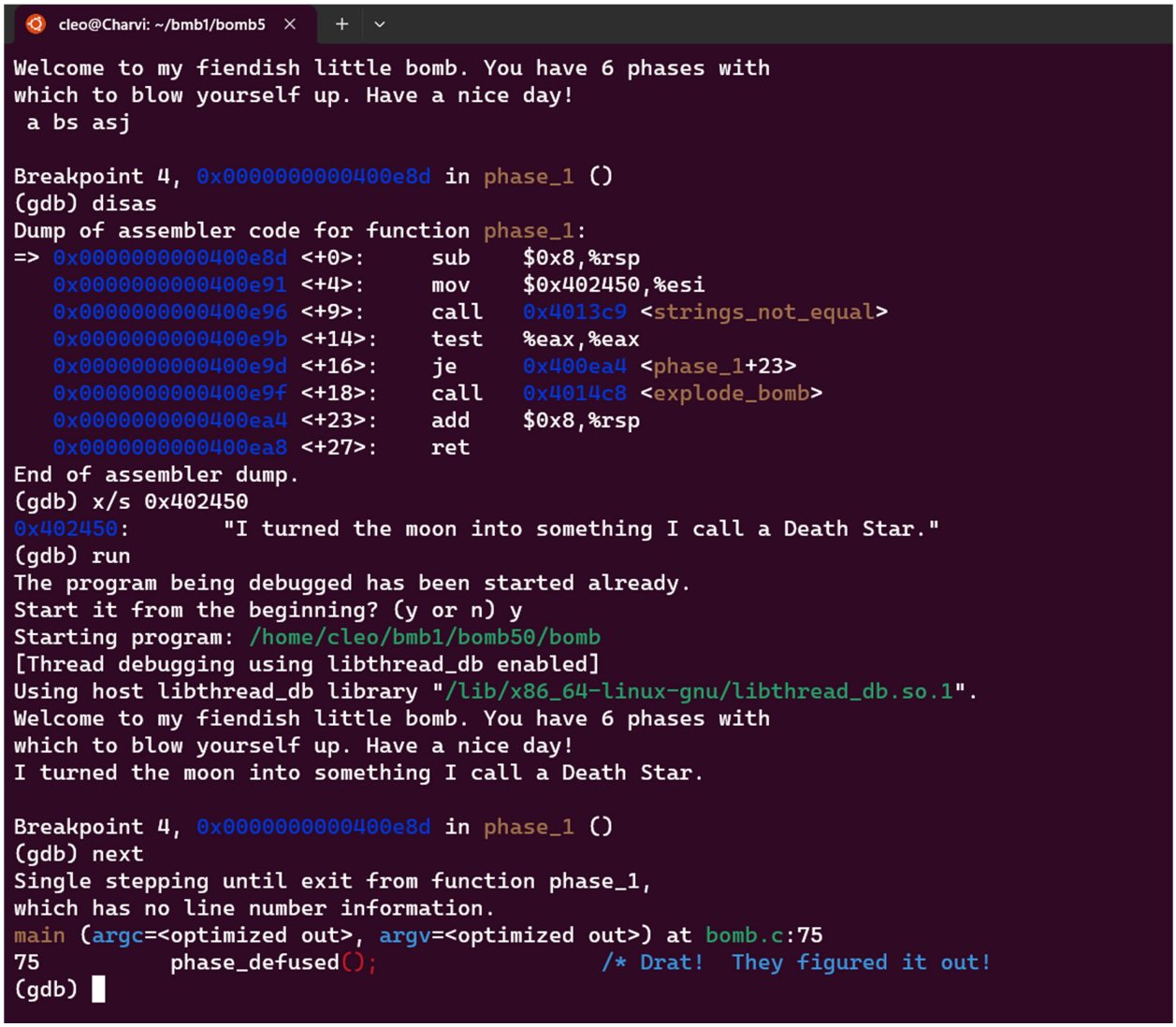
BOMB LAB

BOMB ID: bomb50

S20220010160

Phase 1

First we use GDB bomb command to run the to start gdb debugger in the terminal.

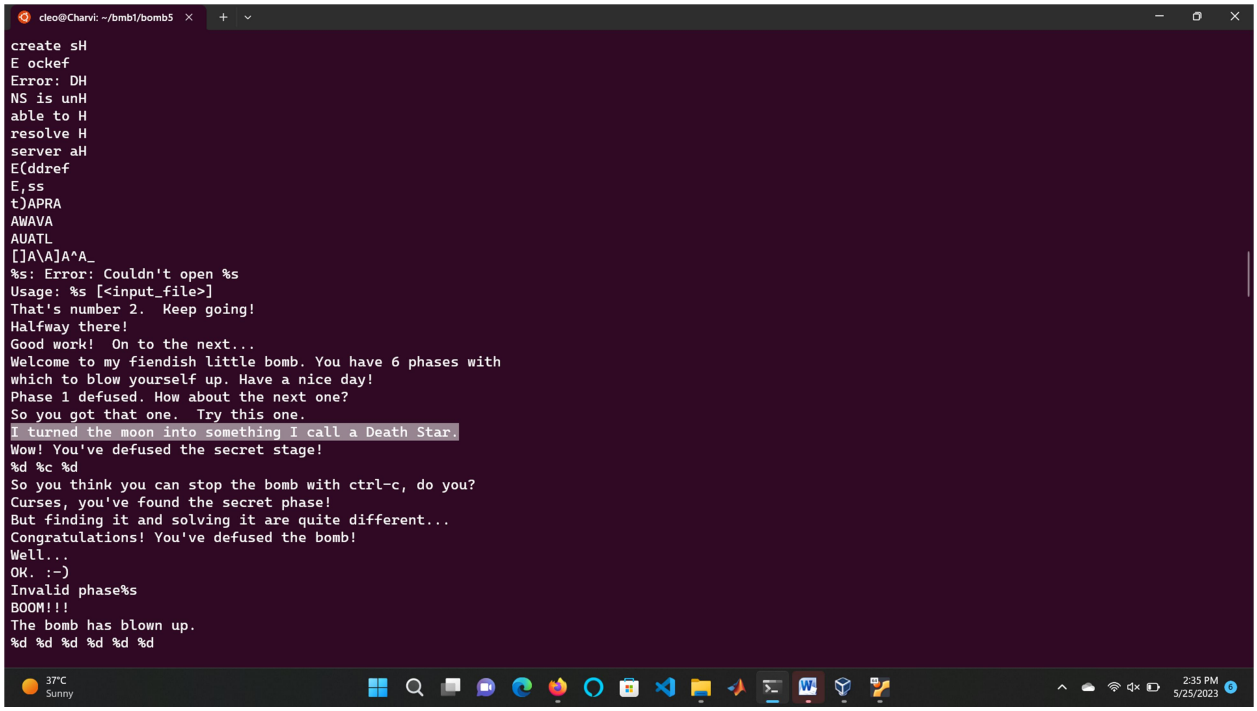


We add break point at phase_1 using “break” command.

Now run the program and enter any dummy input. The program will break at phase1.

Use disas command to see assembly code of phase1.

Check memory $ 0x402450 (x/s 0x402450),it has a string which is being moved into $esi register.And passes onto string_not_equal . The solution key for phase1 is the string that moved into $esi register.



else we can use “strings bomb” command to find the available strings and find the string there as shown in the figure.

ANS: I turned the moon into something I call a Death Star.

Phase 2

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.

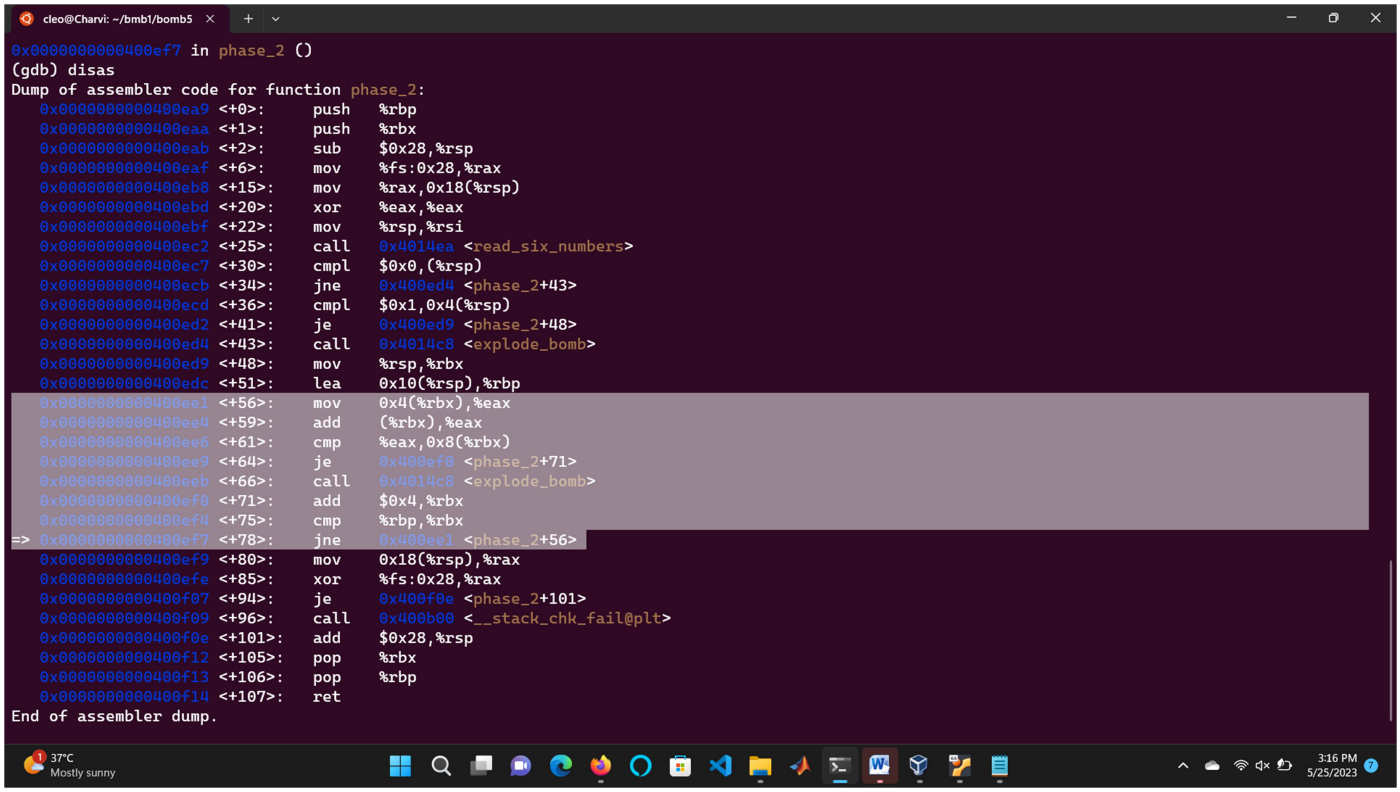
At <+30> we understand that the first value is supposed to be 0 otherwise the bomb explodes.

Similarly at <+36> we infer that the second input is going to be 1 .

<+56> to <+78> we have a loop which runs till %rbp and %rbx are equal.

In the loop at <+59> the rbx value is added to %eax each time.

Hence our next input will become sum of preceding two numbers.



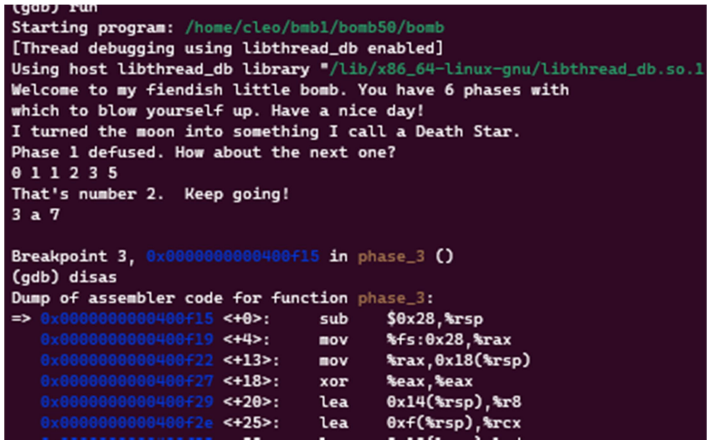
ANS: 0 1 1 2 3 5

Phase 3

In phase 3

At <+35> by checking the memory 0x4024ae using command x/s 0x4024ae we infer inputs must be given as : int ,char, int [0x4024ae: "%d %c %d"]

From the assembly code +45 and +55 we infer that the first input should be greater than 2 & less than 7.



```
   0x000000000400f2e <+25>:    lea    0xf(%rsp),%rcx
   0x000000000400f33 <+30>:    lea    0x10(%rsp),%rdx
   0x000000000400f38 <+35>:    mov    $0x4024ae,%esi
   0x000000000400f3d <+40>:    call   0x400bb0 <__isoc99_sscanf@plt>
   0x000000000400f42 <+45>:    cmp    $0x2,%eax
   0x000000000400f45 <+48>:    jg     0x400f4c <phase_3+55>
   0x000000000400f47 <+50>:    call   0x4014c8 <explode_bomb>
   0x000000000400f4c <+55>:    cmpl   $0x7,0x10(%rsp)
   0x000000000400f51 <+60>:    ja     0x401053 <phase_3+318>
   0x000000000400f57 <+66>:    mov    0x10(%rsp),%eax
   0x000000000400f5b <+70>:    jmp    *0x4024c0(,%rax,8)
   0x000000000400f62 <+77>:    mov    $0x79,%eax
   0x000000000400f67 <+82>:    cmpl   $0x2aa,0x14(%rsp)
   0x000000000400f6f <+90>:    je     0x40105d <phase_3+328>
   0x000000000400f75 <+96>:    call   0x4014c8 <explode_bomb>
   0x000000000400f7a <+101>:   mov    $0x79,%eax
   0x000000000400f7f <+106>:   jmp    0x40105d <phase_3+328>
   0x000000000400f84 <+111>:   mov    $0x75,%eax
   0x000000000400f89 <+116>:   cmpl   $0x351,0x14(%rsp)
   0x000000000400f91 <+124>:   je     0x40105d <phase_3+328>
   0x000000000400f97 <+130>:   call   0x4014c8 <explode_bomb>
   0x000000000400f9c <+135>:   mov    $0x75,%eax
   0x000000000400fa1 <+140>:   jmp    0x40105d <phase_3+328>
   0x000000000400fa6 <+145>:   mov    $0x71,%eax
   0x000000000400fab <+150>:   cmpl   $0x29c,0x14(%rsp)
   0x000000000400fb3 <+158>:   je     0x40105d <phase_3+328>
   0x000000000400fb9 <+164>:   call   0x4014c8 <explode_bomb>
   0x000000000400fbe <+169>:   mov    $0x71,%eax
--Type <RET> for more, q to quit, c to continue without paging--
   0x000000000400fc3 <+174>:   jmp    0x40105d <phase_3+328>
   0x000000000400fc8 <+179>:   mov    $0x6f,%eax
   0x000000000400fcd <+184>:   cmpl   $0x153,0x14(%rsp)
   0x000000000400fd5 <+192>:   je     0x40105d <phase_3+328>
   0x000000000400fdb <+198>:   call   0x4014c8 <explode_bomb>
   0x000000000400fe0 <+203>:   mov    $0x6f,%eax
   0x000000000400fe5 <+208>:   jmp    0x40105d <phase_3+328>
   0x000000000400fe7 <+210>:   mov    $0x77,%eax
   0x000000000400fec <+215>:   cmpl   $0x2ff,0x14(%rsp)
   0x000000000400ff4 <+223>:   je     0x40105d <phase_3+328>
   0x000000000400ff6 <+225>:   call   0x4014c8 <explode_bomb>
   0x000000000400ffb <+230>:   mov    $0x77,%eax
   0x0000000000401000 <+235>:   jmp    0x40105d <phase_3+328>
   0x0000000000401002 <+237>:   mov    $0x63,%eax
   0x0000000000401007 <+242>:   cmpl   $0x274,0x14(%rsp)
   0x000000000040100f <+250>:   je     0x40105d <phase_3+328>
   0x0000000000401011 <+252>:   call   0x4014c8 <explode_bomb>
   0x0000000000401016 <+257>:   mov    $0x63,%eax
   0x000000000040101b <+262>:   jmp    0x40105d <phase_3+328>
   0x000000000040101d <+264>:   mov    $0x6d,%eax
   0x0000000000401022 <+269>:   cmpl   $0x186,0x14(%rsp)
   0x000000000040102a <+277>:   je     0x40105d <phase_3+328>
   0x000000000040102c <+279>:   call   0x4014c8 <explode_bomb>
   0x0000000000401031 <+284>:   mov    $0x6d,%eax
   0x0000000000401036 <+289>:   jmp    0x40105d <phase_3+328>
   0x0000000000401038 <+291>:   mov    $0x62,%eax
   0x000000000040103d <+296>:   cmpl   $0x1f6,0x14(%rsp)
   0x0000000000401045 <+304>:   je     0x40105d <phase_3+328>
   0x0000000000401047 <+306>:   call   0x4014c8 <explode_bomb>
   0x000000000040104c <+311>:   mov    $0x62,%eax
   0x0000000000401051 <+316>:   jmp    0x40105d <phase_3+328>
   0x0000000000401053 <+318>:   call   0x4014c8 <explode_bomb>
   0x0000000000401058 <+323>:   mov    $0x69,%eax
   0x000000000040105d <+328>:   cmp    0xf(%rsp),%al
--Type <RET> for more, q to quit, c to continue without paging--
   0x0000000000401061 <+332>:   je     0x401068 <phase_3+339>
   0x0000000000401063 <+334>:   call   0x4014c8 <explode_bomb>
   0x0000000000401068 <+339>:   mov    0x18(%rsp),%rax
   0x000000000040106d <+344>:   xor    %fs:0x28,%rax
   0x0000000000401076 <+353>:   je     0x40107d <phase_3+360>
   0x0000000000401078 <+355>:   call   0x400b00 <__stack_chk_fail@plt>
   0x000000000040107d <+360>:   add    $0x28,%rsp
   0x0000000000401081 <+364>:   ret
End of assembler dump.
```

The <+70> instruction jumps the header to <+179>

According to the <+184> instruction the second integer must be 0x153 therefore the second integer input is 339.

Now re-run the program from the beginning and now give the input as :

“3 a 339” (cause the next instructions steps into explode_bomb as second input is wrong)

And disas the code again

Now going through the assembly code at <+328> checking memory %al we get the character as ‘o’

Therefore ‘3 o 339’ would be the right input.

ANS: 3 o 339

Phase 4

From 0x40264f we infer that the input must be 2 integers

```
That's number 2.  Keep going!
3 o 339
Halfway there!
3 4

Breakpoint 4, 0x00000000004010c0 in phase_4 ()
(gdb) disas
Dump of assembler code for function phase_4:
=> 0x00000000004010c0 <+0>:     sub    $0x18,%rsp
   0x00000000004010c4 <+4>:     mov    %fs:0x28,%rax
   0x00000000004010cd <+13>:    mov    %rax,0x8(%rsp)
   0x00000000004010d2 <+18>:    xor    %eax,%eax
   0x00000000004010d4 <+20>:    lea    0x4(%rsp),%rcx
   0x00000000004010d9 <+25>:    mov    %rsp,%rdx
   0x00000000004010dc <+28>:    mov    $0x40264f,%esi
   0x00000000004010e1 <+33>:    call   0x400bb0 <__isoc99_sscanf@plt>
   0x00000000004010e6 <+38>:    cmp    $0x2,%eax
   0x00000000004010e9 <+41>:    jne    0x4010f1 <phase_4+49>
   0x00000000004010eb <+43>:    cmpl   $0xe,(%rsp)
   0x00000000004010ef <+47>:    jbe    0x4010f6 <phase_4+54>
   0x00000000004010f1 <+49>:    call   0x4014c8 <explode_bomb>
   0x00000000004010f6 <+54>:    mov    $0xe,%edx
   0x00000000004010fb <+59>:    mov    $0x0,%esi
   0x0000000000401100 <+64>:    mov    (%rsp),%edi
   0x0000000000401103 <+67>:    call   0x401082 <func4>
   0x0000000000401108 <+72>:    cmp    $0x4,%eax
   0x000000000040110b <+75>:    jne    0x401114 <phase_4+84>
   0x000000000040110d <+77>:    cmpl   $0x4,0x4(%rsp)
   0x0000000000401112 <+82>:    je     0x401119 <phase_4+89>
   0x0000000000401114 <+84>:    call   0x4014c8 <explode_bomb>
   0x0000000000401119 <+89>:    mov    0x8(%rsp),%rax
   0x000000000040111e <+94>:    xor    %fs:0x28,%rax
   0x0000000000401127 <+103>:   je     0x40112e <phase_4+110>
   0x0000000000401129 <+105>:   call   0x400b00 <__stack_chk_fail@plt>
   0x000000000040112e <+110>:   add    $0x18,%rsp
   0x0000000000401132 <+114>:   ret
End of assembler dump.
(gdb) x/s 0x40264f
0x40264f:       "%d %d"
```

Instruction <+77> infers that the second input value must be 4 else bomb will explode.

Also at <+43> It states that the first input must be less than 15.

Therefore we give the inputs such that second input is 4 and first input is any number less than 15.

Checking with each possibility(with breakpoint at explode_bomb) the bomb gets defused at input “2 4”

ANS: 2 4

SECRET PHASE STRING

```
(gdb) disas phase_defused
Dump of assembler code for function phase_defused:
   0x000000000040164f <+0>:     sub    $0x78,%rsp
   0x0000000000401653 <+4>:     mov    %fs:0x28,%rax
   0x000000000040165c <+13>:    mov    %rax,0x68(%rsp)
   0x0000000000401661 <+18>:    xor    %eax,%eax
   0x0000000000401663 <+20>:    cmpl   $0x6,0x203122(%rip)        # 0x60478c <num_input_strings>
   0x000000000040166a <+27>:    jne    0x4016ca <phase_defused+123>
   0x000000000040166c <+29>:    lea    0x10(%rsp),%r8
   0x0000000000401671 <+34>:    lea    0xc(%rsp),%rcx
   0x0000000000401676 <+39>:    lea    0x8(%rsp),%rdx
   0x000000000040167b <+44>:    mov    $0x402699,%esi
   0x0000000000401680 <+49>:    mov    $0x604890,%edi
   0x0000000000401685 <+54>:    call   0x400bb0 <__isoc99_sscanf@plt>
   0x000000000040168a <+59>:    cmp    $0x3,%eax
   0x000000000040168d <+62>:    jne    0x4016c0 <phase_defused+113>
   0x000000000040168f <+64>:    mov    $0x4026a2,%esi
   0x0000000000401694 <+69>:    lea    0x10(%rsp),%rdi
   0x0000000000401699 <+74>:    call   0x4013c9 <strings_not_equal>
   0x000000000040169e <+79>:    test   %eax,%eax
   0x00000000004016a0 <+81>:    jne    0x4016c0 <phase_defused+113>
   0x00000000004016a2 <+83>:    mov    $0x402578,%edi
   0x00000000004016a7 <+88>:    call   0x400ae0 <puts@plt>
   0x00000000004016ac <+93>:    mov    $0x4025a0,%edi
   0x00000000004016b1 <+98>:    call   0x400ae0 <puts@plt>
   0x00000000004016b6 <+103>:   mov    $0x0,%eax
   0x00000000004016bb <+108>:   call   0x4012e0 <secret_phase>
   0x00000000004016c0 <+113>:   mov    $0x4025d8,%edi
   0x00000000004016c5 <+118>:   call   0x400ae0 <puts@plt>
   0x00000000004016ca <+123>:   mov    0x68(%rsp),%rax
   0x00000000004016cf <+128>:   xor    %fs:0x28,%rax
   0x00000000004016d8 <+137>:   je     0x4016df <phase_defused+144>
   0x00000000004016da <+139>:   call   0x400b00 <__stack_chk_fail@plt>
   0x00000000004016df <+144>:   add    $0x78,%rsp
   0x00000000004016e3 <+148>:   ret
End of assembler dump.
(gdb) x/s 0x4026a2
0x4026a2:       "DrEvil"
(gdb)
```

The %esi just above the <secret_phase> contains the string to be appended to the phase_4 for unlocking the secret phase.

In this case the string is ‘DrEvil ‘.

Ans: DrEvil

Phase 5

We are supposed to give 6 inputs (We got to know from the function call in line <+4> & the comparison after return)

There is an array starting from address 0x402500.....So, we check the values present in the array

2 10 6 1 12 16

In line <+53>, it is checking if our sum ecx= 0x3f (63).....So, taking numbers from the array, we can take the combinations of 1,2,12,16,16,16 i.e. Index = 3,0,4,5,5,5

Index of 1 is 3, 2 is 0,12 is 4, 16 is 5

Checking the Lower Case alphabets with lower 4bits equal to the index => The solution is 'cpdeee' in any order.

```
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?
0 1 1 2 3 5
That's number 2.  Keep going!
3 o 339
Halfway there!
2 4 DrEvil
So you got that one.  Try this one.
kjdfkj

Breakpoint 5, 0x0000000000401133 in phase_5 ()
(gdb) disas
Dump of assembler code for function phase_5:
=> 0x0000000000401133 <+0>:     push   %rbx
   0x0000000000401134 <+1>:     mov    %rdi,%rbx
   0x0000000000401137 <+4>:     call   0x4013ab <string_length>
   0x000000000040113c <+9>:     cmp    $0x6,%eax
   0x000000000040113f <+12>:    je     0x401146 <phase_5+19>
   0x0000000000401141 <+14>:    call   0x4014c8 <explode_bomb>
   0x0000000000401146 <+19>:    mov    %rbx,%rax
   0x0000000000401149 <+22>:    lea    0x6(%rbx),%rdi
   0x000000000040114d <+26>:    mov    $0x0,%ecx
   0x0000000000401152 <+31>:    movzbl (%rax),%edx
   0x0000000000401155 <+34>:    and    $0xf,%edx
   0x0000000000401158 <+37>:    add    0x402500(,%rdx,4),%ecx
   0x000000000040115f <+44>:    add    $0x1,%rax
   0x0000000000401163 <+48>:    cmp    %rdi,%rax
   0x0000000000401166 <+51>:    jne    0x401152 <phase_5+31>
   0x0000000000401168 <+53>:    cmp    $0x3f,%ecx
   0x000000000040116b <+56>:    je     0x401172 <phase_5+63>
   0x000000000040116d <+58>:    call   0x4014c8 <explode_bomb>
   0x0000000000401172 <+63>:    pop    %rbx
   0x0000000000401173 <+64>:    ret
End of assembler dump.
(gdb) x/6wd 0x402500
0x402500 <array.3600>:  2       10      6       1
0x402510 <array.3600+16>:       12      16
(gdb)
```

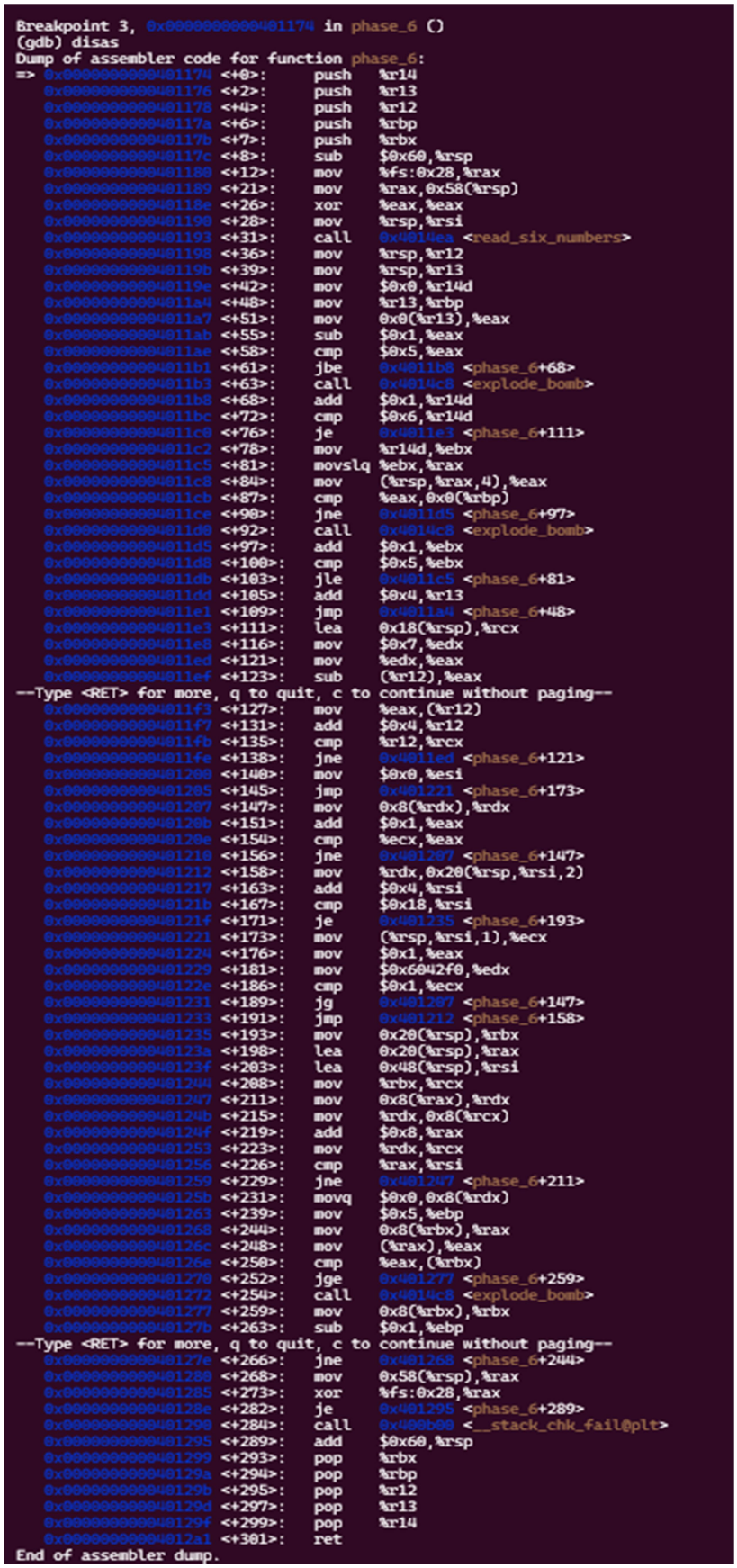
ANS: cpedee

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.

Using x/24xd 0x6042f0 instruction we can find the node values with the indices .



```
Breakpoint 3, 0x0000000000401174 in phase_6 ()
(gdb) disas
Dump of assembler code for function phase_6:
=> 0x0000000000401174 <+0>:     push   %r14
   0x0000000000401176 <+2>:     push   %r13
   0x0000000000401178 <+4>:     push   %r12
   0x000000000040117a <+6>:     push   %rbp
   0x000000000040117b <+7>:     push   %rbx
   0x000000000040117c <+8>:     sub    $0x60,%rsp
   0x0000000000401180 <+12>:    mov    %fs:0x28,%rax
   0x0000000000401189 <+21>:    mov    %rax,0x58(%rsp)
   0x000000000040118e <+26>:    xor    %eax,%eax
   0x0000000000401190 <+28>:    mov    %rsp,%rsi
   0x0000000000401193 <+31>:    call   0x4014ea <read_six_numbers>
   0x0000000000401198 <+36>:    mov    %rsp,%r12
   0x000000000040119b <+39>:    mov    %rsp,%r13
   0x000000000040119e <+42>:    mov    $0x0,%r14d
   0x00000000004011a4 <+48>:    mov    %r13,%rbp
   0x00000000004011a7 <+51>:    mov    0x0(%r13),%eax
   0x00000000004011ab <+55>:    sub    $0x1,%eax
   0x00000000004011ae <+58>:    cmp    $0x5,%eax
   0x00000000004011b1 <+61>:    jbe    0x4011b8 <phase_6+68>
   0x00000000004011b3 <+63>:    call   0x4014c8 <explode_bomb>
   0x00000000004011b8 <+68>:    add    $0x1,%r14d
   0x00000000004011bc <+72>:    cmp    $0x6,%r14d
   0x00000000004011c0 <+76>:    je     0x4011e3 <phase_6+111>
   0x00000000004011c2 <+78>:    mov    %r14d,%ebx
   0x00000000004011c5 <+81>:    movslq %ebx,%rax
   0x00000000004011c8 <+84>:    mov    (%rsp,%rax,4),%eax
   0x00000000004011cb <+87>:    cmp    %eax,0x0(%rbp)
   0x00000000004011ce <+90>:    jne    0x4011d5 <phase_6+97>
   0x00000000004011d0 <+92>:    call   0x4014c8 <explode_bomb>
   0x00000000004011d5 <+97>:    add    $0x1,%ebx
   0x00000000004011d8 <+100>:   cmp    $0x5,%ebx
   0x00000000004011db <+103>:   jle    0x4011c5 <phase_6+81>
   0x00000000004011dd <+105>:   add    $0x4,%r13
   0x00000000004011e1 <+109>:   jmp    0x4011a4 <phase_6+48>
   0x00000000004011e3 <+111>:   lea    0x18(%rsp),%rcx
   0x00000000004011e8 <+116>:   mov    $0x7,%edx
   0x00000000004011ed <+121>:   mov    %edx,%eax
   0x00000000004011ef <+123>:   sub    (%r12),%eax
--Type <RET> for more, q to quit, c to continue without paging--
   0x00000000004011f3 <+127>:   mov    %eax,(%r12)
   0x00000000004011f7 <+131>:   add    $0x4,%r12
   0x00000000004011fb <+135>:   cmp    %r12,%rcx
   0x00000000004011fe <+138>:   jne    0x4011ed <phase_6+121>
   0x0000000000401200 <+140>:   mov    $0x0,%esi
   0x0000000000401205 <+145>:   jmp    0x401221 <phase_6+173>
   0x0000000000401207 <+147>:   mov    0x8(%rdx),%rdx
   0x000000000040120b <+151>:   add    $0x1,%eax
   0x000000000040120e <+154>:   cmp    %ecx,%eax
   0x0000000000401210 <+156>:   jne    0x401207 <phase_6+147>
   0x0000000000401212 <+158>:   mov    %rdx,0x20(%rsp,%rsi,2)
   0x0000000000401217 <+163>:   add    $0x4,%rsi
   0x000000000040121b <+167>:   cmp    $0x18,%rsi
   0x000000000040121f <+171>:   je     0x401235 <phase_6+193>
   0x0000000000401221 <+173>:   mov    (%rsp,%rsi,1),%ecx
   0x0000000000401224 <+176>:   mov    $0x1,%eax
   0x0000000000401229 <+181>:   mov    $0x6042f0,%edx
   0x000000000040122e <+186>:   cmp    $0x1,%ecx
   0x0000000000401231 <+189>:   jg     0x401207 <phase_6+147>
   0x0000000000401233 <+191>:   jmp    0x401212 <phase_6+158>
   0x0000000000401235 <+193>:   mov    0x20(%rsp),%rbx
   0x000000000040123a <+198>:   lea    0x20(%rsp),%rax
   0x000000000040123f <+203>:   lea    0x48(%rsp),%rsi
   0x0000000000401244 <+208>:   mov    %rbx,%rcx
   0x0000000000401247 <+211>:   mov    0x8(%rax),%rdx
   0x000000000040124b <+215>:   mov    %rdx,0x8(%rcx)
   0x000000000040124f <+219>:   add    $0x8,%rax
   0x0000000000401253 <+223>:   mov    %rdx,%rcx
   0x0000000000401256 <+226>:   cmp    %rax,%rsi
   0x0000000000401259 <+229>:   jne    0x401247 <phase_6+211>
   0x000000000040125b <+231>:   movq   $0x0,0x8(%rdx)
   0x0000000000401263 <+239>:   mov    $0x5,%ebp
   0x0000000000401268 <+244>:   mov    0x8(%rbx),%rax
   0x000000000040126c <+248>:   mov    (%rax),%eax
   0x000000000040126e <+250>:   cmp    %eax,(%rbx)
   0x0000000000401270 <+252>:   jge    0x401277 <phase_6+259>
   0x0000000000401272 <+254>:   call   0x4014c8 <explode_bomb>
   0x0000000000401277 <+259>:   mov    0x8(%rbx),%rbx
   0x000000000040127b <+263>:   sub    $0x1,%ebp
--Type <RET> for more, q to quit, c to continue without paging--
   0x000000000040127e <+266>:   jne    0x401268 <phase_6+244>
   0x0000000000401280 <+268>:   mov    0x58(%rsp),%rax
   0x0000000000401285 <+273>:   xor    %fs:0x28,%rax
   0x000000000040128e <+282>:   je     0x401295 <phase_6+289>
   0x0000000000401290 <+284>:   call   0x400b00 <__stack_chk_fail@plt>
   0x0000000000401295 <+289>:   add    $0x60,%rsp
   0x0000000000401299 <+293>:   pop    %rbx
   0x000000000040129a <+294>:   pop    %rbp
   0x000000000040129b <+295>:   pop    %r12
   0x000000000040129d <+297>:   pop    %r13
   0x000000000040129f <+299>:   pop    %r14
   0x00000000004012a1 <+301>:   ret
End of assembler dump.
```

Now arrange the indices according to the node values in descending order which is “3 6 4 2 5 1”

(descending order of node values :999 931 894 648 566 400)

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 inputwould be the 7-y, y=sorted node indices.

Input 1: 7-3 =4

Input 2: 7-6 =1

Input 3: 7-4 =3

Input 4: 7-2 =5

Input 5: 7-5 =2

Input 6: 7-1 =6

Therefore our input string must be 4 1 3 5 2 6

NODE Values:

0x6042f0 <node1>: 400 1

0x604300 <node2>: 648 2

0x604310 <node3>: 999 3

0x604320 <node4>: 894 4

0x604330 <node5>: 566 5

0x604340 <node6>: 931 6

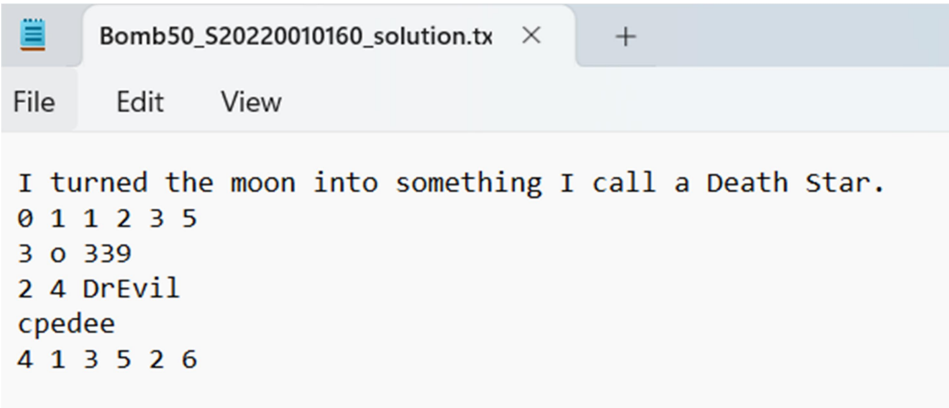
```
   0x000000000040129d <+297>:   pop    %r13
   0x000000000040129f <+299>:   pop    %r14
   0x00000000004012a1 <+301>:   ret
End of assembler dump.
(gdb) x/24xd 0x6042f0
0x6042f0 <node1>:       400     1       6308608 0
0x604300 <node2>:       648     2       6308624 0
0x604310 <node3>:       999     3       6308640 0
0x604320 <node4>:       894     4       6308656 0
0x604330 <node5>:       566     5       6308672 0
0x604340 <node6>:       931     6       0       0
(gdb) x/24x 0x6042f0
0x6042f0 <node1>:       0x00000190      0x00000001      0x00604300      0x00000000
0x604300 <node2>:       0x00000288      0x00000002      0x00604310      0x00000000
0x604310 <node3>:       0x000003e7      0x00000003      0x00604320      0x00000000
0x604320 <node4>:       0x0000037e      0x00000004      0x00604330      0x00000000
0x604330 <node5>:       0x00000236      0x00000005      0x00604340      0x00000000
0x604340 <node6>:       0x000003a3      0x00000006      0x00000000      0x00000000
(gdb) r solutions.txt
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/cleo/bmb1/bomb50/bomb solutions.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...
4 1 3 5 2 6

Breakpoint 3, 0x0000000000401174 in phase_6 ()
(gdb) next
Single stepping until exit from function phase_6,
which has no line number information.
main (argc=<optimized out>, argv=<optimized out>) at bomb.c:109
109             phase_defused();
(gdb)
```

ANS: 4 1 3 5 2 6

```
Reading symbols from bomb...
(gdb) b explode_bomb
Breakpoint 1 at 0x4014c8
(gdb) b secret_phase
Breakpoint 2 at 0x4012e0
(gdb) r Bomb50_S20220010160_solution.txt
Starting program: /home/cleo/bmb1/bomb50/bomb Bomb50_S20220010160_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...

Breakpoint 2, 0x00000000004012e0 in secret_phase ()
(gdb)
```



Solutions

1) I turned the moon into something I call a Death Star.

2) 0 1 1 2 3 5

3) 3 o 339

4) 2 4 DrEvil

5) cpedee

6) 4 1 3 5 2 6