COMPUTER ARCHITECTURE

LAB REPORT

ON

DIFFUSING BOMB

S20220010144 – M. SHREERAJ

CONTENTS :

❖ PHASE -1 :

- Firstly,we'll start with the command "gdb bomb".
- When we disassemble function Phase_1,we get the following result.

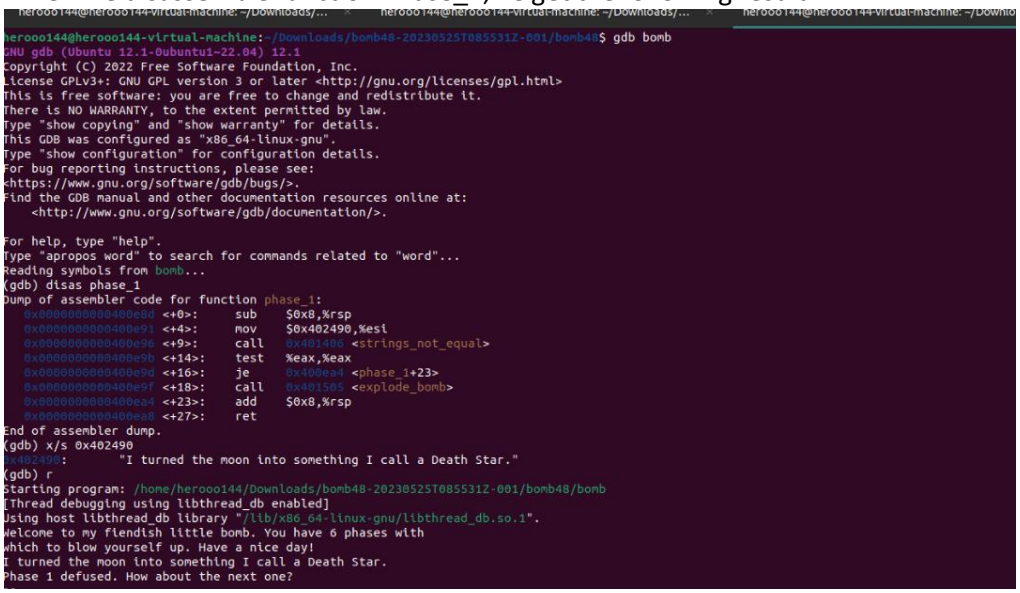

Figure 1: Phase1

- We can observe something is being moved into **%esi.** We need to examine what is inside 0x402490,so we use x/s to examine the string inside the address.
- We got the solution of Phase_1 i.e. **"I turned the moon into something I call a Death Star."**

❖ PHASE -2 :

- Start the gdb debugger
- Add break point to it using the command break phase_2
- Now run the program and enter the solution of previous phase.
- Now enter dummy input for phase_2

```
herooo144@herooo144-virtual-machine:~/Downloads/bomb48-20230525T085531Z-001/bomb48$ 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) break phase_2
Breakpoint 1 at 0x400ea9
(gdb) r
Starting program: /home/herooo144/Downloads/bomb48-20230525T085531Z-001/bomb48/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!
I turned the moon into something I call a Death Star.
Phase 1 defused. How about the next one?
jj

Breakpoint 1, 0x0000000000400ea9 in phase_2 ()
(gdb) disass
```

Figure 2: Phase 2

- The program will break at phase_2.
- Now use the command “disas phase_2” to get the dump of assembler code for for function phase_2.

```
(gdb) disass
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   0x401527 <read_six_numbers>
   0x0000000000400ec7 <+30>:    cmpl   $0x1,(%rsp)
   0x0000000000400ecb <+34>:    je     0x400ed2 <phase_2+41>
   0x0000000000400ecd <+36>:    call   0x401505 <explode_bomb>
   0x0000000000400ed2 <+41>:    mov    %rsp,%rbx
   0x0000000000400ed5 <+44>:    lea    0x14(%rsp),%rbp
   0x0000000000400eda <+49>:    mov    (%rbx),%eax
   0x0000000000400edc <+51>:    add    %eax,%eax
   0x0000000000400ede <+53>:    cmp    %eax,0x4(%rbx)
   0x0000000000400ee1 <+56>:    je     0x400ee8 <phase_2+63>
   0x0000000000400ee3 <+58>:    call   0x401505 <explode_bomb>
   0x0000000000400ee8 <+63>:    add    $0x4,%rbx
   0x0000000000400eec <+67>:    cmp    %rbp,%rbx
   0x0000000000400eef <+70>:    jne    0x400eda <phase_2+49>
   0x0000000000400ef1 <+72>:    mov    0x18(%rsp),%rax
   0x0000000000400ef6 <+77>:    xor    %fs:0x28,%rax
   0x0000000000400eff <+86>:    je     0x400f06 <phase_2+93>
   0x0000000000400f01 <+88>:    call   0x400b00 <__stack_chk_fail@plt>
   0x0000000000400f06 <+93>:    add    $0x28,%rsp
   0x0000000000400f0a <+97>:    pop    %rbx
   0x0000000000400f0b <+98>:    pop    %rbp
   0x0000000000400f0c <+99>:    ret
End of assembler dump.
```

Figure 3: Phase 2.1

- When we Disassemble Function Phase_2, we get this.We can see that it is calling another function read_six_numbers.
- Nothing special, it just reads in 6 numbers.
- We have to give break points at appropriate places such as cmp (command) 0x 0400ede.Then we check the contents of %eax and %rbx as it is being compared and the result decides whether or not the bomb get exploded.
- So,we check values of registers at this point and verify the value of %eax and (%rbx + 0x4) is same or not.(%eax holds the return value)
- Then we check for the next required number(next iteration).Similarly,we get check the all iterations of the string and get the required number string.

```
herooo144@herooo144-virtual-machine: ~/Downloads/...   herooo144@herooo144-virtual-machine: ~/Downloa
   0x0000000000400f0a <+97>:    pop    %rbx
   0x0000000000400f0b <+98>:    pop    %rbp
   0x0000000000400f0c <+99>:    ret
End of assembler dump.
(gdb) until *0x0000000000400ede
0x0000000000400ede in phase_2 ()
(gdb) print $eax
$7 = 2
(gdb) until *0x0000000000400ede
0x0000000000400ede in phase_2 ()
(gdb) print $eax
$8 = 4
(gdb) until *0x0000000000400ede
0x0000000000400ede in phase_2 ()
(gdb) print $eax
$9 = 8
(gdb) until *0x0000000000400ede
0x0000000000400ede in phase_2 ()
(gdb) print $eax
$10 = 16
```

Figure 4: Phase 2.2

- The required string input is **"1 2 4 8 16 32".**

```
Undefined command: "R".  Try "help".
(gdb) r
Starting program: /home/herooo144/Downloads/bomb48-20230525T085531Z-001/bomb48/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!
I turned the moon into something I call a Death Star.
Phase 1 defused. How about the next one?
1 2 4 8 16 32
That's number 2.  Keep going!
```

Figure 5: Phase 2.3

❖ PHASE -3 :

➢ Here's the assembly code of phase 3 :

```
(gdb) disass
Dump of assembler code for function phase_3:
=> 0x0000000000400f0d <+0>:     sub    $0x28,%rsp
   0x0000000000400f11 <+4>:     mov    %fs:0x28,%rax
   0x0000000000400f1a <+13>:    mov    %rax,0x18(%rsp)
   0x0000000000400f1f <+18>:    xor    %eax,%eax
   0x0000000000400f21 <+20>:    lea    0x14(%rsp),%r8
   0x0000000000400f26 <+25>:    lea    0xf(%rsp),%rcx
   0x0000000000400f2b <+30>:    lea    0x10(%rsp),%rdx
   0x0000000000400f30 <+35>:    mov    $0x4024ee,%esi
   0x0000000000400f35 <+40>:    call   0x400bb0 <__isoc99_sscanf@plt>
   0x0000000000400f3a <+45>:    cmp    $0x2,%eax
   0x0000000000400f3d <+48>:    jg     0x400f44 <phase_3+55>
   0x0000000000400f3f <+50>:    call   0x401505 <explode_bomb>
   0x0000000000400f44 <+55>:    cmpl   $0x7,0x10(%rsp)
   0x0000000000400f49 <+60>:    ja     0x401044 <phase_3+311>
   0x0000000000400f4f <+66>:    mov    0x10(%rsp),%eax
   0x0000000000400f53 <+70>:    jmp    *0x402500(,%rax,8)
   0x0000000000400f5a <+77>:    mov    $0x69,%eax
   0x0000000000400f5f <+82>:    cmpl   $0x2e1,0x14(%rsp)
   0x0000000000400f67 <+90>:    je     0x40104e <phase_3+321>
   0x0000000000400f6d <+96>:    call   0x401505 <explode_bomb>
   0x0000000000400f72 <+101>:   mov    $0x69,%eax
   0x0000000000400f77 <+106>:   jmp    0x40104e <phase_3+321>
   0x0000000000400f7c <+111>:   mov    $0x73,%eax
   0x0000000000400f81 <+116>:   cmpl   $0x197,0x14(%rsp)
   0x0000000000400f89 <+124>:   je     0x40104e <phase_3+321>
   0x0000000000400f8f <+130>:   call   0x401505 <explode_bomb>
   0x0000000000400f94 <+135>:   mov    $0x73,%eax
   0x0000000000400f99 <+140>:   jmp    0x40104e <phase_3+321>
   0x0000000000400f9e <+145>:   mov    $0x62,%eax
   0x0000000000400fa3 <+150>:   cmpl   $0x122,0x14(%rsp)
```

```
   0x0000000000400fa3 <+150>:   cmpl   $0x122,0x14(%rsp)
   0x0000000000400fab <+158>:   je     0x40104e <phase_3+321>
   0x0000000000400fb1 <+164>:   call   0x401505 <explode_bomb>
   0x0000000000400fb6 <+169>:   mov    $0x62,%eax
   0x0000000000400fbb <+174>:   jmp    0x40104e <phase_3+321>
   0x0000000000400fc0 <+179>:   mov    $0x68,%eax
   0x0000000000400fc5 <+184>:   cmpl   $0x3ac,0x14(%rsp)
   0x0000000000400fcd <+192>:   je     0x40104e <phase_3+321>
   0x0000000000400fcf <+194>:   call   0x401505 <explode_bomb>
   0x0000000000400fd4 <+199>:   mov    $0x68,%eax
   0x0000000000400fd9 <+204>:   jmp    0x40104e <phase_3+321>
   0x0000000000400fdb <+206>:   mov    $0x70,%eax
   0x0000000000400fe0 <+211>:   cmpl   $0xc6,0x14(%rsp)
   0x0000000000400fe8 <+219>:   je     0x40104e <phase_3+321>
   0x0000000000400fea <+221>:   call   0x401505 <explode_bomb>
   0x0000000000400fef <+226>:   mov    $0x70,%eax
   0x0000000000400ff4 <+231>:   jmp    0x40104e <phase_3+321>
   0x0000000000400ff6 <+233>:   mov    $0x77,%eax
   0x0000000000400ffb <+238>:   cmpl   $0x202,0x14(%rsp)
   0x0000000000401003 <+246>:   je     0x40104e <phase_3+321>
--Type <RET> for more, q to quit, c to continue without paging--c
   0x0000000000401005 <+248>:   call   0x401505 <explode_bomb>
   0x000000000040100a <+253>:   mov    $0x77,%eax
   0x000000000040100f <+258>:   jmp    0x40104e <phase_3+321>
   0x0000000000401011 <+260>:   mov    $0x65,%eax
   0x0000000000401016 <+265>:   cmpl   $0x1db,0x14(%rsp)
   0x000000000040101e <+273>:   je     0x40104e <phase_3+321>
   0x0000000000401020 <+275>:   call   0x401505 <explode_bomb>
   0x0000000000401025 <+280>:   mov    $0x65,%eax
   0x000000000040102a <+285>:   jmp    0x40104e <phase_3+321>
   0x000000000040102c <+287>:   mov    $0x75,%eax
   0x0000000000401031 <+292>:   cmpl   $0x4c,0x14(%rsp)
   0x0000000000401036 <+297>:   je     0x40104e <phase_3+321>
   0x0000000000401038 <+299>:   call   0x401505 <explode_bomb>
   0x000000000040103d <+304>:   mov    $0x75,%eax
   0x0000000000401042 <+309>:   jmp    0x40104e <phase_3+321>
   0x0000000000401044 <+311>:   call   0x401505 <explode_bomb>
   0x0000000000401049 <+316>:   mov    $0x78,%eax
   0x000000000040104e <+321>:   cmp    0xf(%rsp),%al
   0x0000000000401052 <+325>:   je     0x401059 <phase_3+332>
   0x0000000000401054 <+327>:   call   0x401505 <explode_bomb>
   0x0000000000401059 <+332>:   mov    0x18(%rsp),%rax
   0x000000000040105e <+337>:   xor    %fs:0x28,%rax
   0x0000000000401067 <+346>:   je     0x40106e <phase_3+353>
   0x0000000000401069 <+348>:   call   0x400b00 <__stack_chk_fail@plt>
   0x000000000040106e <+353>:   add    $0x28,%rsp
   0x0000000000401072 <+357>:   ret
End of assembler dump.
```

Figure 6 : Phase 3

- Use x/s <specific address> near scanf function to see what data type it accepts.

```
End of assembler dump.
(gdb) x/s 0x4024ee
0x4024ee:       "%d %c %d"
```

Figure 7 : Phase 3.1

- So we can say that it take a number _ a character_a number as input
- Lets dive into this part of code

```
Dump of assembler code for function phase_3:
=> 0x0000000000400f0d <+0>:     sub    $0x28,%rsp
   0x0000000000400f11 <+4>:     mov    %fs:0x28,%rax
   0x0000000000400f1a <+13>:    mov    %rax,0x18(%rsp)
   0x0000000000400f1f <+18>:    xor    %eax,%eax
   0x0000000000400f21 <+20>:    lea    0x14(%rsp),%r8
   0x0000000000400f26 <+25>:    lea    0xf(%rsp),%rcx
   0x0000000000400f2b <+30>:    lea    0x10(%rsp),%rdx
   0x0000000000400f30 <+35>:    mov    $0x4024ee,%esi
   0x0000000000400f35 <+40>:    call   0x400bb0 <__isoc99_sscanf@plt>
   0x0000000000400f3a <+45>:    cmp    $0x2,%eax
   0x0000000000400f3d <+48>:    jg     0x400f44 <phase_3+55>
   0x0000000000400f3f <+50>:    call   0x401505 <explode_bomb>
   0x0000000000400f44 <+55>:    cmpl   $0x7,0x10(%rsp)
   0x0000000000400f49 <+60>:    ja     0x401044 <phase_3+311>
   0x0000000000400f4f <+66>:    mov    0x10(%rsp),%eax
   0x0000000000400f53 <+70>:    jmp    *0x402500(,%rax,8)
   0x0000000000400f5a <+77>:    mov    $0x69,%eax
   0x0000000000400f5f <+82>:    cmpl   $0x2e1,0x14(%rsp)
```

Figure 8 : Phase 3.2

- This part of code says that its like a conditional statement with a character and number . here the cases are less then seven. In above code there is a part of only first code . in the (eax) the character is stored and in next statement cmpl $0x2e1,0x14(%$rsp) the second number is stored and it's the case 0.
- Therefore the input answer for phase_3 is **"0 i 737 "**
- There will be not only a singles answer but it also have many answers like 1 s 407 etc……

❖ PHASE -4:

- Here's the assembly code of phase 4 :

```
(gdb) disass phase_4
Dump of assembler code for function phase_4:
   0x00000000004010b1 <+0>:     sub    $0x18,%rsp
   0x00000000004010b5 <+4>:     mov    %fs:0x28,%rax
   0x00000000004010be <+13>:    mov    %rax,0x8(%rsp)
   0x00000000004010c3 <+18>:    xor    %eax,%eax
   0x00000000004010c5 <+20>:    lea    0x4(%rsp),%rcx
   0x00000000004010ca <+25>:    mov    %rsp,%rdx
   0x00000000004010cd <+28>:    mov    $0x40268f,%esi
   0x00000000004010d2 <+33>:    call   0x400bb0 <__isoc99_sscanf@plt>
   0x00000000004010d7 <+38>:    cmp    $0x2,%eax
   0x00000000004010da <+41>:    jne    0x4010e2 <phase_4+49>
   0x00000000004010dc <+43>:    cmpl   $0xe,(%rsp)
   0x00000000004010e0 <+47>:    jbe    0x4010e7 <phase_4+54>
   0x00000000004010e2 <+49>:    call   0x401505 <explode_bomb>
   0x00000000004010e7 <+54>:    mov    $0xe,%edx
   0x00000000004010ec <+59>:    mov    $0x0,%esi
   0x00000000004010f1 <+64>:    mov    (%rsp),%edi
   0x00000000004010f4 <+67>:    call   0x401073 <func4>
   0x00000000004010f9 <+72>:    cmp    $0x6,%eax
   0x00000000004010fc <+75>:    jne    0x401105 <phase_4+84>
   0x00000000004010fe <+77>:    cmpl   $0x6,0x4(%rsp)
   0x0000000000401103 <+82>:    je     0x40110a <phase_4+89>
   0x0000000000401105 <+84>:    call   0x401505 <explode_bomb>
   0x000000000040110a <+89>:    mov    0x8(%rsp),%rax
   0x000000000040110f <+94>:    xor    %fs:0x28,%rax
   0x0000000000401118 <+103>:   je     0x40111f <phase_4+110>
   0x000000000040111a <+105>:   call   0x400b00 <__stack_chk_fail@plt>
   0x000000000040111f <+110>:   add    $0x18,%rsp
   0x0000000000401123 <+114>:   ret
End of assembler dump.
```

Figure 9 : Phase 4

- By using the 'x/s 0x4025cf' command we get to know that we have to give two numbers as inputs
- In the code we can see 'cmpl $0x6,0x4(%rsp)' command from which we can say that the secound input is 6(decimal representation of 0x6), we can also say that the value of first input is less than 14.
- So by trial and error we can say find that the value of first input is 6.
- So , the fourth phase can be diffused by using the integers: "6 6"

❖ PHASE -5:

When we Disassemble Function Phase_5, we get this. We can observe that something is being moved into %esi. We need to examine what is inside 0x4025af, so we examine string inside the address. We get to know that it requires a string of 2 integers. So, we give random inputs to begin.

```
(gdb) disas phase_5
Dump of assembler code for function phase_5:
   0x0000000000401124 <+0>:     sub    $0x18,%rsp
   0x0000000000401128 <+4>:     mov    %fs:0x28,%rax
   0x0000000000401131 <+13>:    mov    %rax,0x8(%rsp)
   0x0000000000401136 <+18>:    xor    %eax,%eax
   0x0000000000401138 <+20>:    lea    0x4(%rsp),%rcx
   0x000000000040113d <+25>:    mov    %rsp,%rdx
   0x0000000000401140 <+28>:    mov    $0x40268f,%esi
   0x0000000000401145 <+33>:    call   0x400bb0 <__isoc99_sscanf@plt>
   0x000000000040114a <+38>:    cmp    $0x1,%eax
   0x000000000040114d <+41>:    jg     0x401154 <phase_5+48>
   0x000000000040114f <+43>:    call   0x401505 <explode_bomb>
   0x0000000000401154 <+48>:    mov    (%rsp),%eax
   0x0000000000401157 <+51>:    and    $0xf,%eax
   0x000000000040115a <+54>:    mov    %eax,(%rsp)
   0x000000000040115d <+57>:    cmp    $0xf,%eax
   0x0000000000401160 <+60>:    je     0x401191 <phase_5+109>
   0x0000000000401162 <+62>:    mov    $0x0,%ecx
   0x0000000000401167 <+67>:    mov    $0x0,%edx
   0x000000000040116c <+72>:    add    $0x1,%edx
   0x000000000040116f <+75>:    cltq
   0x0000000000401171 <+77>:    mov    0x402540(,%rax,4),%eax
   0x0000000000401178 <+84>:    add    %eax,%ecx
   0x000000000040117a <+86>:    cmp    $0xf,%eax
   0x000000000040117d <+89>:    jne    0x40116c <phase_5+72>
   0x000000000040117f <+91>:    movl   $0xf,(%rsp)
   0x0000000000401186 <+98>:    cmp    $0xf,%edx
   0x0000000000401189 <+101>:   jne    0x401191 <phase_5+109>
   0x000000000040118b <+103>:   cmp    0x4(%rsp),%ecx
   0x000000000040118f <+107>:   je     0x401196 <phase_5+114>
   0x0000000000401191 <+109>:   call   0x401505 <explode_bomb>
   0x0000000000401196 <+114>:   mov    0x8(%rsp),%rax
   0x000000000040119b <+119>:   xor    %fs:0x28,%rax
   0x00000000004011a4 <+128>:   je     0x4011ab <phase_5+135>
   0x00000000004011a6 <+130>:   call   0x400b00 <__stack_chk_fail@plt>
   0x00000000004011ab <+135>:   add    $0x18,%rsp
   0x00000000004011af <+139>:   ret
End of assembler dump.
```

Figure 10 : Phase 5

Everytime we are comparing %eax with 0xf in (0x401099) i.e. we are looping for 15 times. Here, we can see that we are moving something from address starting from (0x4010ad). So, we need to check what is in there

```
(gdb) x/g 0x402460
0x402460 <array.3597>:  8589934602
```

Figure 11 : Phase 5.1

We can see that there is an array in the said address. So, we need to check the first 15 elements of the array, as we are only looping for 15 times. We can use x/15w which examines the first 15 4-byte words starting from the given address.

```
(gdb) x/15w 0x402460
0x402460 <array.3597>:  10      2       14      7
0x402470 <array.3597+16>:       8       12      15      11
0x402480 <array.3597+32>:       0       4       1       13
0x402490 <array.3597+48>:       3       9       6
(gdb)
```

Figure 12 : Phase 5.2

In the assembly code, we can check that the elements are getting added up in each loop. So, we simply need to add first 15 elements of the array, which is equal to 115. The required string input is “5 115”

❖ PHASE -6 :

Code Analysis :

```
00000000004010f9 <phase_6>:
  4010f9:       41 56                   push   %r14
  .....
  401113:       31 c0                   xor    %eax,%eax
```

Initializing the stack and placing canary for stack/buffer overflow.

```
00000000004010f9 <phase_6>:
  .....
  401115:       48 89 e6                mov    %rsp,%rsi
  .....
  401118:       e8 53 03 00 00          call   401470 <read_six_numbers>
  .....
```

Take 6 Integer-Inputs and store it in `0xn(%rsp)`, where `n = 0,4,8,12,16,20`

```
00000000004010f9 <phase_6>:
  .....
  40111d:       49 89 e4                mov    %rsp,%r12
  .....
  .....
  401138:       e8 11 03 00 00          call   40144e <explode_bomb>
  .....
```

Check if all the numbers are between 1 and 6 (inclusive) or not. It also verifies whether all the numbers are unique or not. So, our solution is a `Permutation of 1, 2, 3, 4, 5, 6.`

```
00000000004010f9 <phase_6>:
  .....
  40111d:       49 89 e4                mov    %rsp,%r12
  .....
  .....
  401138:       e8 11 03 00 00          call   40144e <explode_bomb>
  .....
```

Check if all the numbers are between 1 and 6 (inclusive) or not. It also verifies whether all the numbers are unique or not. So, our solution is a `Permutation of 1, 2, 3, 4, 5, 6.`

```
00000000004010f9 <phase_6>:
  .....
  40114a:       48 63 c3                movslq %ebx,%rax
  .....
  .....
  401168:       48 8d 4c 24 18          lea    0x18(%rsp),%rcx
  .....
```

Here, the code sets `%rsp + 0x20 + 8 * i` to the address of the nth node in the linked-list.

So, if the input is `1 6 2 4 3 5` and the linked-list is like `0xF => 0xE => 0xD => 0xC => 0xB => 0xA`, the addresses of the linked-list nodes. Then,

```
(%rsp + 0x20 + 8 * 0) = 0xF;
(%rsp + 0x20 + 8 * 1) = 0xA;
(%rsp + 0x20 + 8 * 2) = 0xE;
(%rsp + 0x20 + 8 * 3) = 0xC;
(%rsp + 0x20 + 8 * 4) = 0xD;
(%rsp + 0x20 + 8 * 5) = 0xB;
```

```
00000000004010f9 <phase_6>:
  40116d:       ba 07 00 00 00          mov    $0x7,%edx
  .....
  .....
  40118a:       eb 1a                   jmp    4011a6 <phase_6+0xad>
```

Here, we are rearranging our inputs by Subtracting our inputs from 7 i.e. " `ActualNum = 7 - InputNum` " is our formula to get the order of nodes.

```
00000000004010f9 <phase_6>:
  .....
  401197:       48 89 54 74 20          mov    %rdx,0x20(%rsp,%rsi,2)
  .....
  .....
  4011b8:       eb dd                   jmp    401197 <phase_6+0x9e>
  .....
  .....
  4011e0:       48 c7 42 08 00 00 00    movq   $0x0,0x8(%rdx)
  4011e7:       00
  .....
```

Here, all the available nodes are combined to form a Linked-List. All the addresses are stored in `%rsp + 0x20`.
All these addresses holds some value and are getting joined to form a Linked-List by setting `next node pointer of current node` equal to `next node`.

```
// Basically It Is Doing This
node1->next = node2;
node2->next = node3;
```

A loop is running `5-Times` to set all the nodes at its correct location. At the end, it sets the `next node pointer` of the `Last_node` to `NULL`.

Linked List :

We can check the Linked-List by checking the initial address & next node address (which we can get by checking the next node pointer of the current node).

The Structure of each node is given by

```
4-Byte Integer | 4-Byte Padding | 8-Byte Pointer To The Next Node (Address)
```

```
.....
(gdb) break *0x4011ba
Breakpoint 1 at 0x4011ba
.....
.....
(gdb) x/w 0x6032f0
0x6032f0 <node1>:   987
(gdb) x/w 0x6032f0+8
0x6032f8 <node1+8>: 6304512
(gdb) x/w 6304512
0x603300 <node2>:   478
(gdb) x/w 6304512+8
0x603308 <node2+8>: 6304528
(gdb) x/w 6304528
0x603310 <node3>:   516
(gdb) x/w 6304528+8
0x603318 <node3+8>: 6304544
(gdb) x/w 6304544
0x603320 <node4>:   494
(gdb) x/w 6304544+8
0x603328 <node4+8>: 6304560
(gdb) x/w 6304560
0x603330 <node5>:   878
(gdb) x/w 6304560+8
0x603338 <node5+8>: 6304576
(gdb) x/w 6304576
0x603340 <node6>:   757
(gdb) x/w 6304576+8
0x603348 <node6+8>: 0   <-------NULL POINTER
```

Final Solution :

The values of the nodes are 987, 478, 516, 494, 878, 757. It is asking us to rearrange it in Ascending Order. So, we can get the ordering sequence as 1 5 6 3 4 2. And hence our actual input string can be calculated by subtracting this sequence with 7 and our result would be 6 2 1 4 3 5. [According to the Formula we had derived earlier]

Screenshort of all phases defused :

Answer is stored in a txt file named **Bomb48_S20220010144_solution.txt**

```
herooo144@herooo144-virtual-machine:~/Downloads/bomb48-20230525T085531Z-001/bomb48$ 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 Bomb48_S20220010144_solution.txt
Starting program: /home/herooo144/Downloads/bomb48-20230525T085531Z-001/bomb48/bomb Bomb48_S20220010144_solution.txt
[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 4322) exited normally]
(gdb) S
```

Figure 13 : Phase 5.2

❖ SECRET PHASE :

It can be solved using the following commands mentioned in the screenshorts it is not the appropriate way but it can also be considered as a way

```
(gdb) disas phase_defused
Dump of assembler code for function phase_defused:
   0x000000000040168c <+0>:     sub    $0x78,%rsp
   0x0000000000401690 <+4>:     mov    %fs:0x28,%rax
   0x0000000000401699 <+13>:    mov    %rax,0x68(%rsp)
   0x000000000040169e <+18>:    xor    %eax,%eax
   0x00000000004016a0 <+20>:    cmpl   $0x6,0x2030e5(%rip)        # 0x60478c <num_input_strings>
   0x00000000004016a7 <+27>:    jne    0x401707 <phase_defused+123>
   0x00000000004016a9 <+29>:    lea    0x10(%rsp),%r8
   0x00000000004016ae <+34>:    lea    0xc(%rsp),%rcx
   0x00000000004016b3 <+39>:    lea    0x8(%rsp),%rdx
   0x00000000004016b8 <+44>:    mov    $0x4026d9,%esi
   0x00000000004016bd <+49>:    mov    $0x604890,%edi
   0x00000000004016c2 <+54>:    call   0x400bb0 <__isoc99_sscanf@plt>
   0x00000000004016c7 <+59>:    cmp    $0x3,%eax
   0x00000000004016ca <+62>:    jne    0x4016fd <phase_defused+113>
   0x00000000004016cc <+64>:    mov    $0x4026e2,%esi
   0x00000000004016d1 <+69>:    lea    0x10(%rsp),%rdi
   0x00000000004016d6 <+74>:    call   0x401406 <strings_not_equal>
   0x00000000004016db <+79>:    test   %eax,%eax
   0x00000000004016dd <+81>:    jne    0x4016fd <phase_defused+113>
   0x00000000004016df <+83>:    mov    $0x4025b8,%edi
   0x00000000004016e4 <+88>:    call   0x400ae0 <puts@plt>
   0x00000000004016e9 <+93>:    mov    $0x4025e0,%edi
   0x00000000004016ee <+98>:    call   0x400ae0 <puts@plt>
   0x00000000004016f3 <+103>:   mov    $0x0,%eax
   0x00000000004016f8 <+108>:   call   0x40131c <secret_phase>
   0x00000000004016fd <+113>:   mov    $0x402618,%edi
   0x0000000000401702 <+118>:   call   0x400ae0 <puts@plt>
   0x0000000000401707 <+123>:   mov    0x68(%rsp),%rax
   0x000000000040170c <+128>:   xor    %fs:0x28,%rax
   0x0000000000401715 <+137>:   je     0x40171c <phase_defused+144>
   0x0000000000401717 <+139>:   call   0x400b00 <__stack_chk_fail@plt>
   0x000000000040171c <+144>:   add    $0x78,%rsp
   0x0000000000401720 <+148>:   ret
End of assembler dump.
```

```
(gdb) disas phase_1
Dump of assembler code for function phase_1:
   0x0000000000400e8d <+0>:     sub    $0x8,%rsp
   0x0000000000400e91 <+4>:     mov    $0x402490,%esi
   0x0000000000400e96 <+9>:     call   0x401406 <strings_not_equal>
   0x0000000000400e9b <+14>:    test   %eax,%eax
   0x0000000000400e9d <+16>:    je     0x400ea4 <phase_1+23>
   0x0000000000400e9f <+18>:    call   0x401505 <explode_bomb>
   0x0000000000400ea4 <+23>:    add    $0x8,%rsp
   0x0000000000400ea8 <+27>:    ret
End of assembler dump.
(gdb) x/s 0x4026e2
0x4026e2:       "DrEvil"
(gdb) run
Starting program: /home/herooo144/Downloads/bomb48-20230525T085531Z-001/bomb48/bomb a.txt
[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...
Curses, you've found the secret phase!
But finding it and solving it are quite different...
^C
Program received signal SIGINT, Interrupt.
```

The secret phase is **"DrEvil"**

We must append it in phase_4 answer

```
herooo144@herooo144-virtual-machine:~/Downloads/bomb48-20230525T085531Z-001/bomb48$ 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 Bomb48_S20220010144_solution.txt
Starting program: /home/herooo144/Downloads/bomb48-20230525T085531Z-001/bomb48/bomb Bomb48_S20220010144_solution.txt
[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...
Curses, you've found the secret phase!
But finding it and solving it are quite different...
```