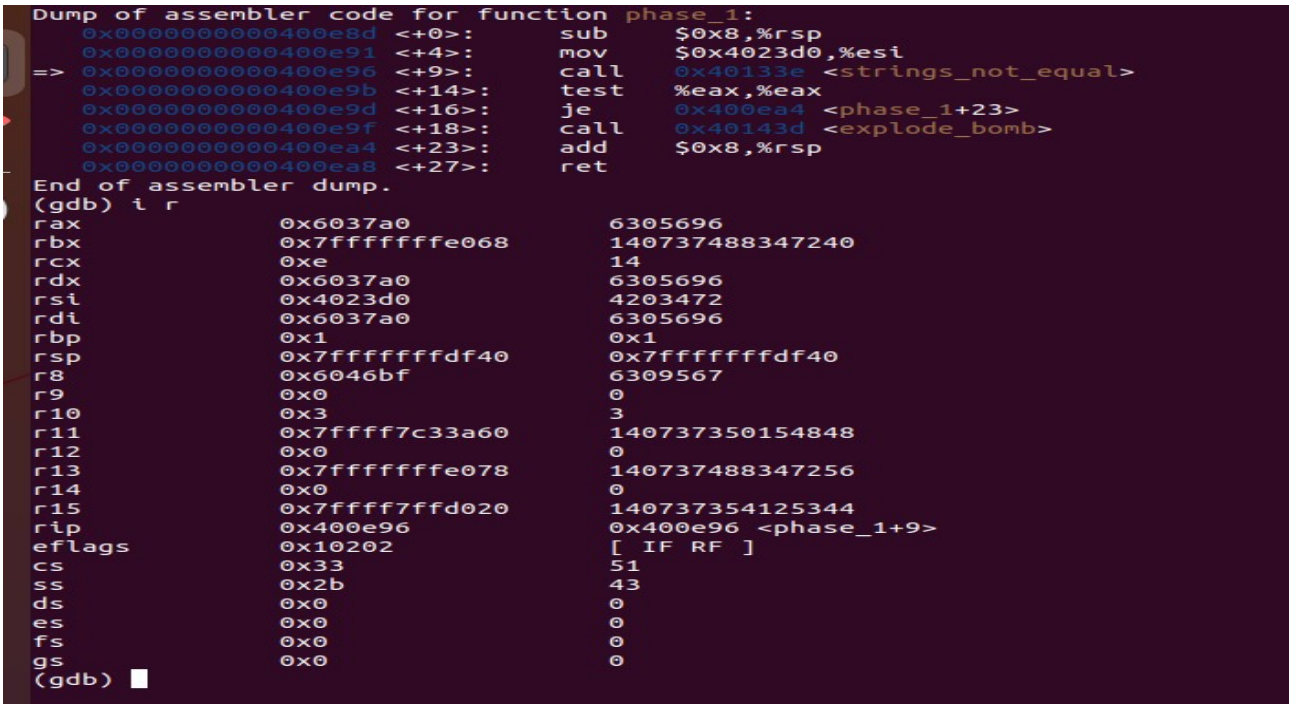
S20220010168 P.Jaswanth
BOMB LAB DOCUMENTATION:

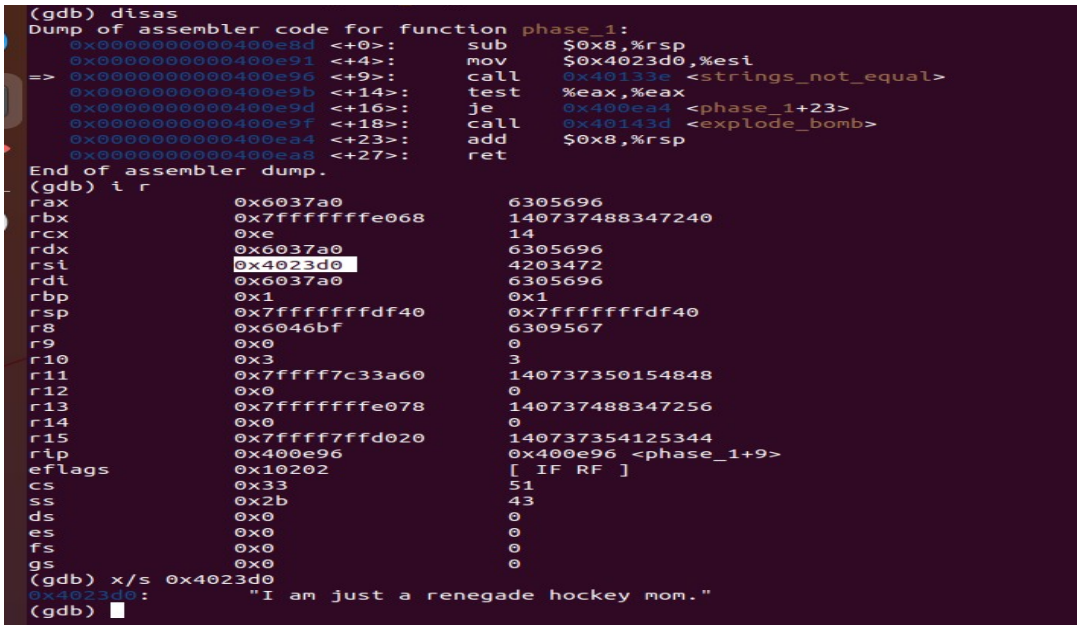
Before going to all phases.
Make sure to see the file location of bomb.c was opened it in the terminal, then it can be easy to access the terminal while we diffusing an each phases of bomb.
Now Let's Start the Diffusion of All Bombs.

Phase_1 :

Step -1 Open the terminal and type gdb bomb.
Step -2 Then you will enter to the (gdb)__.
Step -3 Now, Type "b phase_1" or "break phase_1". By typing this "b phase_1" or "break phase_1" it access your phase_1 breakpoints.
Step -4 Now in the next (gdb) type run.Then you can see the text as:
"Welcome to my fiendish little bomb. You have 6 phases with



which to blow yourself up. Have a nice day!"
Step -5 Type a random string or any integers to check the right one. Then you will get a "breakpoint 1, 0x0000000000400exx in phase_1()
(gdb)____"
Step -6 Now in gdb type disas
Step -7 By clicking disas it opens all the phase_1 registers.



Step -8 %esi stores $0x4023d0. To check that we will type “x/s $0x4023d0.

Then it shows and reads the string.(“I am just a renegade hockey mom.”)
Phase 1 reads in a string and explodes unless the string matches the predetermined password string.
So that is the phase_1 output and the phase_1 is diffused.
Now type Quit in gdb and Check the next bomb..

Phase_2:

Step -1 Open the terminal and type gdb bomb.
Step -2 Then you will enter to the (gdb)__.
Step -3 Now, Type "b phase_2" or "break phase_2". By typing this "b phase_2" or "break phase_2" it access your phase_2 breakpoints.
Step -4 Now in the next (gdb) type run.Then you can see the text as:
"Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!"
Step -5 Type a random string or any integers to check the right one. Then you will get a "breakpoint 1, 0x0000000000400exx in phase_2()
(gdb)____"
Step -6 Now in gdb type disas
Step -7 By clicking disas it opens all the phase_2 registers.

```
Breakpoint 1, 0x0000000000400ea9 in phase_2 ()
(gdb) disas
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   0x40145f <read_six_numbers>
   0x0000000000400ec7 <+30>:    cmpl   $0x1,(%rsp)
   0x0000000000400ecb <+34>:    je     0x400ed2 <phase_2+41>
   0x0000000000400ecd <+36>:    call   0x40143d <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   0x40143d <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.
```

Phase 2 reads in six numbers and it makes a each loop to be continued and executed by checking the given inputs.If the input is wrong then the function call explode_bomb.

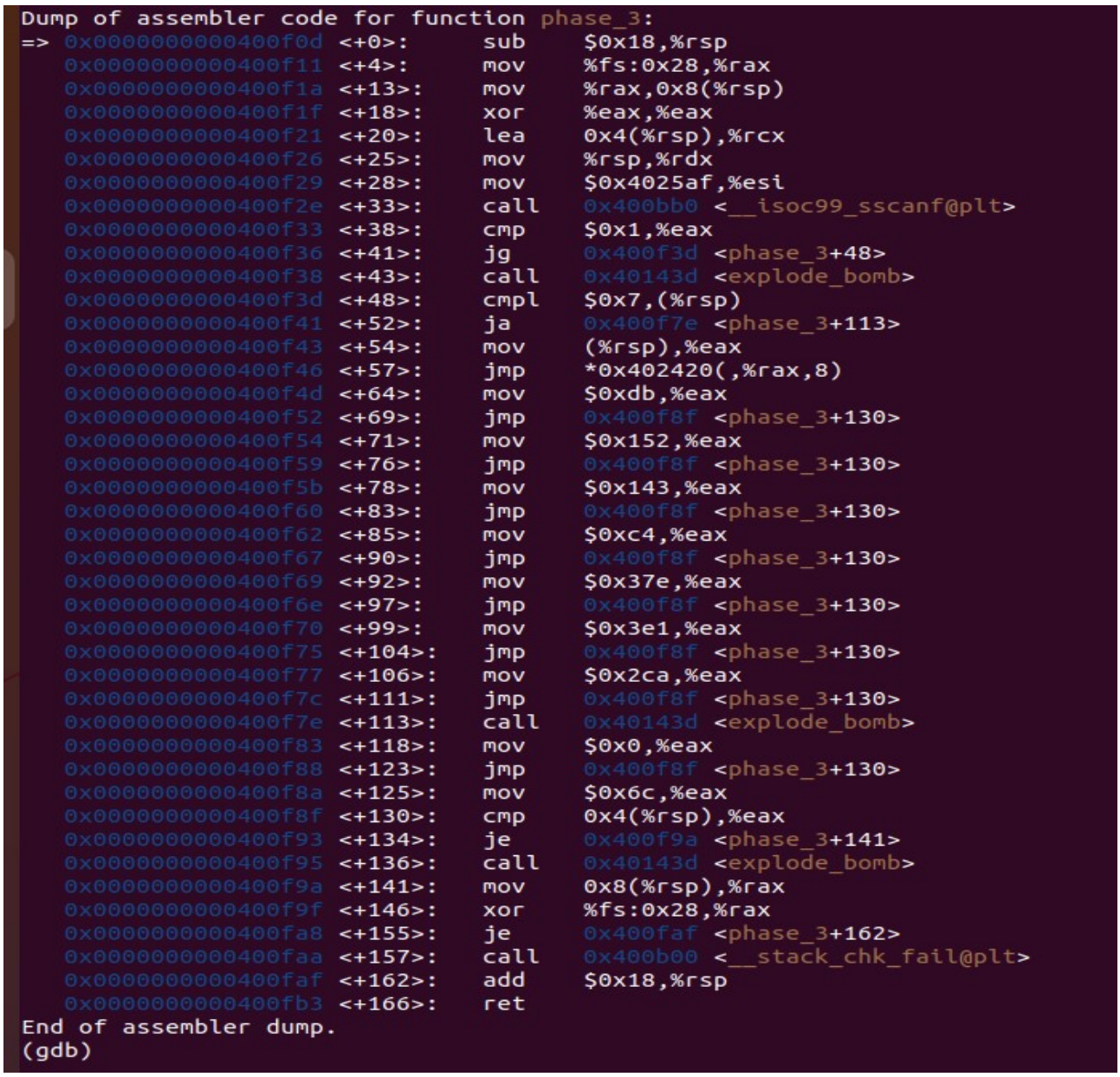
Due to that it goes to the sequence as: 2 power n
i.e 2^0 = 1 if n=0
i.e 2^1 = 2 if n=1
i.e 2^2 = 4 if n=2
i.e 2^3 = 8 if n=3.... and so on.
So that is the phase_2 output and the phase_2 is diffused.
Now type Quit in gdb and Check the next bomb..

```
Reading symbols from bomb...
(gdb) b phase_3
Breakpoint 1 at 0x400f0d
(gdb) run
Starting program: /home/jaswanth/Desktop/bomb55/bomb

This GDB supports auto-downloading debuginfo from the following URLs:
https://debuginfod.ubuntu.com
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
[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 am just a renegade hockey mom.
Phase 1 defused. How about the next one?
1 2 4 8 16 32
That's number 2.  Keep going!
```

Phase 3 :

Step -1 Open the terminal and type gdb bomb.
Step -2 Then you will enter to the (gdb)__.
Step -3 Now, Type "b phase_3" or "break phase_3". By typing this "b phase_3" or "break phase_3" it access your phase_3 breakpoints.
Step -4 Now in the next (gdb) type run.Then you can see the text as:
"Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!"
Step -5 Type a random string or any integers to check the right one. Then you will get a "breakpoint 1, 0x0000000000400exx in phase_3()
(gdb)____"
Step -6 Now in gdb type disas
Step -7 By clicking disas it opens all the phase_3 registers.



Step -8: We will see directly in call function, it tells that is “scanf”. So for that we need to check the before address what it is stored for that we will check it in “mov $0x4025af, %esi”. The given mov instruction tells that esi is storing “$0x4025af”.
Step -9: so we will check that “$0x4025af” what it is stored. For that we will type in gdb as__
“x/s $0x4025af” and then press enter.
So it shows that it stores “%d %d”.
That means it stores two integers.

By checking the first integer of which is used to determine the password by use of a switch statement. The bomb explodes if the value of the second number does not match this password .

So that is the phase_3 output and the phase_3 is diffused.
Now type Quit in gdb and Check the next bomb..

```
Reading symbols from bomb...
(gdb) b phase_4
Breakpoint 1 at 0x400fef
(gdb) run
Starting program: /home/jaswanth/Desktop/bomb55/bomb

This GDB supports auto-downloading debuginfo from the following URLs:
https://debuginfod.ubuntu.com
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
[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 am just a renegade hockey mom.
Phase 1 defused. How about the next one?
1 2 4 8 16 32
That's number 2.  Keep going!
7 714
Halfway there!
```

Phase 4 :

Step -1 Open the terminal and type gdb bomb.
Step -2 Then you will enter to the (gdb)__.
Step -3 Now, Type "b phase_4" or "break phase_4". By typing this "b phase_4" or "break phase_4" it access your phase_4 breakpoints.
Step -4 Now in the next (gdb) type run.Then you can see the text as:
"Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!"
Step -5 Type a random string or any integers to check the right one. Then you will get a "breakpoint 1, 0x0000000000400exx in phase_4()
(gdb)____"
Step -6 Now in gdb type disas
Step -7 By clicking disas it opens all the phase_4 registers.

```
Dump of assembler code for function phase_4:
=> 0x0000000000400fef <+0>:     sub    $0x18,%rsp
   0x0000000000400ff3 <+4>:     mov    %fs:0x28,%rax
   0x0000000000400ffc <+13>:    mov    %rax,0x8(%rsp)
   0x0000000000401001 <+18>:    xor    %eax,%eax
   0x0000000000401003 <+20>:    mov    %rsp,%rcx
   0x0000000000401006 <+23>:    lea    0x4(%rsp),%rdx
   0x000000000040100b <+28>:    mov    $0x4025af,%esi
   0x0000000000401010 <+33>:    call   0x400bb0 <__isoc99_sscanf@plt>
   0x0000000000401015 <+38>:    cmp    $0x2,%eax
   0x0000000000401018 <+41>:    jne    0x401025 <phase_4+54>
   0x000000000040101a <+43>:    mov    (%rsp),%eax
   0x000000000040101d <+46>:    sub    $0x2,%eax
   0x0000000000401020 <+49>:    cmp    $0x2,%eax
   0x0000000000401023 <+52>:    jbe    0x40102a <phase_4+59>
   0x0000000000401025 <+54>:    call   0x40143d <explode_bomb>
   0x000000000040102a <+59>:    mov    (%rsp),%esi
   0x000000000040102d <+62>:    mov    $0x9,%edi
   0x0000000000401032 <+67>:    call   0x400fb4 <func4>
   0x0000000000401037 <+72>:    cmp    0x4(%rsp),%eax
   0x000000000040103b <+76>:    je     0x401042 <phase_4+83>
   0x000000000040103d <+78>:    call   0x40143d <explode_bomb>
   0x0000000000401042 <+83>:    mov    0x8(%rsp),%rax
   0x0000000000401047 <+88>:    xor    %fs:0x28,%rax
   0x0000000000401050 <+97>:    je     0x401057 <phase_4+104>
   0x0000000000401052 <+99>:    call   0x400b00 <__stack_chk_fail@plt>
   0x0000000000401057 <+104>:   add    $0x18,%rsp
   0x000000000040105b <+108>:   ret
End of assembler dump.
(gdb)
```

Step -8: We will see directly in call function, it tells that is “scanf”. So for that we need to check the before address what it is stored for that we will check it in “mov $0x4025af, %esi”. The given mov instruction tells that esi is storing “$0x4025af”.
Step -9: so we will check that “$0x4025af” what it is stored. For that we will type in gdb as__
“x/s $0x4025af” and then press enter.
So it shows that it stores “%d %d”.
That means it stores two integers.(picture is in the below)

```
   0x000000000040100b <+28>:    mov    $0x4025af,%esi
=> 0x0000000000401010 <+33>:    call   0x400bb0 <__isoc99_sscanf@plt>
   0x0000000000401015 <+38>:    cmp    $0x2,%eax
   0x0000000000401018 <+41>:    jne    0x401025 <phase_4+54>
   0x000000000040101a <+43>:    mov    (%rsp),%eax
   0x000000000040101d <+46>:    sub    $0x2,%eax
   0x0000000000401020 <+49>:    cmp    $0x2,%eax
   0x0000000000401023 <+52>:    jbe    0x40102a <phase_4+59>
   0x0000000000401025 <+54>:    call   0x40143d <explode_bomb>
   0x000000000040102a <+59>:    mov    (%rsp),%esi
   0x000000000040102d <+62>:    mov    $0x9,%edi
   0x0000000000401032 <+67>:    call   0x400fb4 <func4>
   0x0000000000401037 <+72>:    cmp    0x4(%rsp),%eax
   0x000000000040103b <+76>:    je     0x401042 <phase_4+83>
   0x000000000040103d <+78>:    call   0x40143d <explode_bomb>
   0x0000000000401042 <+83>:    mov    0x8(%rsp),%rax
   0x0000000000401047 <+88>:    xor    %fs:0x28,%rax
   0x0000000000401050 <+97>:    je     0x401057 <phase_4+104>
   0x0000000000401052 <+99>:    call   0x400b00 <__stack_chk_fail@plt>
   0x0000000000401057 <+104>:   add    $0x18,%rsp
   0x000000000040105b <+108>:   ret
End of assembler dump.
(gdb) i r
rax            0x0                 0
rbx            0x7fffffffe068      140737488347240
rcx            0x7fffffffdf30      140737488346928
rdx            0x7fffffffdf34      140737488346932
rsi            0x4025af            4203951
rdi            0x603890            6305936
rbp            0x1                 0x1
rsp            0x7fffffffdf30      0x7fffffffdf30
r8             0x6046b5            6309557
r9             0x0                 0
r10            0x7ffff7d9cac0      140737351633600
r11            0x246               582
r12            0x0                 0
r13            0x7fffffffe078      140737488347256
r14            0x0                 0
r15            0x7ffff7ffd020      140737354125344
rip            0x401010            0x401010 <phase_4+33>
eflags         0x10246             [ PF ZF IF RF ]
cs             0x33                51
ss             0x2b                43
ds             0x0                 0
es             0x0                 0
fs             0x0                 0
gs             0x0                 0
(gdb) x/d 0x4025af
0x4025af:       622879781
(gdb) x/s 0x4025af
0x4025af:       "%d %d"
(gdb)
```

In Phase 4 it reads in one number and runs a recursive calculation function, with the number of recursions equalling the input number.

The recursive function, func4, starts with a value of one and as the stack collapses each function collapse. The bomb explodes if the number calculated by this function does not equal 352.

So that is the phase_4 output and the phase_4 is diffused.
Now type Quit in gdb and Check the next bomb..

```
Reading symbols from bomb...
(gdb) b phase_5
Breakpoint 1 at 0x40105c
(gdb) run
Starting program: /home/jaswanth/Desktop/bomb55/bomb

This GDB supports auto-downloading debuginfo from the following URLs:
https://debuginfod.ubuntu.com
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
[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 am just a renegade hockey mom.
Phase 1 defused. How about the next one?
1 2 4 8 16 32
That's number 2.  Keep going!
7 714
Halfway there!
352 4
So you got that one.  Try this one.
```

Phase 5 :

Step -1 Open the terminal and type gdb bomb.
Step -2 Then you will enter to the (gdb)__.
Step -3 Now, Type "b phase_5" or "break phase_5". By typing this "b phase_5" or "break phase_5" it access your phase_5 breakpoints.
Step -4 Now in the next (gdb) type run.Then you can see the text as:
"Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!"
Step -5 Type a random string or any integers to check the right one. Then you will get a "breakpoint 1, 0x0000000000400exx in phase_5()
(gdb)____"
Step -6 Now in gdb type disas
Step -7 By clicking disas it opens all the phase_5 registers.

```
Breakpoint 3, 0x000000000040105c in phase_5 ()
(gdb) disas
Dump of assembler code for function phase_5:
=> 0x000000000040105c <+0>:     sub    $0x18,%rsp
   0x0000000000401060 <+4>:     mov    %fs:0x28,%rax
   0x0000000000401069 <+13>:    mov    %rax,0x8(%rsp)
   0x000000000040106e <+18>:    xor    %eax,%eax
   0x0000000000401070 <+20>:    lea    0x4(%rsp),%rcx
   0x0000000000401075 <+25>:    mov    %rsp,%rdx
   0x0000000000401078 <+28>:    mov    $0x4025af,%esi
   0x000000000040107d <+33>:    call   0x400bb0 <__isoc99_sscanf@plt>
   0x0000000000401082 <+38>:    cmp    $0x1,%eax
   0x0000000000401085 <+41>:    jg     0x40108c <phase_5+48>
   0x0000000000401087 <+43>:    call   0x40143d <explode_bomb>
   0x000000000040108c <+48>:    mov    (%rsp),%eax
   0x000000000040108f <+51>:    and    $0xf,%eax
   0x0000000000401092 <+54>:    mov    %eax,(%rsp)
   0x0000000000401095 <+57>:    cmp    $0xf,%eax
   0x0000000000401098 <+60>:    je     0x4010c9 <phase_5+109>
   0x000000000040109a <+62>:    mov    $0x0,%ecx
   0x000000000040109f <+67>:    mov    $0x0,%edx
   0x00000000004010a4 <+72>:    add    $0x1,%edx
   0x00000000004010a7 <+75>:    cltq
   0x00000000004010a9 <+77>:    mov    0x402460(,%rax,4),%eax
   0x00000000004010b0 <+84>:    add    %eax,%ecx
   0x00000000004010b2 <+86>:    cmp    $0xf,%eax
   0x00000000004010b5 <+89>:    jne    0x4010a4 <phase_5+72>
   0x00000000004010b7 <+91>:    movl   $0xf,(%rsp)
   0x00000000004010be <+98>:    cmp    $0xf,%edx
   0x00000000004010c1 <+101>:   jne    0x4010c9 <phase_5+109>
   0x00000000004010c3 <+103>:   cmp    0x4(%rsp),%ecx
   0x00000000004010c7 <+107>:   je     0x4010ce <phase_5+114>
   0x00000000004010c9 <+109>:   call   0x40143d <explode_bomb>
   0x00000000004010ce <+114>:   mov    0x8(%rsp),%rax
   0x00000000004010d3 <+119>:   xor    %fs:0x28,%rax
   0x00000000004010dc <+128>:   je     0x4010e3 <phase_5+135>
   0x00000000004010de <+130>:   call   0x400b00 <__stack_chk_fail@plt>
   0x00000000004010e3 <+135>:   add    $0x18,%rsp
   0x00000000004010e7 <+139>:   ret
End of assembler dump.
(gdb)
```

Step -8: We will see directly in call function, it tells that is “scanf”. So for that we need to check the before address what it is stored for that we will check it in “mov $0x4025af, %esi”. The given mov instruction tells that esi is storing “$0x4025af”.
Step -9: so we will check that “$0x4025af” what it is stored. For that we will type in gdb as__ “x/s $0x4025af” and then press enter.
So it shows that it stores “%d %d”.

We can tell that Phase 5 reads in two numbers, the first of which is used as a starting point within a sequence of numbers. The bomb explodes if the number of steps to get to the number 15 in the sequence does not equal 15, or if the second input number does not equal the sum of the numbers stepped along to reach 15 (including 15).

```
(gdb) x/100d 0x402460
0x402460 <array.3597>:  10      0       0       0       2       0       0       0
0x402468 <array.3597+8>:        14      0       0       0       7       0       0       0
0x402470 <array.3597+16>:       8       0       0       0       12      0       0       0
0x402478 <array.3597+24>:       15      0       0       0       11      0       0       0
0x402480 <array.3597+32>:       0       0       0       0       4       0       0       0
0x402488 <array.3597+40>:       1       0       0       0       13      0       0       0
0x402490 <array.3597+48>:       3       0       0       0       9       0       0       0
0x402498 <array.3597+56>:       6       0       0       0       5       0       0       0
0x4024a0:       83      111     32      121     111     117     32      116
0x4024a8:       104     105     110     107     32      121     111     117
0x4024b0:       32      99      97      110     32      115     116     111
0x4024b8:       112     32      116     104     101     32      98      111
0x4024c0:       109     98      32      119
(gdb) x/2d 0x402460
0x402460 <array.3597>:  10      0
```

As we can see from the above picture for “0x402460” we check the each steps to get the number 15. and if it the number doesn”t call in the sequence then the bomb explodes.
From the above picture, it tells that looping continues 14 times (without including 15) as :
15<-6<-14<-2<-1<-10<-0<-8 <-4<-9<-13<-11<-7<-3<-12<-5
from the above numbers we can tell the looping was continued from 5 to 15 i.e 14 times to check the number.
So by checking like this my first number is 5 and after checking the loop correctly then it goes to another instruction and in another instruction it stores as 115.

So that is the phase_5 output and the phase_5 is diffused.
Now type Quit in gdb and Check the next bomb..

```
Reading symbols from bomb...
(gdb) b phase_6
Breakpoint 1 at 0x4010e8
(gdb) run
Starting program: /home/jaswanth/Desktop/bomb55/bomb

This GDB supports auto-downloading debuginfo from the following URLs:
https://debuginfod.ubuntu.com
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
[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 am just a renegade hockey mom.
Phase 1 defused. How about the next one?
1 2 4 8 16 32
That's number 2.  Keep going!
7 714
Halfway there!
352 4
So you got that one.  Try this one.
5 115
Good work!  On to the next...
```