Sheldon Phase 2- Writeup

Objective of this report is to introduce the used approach to find the passphrase for the phase_2 of the Sheldon_1 binary file.

Since we already know how the program works, let's start with disassembling the phase_2 to analyze its functionality. Also, notice the 'b phase_2' command which was used to set a breakpoint at phase_2 [Figure 1].

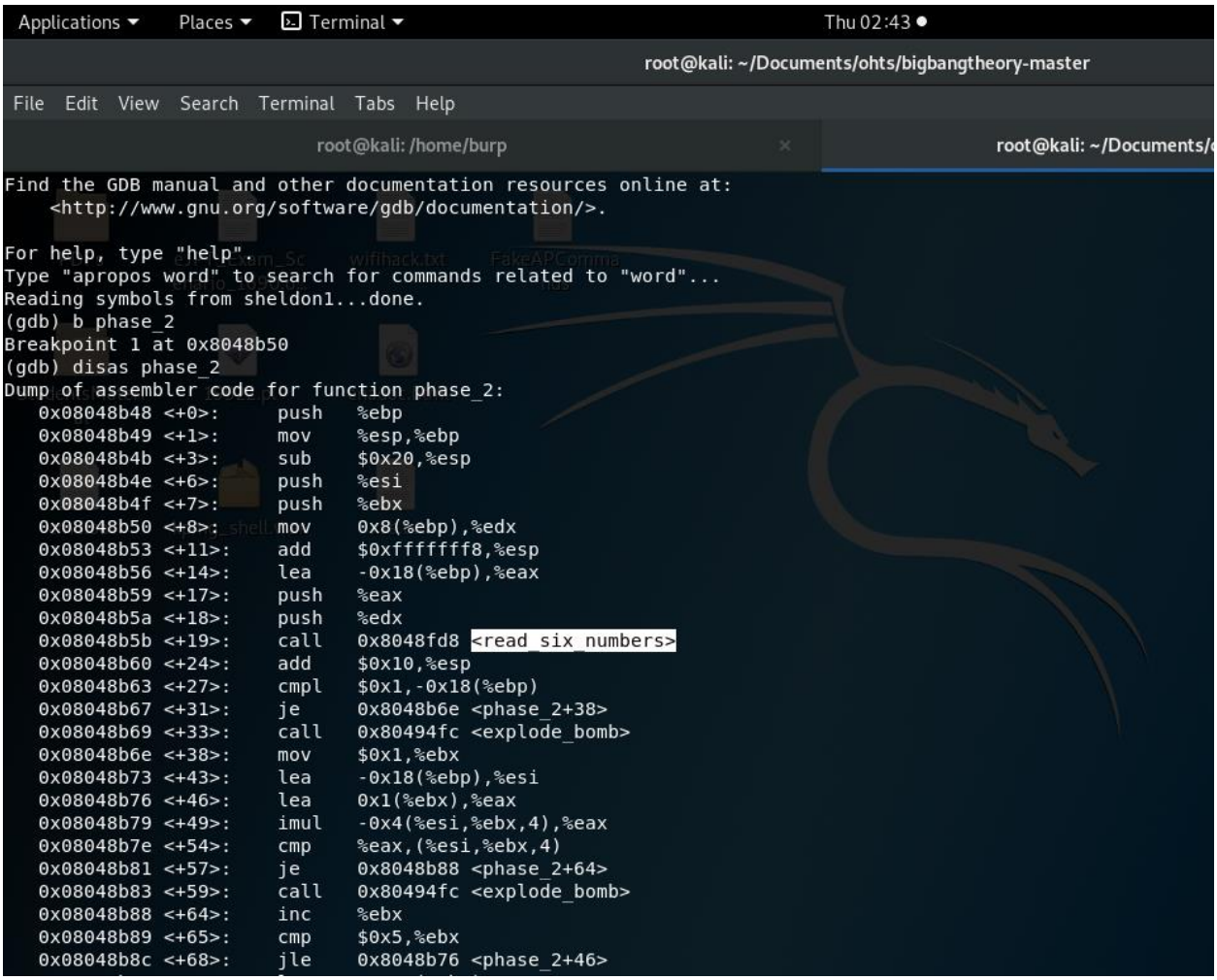

Figure 1: Disassembled phase_2 function

After going through line by line we can see that at the line number 19, there's a call for a function named as read_six_numbers. Now we have an idea about the passphrase to this level (6 numbers). To examine more, let's see what we can find inside the read_six_numbers function [Figure 2].

```
   0x08048b95 <+77>:    pop    %ebp
   0x08048b96 <+78>:    ret
End of assembler dump.
(gdb) disas read_six_numbers
Dump of assembler code for function read_six_numbers:
   0x08048fd8 <+0>:     push   %ebp
   0x08048fd9 <+1>:     mov    %esp,%ebp
   0x08048fdb <+3>:     sub    $0x8,%esp
   0x08048fde <+6>:     mov    0x8(%ebp),%ecx
   0x08048fe1 <+9>:     mov    0xc(%ebp),%edx
   0x08048fe4 <+12>:    lea    0x14(%edx),%eax
   0x08048fe7 <+15>:    push   %eax
   0x08048fe8 <+16>:    lea    0x10(%edx),%eax
   0x08048feb <+19>:    push   %eax
   0x08048fec <+20>:    lea    0xc(%edx),%eax
   0x08048fef <+23>:    push   %eax
   0x08048ff0 <+24>:    lea    0x8(%edx),%eax
   0x08048ff3 <+27>:    push   %eax
   0x08048ff4 <+28>:    lea    0x4(%edx),%eax
   0x08048ff7 <+31>:    push   %eax
   0x08048ff8 <+32>:    push   %edx
   0x08048ff9 <+33>:    push   $0x8049b1b
   0x08048ffe <+38>:    push   %ecx
   0x08048fff <+39>:    call   0x8048860 <sscanf@plt>
   0x08049004 <+44>:    add    $0x20,%esp
   0x08049007 <+47>:    cmp    $0x5,%eax
   0x0804900a <+50>:    jg     0x8049011 <read_six_numbers+57>
   0x0804900c <+52>:    call   0x80494fc <explode_bomb>
   0x08049011 <+57>:    mov    %ebp,%esp
   0x08049013 <+59>:    pop    %ebp
   0x08049014 <+60>:    ret
End of assembler dump.
(gdb) x/s 0x8049b1b
0x8049b1b:      "%d %d %d %d %d %d"
(gdb)
```

Figure 2: Disassembled read_six_numbers function

At line number 33, we can see that a variable is being pushed to the stack before calling for the scanf function which is used to grab the input. Let's see what's inside the address by using the x/s command. Even though we had an idea that the input consists of 6 numbers, it wasn't clear enough about the input pattern. After digging the function, we can see that 6 integers which are separated with spaces are needed as the valid passphrase [Figure 2].

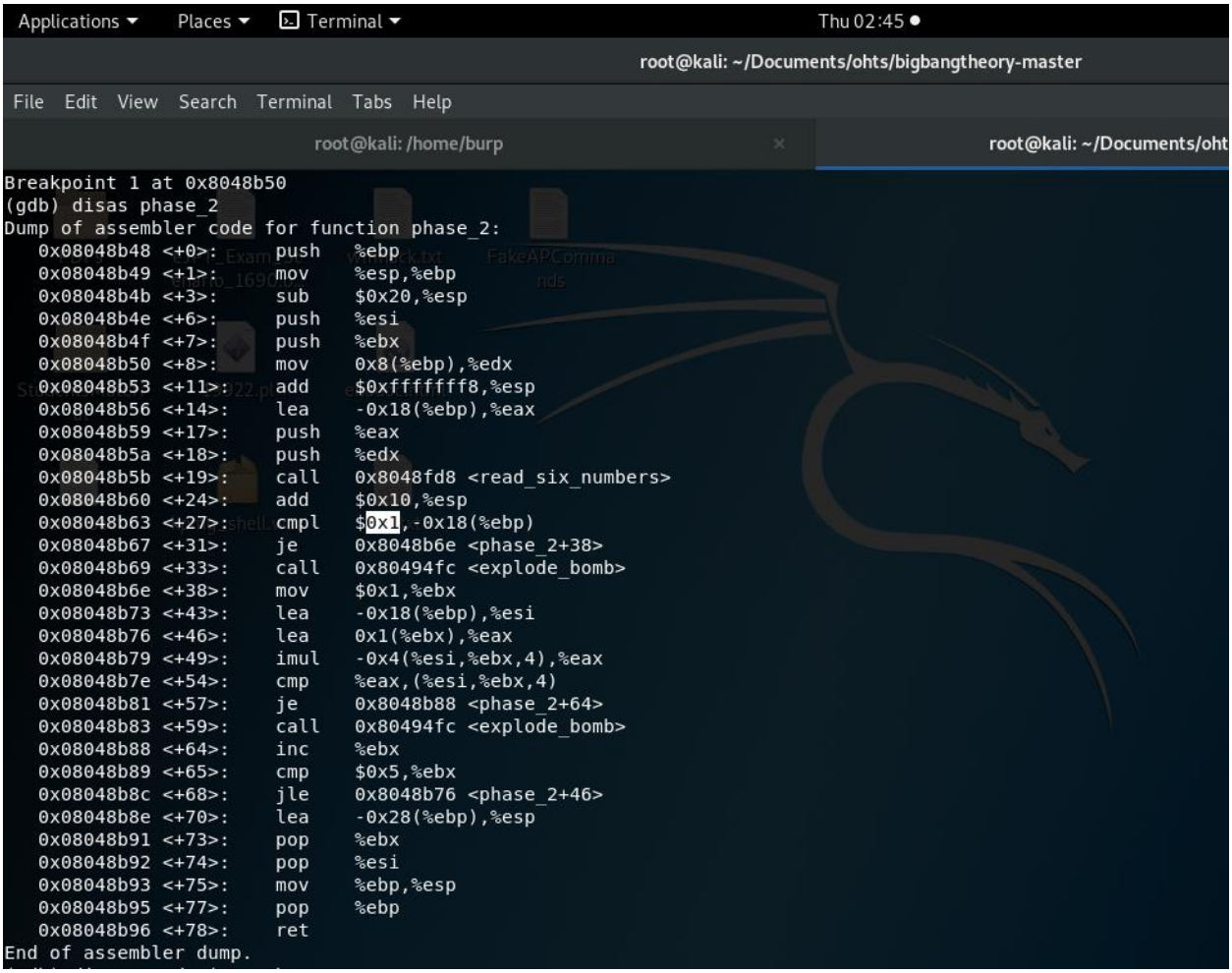

Figure 3: Identification of the first integer

In the phase_2 at line 27, we can see a comparison with a variable ($0x1) and if the condition is correct the code will jump to line number 38 to continue with the execution. If the condition turned out as false it'll execute the explode_bomb function and it will force to end the program. We can use print command (p 0x1) to find the decimal value of any hex value. For the variable $0x1 the decimal value is 1. Now we have a clue about which might be the first number of the passphrase [Figure 3].

Since we have set the breakpoint from the beginning and we found the first integer as 1, let's run the program with number 1 as the first integer (put rest of the numbers as you like).Note that pass is a text file which contains the password of the phase1 [Figure 4].

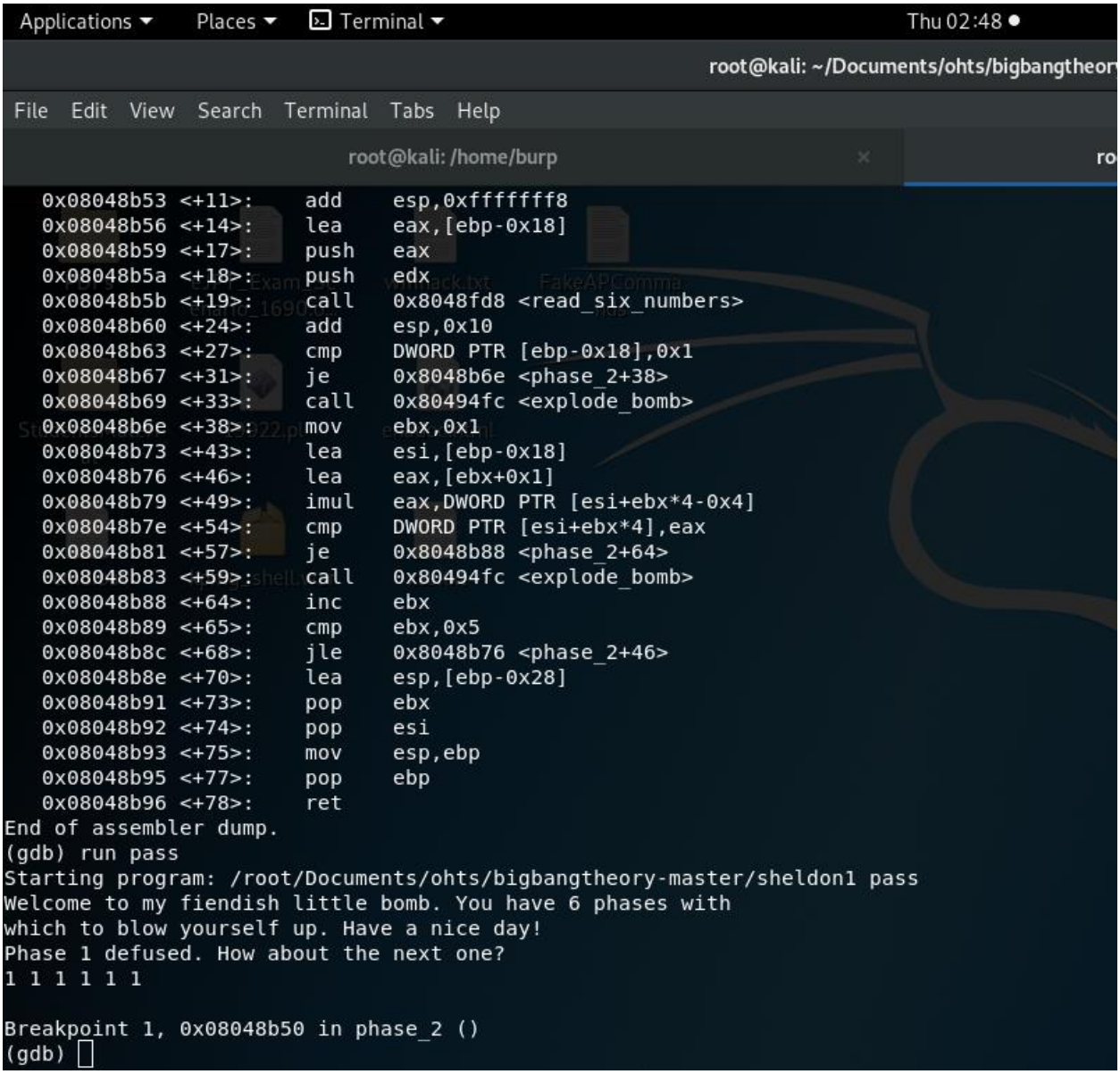

Figure 4: Execution of the program with the first integer

Now let's jump to the phase_2 function's line 57 by using the until command with the respective memory address [Figure 5].

```
Applications ▾   Places ▾   Terminal ▾                                   Thu 02:49 ●
                                                   root@kali: ~/Documents/ohts/bigbang
File  Edit  View  Search  Terminal  Tabs  Help
                                  root@kali: /home/burp
   0x08048b76 <+46>:    lea    eax,[ebx+0x1]
   0x08048b79 <+49>:    imul   eax,DWORD PTR [esi+ebx*4-0x4]
   0x08048b7e <+54>:    cmp    DWORD PTR [esi+ebx*4],eax
   0x08048b81 <+57>:    je     0x8048b88 <phase_2+64>
   0x08048b83 <+59>:    call   0x80494fc <explode_bomb>
   0x08048b88 <+64>:    inc    ebx
   0x08048b89 <+65>:    cmp    ebx,0x5
   0x08048b8c <+68>:    jle    0x8048b76 <phase_2+46>
   0x08048b8e <+70>:    lea    esp,[ebp-0x28]
   0x08048b91 <+73>:    pop    ebx
   0x08048b92 <+74>:    pop    esi
   0x08048b93 <+75>:    mov    esp,ebp
   0x08048b95 <+77>:    pop    ebp
   0x08048b96 <+78>:    ret
End of assembler dump.
(gdb) until *0x08048b81
0x08048b81 in phase_2 ()
(gdb) i r
eax            0x2                 2
ecx            0x0                 0
edx            0x0                 0
ebx            0x1                 1
esp            0xffffd280          0xffffd280
ebp            0xffffd2a8          0xffffd2a8
esi            0xffffd290          -11632
edi            0xf7fad000          -134557696
eip            0x8048b81           0x8048b81 <phase_2+57>
eflags         0x297               [ CF PF AF SF IF ]
cs             0x23                35
ss             0x2b                43
ds             0x2b                43
es             0x2b                43
fs             0x0                 0
gs             0x63                99
(gdb)
```

Figure 5: Identification of the second integer

At line 54, we can see a comparison happening between the register values. If the input value is correct the next value will be passed to the eax register. Also, we can see that the line number 49 is where this transformation happens to create the next value. To find the register values we can use the info registers command [Figure 5]. TADA! We found the next value as '2' and if we continue the execution using next instruction (ni) command we can see that the bomb explodes [Figure 6]. Reason for that is after executing line number 68 it jumps to 48 to check the next value. Since our input was 1 1 1 1 1 1, it generates the second value in the register (which we found as '2') and check with the input value. If it's not a match then the bomb explodes.

```
Applications ▾   Places ▾   Terminal ▾                    Thu 02:50 ●
                                          root@kali: ~/Documents/ohts/bigban
File  Edit  View  Search  Terminal  Tabs  Help
                 root@kali: /home/burp        root@kali: ~/Documents/ohts/bigbangtheory
fs             0x0                 0
gs             0x63                99
(gdb) ni
0x08048b83 in phase_2 ()
(gdb) i r
eax            0x2                 2
ecx            0x0                 0
edx            0x0                 0
ebx            0x1                 1
esp            0xffffd280          0xffffd280
ebp            0xffffd2a8          0xffffd2a8
esi            0xffffd290          -11632
edi            0xf7fad000          -134557696
eip            0x8048b83           0x8048b83 <phase_2+59>
eflags         0x297               [ CF PF AF SF IF ]
cs             0x23                35
ss             0x2b                43
ds             0x2b                43
es             0x2b                43
fs             0x0                 0
gs             0x63                99
(gdb) ni

BOOM!!!
The bomb has blown up.
[Inferior 1 (process 3835) exited with code 010]
(gdb) run pass
Starting program: /root/Documents/ohts/bigbangtheory-master/sheldon1 pass
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?
1 2 1 1 1 1

Breakpoint 1, 0x08048b50 in phase_2 ()
(gdb)
```

Figure 6:Execution of the program with the second integer

In order to get the next value, we must run the program again with the found second value [Figure 6]. It generates the next value as 6 and the bomb explodes because of the aforementioned scenario [Figure 7].

```
Applications ▾   Places ▾   Terminal ▾                                Thu 02:59
root@kali: ~/Documents/ohts/big
File  Edit  View  Search  Terminal  Tabs  Help
root@kali: /home/burp                    root@kali: ~/Documents/ohts/bigbangth
0x08048b76 in phase_2 ()
(gdb)
0x08048b79 in phase_2 ()
(gdb)
0x08048b7e in phase_2 ()
(gdb) i r
eax            0x6                 6
ecx            0x0                 0
edx            0x0                 0
ebx            0x2                 2
esp            0xffffd280          0xffffd280
ebp            0xffffd2a8          0xffffd2a8
esi            0xffffd290          -11632
edi            0xf7fad000          -134557696
eip            0x8048b7e           0x8048b7e <phase_2+54>
eflags         0x206               [ PF IF ]
cs             0x23                35
ss             0x2b                43
ds             0x2b                43
es             0x2b                43
fs             0x0                 0
gs             0x63                99
(gdb) ni
0x08048b81 in phase_2 ()
(gdb) x/i Quit
(gdb) x/i 0x08048b81
=> 0x8048b81 <phase_2+57>:      je     0x8048b88 <phase_2+64>
(gdb) ni
0x08048b83 in phase_2 ()
(gdb) ni

BOOM!!!
The bomb has blown up.
[Inferior 1 (process 4031) exited with code 010]
(gdb)
```

Figure 7: Identification of the third integer

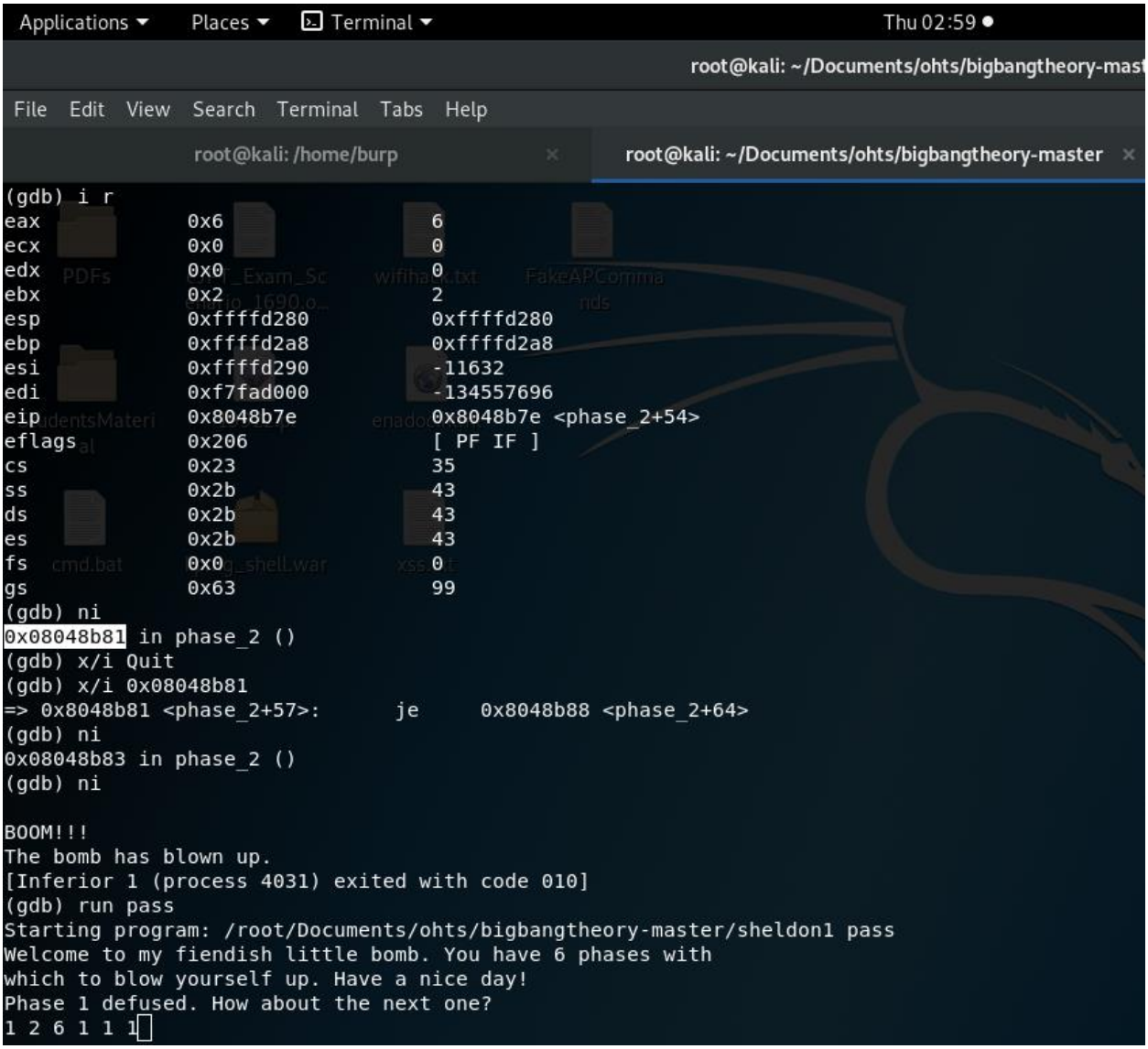

Figure 8: Execution of the program with the third integer

Let's run the program again by inserting the found value 6 to the input section [Figure 8].

```
Applications ▾   Places ▾   Terminal ▾
                                                    root@kali: ~/Docu
File  Edit  View  Search  Terminal  Tabs  Help
                 root@kali: /home/burp                 root@kali: ~/Documents/
ss             0x2b                43
ds             0x2b                43
es             0x2b                43
fs             0x0                 0
gs             0x63                99
(gdb) ni
0x08048b88 in phase_2 ()
(gdb)
0x08048b89 in phase_2 ()
(gdb)
0x08048b8c in phase_2 ()
(gdb)
0x08048b76 in phase_2 ()
(gdb)
0x08048b79 in phase_2 ()
(gdb)
0x08048b7e in phase_2 ()
(gdb) i r
eax            0x18                24
ecx            0x0                 0
edx            0x0                 0
ebx            0x3                 3
esp            0xffffd280          0xffffd280
ebp            0xffffd2a8          0xffffd2a8
esi            0xffffd290          -11632
edi            0xf7fad000          -134557696
eip            0x8048b7e           0x8048b7e <phase_2+54>
eflags         0x206               [ PF IF ]
cs             0x23                35
ss             0x2b                43
ds             0x2b                43
es             0x2b                43
fs             0x0                 0
gs             0x63                99
(gdb)
```

Figure 9: Identification of the fourth integer

By executing the ni and i r commands we were able to find the next value from the eax register [Figure 9]. After running the program again by using the input as 1 2 6 24 1 1 we can get the fifth value [Figure 10]. Note that we can use until command to directly jump to the line 49 in order to execute the line which generates the next value from the given input.

```
root@kali: /home/burp                                  root@kali: ~/Document
ds             0x2b                43
es             0x2b                43
fs             0x0                 0
gs             0x63                99
(gdb) until *0x08048b79
0x08048b79 in phase_2 ()
(gdb) ni
0x08048b7e in phase_2 ()
(gdb) i r
eax            0x78                120
ecx            0x0                 0
edx            0x0                 0
ebx            0x4                 4
esp            0xffffd280          0xffffd280
ebp            0xffffd2a8          0xffffd2a8
esi            0xffffd290          -11632
edi            0xf7fad000          -134557696
eip            0x8048b7e           0x8048b7e <phase_2+54>
eflags         0x206               [ PF IF ]
cs             0x23                35
ss             0x2b                43
ds             0x2b                43
es             0x2b                43
fs             0x0                 0
gs             0x63                99
(gdb) ni
0x08048b81 in phase_2 ()
(gdb)
0x08048b83 in phase_2 ()
(gdb)

BOOM!!!
The bomb has blown up.
[Inferior 1 (process 4157) exited with code 010]
(gdb)
```

Figure 10: Identification of the fifth integer

By performing the same process, we were able to get all the six numbers through the eax register [Figure 11].

```
Applications ▾   Places ▾   Terminal ▾
                                                    root@kali: ~/Docum
File  Edit  View  Search  Terminal  Tabs  Help
                    root@kali: /home/burp          root@kali: ~/Documents/oh
ebx            0x4                 4
esp            0xffffd280          0xffffd280
ebp            0xffffd2a8          0xffffd2a8
esi            0xffffd290          -11632
edi            0xf7fad000          -134557696
eip            0x8048b7e           0x8048b7e <phase_2+54>
eflags         0x206               [ PF IF ]
cs             0x23                35
ss             0x2b                43
ds             0x2b                43
es             0x2b                43
fs             0x0                 0
gs             0x63                99
(gdb) until *0x08048b79
0x08048b79 in phase_2 ()
(gdb) ni
0x08048b7e in phase_2 ()
(gdb) i r
eax            0x2d0               720
ecx            0x0                 0
edx            0x0                 0
ebx            0x5                 5
esp            0xffffd280          0xffffd280
ebp            0xffffd2a8          0xffffd2a8
esi            0xffffd290          -11632
edi            0xf7fad000          -134557696
eip            0x8048b7e           0x8048b7e <phase_2+54>
eflags         0x202               [ IF ]
cs             0x23                35
ss             0x2b                43
ds             0x2b                43
es             0x2b                43
fs             0x0                 0
gs             0x63                99
(gdb)
```

Figure 11: Identification of the sixth integer

Now we have all the six numbers, to run the program again we must delete or disable the breakpoints.

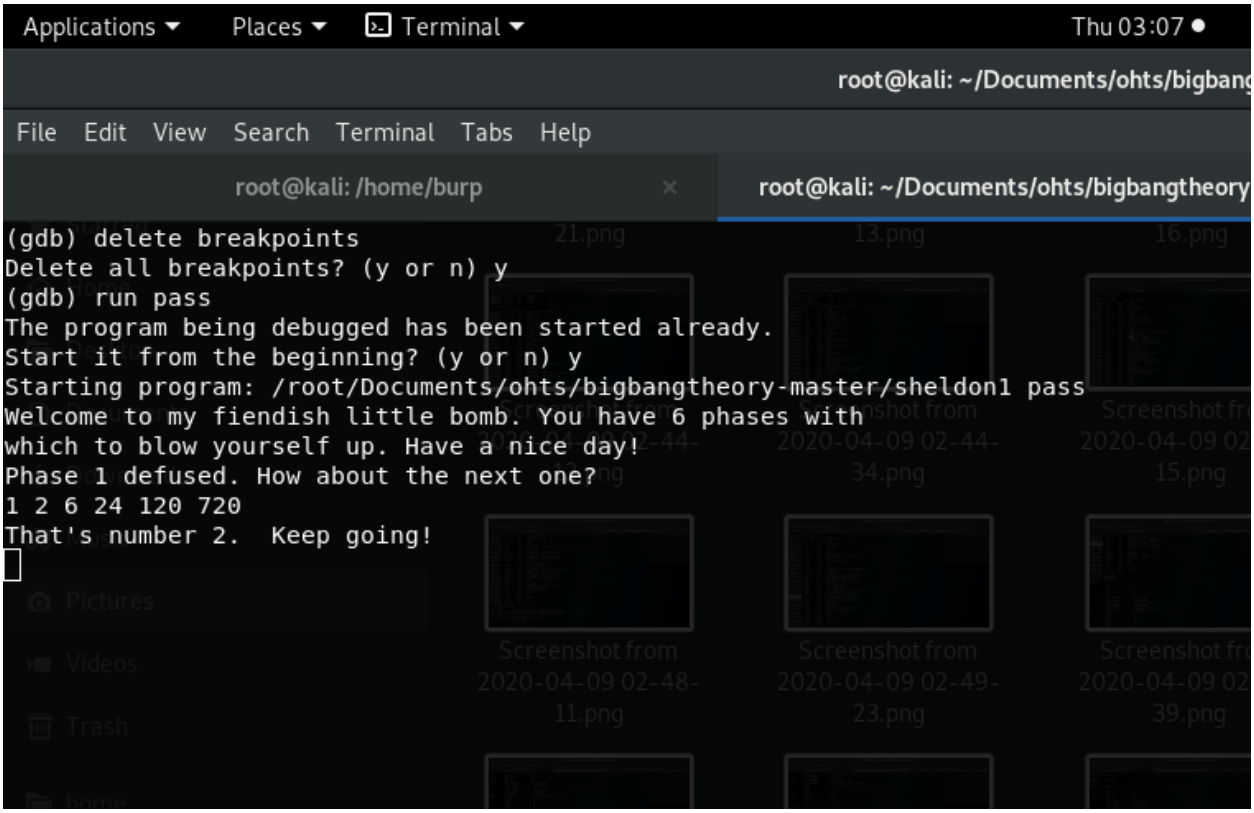

Figure 12: Testing the program with the found values

There we go! Phase_2 is diffused [Figure 12].