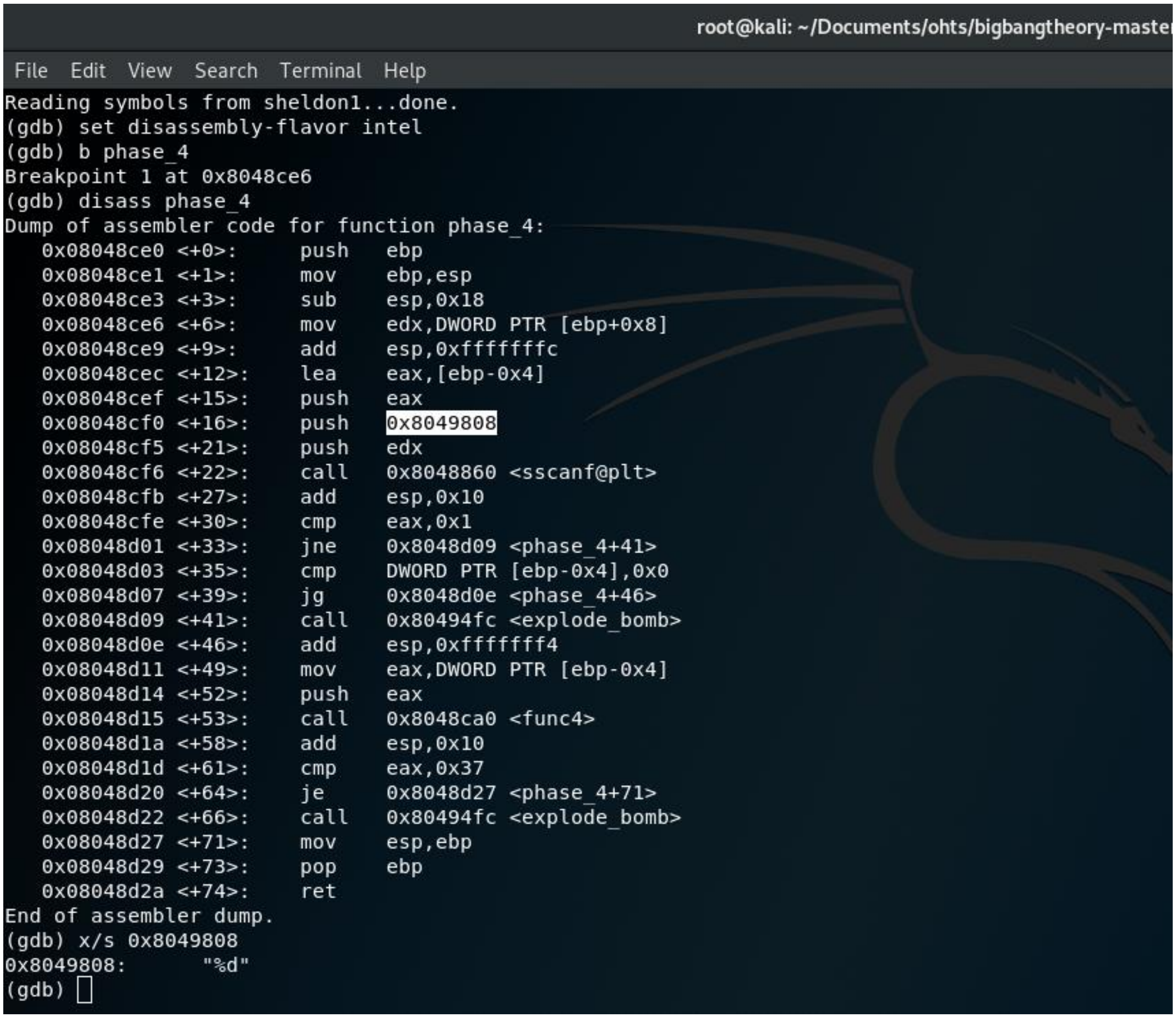
Sheldon Phase 4- Writeup

Objective of this report is to introduce the used approach to find the passphrase for the phase_4 of the Sheldon_1 binary file.

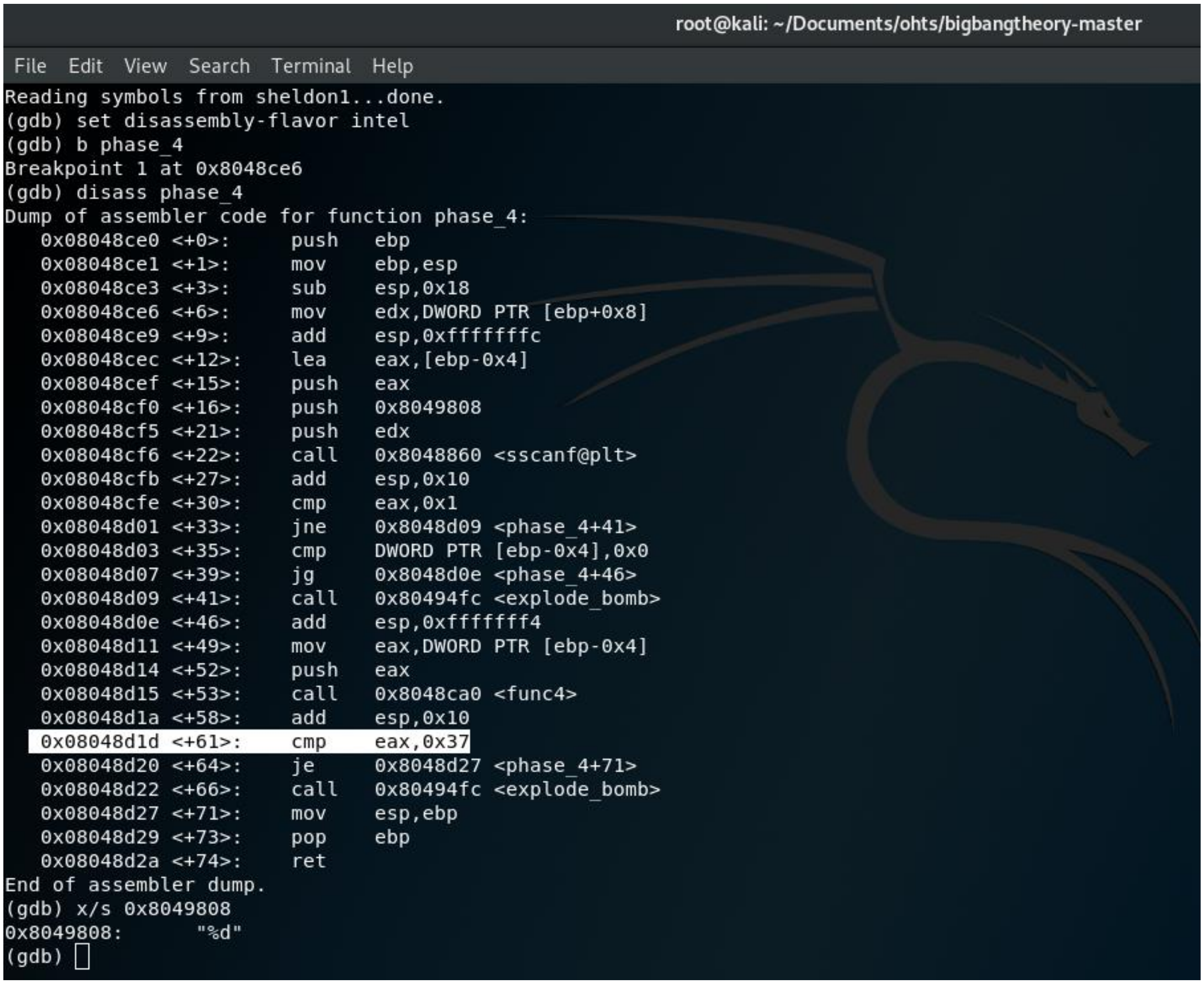
Since we already know how the program works, let’s start with disassembling the phase_4 to analyze its functionality. Also, notice that the ‘b phase_4’ command was used to set a breakpoint at phase_4 from the beginning [Figure 1].



```
Reading symbols from sheldon1...done.
(gdb) set disassembly-flavor intel
(gdb) b phase_4
Breakpoint 1 at 0x8048ce6
(gdb) disass phase_4
Dump of assembler code for function phase_4:
   0x08048ce0 <+0>:     push   ebp
   0x08048ce1 <+1>:     mov    ebp,esp
   0x08048ce3 <+3>:     sub    esp,0x18
   0x08048ce6 <+6>:     mov    edx,DWORD PTR [ebp+0x8]
   0x08048ce9 <+9>:     add    esp,0xfffffffc
   0x08048cec <+12>:    lea    eax,[ebp-0x4]
   0x08048cef <+15>:    push   eax
   0x08048cf0 <+16>:    push   0x8049808
   0x08048cf5 <+21>:    push   edx
   0x08048cf6 <+22>:    call   0x8048860 <sscanf@plt>
   0x08048cfb <+27>:    add    esp,0x10
   0x08048cfe <+30>:    cmp    eax,0x1
   0x08048d01 <+33>:    jne    0x8048d09 <phase_4+41>
   0x08048d03 <+35>:    cmp    DWORD PTR [ebp-0x4],0x0
   0x08048d07 <+39>:    jg     0x8048d0e <phase_4+46>
   0x08048d09 <+41>:    call   0x80494fc <explode_bomb>
   0x08048d0e <+46>:    add    esp,0xfffffff4
   0x08048d11 <+49>:    mov    eax,DWORD PTR [ebp-0x4]
   0x08048d14 <+52>:    push   eax
   0x08048d15 <+53>:    call   0x8048ca0 <func4>
   0x08048d1a <+58>:    add    esp,0x10
   0x08048d1d <+61>:    cmp    eax,0x37
   0x08048d20 <+64>:    je     0x8048d27 <phase_4+71>
   0x08048d22 <+66>:    call   0x80494fc <explode_bomb>
   0x08048d27 <+71>:    mov    esp,ebp
   0x08048d29 <+73>:    pop    ebp
   0x08048d2a <+74>:    ret
End of assembler dump.
(gdb) x/s 0x8049808
0x8049808:      "%d"
(gdb) 
```

Figure 1: Identification of the input pattern

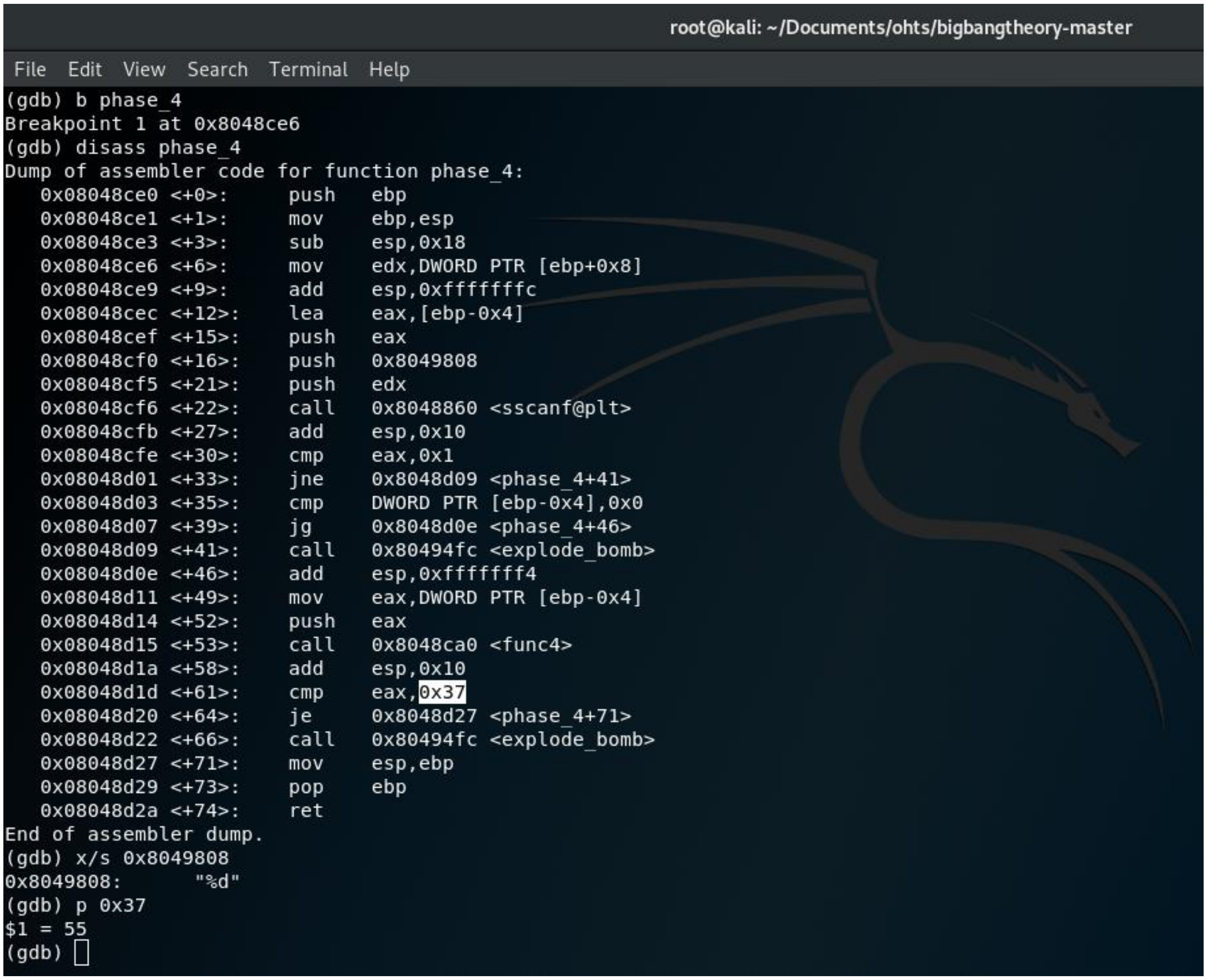
We can see that a value is being pushed to the stack before calling the scanf function. If we check that address it is possible to see the input pattern for the phase_4 as an integer [Figure 1].


root@kali: ~/Documents/ohts/bigbangtheory-master
File Edit View Search Terminal Help

```
Reading symbols from sheldon1...done.
(gdb) set disassembly-flavor intel
(gdb) b phase_4
Breakpoint 1 at 0x8048ce6
(gdb) disass phase_4
Dump of assembler code for function phase_4:
   0x08048ce0 <+0>:     push   ebp
   0x08048ce1 <+1>:     mov    ebp,esp
   0x08048ce3 <+3>:     sub    esp,0x18
   0x08048ce6 <+6>:     mov    edx,DWORD PTR [ebp+0x8]
   0x08048ce9 <+9>:     add    esp,0xfffffffc
   0x08048cec <+12>:    lea    eax,[ebp-0x4]
   0x08048cef <+15>:    push   eax
   0x08048cf0 <+16>:    push   0x8049808
   0x08048cf5 <+21>:    push   edx
   0x08048cf6 <+22>:    call   0x8048860 <sscanf@plt>
   0x08048cfb <+27>:    add    esp,0x10
   0x08048cfe <+30>:    cmp    eax,0x1
   0x08048d01 <+33>:    jne    0x8048d09 <phase_4+41>
   0x08048d03 <+35>:    cmp    DWORD PTR [ebp-0x4],0x0
   0x08048d07 <+39>:    jg     0x8048d0e <phase_4+46>
   0x08048d09 <+41>:    call   0x80494fc <explode_bomb>
   0x08048d0e <+46>:    add    esp,0xfffffff4
   0x08048d11 <+49>:    mov    eax,DWORD PTR [ebp-0x4]
   0x08048d14 <+52>:    push   eax
   0x08048d15 <+53>:    call   0x8048ca0 <func4>
   0x08048d1a <+58>:    add    esp,0x10
   0x08048d1d <+61>:    cmp    eax,0x37
   0x08048d20 <+64>:    je     0x8048d27 <phase_4+71>
   0x08048d22 <+66>:    call   0x80494fc <explode_bomb>
   0x08048d27 <+71>:    mov    esp,ebp
   0x08048d29 <+73>:    pop    ebp
   0x08048d2a <+74>:    ret
End of assembler dump.
(gdb) x/s 0x8049808
0x8049808:      "%d"
(gdb) 
```

Figure 2: Finding the comparison to satisfy

Also, from the disassembled phase_4 code, we can identify a comparison at the line 61 and it’s happening before the execution of the explode_bomb function. This is interesting and from this finding, we can think that this comparison is the key point in this phase. Let’s print the value in decimal to check what’s the value that satisfies the phase_4 to be successful [Figure 3].

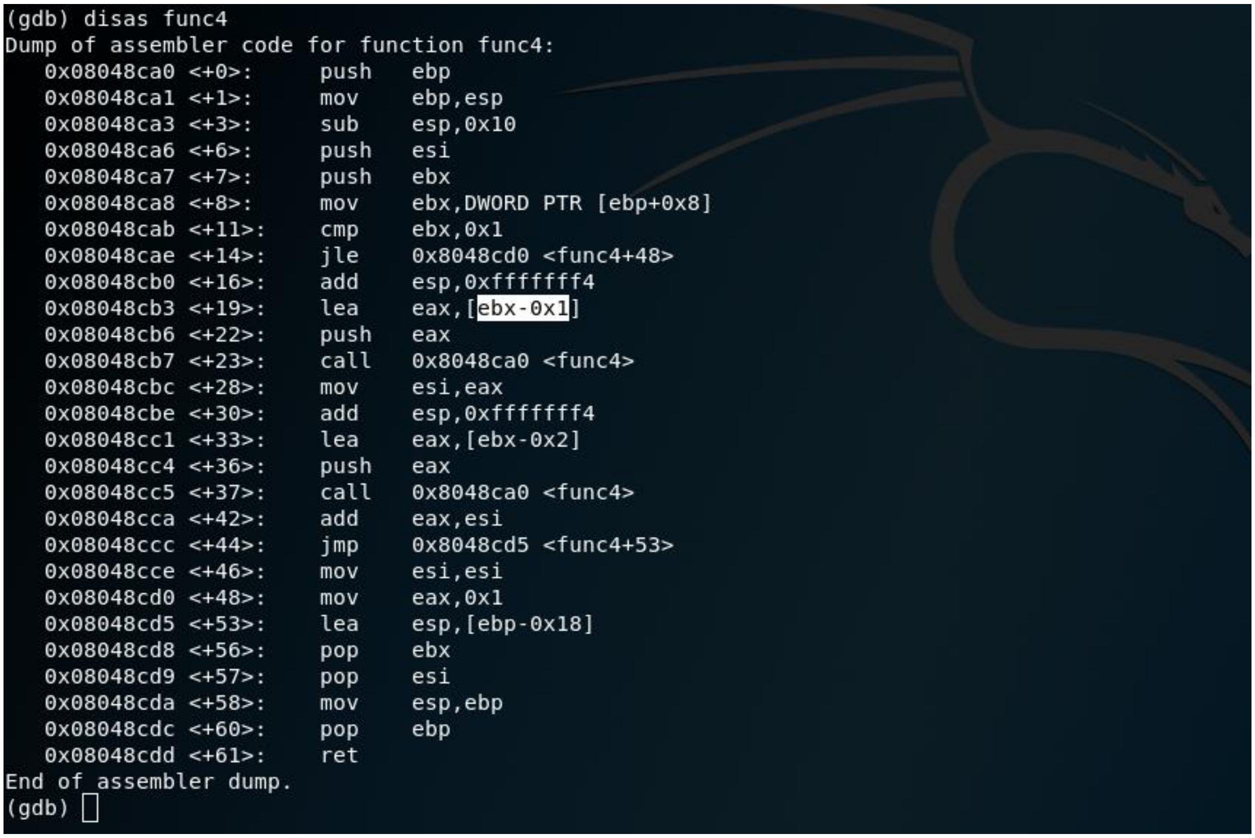


root@kali: ~/Documents/ohts/bigbangtheory-master
File Edit View Search Terminal Help

```
(gdb) b phase_4
Breakpoint 1 at 0x8048ce6
(gdb) disass phase_4
Dump of assembler code for function phase_4:
   0x08048ce0 <+0>:     push   ebp
   0x08048ce1 <+1>:     mov    ebp,esp
   0x08048ce3 <+3>:     sub    esp,0x18
   0x08048ce6 <+6>:     mov    edx,DWORD PTR [ebp+0x8]
   0x08048ce9 <+9>:     add    esp,0xfffffffc
   0x08048cec <+12>:    lea    eax,[ebp-0x4]
   0x08048cef <+15>:    push   eax
   0x08048cf0 <+16>:    push   0x8049808
   0x08048cf5 <+21>:    push   edx
   0x08048cf6 <+22>:    call   0x8048860 <sscanf@plt>
   0x08048cfb <+27>:    add    esp,0x10
   0x08048cfe <+30>:    cmp    eax,0x1
   0x08048d01 <+33>:    jne    0x8048d09 <phase_4+41>
   0x08048d03 <+35>:    cmp    DWORD PTR [ebp-0x4],0x0
   0x08048d07 <+39>:    jg     0x8048d0e <phase_4+46>
   0x08048d09 <+41>:    call   0x80494fc <explode_bomb>
   0x08048d0e <+46>:    add    esp,0xfffffff4
   0x08048d11 <+49>:    mov    eax,DWORD PTR [ebp-0x4]
   0x08048d14 <+52>:    push   eax
   0x08048d15 <+53>:    call   0x8048ca0 <func4>
   0x08048d1a <+58>:    add    esp,0x10
   0x08048d1d <+61>:    cmp    eax,0x37
   0x08048d20 <+64>:    je     0x8048d27 <phase_4+71>
   0x08048d22 <+66>:    call   0x80494fc <explode_bomb>
   0x08048d27 <+71>:    mov    esp,ebp
   0x08048d29 <+73>:    pop    ebp
   0x08048d2a <+74>:    ret
End of assembler dump.
(gdb) x/s 0x8049808
0x8049808:      "%d"
(gdb) p 0x37
$1 = 55
(gdb) 
```

Figure 3: Printing the value that must be met

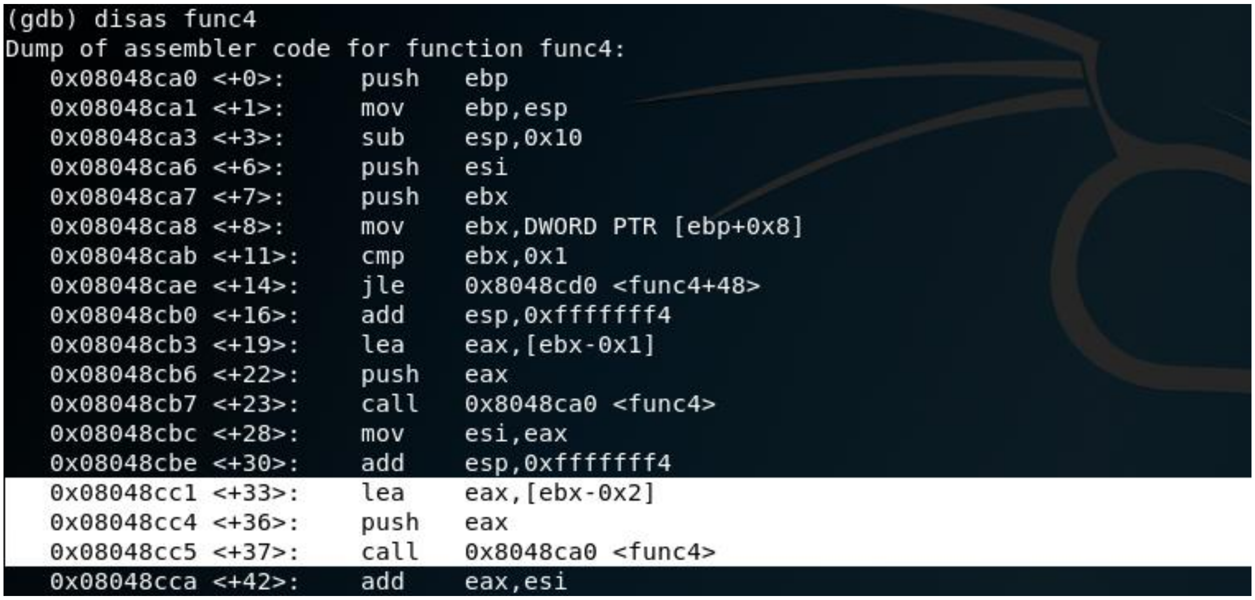
We found the required value as 55 by using the print command [Figure 3]. Now we have a question! Is this the value that we have to enter as the input? You can test the phase_4 by using the value 55 and it will get failed. So, how’s the value getting generated? If we look at the line 53, we can see there a call for a function before this comparison [Figure 3]. Let’s find out what’s inside the ‘func4’ function.



```
(gdb) disas func4
Dump of assembler code for function func4:
   0x08048ca0 <+0>:     push   ebp
   0x08048ca1 <+1>:     mov    ebp,esp
   0x08048ca3 <+3>:     sub    esp,0x10
   0x08048ca6 <+6>:     push   esi
   0x08048ca7 <+7>:     push   ebx
   0x08048ca8 <+8>:     mov    ebx,DWORD PTR [ebp+0x8]
   0x08048cab <+11>:    cmp    ebx,0x1
   0x08048cae <+14>:    jle    0x8048cd0 <func4+48>
   0x08048cb0 <+16>:    add    esp,0xfffffff4
   0x08048cb3 <+19>:    lea    eax,[ebx-0x1]
   0x08048cb6 <+22>:    push   eax
   0x08048cb7 <+23>:    call   0x8048ca0 <func4>
   0x08048cbc <+28>:    mov    esi,eax
   0x08048cbe <+30>:    add    esp,0xfffffff4
   0x08048cc1 <+33>:    lea    eax,[ebx-0x2]
   0x08048cc4 <+36>:    push   eax
   0x08048cc5 <+37>:    call   0x8048ca0 <func4>
   0x08048cca <+42>:    add    eax,esi
   0x08048ccc <+44>:    jmp    0x8048cd5 <func4+53>
   0x08048cce <+46>:    mov    esi,esi
   0x08048cd0 <+48>:    mov    eax,0x1
   0x08048cd5 <+53>:    lea    esp,[ebp-0x18]
   0x08048cd8 <+56>:    pop    ebx
   0x08048cd9 <+57>:    pop    esi
   0x08048cda <+58>:    mov    esp,ebp
   0x08048cdc <+60>:    pop    ebp
   0x08048cdd <+61>:    ret
End of assembler dump.
(gdb) 
```

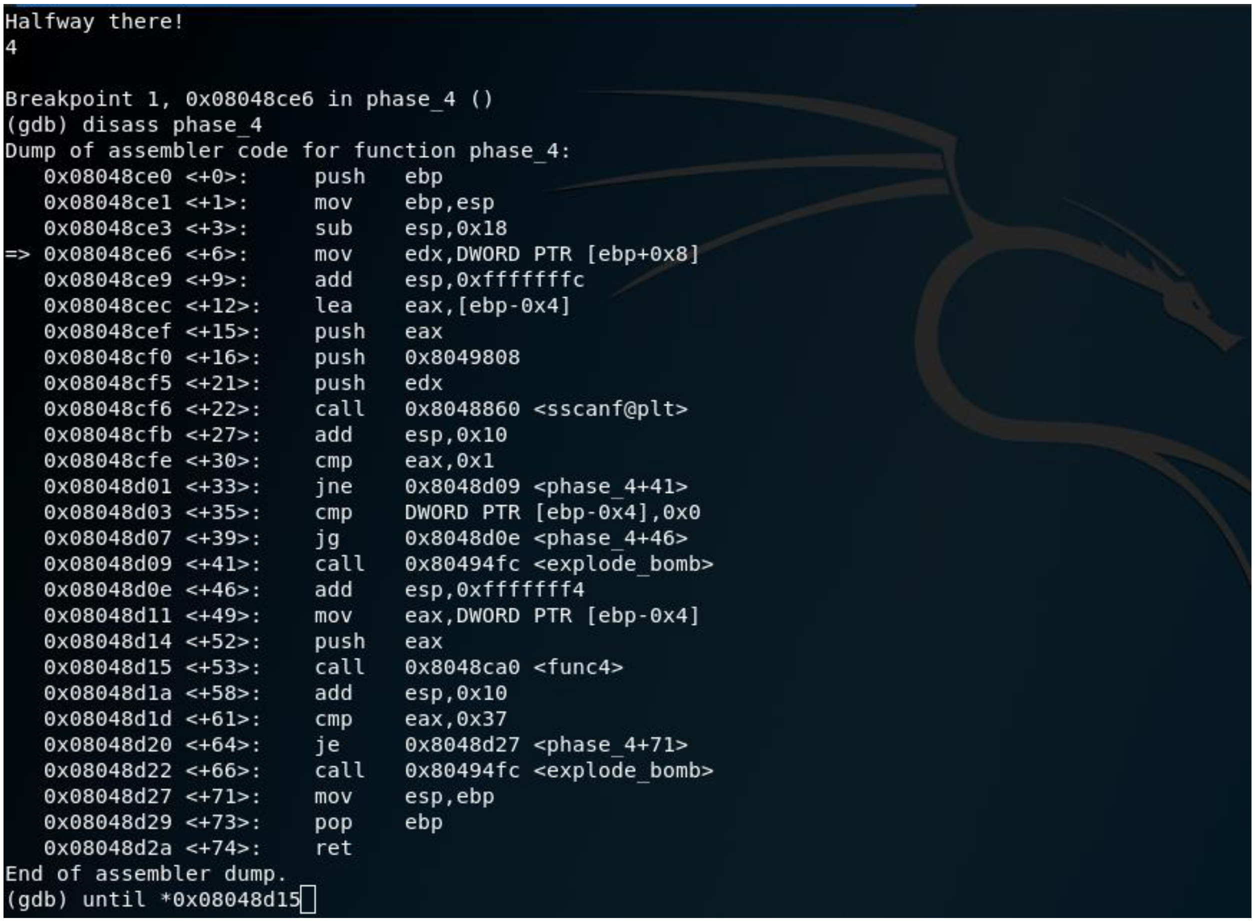
Figure 4 : Identification of the deduction part 1

After disassembling the func4, we can see that the inserted value is going through several transformations. In overall, the inserted value is getting inserted to the eax register, the ebx register holds the values by deducting 1 [Figure 4] from the input at line number 19 and by deducting 2 [Figure 5] at line number 33. Also, we can see that the func4 is getting called two times which transforms it to a recursive function. At the end of the code the function returns a value [Figure 4] and now we know that this value must be 55 to successfully execute the phase.



```
(gdb) disas func4
Dump of assembler code for function func4:
   0x08048ca0 <+0>:     push   ebp
   0x08048ca1 <+1>:     mov    ebp,esp
   0x08048ca3 <+3>:     sub    esp,0x10
   0x08048ca6 <+6>:     push   esi
   0x08048ca7 <+7>:     push   ebx
   0x08048ca8 <+8>:     mov    ebx,DWORD PTR [ebp+0x8]
   0x08048cab <+11>:    cmp    ebx,0x1
   0x08048cae <+14>:    jle    0x8048cd0 <func4+48>
   0x08048cb0 <+16>:    add    esp,0xfffffff4
   0x08048cb3 <+19>:    lea    eax,[ebx-0x1]
   0x08048cb6 <+22>:    push   eax
   0x08048cb7 <+23>:    call   0x8048ca0 <func4>
   0x08048cbc <+28>:    mov    esi,eax
   0x08048cbe <+30>:    add    esp,0xfffffff4
   0x08048cc1 <+33>:    lea    eax,[ebx-0x2]
   0x08048cc4 <+36>:    push   eax
   0x08048cc5 <+37>:    call   0x8048ca0 <func4>
   0x08048cca <+42>:    add    eax,esi
```

Figure 5: Identification of the deduction part 2

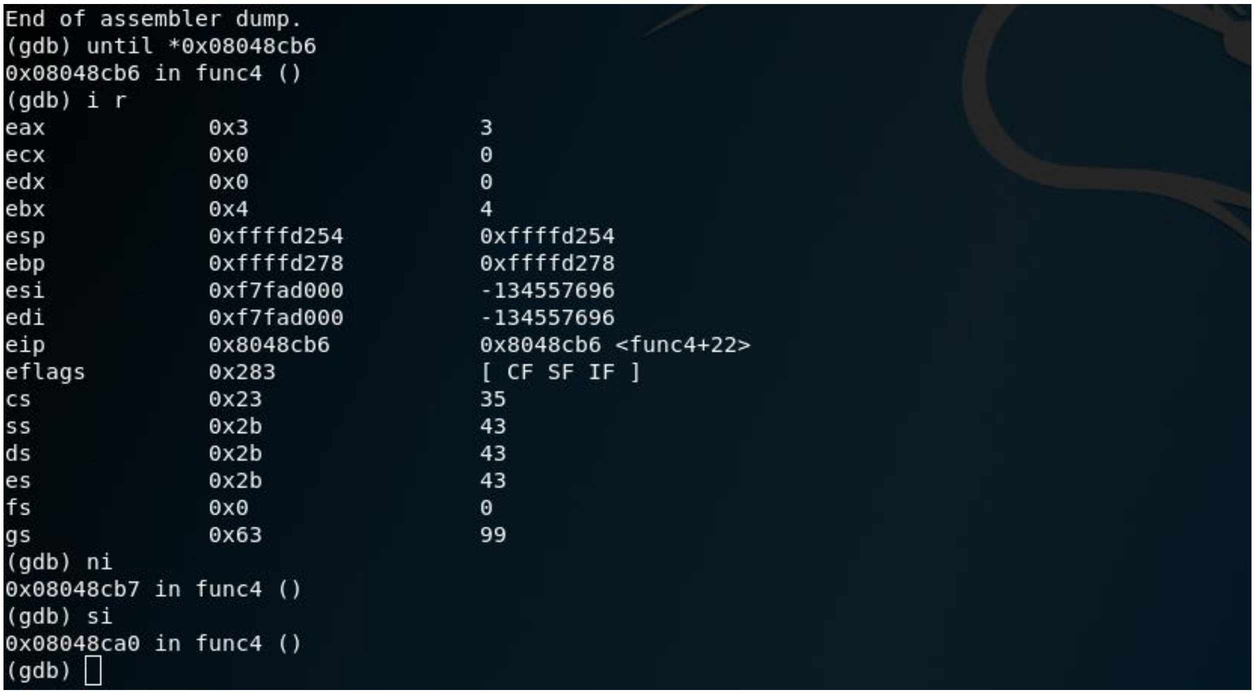


```
Halfway there!
4

Breakpoint 1, 0x08048ce6 in phase_4 ()
(gdb) disass phase_4
Dump of assembler code for function phase_4:
   0x08048ce0 <+0>:     push   ebp
   0x08048ce1 <+1>:     mov    ebp,esp
   0x08048ce3 <+3>:     sub    esp,0x18
=> 0x08048ce6 <+6>:     mov    edx,DWORD PTR [ebp+0x8]
   0x08048ce9 <+9>:     add    esp,0xfffffffc
   0x08048cec <+12>:    lea    eax,[ebp-0x4]
   0x08048cef <+15>:    push   eax
   0x08048cf0 <+16>:    push   0x8049808
   0x08048cf5 <+21>:    push   edx
   0x08048cf6 <+22>:    call   0x8048860 <sscanf@plt>
   0x08048cfb <+27>:    add    esp,0x10
   0x08048cfe <+30>:    cmp    eax,0x1
   0x08048d01 <+33>:    jne    0x8048d09 <phase_4+41>
   0x08048d03 <+35>:    cmp    DWORD PTR [ebp-0x4],0x0
   0x08048d07 <+39>:    jg     0x8048d0e <phase_4+46>
   0x08048d09 <+41>:    call   0x80494fc <explode_bomb>
   0x08048d0e <+46>:    add    esp,0xfffffff4
   0x08048d11 <+49>:    mov    eax,DWORD PTR [ebp-0x4]
   0x08048d14 <+52>:    push   eax
   0x08048d15 <+53>:    call   0x8048ca0 <func4>
   0x08048d1a <+58>:    add    esp,0x10
   0x08048d1d <+61>:    cmp    eax,0x37
   0x08048d20 <+64>:    je     0x8048d27 <phase_4+71>
   0x08048d22 <+66>:    call   0x80494fc <explode_bomb>
   0x08048d27 <+71>:    mov    esp,ebp
   0x08048d29 <+73>:    pop    ebp
   0x08048d2a <+74>:    ret
End of assembler dump.
(gdb) until *0x08048d15
```

Figure 6 : Testing the program with random values

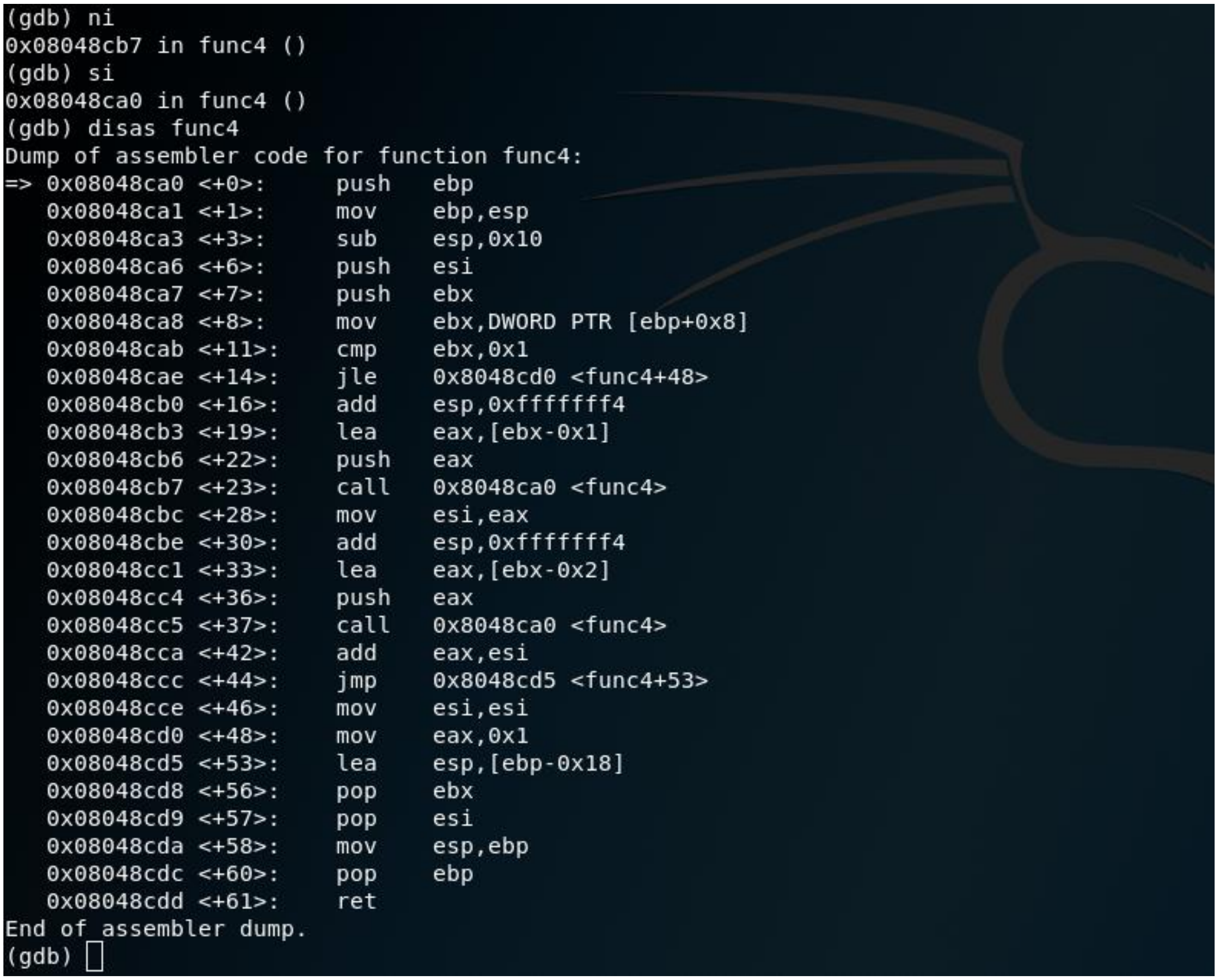
Let’s insert a random value to understand the value generating process. I have inserted number 4 as the input and used until command to jump into the func4 address because that’s where the magic happens [Figure 6].



```
End of assembler dump.
(gdb) until *0x08048cb6
0x08048cb6 in func4 ()
(gdb) i r
eax            0x3                 3
ecx            0x0                 0
edx            0x0                 0
ebx            0x4                 4
esp            0xffffd254          0xffffd254
ebp            0xffffd278          0xffffd278
esi            0xf7fad000          -134557696
edi            0xf7fad000          -134557696
eip            0x8048cb6           0x8048cb6 <func4+22>
eflags         0x283               [ CF SF IF ]
cs             0x23                35
ss             0x2b                43
ds             0x2b                43
es             0x2b                43
fs             0x0                 0
gs             0x63                99
(gdb) ni
0x08048cb7 in func4 ()
(gdb) si
0x08048ca0 in func4 ()
(gdb) 
```

Figure 7: Analyzing the register values

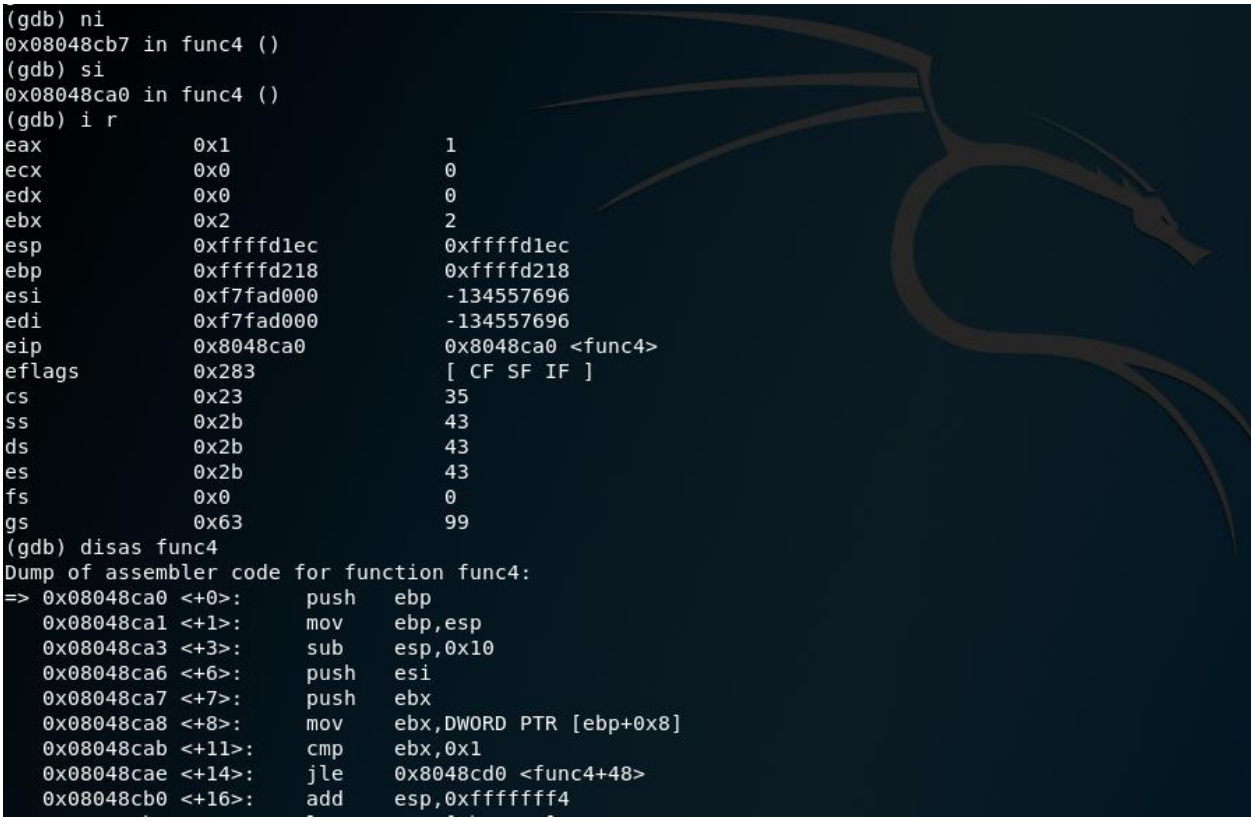
After disassembling the func4 and jumping to the middle section (jumped address is in the figure) of the func4, we can check the register values to see the changes. In the first iteration we can see the eax got reduced by 1 and the previous eax value (which is the inserted value ‘4’) gets moved to the ebx register [Figure 7].



```
(gdb) ni
0x08048cb7 in func4 ()
(gdb) si
0x08048ca0 in func4 ()
(gdb) disas func4
Dump of assembler code for function func4:
=> 0x08048ca0 <+0>:     push   ebp
   0x08048ca1 <+1>:     mov    ebp,esp
   0x08048ca3 <+3>:     sub    esp,0x10
   0x08048ca6 <+6>:     push   esi
   0x08048ca7 <+7>:     push   ebx
   0x08048ca8 <+8>:     mov    ebx,DWORD PTR [ebp+0x8]
   0x08048cab <+11>:    cmp    ebx,0x1
   0x08048cae <+14>:    jle    0x8048cd0 <func4+48>
   0x08048cb0 <+16>:    add    esp,0xfffffff4
   0x08048cb3 <+19>:    lea    eax,[ebx-0x1]
   0x08048cb6 <+22>:    push   eax
   0x08048cb7 <+23>:    call   0x8048ca0 <func4>
   0x08048cbc <+28>:    mov    esi,eax
   0x08048cbe <+30>:    add    esp,0xfffffff4
   0x08048cc1 <+33>:    lea    eax,[ebx-0x2]
   0x08048cc4 <+36>:    push   eax
   0x08048cc5 <+37>:    call   0x8048ca0 <func4>
   0x08048cca <+42>:    add    eax,esi
   0x08048ccc <+44>:    jmp    0x8048cd5 <func4+53>
   0x08048cce <+46>:    mov    esi,esi
   0x08048cd0 <+48>:    mov    eax,0x1
   0x08048cd5 <+53>:    lea    esp,[ebp-0x18]
   0x08048cd8 <+56>:    pop    ebx
   0x08048cd9 <+57>:    pop    esi
   0x08048cda <+58>:    mov    esp,ebp
   0x08048cdc <+60>:    pop    ebp
   0x08048cdd <+61>:    ret
End of assembler dump.
(gdb) 
```

Figure 8: Disassembled code of func4

By using the si,ni and i r commands repitively we can see that the the first func4 at line 23 moving to the next line 28 once the ebx value hits as 1. Execution will continue from the 28th line and this time it will reduce 2 from the ebx register and do the same exact process to jumps to the 42 line. This is where the the eax and esi values gets added to produce the value 55 to return as the output of func4 [Figure 8].



```
(gdb) ni
0x08048cb7 in func4 ()
(gdb) si
0x08048ca0 in func4 ()
(gdb) i r
eax            0x1                 1
ecx            0x0                 0
edx            0x0                 0
ebx            0x2                 2
esp            0xffffd1ec          0xffffd1ec
ebp            0xffffd218          0xffffd218
esi            0xf7fad000          -134557696
edi            0xf7fad000          -134557696
eip            0x8048ca0           0x8048ca0 <func4>
eflags         0x283               [ CF SF IF ]
cs             0x23                35
ss             0x2b                43
ds             0x2b                43
es             0x2b                43
fs             0x0                 0
gs             0x63                99
(gdb) disas func4
Dump of assembler code for function func4:
=> 0x08048ca0 <+0>:     push   ebp
   0x08048ca1 <+1>:     mov    ebp,esp
   0x08048ca3 <+3>:     sub    esp,0x10
   0x08048ca6 <+6>:     push   esi
   0x08048ca7 <+7>:     push   ebx
   0x08048ca8 <+8>:     mov    ebx,DWORD PTR [ebp+0x8]
   0x08048cab <+11>:    cmp    ebx,0x1
   0x08048cae <+14>:    jle    0x8048cd0 <func4+48>
   0x08048cb0 <+16>:    add    esp,0xfffffff4
```

Figure 9 : Checking register values recursively

Let’s analyze what we have found. Below are the register value changes [Figure 9] (only for the first 3 iterations) from first call for the func4 at line 23.

First Iteration – eax → 3

ebx → 4

Second Iteration – eax → 2

ebx → 3

Third Iteration – eax → 1

ebx → 2

The same process happens in the line 37 but this time it starts by deducting 2 from the inserted value.

I’ve inserted 4 as the input value and the compared value (final output of the func4) was 5 against the 55 value. Also, I’ve mentioned that we can notice the function ‘func4’ getting called two times recursively by deducting 1 and 2 in an ordered manner. So, if we put the findings into a mathematical equation it should be like this.

Consider n as the input value,

F(n) = f (n-1) + f (n-2)

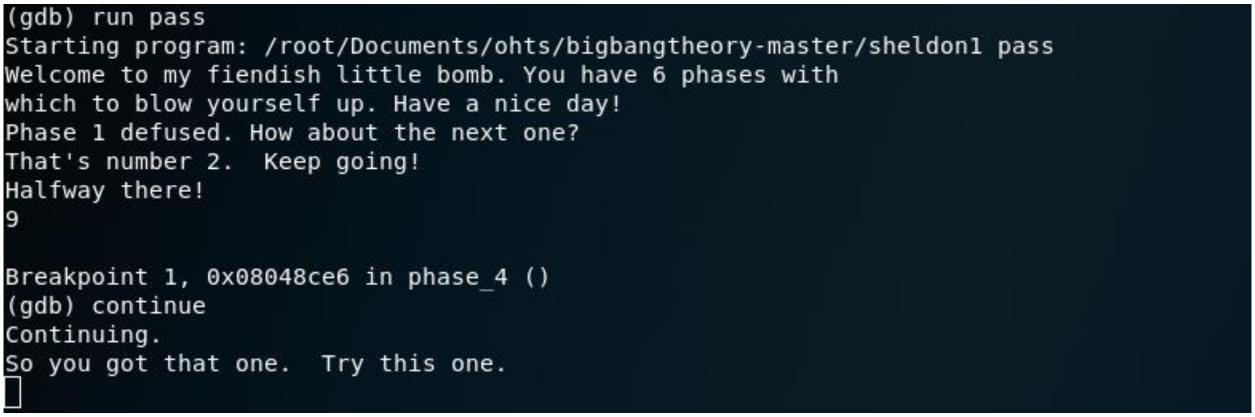
Interesting!!! This function looks like something we have learned in our school days. Yes, It's the Fibonacci number formula.

Fibonacci sequence,

1,1,2,3,5,8,13,21,34,55….

And there we go we have a 55 there. But when we inserted 4 as the input, we got the final func4 output as 5. That's because the final eax + esi value addition happened by using number two and three which has the positions three and four in the sequence.

This implies that to create 55, we must insert number 9 not 10 as the input. Only then the final addition happens with number 21 and 34. So let's test the program with the found value 9 [Figure 10].



```
(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?
That's number 2.  Keep going!
Halfway there!
9

Breakpoint 1, 0x08048ce6 in phase_4 ()
(gdb) continue
Continuing.
So you got that one.  Try this one.
█
```

Figure 10: Testing the program

There we go! Phase_4 diffused!