

Big Bang Theory-master

Sheldon 1

Introduction

First, download big bang theory from git hub and extract it.

There are 4 files.

- learnord_win.exe
- README.md
- sheldon1
- sheldon2

Then launching the sheldon1 file under gdb

```
root@kali:~/Downloads/bigbangtheory-master# gdb sheldon1
GNU gdb (Debian 7.12-6) 7.12.0.20161007-git
Copyright (C) 2016 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:
<http://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 sheldon1...done.
```

Then we can find 6 phases in that assembly code.

```
--Type <return> to continue, or q <return> to quit--Type
0x08048a3d <+141>: call 0x8048810 <printf@plt>
0x08048a42 <+146>: add esp,0xffffffff
0x08048a45 <+149>: push 0x80496a0
0x08048a4a <+154>: call 0x8048810 <printf@plt>
0x08048a4f <+159>: add esp,0x20
0x08048a52 <+162>: call 0x80491fc <read_line>
0x08048a57 <+167>: add esp,0xffffffff
0x08048a5a <+170>: push eax
0x08048a5b <+171>: call 0x8048b20 <phase_1>
0x08048a60 <+176>: call 0x804952c <phase_defused>
0x08048a65 <+181>: add esp,0xffffffff
0x08048a68 <+184>: push 0x80496e0
0x08048a6d <+189>: call 0x8048810 <printf@plt>
0x08048a72 <+194>: add esp,0x20
0x08048a75 <+197>: call 0x80491fc <read_line>
0x08048a7a <+202>: add esp,0xffffffff
0x08048a7d <+205>: push eax
0x08048a7e <+206>: call 0x8048b48 <phase_2>
0x08048a83 <+211>: call 0x804952c <phase_defused>
0x08048a88 <+216>: add esp,0xffffffff
0x08048a8b <+219>: push 0x8049720
0x08048a90 <+224>: call 0x8048810 <printf@plt>
0x08048a95 <+229>: add esp,0x20
--Type <return> to continue, or q <return> to quit--Type
0x08048a98 <+232>: call 0x80491fc <read_line>
0x08048a9d <+237>: add esp,0xffffffff
0x08048aa0 <+240>: push 0x0
0x08048aa1 <+241>: call 0x8048b98 <phase_3>
0x08048aa6 <+246>: call 0x804952c <phase_defused>
0x08048aab <+251>: add esp,0xffffffff
0x08048aae <+254>: push 0x804973f
0x08048ab3 <+259>: call 0x8048810 <printf@plt>
0x08048ab8 <+264>: add esp,0x20
0x08048abb <+267>: call 0x80491fc <read_line>
0x08048ac0 <+272>: add esp,0xffffffff
0x08048ac3 <+275>: push eax
0x08048ac4 <+276>: call 0x80496e0 <phase_4>
```

Now the challenge starts. You need to find the password to that phases.

Phase 01

First, run the sheldon1 file using **chmod +x sheldon1** command and **./sheldon1** command. Then open it using **gdb**.

```
kali@kali:~/Downloads/bigbangtheory-master$ chmod +x sheldon1
kali@kali:~/Downloads/bigbangtheory-master$ ./sheldon1
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
^CSo you think you can stop the bomb with ctrl-c, do you?
Well... OK. :-)
```

```
kali@kali:~/Downloads/bigbangtheory-master$ gdb sheldon1
GNU gdb (Debian 8.3.1-1) 8.3.1
Copyright (C) 2019 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:
<http://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 sheldon1...
```

Look for the functions inside sheldon1 using the command “info functions”. As we can see there are separate functions for each phase there.

```
(gdb) info functions
All defined functions:

File bomb.c:
36:   int main(int, char **);

Non-debugging symbols:
0x080486e0  _init
0x08048720  __register_frame_info@plt
0x08048730  close@plt
0x08048740  fprintf@plt
0x08048750  tmpfile@plt
0x08048760  getenv@plt
0x08048770  signal
0x08048770  signal@plt
0x08048780  fflush
0x08048780  fflush@plt
0x08048790  bcopy
0x08048790  bcopy@plt
0x080487a0  rewind
0x080487a0  rewind@plt
0x080487b0  system
0x080487b0  system@plt
0x080487c0  __deregister_frame_info
0x080487c0  __deregister_frame_info@plt
0x080487d0  fgets
0x080487d0  fgets@plt
0x080487e0  sleep
0x080487e0  sleep@plt
0x080487f0  __strtol_internal
0x080487f0  __strtol_internal@plt
0x08048800  __libc_start_main
0x08048800  __libc_start_main@plt
0x08048810  printf
0x08048810  printf@plt
0x08048820  fclose
0x08048820  fclose@plt
```

```
kali@kali: ~/Downloads/bigbangtheory-master

File  Actions  Edit  View  Help

0x08048850  exit
0x08048850  exit@plt
0x08048860  sscanf
0x08048860  sscanf@plt
0x08048870  connect
0x08048870  connect@plt
--Type <RET> for more, q to quit, c to continue without paging--c
0x08048880  fopen
0x08048880  fopen@plt
0x08048890  dup
0x08048890  dup@plt
0x080488a0  sprintf
0x080488a0  sprintf@plt
0x080488b0  socket
0x080488b0  socket@plt
0x080488c0  cuserid
0x080488c0  cuserid@plt
0x080488d0  strcpy
0x080488d0  strcpy@plt
0x080488e0  _start
0x08048910  __do_global_dtors_aux
0x08048964  fini_dummy
0x08048970  frame_dummy
0x08048998  init_dummy
0x08048b20  phase_1
0x08048b48  phase_2
0x08048b98  phase_3
0x08048ca0  func4
0x08048ce0  phase_4
0x08048d2c  phase_5
0x08048d98  phase_6
0x08048e94  fun7
0x08048ee8  secret_phase
0x08048f50  sig_handler
0x08048fb4  invalid_phase
0x08048fd8  read_six_numbers
0x08049018  string_length
0x08049030  strings_not_equal
0x0804908c  open_clientfd
0x08049160  initialize_bomb
0x0804917c  blank_line
0x080491b0  skip
0x080491fc  read_line
0x080492c0  send_msg
0x080494fc  explode_bomb
0x0804952c  phase_defused
0x080495b0  __do_global_ctors_aux
```

Then view the assembly code of the main function using **disassemble main** command.

```
kali@kali: ~/Downloads/bigbangtheory-master
File Actions Edit View Help
0x080495e4 _fini
(gdb) disassemble main
Dump of assembler code for function main:
0x080489b0 <+0>: push %ebp
0x080489b1 <+1>: mov %esp,%ebp
0x080489b3 <+3>: sub $0x14,%esp
0x080489b6 <+6>: push %ebx
0x080489b7 <+7>: mov 0x8(%ebp),%eax
0x080489ba <+10>: mov 0xc(%ebp),%ebx
0x080489bd <+13>: cmp $0x1,%eax
0x080489c0 <+16>: jne 0x80489d0 <main+32>
0x080489c2 <+18>: mov 0x804b648,%eax
0x080489c7 <+23>: mov %eax,0x804b664
0x080489cc <+28>: jmp 0x8048a30 <main+128>
0x080489ce <+30>: mov %esi,%esi
0x080489d0 <+32>: cmp $0x2,%eax
0x080489d3 <+35>: jne 0x8048a10 <main+96>
0x080489d5 <+37>: add $0xffffffff,%esp
0x080489d8 <+40>: push $0x8049620
0x080489dd <+45>: mov 0x4(%ebx),%eax
0x080489e0 <+48>: push %eax
0x080489e1 <+49>: call 0x8048880 <fopen@plt>
0x080489e6 <+54>: mov %eax,0x804b664
0x080489eb <+59>: add $0x10,%esp
0x080489ee <+62>: test %eax,%eax
0x080489f0 <+64>: jne 0x8048a30 <main+128>
0x080489f2 <+66>: add $0xffffffff,%esp
0x080489f5 <+69>: mov 0x4(%ebx),%eax
0x080489f8 <+72>: push %eax
0x080489f9 <+73>: mov (%ebx),%eax
0x080489fb <+75>: push %eax
0x080489fc <+76>: push $0x8049622
0x08048a01 <+81>: call 0x8048810 <printf@plt>
0x08048a06 <+86>: add $0xffffffff,%esp
0x08048a09 <+89>: push $0x8
0x08048a0b <+91>: call 0x8048850 <exit@plt>
0x08048a10 <+96>: add $0xffffffff,%esp
0x08048a13 <+99>: mov (%ebx),%eax
0x08048a15 <+101>: push %eax
0x08048a16 <+102>: push $0x804963f
0x08048a1b <+107>: call 0x8048810 <printf@plt>
0x08048a20 <+112>: add $0xffffffff,%esp
0x08048a23 <+115>: push $0x8
0x08048a25 <+117>: call 0x8048850 <exit@plt>
0x08048a2a <+122>: lea 0x0(%esi),%esi
0x08048a30 <+128>: call 0x8049160 <initialize_bomb>
0x08048a35 <+133>: add $0xffffffff,%esp
```

View the assembly code of the phase_1 using **disassemble phase_1** command.

```
kali@kali: ~/Downloads/bigbangtheory-master
File Actions Edit View Help
End of assembler dump.
(gdb) disass phase_1
Dump of assembler code for function phase_1:
0x08048b20 <+0>: push ebp
0x08048b21 <+1>: mov ebp,esp
0x08048b23 <+3>: sub esp,0x8
0x08048b26 <+6>: mov eax,DWORD PTR [ebp+0x8]
0x08048b29 <+9>: add esp,0xffffffff
0x08048b2c <+12>: push 0x80497c0
0x08048b31 <+17>: push eax
0x08048b32 <+18>: call 0x8049030 <strings_not_equal>
0x08048b37 <+23>: add esp,0x10
0x08048b3a <+26>: test eax,eax
0x08048b3c <+28>: je 0x8048b43 <phase_1+35>
0x08048b3e <+30>: call 0x80494fc <explode_bomb>
0x08048b43 <+35>: mov esp,ebp
0x08048b45 <+37>: pop ebp
0x08048b46 <+38>: ret
End of assembler dump.
```

In the first phase, there's a simple string comparison in place. Very straight-forward. The first string being compared is our input. The second string is the password for phase_1.

```
(gdb) x /x $ebp+0x8
No registers.
(gdb) run
Starting program: /home/kali/Downloads/bigbangtheory-master/sheldon1
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
^C
Program received signal SIGINT, Interrupt.
0xf7fd3b59 in __kernel_vsyscall ()
(gdb) x /x $ebp+0x8
0xffffd1c0: 0xf7fb65c0
(gdb) x /s 0x0804b680
0x804b680 <input_strings>: ""
(gdb) x /s 0x080497c0
0x080497c0: "Public speaking is very easy."
```

Password for phase 01: **Public speaking is very easy.**

```
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/kali/Downloads/bigbangtheory-master/sheldon1
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Public speaking is very easy.
Phase 1 defused. How about the next one?
█
```


Phase 02

View the assembly code of the phase_2 using **disassemble phase_2** command.

```
(gdb) disass phase_2
Dump of assembler code for function phase_2:
0x08048b48 <+0>:  push    %ebp
0x08048b49 <+1>:  mov     %esp,%ebp
0x08048b4b <+3>:  sub     $0x20,%esp
0x08048b4e <+6>:  push    %esi
0x08048b4f <+7>:  push    %ebx
0x08048b50 <+8>:  mov     0x8(%ebp),%edx
0x08048b53 <+11>: add     $0xffffffff,%esp
0x08048b56 <+14>: lea     -0x18(%ebp),%eax
0x08048b59 <+17>: push    %eax
0x08048b5a <+18>: push    %edx
0x08048b5b <+19>: call    0x8048fd8 <read_six_numbers>
0x08048b60 <+24>: add     $0x10,%esp
0x08048b63 <+27>: cmpl    $0x1,-0x18(%ebp)
0x08048b67 <+31>: je      0x8048b6e <phase_2+38>
0x08048b69 <+33>: call    0x80494fc <explode_bomb>
0x08048b6e <+38>: mov     $0x1,%ebx
0x08048b73 <+43>: lea     -0x18(%ebp),%esi
0x08048b76 <+46>: lea     0x1(%ebx),%eax
0x08048b79 <+49>: imul    -0x4(%esi,%ebx,4),%eax
0x08048b7e <+54>: cmp     %eax,(%esi,%ebx,4)
0x08048b81 <+57>: je      0x8048b88 <phase_2+64>
0x08048b83 <+59>: call    0x80494fc <explode_bomb>
0x08048b88 <+64>: inc     %ebx
0x08048b89 <+65>: cmp     $0x5,%ebx
0x08048b8c <+68>: jle     0x8048b76 <phase_2+46>
0x08048b8e <+70>: lea     -0x28(%ebp),%esp
0x08048b91 <+73>: pop     %ebx
0x08048b92 <+74>: pop     %esi
0x08048b93 <+75>: mov     %ebp,%esp
0x08048b95 <+77>: pop     %ebp
0x08048b96 <+78>: ret
End of assembler dump.
```

Using the **i r** command to see the contents of registers.

```
(gdb) i r
eax             0xfffffe00             -512
ecx             0x804c5b0             134530480
edx             0x400                 1024
ebx             0x0                   0
esp             0xfffffd150           0xfffffd150
ebp             0xfffffd1c8           0xfffffd1c8
esi             0xf7fb65c0            -134519360
edi             0x0                   0
eip             0xf7fd3b59            0xf7fd3b59 <__kernel_vsyscall+9>
eflags         0x206                 [ PF IF ]
cs              0x23                 35
ss              0x2b                 43
ds              0x2b                 43
es              0x2b                 43
fs              0x0                   0
gs              0x63                 99
```

In assembly language basics a few things should jump out at you right away. Before anything else happens, a number is getting compared to 1, and the bomb is going off this number isn't also 1. This means that the first integer in our secret phrase is undoubtedly 1.

```
(gdb) i r eax
eax             0x1                   1
(gdb)
```

Once we've arrived at the comparison statement, we can use the **i r** command to see the contents of our registers. **eax**, which is the register to which our value is being compared, is equal to 2. Therefore, the second integer in our passphrase should be 2.

The simplest way to solve this level completely is by continuing to step through the code, seeing what **eax** is equal to after each iteration. Next, you'll find 6, then 24, then 120 followed by 720. The assembly code is implementing the following algorithm:

$$v[0] = 1$$

$$v[i] = (i+i) * v[i-1]$$

Password for phase 02 : **1 2 6 24 120 720**

```
kali@kali:~/Downloads/bigbangtheory-master$ ./sheldon1
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Public speaking is very easy.
Phase 1 defused. How about the next one?
1 2 6 24 120 720
That's number 2. Keep going!
```

Phase 03

View the assembly code of the phase_3 using **disassemble phase_3** command.

```
kali@kali: ~/Downloads/bigbangtheory-master
File Actions Edit View Help
(gdb) disass phase_3
Dump of assembler code for function phase_3:
0x08048b98 <+0>: push %ebp
0x08048b99 <+1>: mov %esp,%ebp
0x08048b9b <+3>: sub $0x14,%esp
0x08048b9e <+6>: push %ebx
0x08048b9f <+7>: mov 0x8(%ebp),%edx
0x08048ba2 <+10>: add $0xffffffff,%esp
0x08048ba5 <+13>: lea -0x4(%ebp),%eax
0x08048ba8 <+16>: push %eax
0x08048ba9 <+17>: lea -0x5(%ebp),%eax
0x08048bac <+20>: push %eax
0x08048bad <+21>: lea -0xc(%ebp),%eax
0x08048bb0 <+24>: push %eax
0x08048bb1 <+25>: push $0x80497de
0x08048bb6 <+30>: push %edx
0x08048bb7 <+31>: call 0x8048860 <sscanf@plt>
0x08048bb8 <+32>: add $0x20,%esp
0x08048bbf <+39>: cmp $0x2,%eax
0x08048bc2 <+42>: jg 0x8048bc9 <phase_3+49>
0x08048bc4 <+44>: call 0x80494fc <explode_bomb>
0x08048bc9 <+49>: cmpl $0x7,-0xc(%ebp)
0x08048bcd <+53>: ja 0x8048c88 <phase_3+240>
0x08048bd3 <+59>: mov -0xc(%ebp),%eax
0x08048bd6 <+62>: jmp *0x80497e8(,%eax,4)
0x08048bdd <+69>: lea 0x0(%esi),%esi
0x08048be0 <+72>: mov $0x71,%bl
0x08048be2 <+74>: cmpl $0x309,-0x4(%ebp)
0x08048be9 <+81>: je 0x8048c8f <phase_3+247>
0x08048bef <+87>: call 0x80494fc <explode_bomb>
0x08048bf4 <+92>: jmp 0x8048c8f <phase_3+247>
0x08048bf9 <+97>: lea 0x0(%esi,%eiz,1),%esi
0x08048c00 <+104>: mov $0x62,%bl
0x08048c02 <+106>: cmpl $0xd6,-0x4(%ebp)
0x08048c09 <+113>: je 0x8048c8f <phase_3+247>
0x08048c0f <+119>: call 0x80494fc <explode_bomb>
0x08048c14 <+124>: jmp 0x8048c8f <phase_3+247>
0x08048c16 <+126>: mov $0x62,%bl
0x08048c18 <+128>: cmpl $0x2f3,-0x4(%ebp)
0x08048c1f <+135>: je 0x8048c8f <phase_3+247>
0x08048c21 <+137>: call 0x80494fc <explode_bomb>
0x08048c26 <+142>: jmp 0x8048c8f <phase_3+247>
0x08048c28 <+144>: mov $0x6b,%bl
0x08048c2a <+146>: cmpl $0xfb,-0x4(%ebp)
0x08048c31 <+153>: je 0x8048c8f <phase_3+247>
0x08048c33 <+155>: call 0x80494fc <explode_bomb>
0x08048c38 <+160>: jmp 0x8048c8f <phase_3+247>
```

This expects as: “an integer”- “a character”- “an integer”

```
(gdb) x/s 0x80497de
0x80497de: "%d %c %d"
```


Input the passphrase as “1 a 7”. Results:

```
(gdb) p /x $eax
$1 = 0xf7fb7548
(gdb) x/d $ebp-4
0xffffd2d4:      0
(gdb) x/c $ebp-5
0xffffd2d3:      8 '\b'
(gdb) x/d $ebp-0xc
0xffffd2cc:     -40
(gdb) █
```

The first condition to not explode the bomb: we have to fill all the 3 variables passed to **sscanf**.

```
0x08048bbf <+39>: cmp    eax,0x2
0x08048bc2 <+42>: jg     0x08048bc9 <phase_3+49>
0x08048bc4 <+44>: call  0x080494fc <explode_bomb>
```

Second condition: the first number must be ≤ 7

```
0x08048bc9 <+49>: cmp    DWORD PTR [ebp-0xc],0x7
0x08048bcd <+53>: ja     0x08048c88 <phase_3+240>
```

The last part of the function looks like a case structure. We have the following table of addresses:

```
0x08048bd6 <+62>: jmp    DWORD PTR [eax*4+0x80497e8]
```

In **eax**, we have the first number, which we chose as 7.

```
(gdb) x/10wx 0x80497e8
0x80497e8:      0x08048be0      0x08048c00      0x08048c16      0x08048c28
0x80497f8:      0x08048c40      0x08048c52      0x08048c64      0x08048c76
0x8049808:      0x67006425      0x746e6169
```

In our case, when the first parameter was 7, we'll jump to 0x08048c76.

```
(gdb) x /x $eax*4+0x80497e8
```

```
0x8049804: 0x08048c76
```

```
0x08048c76 <+222>: mov    bl,0x62 ..... ascii letter 'b'
```

```
0x08048c78 <+224>: cmp    DWORD PTR [ebp-0x4],0x20c .....524 in decimal
```

```
0x08048c7f <+231>: je     0x8048c8f <phase_3+247>
```

```
0x08048c81 <+233>: call  0x80494fc <explode_bomb>
```

```
0x08048c86 <+238>: jmp    0x8048c8f <phase_3+247>
```

Password for phase 03: 7 b 524

```
Quit anyway? (y or n) y
kali@kali:~/Downloads/bigbangtheory-master$ ./sheldon1
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Public speaking is very easy.
Phase 1 defused. How about the next one?
1 2 6 24 120 720
That's number 2. Keep going!
7 b 524
Halfway there!
█
```

Phase 04

View the assembly code of the phase_4 using **disassemble phase_4** command.

```
(gdb) disassemble phase_4
Dump of assembler code for function phase_4:
0x08048ce0 <+0>:  push    %ebp
0x08048ce1 <+1>:  mov     %esp,%ebp
0x08048ce3 <+3>:  sub     $0x18,%esp
0x08048ce6 <+6>:  mov     0x8(%ebp),%edx
0x08048ce9 <+9>:  add     $0xffffffff,%esp
0x08048cec <+12>: lea     -0x4(%ebp),%eax
0x08048cef <+15>:  push    %eax
0x08048cf0 <+16>:  push    $0x8049808
0x08048cf3 <+19>:  push    %edx
0x08048cf6 <+22>:  call    0x8048860 <scanf@plt>
0x08048cfb <+27>:  add     $0x10,%esp
0x08048cfe <+30>:  cmp     $0x1,%eax
0x08048d01 <+33>:  jne     0x8048d09 <phase_4+41>
0x08048d03 <+35>:  cmpl    $0x0,-0x4(%ebp)
0x08048d07 <+39>:  jg      0x8048d0e <phase_4+46>
0x08048d09 <+41>:  call    0x80494fc <explode_bomb>
0x08048d0e <+46>:  add     $0xffffffff,%esp
0x08048d11 <+49>:  mov     -0x4(%ebp),%eax
0x08048d14 <+52>:  push    %eax
0x08048d15 <+53>:  call    0x8048ca0 <func4>
0x08048d1a <+58>:  add     $0x10,%esp
0x08048d1d <+61>:  cmp     $0x37,%eax
0x08048d20 <+64>:  je      0x8048d27 <phase_4+71>
0x08048d22 <+66>:  call    0x80494fc <explode_bomb>
0x08048d27 <+71>:  mov     %ebp,%esp
0x08048d29 <+73>:  pop     %ebp
0x08048d2a <+74>:  ret
End of assembler dump.
```

On line phase_4+16, the phase_4 function is pushing a fixed value stored at memory address 0x8049808 onto the stack right before a call to **scanf** is made. As we have learned from the past phases, fixed values are almost always important. Lo and behold, when we dump the contents of the memory address we get “%d”, which tells us that the answer to this phase should be a single integer.

```
kali@kali: ~/Downloads/bigbangtheory-master
File  Actions  Edit  View  Help
0x08048d2a <+74>:  ret
End of assembler dump.
(gdb) x/s 0x8049808
0x8049808:  "%d"
```

The second important feature of this code occurs phase_4+53. Input value, which is stored in **eax**, is getting input into this other function called func4. Right now func4 is a black box, so we will need to dig into it using **si** to figure out what it is doing to our integer.

The last thing we need to consider is that, after <func4> gets called and returns our potentially altered input to **eax**, **eax** then gets compared to the hex value 37, which in decimal is 55. This means that, whatever our input is, to begin with, it needs to be turned into 55 by func4.

Now dig into func4 to figure out what we need to input to generate 55.

```

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

```

The next condition is that the func4 should return 0x37

```

0x08048d1d <+61>: cmp    eax,0x37
0x08048d20 <+64>: je     0x8048d27 <phase_4+71>

```

In func4, if argument ≤ 1 , the return value is 0x1:

```

0x08048cab <+11>: cmp    ebx,0x1
0x08048cae <+14>: jle    0x8048cd0 <func4+48>

0x08048cd0 <+48>: mov    eax,0x1

```

Func4 :

func4(x):

if $x \leq 1$:

return 1

else :

$y = \text{func4}(x-1)$

$z = \text{func4}(x-2)$

return $y + z$

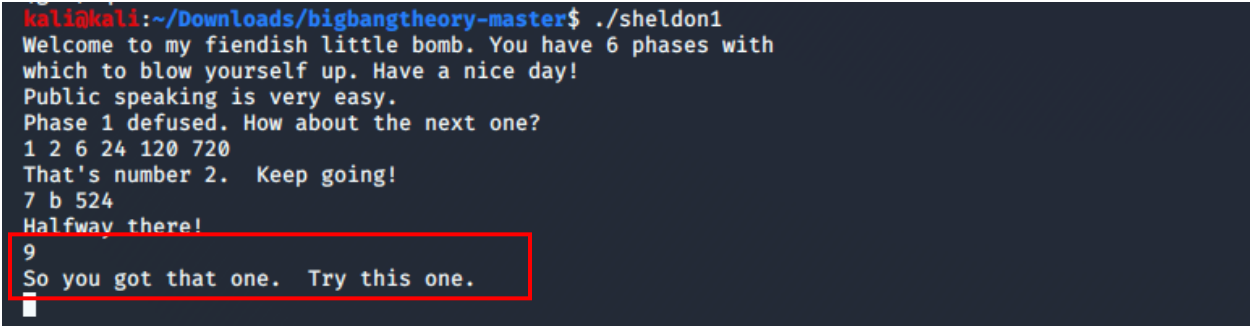
It's the Fibonacci function, implemented recursively. We quickly convert it to python code:

```
#!/usr/bin/python
```

```
def func4(x):  
    if x <= 1 :  
        return 1  
    else :  
        y = func4(x-1)  
        z = func4(x-2)  
        return y + z  
  
if __name__ == "__main__":  
    print func4(9)
```

We're expecting 0x37, which is 55 decimal, which is Fibonacci(9).

Password for phase 04: **9**



```
kali@kali:~/Downloads/bigbangtheory-master$ ./sheldon1  
Welcome to my fiendish little bomb. You have 6 phases with  
which to blow yourself up. Have a nice day!  
Public speaking is very easy.  
Phase 1 defused. How about the next one?  
1 2 6 24 120 720  
That's number 2. Keep going!  
7 b 524  
Halfway there!  
9  
So you got that one. Try this one.  
█
```