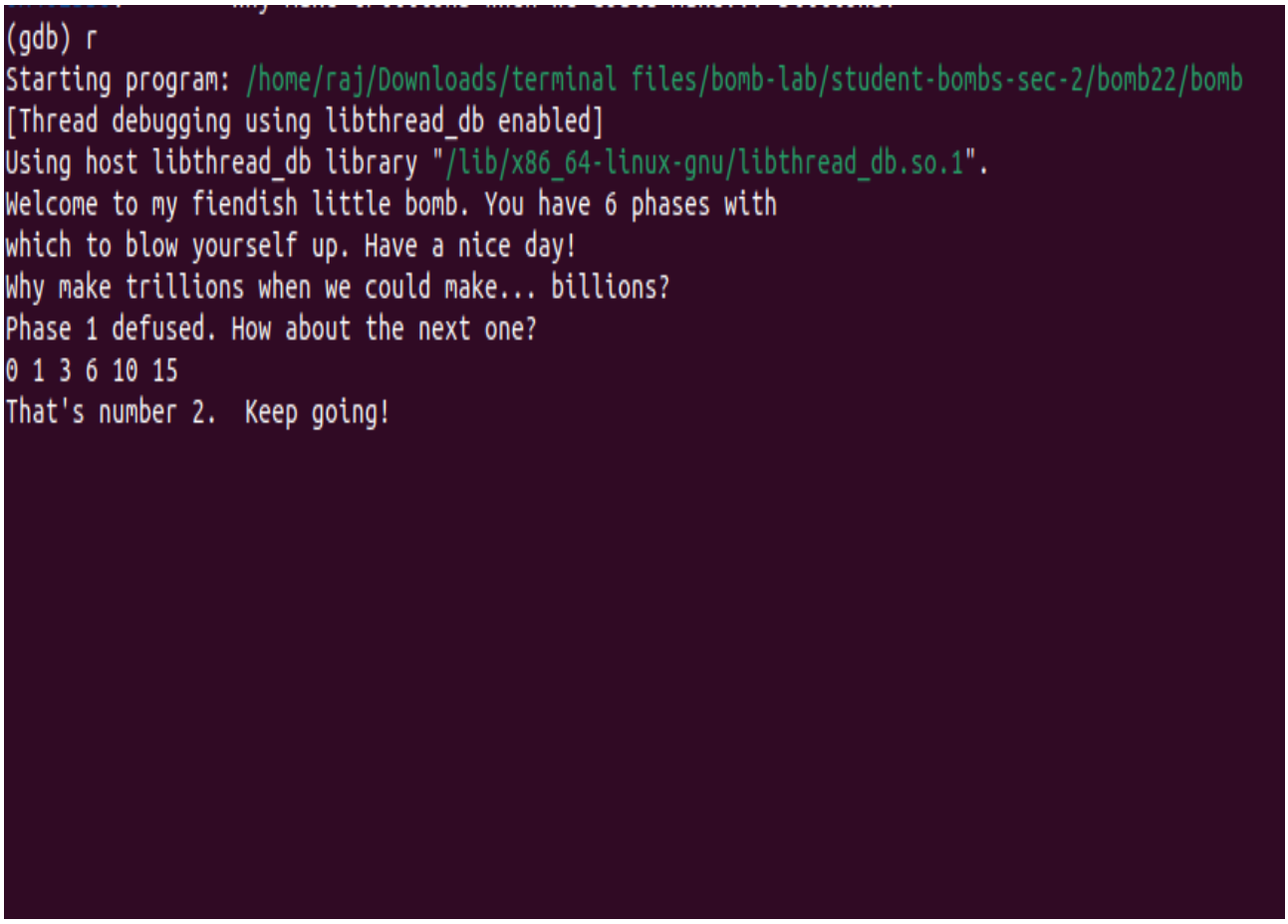
PHASE-2

- Start the gdb debugger and add break point at phase_2.
- Now use disas phase_2 command to see the assembly code of phase2.

```
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from bomb...
(gdb) disas phase_2
Dump of assembler code for function phase_2:
   0x0000000000400ea9 <+0>:     push   %rbp
   0x0000000000400eaa <+1>:     push   %rbx
   0x0000000000400eab <+2>:     sub    $0x28,%rsp
   0x0000000000400eaf <+6>:     mov    %fs:0x28,%rax
   0x0000000000400eb8 <+15>:    mov    %rax,0x18(%rsp)
   0x0000000000400ebd <+20>:    xor    %eax,%eax
   0x0000000000400ebf <+22>:    mov    %rsp,%rsi
   0x0000000000400ec2 <+25>:    call   0x40143c <read_six_numbers>
   0x0000000000400ec7 <+30>:    cmpl   $0x0,(%rsp)
   0x0000000000400ecb <+34>:    jns    0x400ed2 <phase_2+41>
   0x0000000000400ecd <+36>:    call   0x40141a <explode_bomb>
   0x0000000000400ed2 <+41>:    mov    %rsp,%rbp
   0x0000000000400ed5 <+44>:    mov    $0x1,%ebx
   0x0000000000400eda <+49>:    mov    %ebx,%eax
   0x0000000000400edc <+51>:    add    0x0(%rbp),%eax
   0x0000000000400edf <+54>:    cmp    %eax,0x4(%rbp)
   0x0000000000400ee2 <+57>:    je     0x400ee9 <phase_2+64>
   0x0000000000400ee4 <+59>:    call   0x40141a <explode_bomb>
   0x0000000000400ee9 <+64>:    add    $0x1,%ebx
   0x0000000000400eec <+67>:    add    $0x4,%rbp
   0x0000000000400ef0 <+71>:    cmp    $0x6,%ebx
   0x0000000000400ef3 <+74>:    jne    0x400eda <phase_2+49>
   0x0000000000400ef5 <+76>:    mov    0x18(%rsp),%rax
   0x0000000000400efa <+81>:    xor    %fs:0x28,%rax
   0x0000000000400f03 <+90>:    je     0x400f0a <phase_2+97>
   0x0000000000400f05 <+92>:    call   0x400b00 <__stack_chk_fail@plt>
   0x0000000000400f0a <+97>:    add    $0x28,%rsp
   0x0000000000400f0e <+101>:   pop    %rbx
   0x0000000000400f0f <+102>:   pop    %rbp
   0x0000000000400f10 <+103>:   ret
End of assembler dump.
(gdb)
```

- We can see read_six_numbers function.If we examine assembly code of this function we can conclude that solution format for phase2 is six numbers.
- In phase2 there is loop executing,it is doubling the input in $eax and comparing it with $rbx+4,which is the next input.So the input pattern is it should be double of its previous input
- • Let’s try 0 1 3 6 10 15.And it worked!.

3



----------DONE WITH THE PHASE-2 ----------

PHASE-3:

- Enter the command disas phase_3
- Examine the assembly code of Phase_3.
- • Observe memory $0x4025af which has input format for phase3
- that is two integers.
- • Its checking whether the input 1 is greater tha equal to 1 and
- less than 7 ,if not it jumps to explode_bomb.
- • So,the first input can be any number in between 1 and
- 6(including 1,6).
- • There is a switch case being executed based upon the first
- input it a moves a immediate value into $eax which is being
- compared to user input two.

4

```
(gdb) disas phase_3
Dump of assembler code for function phase_3:
   0x0000000000400f11 <+0>:     sub    $0x18,%rsp
   0x0000000000400f15 <+4>:     mov    %fs:0x28,%rax
   0x0000000000400f1e <+13>:    mov    %rax,0x8(%rsp)
   0x0000000000400f23 <+18>:    xor    %eax,%eax
   0x0000000000400f25 <+20>:    lea    0x4(%rsp),%rcx
   0x0000000000400f2a <+25>:    mov    %rsp,%rdx
   0x0000000000400f2d <+28>:    mov    $0x4025af,%esi
   0x0000000000400f32 <+33>:    call   0x400bb0 <__isoc99_sscanf@plt>
   0x0000000000400f37 <+38>:    cmp    $0x1,%eax
   0x0000000000400f3a <+41>:    jg     0x400f41 <phase_3+48>
   0x0000000000400f3c <+43>:    call   0x40141a <explode_bomb>
   0x0000000000400f41 <+48>:    cmpl   $0x7,(%rsp)
   0x0000000000400f45 <+52>:    ja     0x400f82 <phase_3+113>
   0x0000000000400f47 <+54>:    mov    (%rsp),%eax
   0x0000000000400f4a <+57>:    jmp    *0x402420(,%rax,8)
   0x0000000000400f51 <+64>:    mov    $0x308,%eax
   0x0000000000400f56 <+69>:    jmp    0x400f93 <phase_3+130>
   0x0000000000400f58 <+71>:    mov    $0x303,%eax
   0x0000000000400f5d <+76>:    jmp    0x400f93 <phase_3+130>
   0x0000000000400f5f <+78>:    mov    $0x198,%eax
   0x0000000000400f64 <+83>:    jmp    0x400f93 <phase_3+130>
   0x0000000000400f66 <+85>:    mov    $0x3cc,%eax
   0x0000000000400f6b <+90>:    jmp    0x400f93 <phase_3+130>
   0x0000000000400f6d <+92>:    mov    $0x151,%eax
   0x0000000000400f72 <+97>:    jmp    0x400f93 <phase_3+130>
   0x0000000000400f74 <+99>:    mov    $0x3b2,%eax
   0x0000000000400f79 <+104>:   jmp    0x400f93 <phase_3+130>
   0x0000000000400f7b <+106>:   mov    $0x130,%eax
   0x0000000000400f80 <+111>:   jmp    0x400f93 <phase_3+130>
   0x0000000000400f82 <+113>:   call   0x40141a <explode_bomb>
   0x0000000000400f87 <+118>:   mov    $0x0,%eax
   0x0000000000400f8c <+123>:   jmp    0x400f93 <phase_3+130>
   0x0000000000400f8e <+125>:   mov    $0x135,%eax
   0x0000000000400f93 <+130>:   cmp    0x4(%rsp),%eax
   0x0000000000400f97 <+134>:   je     0x400f9e <phase_3+141>
   0x0000000000400f99 <+136>:   call   0x40141a <explode_bomb>
   0x0000000000400f9e <+141>:   mov    0x8(%rsp),%rax
   0x0000000000400fa3 <+146>:   xor    %fs:0x28,%rax
   0x0000000000400fac <+155>:   je     0x400fb3 <phase_3+162>
   0x0000000000400fae <+157>:   call   0x400b00 <__stack_chk_fail@plt>
   0x0000000000400fb3 <+162>:   add    $0x18,%rsp
   0x0000000000400fb7 <+166>:   ret
End of assembler dump.
```

- So, based on input 1 ,you can conclude the input 2 value.
- In my case input1=1,then input2 should be 0x308==776.
- Otherwise input1=2,then input2 should be 0x303==771

```
(gdb) r
Starting program: /home/raj/Downloads/terminal files/bomb-lab/student-bombs-sec-2/bomb22/bomb
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Why make trillions when we could make... billions?
Phase 1 defused. How about the next one?
0 1 3 6 10 15
That's number 2.  Keep going!
2 771
Halfway there!
```

----------DONE WITH THE PHASE-3 ----------

PHASE-4:

- Enter the command disas phase_4.
- Examine phase_4 assembly code.It can be found that format input is two
- integers from the $0x4025af.
- We can observe that $rsp holds our first input. $rsp+4 holds our
- second input.And the code is checking for input1 to be between 2 ,14

from the code at phase_4+84 my second input is being compared with

- $0xf,if not equal it jumps to explode bomb ,so my input 2 is 15.

```
(gdb) disas phase_4
Dump of assembler code for function phase_4:
   0x0000000000400feb <+0>:     sub    $0x18,%rsp
   0x0000000000400fef <+4>:     mov    %fs:0x28,%rax
   0x0000000000400ff8 <+13>:    mov    %rax,0x8(%rsp)
   0x0000000000400ffd <+18>:    xor    %eax,%eax
   0x0000000000400fff <+20>:    lea    0x4(%rsp),%rcx
   0x0000000000401004 <+25>:    mov    %rsp,%rdx
   0x0000000000401007 <+28>:    mov    $0x4025af,%esi
   0x000000000040100c <+33>:    call   0x400bb0 <__isoc99_sscanf@plt>
   0x0000000000401011 <+38>:    cmp    $0x2,%eax
   0x0000000000401014 <+41>:    jne    0x40101c <phase_4+49>
   0x0000000000401016 <+43>:    cmpl   $0xe,(%rsp)
   0x000000000040101a <+47>:    jbe    0x401021 <phase_4+54>
   0x000000000040101c <+49>:    call   0x40141a <explode_bomb>
   0x0000000000401021 <+54>:    mov    $0xe,%edx
   0x0000000000401026 <+59>:    mov    $0x0,%esi
   0x000000000040102b <+64>:    mov    (%rsp),%edi
   0x000000000040102e <+67>:    call   0x400fb8 <func4>
   0x0000000000401033 <+72>:    cmp    $0xf,%eax
   0x0000000000401036 <+75>:    jne    0x40103f <phase_4+84>
   0x0000000000401038 <+77>:    cmpl   $0xf,0x4(%rsp)
   0x000000000040103d <+82>:    je     0x401044 <phase_4+89>
   0x000000000040103f <+84>:    call   0x40141a <explode_bomb>
   0x0000000000401044 <+89>:    mov    0x8(%rsp),%rax
   0x0000000000401049 <+94>:    xor    %fs:0x28,%rax
   0x0000000000401052 <+103>:   je     0x401059 <phase_4+110>
   0x0000000000401054 <+105>:   call   0x400b00 <__stack_chk_fail@plt>
   0x0000000000401059 <+110>:   add    $0x18,%rsp
   0x000000000040105d <+114>:   ret
End of assembler dump.
```

- for input 1 if we examine func4 we can find exact value of input1,in my case it is 5.

```
End of assembler dump.
(gdb) disas func4
Dump of assembler code for function func4:
   0x0000000000400fb8 <+0>:     push   %rbx
   0x0000000000400fb9 <+1>:     mov    %edx,%eax
   0x0000000000400fbb <+3>:     sub    %esi,%eax
   0x0000000000400fbd <+5>:     mov    %eax,%ebx
   0x0000000000400fbf <+7>:     shr    $0x1f,%ebx
   0x0000000000400fc2 <+10>:    add    %ebx,%eax
   0x0000000000400fc4 <+12>:    sar    %eax
   0x0000000000400fc6 <+14>:    lea    (%rax,%rsi,1),%ebx
   0x0000000000400fc9 <+17>:    cmp    %edi,%ebx
   0x0000000000400fcb <+19>:    jle    0x400fd9 <func4+33>
   0x0000000000400fcd <+21>:    lea    -0x1(%rbx),%edx
   0x0000000000400fd0 <+24>:    call   0x400fb8 <func4>
   0x0000000000400fd5 <+29>:    add    %ebx,%eax
   0x0000000000400fd7 <+31>:    jmp    0x400fe9 <func4+49>
   0x0000000000400fd9 <+33>:    mov    %ebx,%eax
   0x0000000000400fdb <+35>:    cmp    %edi,%ebx
   0x0000000000400fdd <+37>:    jge    0x400fe9 <func4+49>
   0x0000000000400fdf <+39>:    lea    0x1(%rbx),%esi
   0x0000000000400fe2 <+42>:    call   0x400fb8 <func4>
   0x0000000000400fe7 <+47>:    add    %ebx,%eax
   0x0000000000400fe9 <+49>:    pop    %rbx
   0x0000000000400fea <+50>:    ret
```

- In my case input 1 is 5 and input 2 is 15

```
[Inferior 1 (process 4870) exited with code 010]
(gdb) r
Starting program: /home/raj/Downloads/terminal files/bomb-lab/student-bombs-sec-2/bomb22/bomb
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Why make trillions when we could make... billions?
Phase 1 defused. How about the next one?
0 1 3 6 10 15
That's number 2.  Keep going!
2 771
Halfway there!
5 15
So you got that one.  Try this one.
```

----------DONE WITH THE PHASE-4 ----------

PHASE-5:

- Enter the command disas phase_5.
- Examine phase_5 assembly code.it can seen that input should be a string
- of length 6.
- • We can also find a array containing integers
- 0x402460 <array.3599>: 10 2 14 7
- 0x402470 <array.3599+16>: 8 12 15 11
- 0x402480 <array.3599+32>: 0 4 1 13
- 0x402490 <array.3599+48>: 3 9 6.
- There is a loop being executed ,for every iteration its taking last 4 bits of
-
- for array we seen above and sums up the element present at that index.

```
   0x0000000000401094 <+54>:    mov    %eax,(%rsp)
   0x0000000000401097 <+57>:    cmp    $0xf,%eax
   0x000000000040109a <+60>:    je     0x4010cb <phase_5+109>
   0x000000000040109c <+62>:    mov    $0x0,%ecx
   0x00000000004010a1 <+67>:    mov    $0x0,%edx
   0x00000000004010a6 <+72>:    add    $0x1,%edx
   0x00000000004010a9 <+75>:    cltq
   0x00000000004010ab <+77>:    mov    0x402460(,%rax,4),%eax
   0x00000000004010b2 <+84>:    add    %eax,%ecx
   0x00000000004010b4 <+86>:    cmp    $0xf,%eax
   0x00000000004010b7 <+89>:    jne    0x4010a6 <phase_5+72>
   0x00000000004010b9 <+91>:    movl   $0xf,(%rsp)
   0x00000000004010c0 <+98>:    cmp    $0xf,%edx
   0x00000000004010c3 <+101>:   jne    0x4010cb <phase_5+109>
   0x00000000004010c5 <+103>:   cmp    0x4(%rsp),%ecx
   0x00000000004010c9 <+107>:   je     0x4010d0 <phase_5+114>
   0x00000000004010cb <+109>:   call   0x40141a <explode_bomb>
   0x00000000004010d0 <+114>:   mov    0x8(%rsp),%rax
   0x00000000004010d5 <+119>:   xor    %fs:0x28,%rax
   0x00000000004010de <+128>:   je     0x4010e5 <phase_5+135>
   0x00000000004010e0 <+130>:   call   0x400b00 <__stack_chk_fail@plt>
   0x00000000004010e5 <+135>:   add    $0x18,%rsp
   0x00000000004010e9 <+139>:   ret
End of assembler dump.
(gdb) x/15wd 0x402460
0x402460 <array.3599>:  10      2       14      7
0x402470 <array.3599+16>:       8       12      15      11
0x402480 <array.3599+32>:       0       4       1       13
0x402490 <array.3599+48>:       3       9       6
(gdb)
```

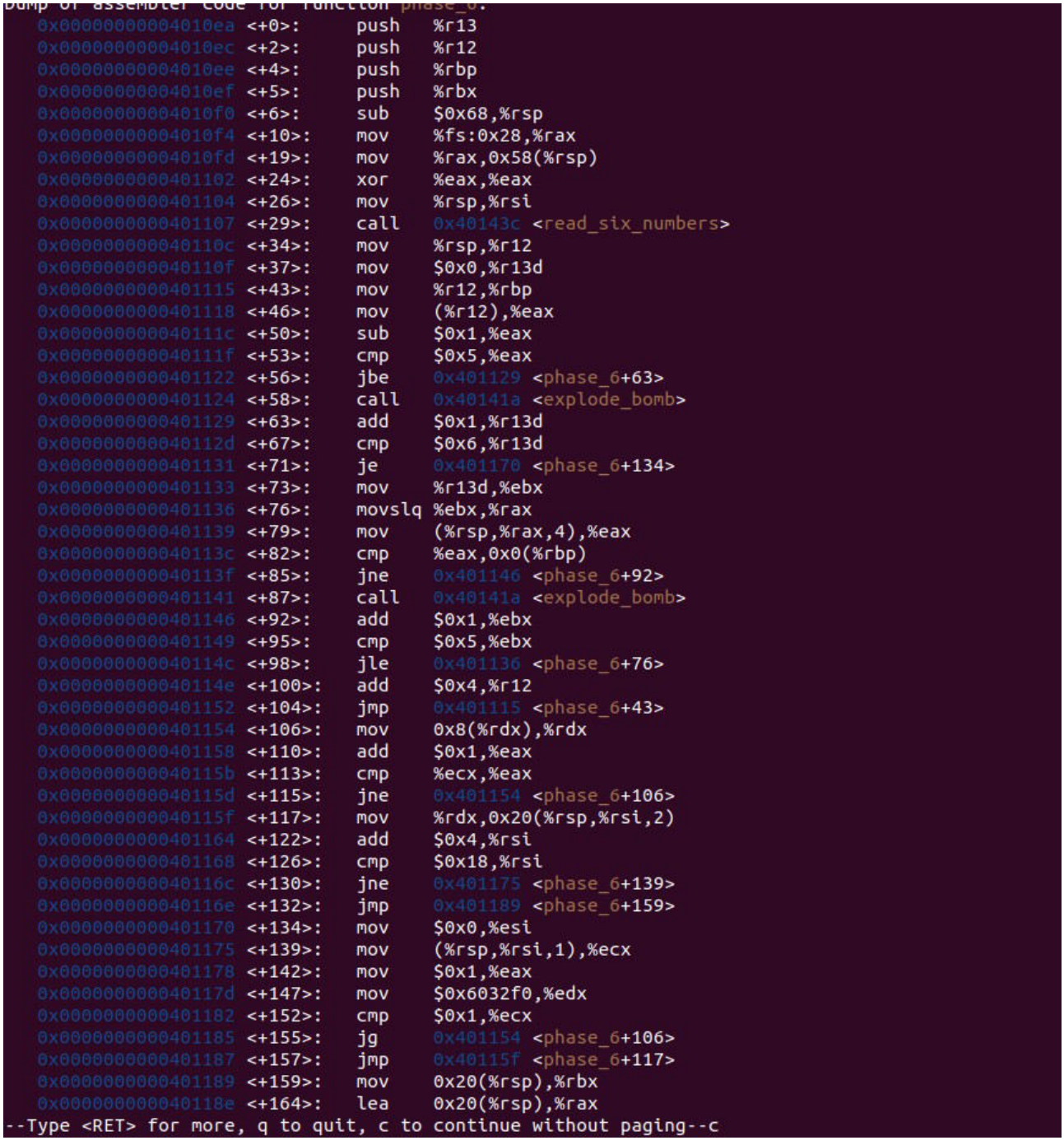
➢ In my case input 1 is 5 and input 2 is sum of all array elements which is equal to 115.

```
(gdb) r
Starting program: /home/raj/Downloads/terminal files/bomb-lab/student-bombs-sec-2/bomb22/bomb
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Why make trillions when we could make... billions?
Phase 1 defused. How about the next one?
0 1 3 6 10 15
That's number 2.  Keep going!
2 771
Halfway there!
5 15
So you got that one.  Try this one.
5 115
Good work!  On to the next...
```

----------DONE WITH THE PHASE-5----------

PHASE-6:

- Enter the command disas phase_6.
- Examine phase_6 assembly code.We can see read_six_numbers
- function .If we go through ,we can conclude that input should
- be 6 integers all different and 1 to 6.
- • If we debug and go through the code and memory we can find
- a list of nodes which has indices and node values.



- Here sort the node indices based on the node values in the decreasing
- order.
- • The order of six indices we got is compared to the input but not directly
- ,before comparing our input is being converted to 7-x.
- • So ,our converted input should be equal to the node indices sorted
- • Therefore, our correct input would be the 7-y,y=sorted node indices.
- • In my case solution is 2,4,6,1,3,5.

```
--Type <RET> for more, q to quit, c to continue without paging--c
   0x0000000000401193 <+169>:   lea    0x48(%rsp),%rsi
   0x0000000000401198 <+174>:   mov    %rbx,%rcx
   0x000000000040119b <+177>:   mov    0x8(%rax),%rdx
   0x000000000040119f <+181>:   mov    %rdx,0x8(%rcx)
   0x00000000004011a3 <+185>:   add    $0x8,%rax
   0x00000000004011a7 <+189>:   mov    %rdx,%rcx
   0x00000000004011aa <+192>:   cmp    %rsi,%rax
   0x00000000004011ad <+195>:   jne    0x40119b <phase_6+177>
   0x00000000004011af <+197>:   movq   $0x0,0x8(%rdx)
   0x00000000004011b7 <+205>:   mov    $0x5,%ebp
   0x00000000004011bc <+210>:   mov    0x8(%rbx),%rax
   0x00000000004011c0 <+214>:   mov    (%rax),%eax
   0x00000000004011c2 <+216>:   cmp    %eax,(%rbx)
   0x00000000004011c4 <+218>:   jle    0x4011cb <phase_6+225>
   0x00000000004011c6 <+220>:   call   0x40141a <explode_bomb>
   0x00000000004011cb <+225>:   mov    0x8(%rbx),%rbx
   0x00000000004011cf <+229>:   sub    $0x1,%ebp
   0x00000000004011d2 <+232>:   jne    0x4011bc <phase_6+210>
   0x00000000004011d4 <+234>:   mov    0x58(%rsp),%rax
   0x00000000004011d9 <+239>:   xor    %fs:0x28,%rax
   0x00000000004011e2 <+248>:   je     0x4011e9 <phase_6+255>
   0x00000000004011e4 <+250>:   call   0x400b00 <__stack_chk_fail@plt>
   0x00000000004011e9 <+255>:   add    $0x68,%rsp
   0x00000000004011ed <+259>:   pop    %rbx
   0x00000000004011ee <+260>:   pop    %rbp
   0x00000000004011ef <+261>:   pop    %r12
   0x00000000004011f1 <+263>:   pop    %r13
   0x00000000004011f3 <+265>:   ret
End of assembler dump.
(gdb)
```

----------DONE WITH THE PHASE-6----------

FINAL KEYS:

i.Why make trillions when we could make... billions?

ii.0 1 3 6 10 15

iii.1 776

iv.5 15

v.5 115

vi.2 4 6 1 3 5

FINAL OUTPUT:

```
raj@raj-HP-Laptop-15s-fq5xxx:~/Downloads/terminal files/bomb-lab/student-bombs-sec-2/bomb22$ gdb bomb
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from bomb...
(gdb) r CA_sol
Starting program: /home/raj/Downloads/terminal files/bomb-lab/student-bombs-sec-2/bomb22/bomb CA_sol
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Phase 1 defused. How about the next one?
That's number 2.  Keep going!
Halfway there!
So you got that one.  Try this one.
Good work!  On to the next...
Congratulations! You've defused the bomb!
[Inferior 1 (process 5392) exited normally]
(gdb)
```