

CA Lab 5

Rahul Katinni

S20200010091

Phase 1 :

To diffuse the 1st phase of the bomb I used strings bomb command to view all the strings present in the bomb and searched for any abnormal string that could be the key to the 1st phase.

I found the key to my phase 1 to be

`"I turned the moon into something I call a Death Star."`

```
rahol@ubuntu-lappy: ~/Desktop/Bomb_lab
$ ./bomb
[+]APRA
ANAVA
AUAU1
[+]AUA]A^A
ks: Error: Couldn't open ks
Usage: %s [-input_file]
That's number 2. Keep going!
Halfway there!
Good work! On to the next...
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?
So you got that one. Try this one.
I turned the moon into something I call a Death Star.
Wow! You've defused the secret stage!
So you think you can stop the bomb with ctrl-c, do you?
Curse, you've found the secret phase!
But finding it and solving it are quite different...
Congratulations! You've defused the bomb!
Well...
OK. :-)
Invalid phaseks
BOOM!!!
The bomb has blown up.
nd nd nd nd nd nd
Error: Premature EOF on stdin
@NADE_BOMB
Error: Input line too long
nd nd ks
DREv11
greatwhite.lcs.cs.cmu.edu
angelshark.lcs.cs.cmu.edu
makoshark.lcs.cs.cmu.edu
whiteshark.lcs.cs.cmu.edu
Program timed out after nd seconds
Error: HTTP request failed with error Nd: %s
GET /ks/submitr.pl/?userId=ks&userpwd=ks&lab=ks&result=ks&submit=submit HTTP/1.0
Error: Unable to connect to server %s
%%N02X
ks Nd %([a-zA-Z ]
changeme.lcs.cs.cmu.edu
AUTORESULT_STRING=ks
csapp
+255
GCC: (Ubuntu 5.4.0-6ubuntu1-10.04.12) 5.4.0 20160609
-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!
Kso you got that one. Try this one.
Good work! On to the next...
/usr/include/x86_64-linux-gnu/bits
/usr/lib/gcc/x86_64-linux-gnu/5/include
/usr/include
```

```
rahol@ubuntu-lappy: ~/Desktop/Bomb_lab$ gdb bomb
GNU gdb (Ubuntu 8.2-0ubuntu1~20.04) 8.2
Copyright (C) 2020 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 bomb...
(gdb) b phase_2
Breakpoint 1 at 0x400e09
(gdb) b phase_3
Breakpoint 2 at 0x400f11
(gdb) b phase_4
Breakpoint 3 at 0x400ff3
(gdb) b phase_5
Breakpoint 4 at 0x401000
(gdb) b phase_6
Breakpoint 5 at 0x4010ec
(gdb) i b
num      Type           Disp Enb Address            What
1         breakpoint     keep y   0x00000000400e09 <phase_2>
2         breakpoint     keep y   0x00000000400f11 <phase_3>
3         breakpoint     keep y   0x00000000400ff3 <phase_4>
4         breakpoint     keep y   0x00000000401000 <phase_5>
5         breakpoint     keep y   0x000000004010ec <phase_6>
(gdb) r
Starting program: /home/rahol/Desktop/Bomb_lab/bomb
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?
```

Phase 2 :

For deciphering phase 2, I debugged the assembly code step by step and found that my 1st number should be positive and the

logic for the remaining five number to be as follows :

Let the numbers be a,b,c,d,e,f.

The logic is :

1st number a is any positive number.

Then $b = a + 1$

$c = b + 2$

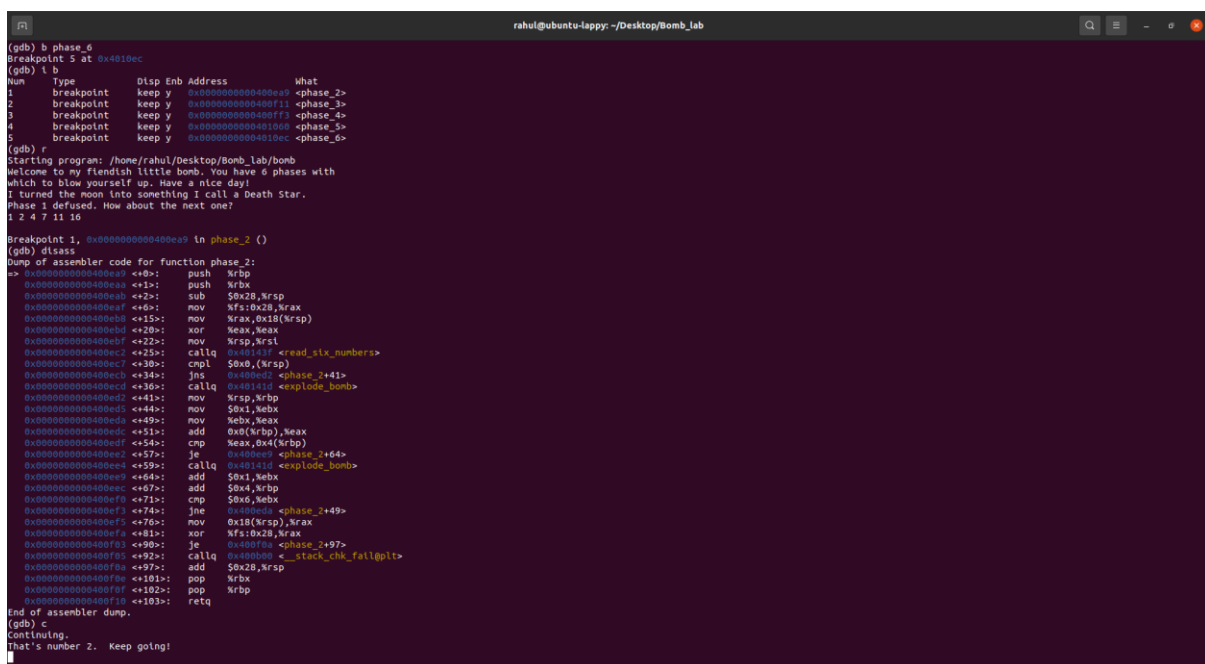
$d = c + 3$

$e = d + 4$

$f = e + 5$

Therefore taking 1st number as 1 the key to the 2nd phase is :

“ 1 , 2 , 4 , 7 , 11 , 16 ”



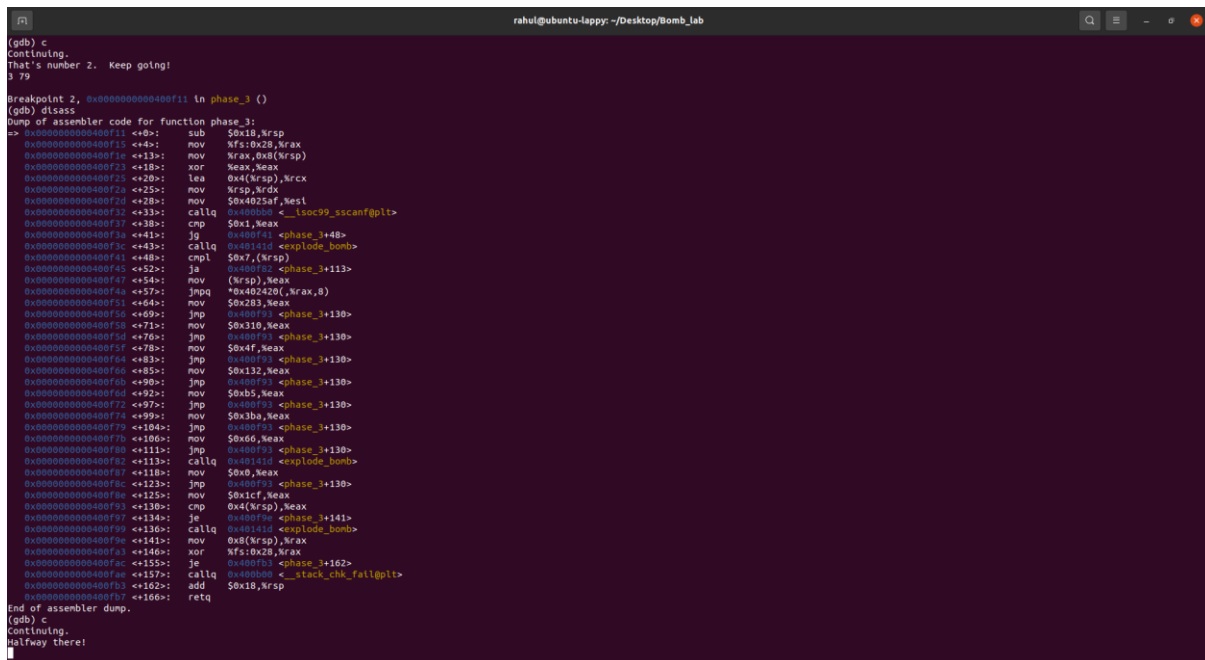
```
(gdb) b phase_6
Breakpoint 5 at 0x4010ec
(gdb) i b
Num Type Disps Enb Address What
1 breakpoint keep y 0x00000000400ea9 <phase_2>
2 breakpoint keep y 0x00000000400ff1 <phase_3>
3 breakpoint keep y 0x00000000400ff3 <phase_4>
4 breakpoint keep y 0x00000000401000 <phase_5>
5 breakpoint keep y 0x0000000040100c <phase_6>
(gdb) r
Starting program: /home/rahu/Desktop/Bomb_lab/bomb
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?
1 2 4 7 11 16
Breakpoint 1, 0x00000000400ea9 in phase_2 ()
(gdb) d/ass
Dump of assembler code for function phase_2:
=> 0x00000000400ea9 <+0>: push rbp
0x00000000400ea9 <+1>: push rbx
0x00000000400ea9 <+2>: sub $0x28,%rsp
0x00000000400ea9 <+3>: mov %fs:0x28,%rax
0x00000000400ea9 <+15>: mov %rax,0x10(%rsp)
0x00000000400ea9 <+20>: xor %eax,%eax
0x00000000400ea9 <+22>: mov %rsp,%rsi
0x00000000400ea9 <+25>: callq 0x40101f <read_six_numbers>
0x00000000400ea9 <+30>: cmpl $0x0,(%rsi)
0x00000000400ea9 <+34>: jns 0x400ee2 <phase_2+41>
0x00000000400ea9 <+36>: callq 0x40101d <explode_bomb>
0x00000000400ea9 <+41>: mov %rsp,%rbp
0x00000000400ea9 <+44>: mov $0x1,%ebx
0x00000000400ea9 <+49>: mov %ebx,%eax
0x00000000400ea9 <+51>: add 0x0(%rbp),%eax
0x00000000400ea9 <+54>: cmpl %eax,0x4(%rbp)
0x00000000400ea9 <+57>: je 0x400ee9 <phase_2+64>
0x00000000400ea9 <+59>: callq 0x40101d <explode_bomb>
0x00000000400ea9 <+64>: add $0x1,%ebx
0x00000000400ea9 <+67>: add $0x4,%rbp
0x00000000400ea9 <+71>: cmpl $0x6,%ebx
0x00000000400ea9 <+74>: jne 0x400ee9 <phase_2+49>
0x00000000400ea9 <+76>: mov 0x18(%rsp),%rax
0x00000000400ea9 <+81>: xor %fs:0x28,%rax
0x00000000400ea9 <+90>: je 0x400ff0a <phase_2+97>
0x00000000400ea9 <+92>: callq 0x400ee9 <__stack_chk_fail@plt>
0x00000000400ea9 <+97>: add $0x28,%rsp
0x00000000400ea9 <+101>: pop %rbx
0x00000000400ea9 <+102>: pop %rbp
0x00000000400ea9 <+103>: retq
End of assembler dump.
(gdb) c
Continuing.
That's number 2. Keep going!
```

Phase 3 :

For phase 3 after looking at the assembly code I found that the 1st number should be less than 7 and to find the 2nd number I took a random input and ran it through the code step by step and at the final comparison I extracted the correct input from the registers.

The key I got for phase 3 is :

“ 3 , 79 ”



```
(gdb) c
Continuing.
That's number 2. Keep going!
3 79

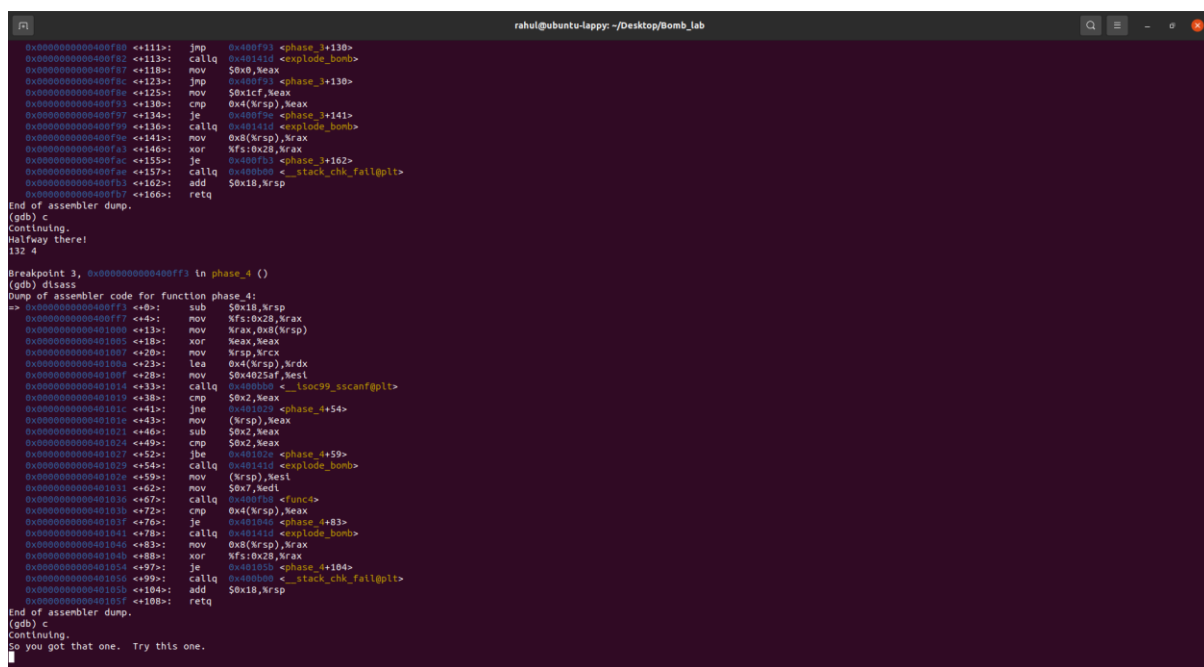
Breakpoint 2, 0x00000000400f11 in phase_3 ()
(gdb) disass
Dump of assembler code for function phase_3:
-> 0x00000000400f11 <+0>: sub    $0x18,%rsp
0x00000000400f15 <+4>: mov    $0x28,%rax
0x00000000400f1e <+13>: mov    %rax,%r9
0x00000000400f23 <+18>: xor    %eax,%eax
0x00000000400f25 <+20>: lea    0x4(%rsp),%rcx
0x00000000400f2a <+25>: mov    %rsp,%rdi
0x00000000400f2d <+28>: mov    $0x4025af,%esi
0x00000000400f32 <+33>: callq  0x400000 <__isoc99_sscanf@plt>
0x00000000400f37 <+38>: cmp    $0x1,%eax
0x00000000400f3a <+41>: jg     0x400f41 <phase_3+48>
0x00000000400f3c <+43>: callq  0x40141d <explode_bomb>
0x00000000400f41 <+48>: cmpl   $0x7,(%rsp)
0x00000000400f45 <+52>: ja     0x400f82 <phase_3+113>
0x00000000400f47 <+54>: mov    (%rsp),%eax
0x00000000400f4a <+57>: *0x402420(,%rax,8)
0x00000000400f51 <+64>: mov    $0x23,%eax
0x00000000400f56 <+69>: jmp    0x400f93 <phase_3+130>
0x00000000400f58 <+71>: mov    $0x310,%eax
0x00000000400f5d <+76>: jmp    0x400f93 <phase_3+130>
0x00000000400f5f <+78>: mov    $0x4f,%eax
0x00000000400f64 <+83>: jmp    0x400f93 <phase_3+130>
0x00000000400f66 <+85>: mov    $0x132,%eax
0x00000000400f6a <+90>: jmp    0x400f93 <phase_3+130>
0x00000000400f6d <+92>: mov    $0xb5,%eax
0x00000000400f72 <+97>: jmp    0x400f93 <phase_3+130>
0x00000000400f74 <+99>: mov    $0x3ba,%eax
0x00000000400f79 <+104>: jmp    0x400f93 <phase_3+130>
0x00000000400f7b <+106>: mov    $0xd6,%eax
0x00000000400f7e <+111>: jmp    0x400f93 <phase_3+130>
0x00000000400f82 <+113>: callq  0x40141d <explode_bomb>
0x00000000400f87 <+118>: mov    $0x0,%eax
0x00000000400f8c <+123>: jmp    0x400f93 <phase_3+130>
0x00000000400f8e <+125>: mov    $0xc1f,%eax
0x00000000400f93 <+130>: cmp    0x4(%rsp),%eax
0x00000000400f97 <+134>: je     0x400f9e <phase_3+141>
0x00000000400f99 <+136>: callq  0x40141d <explode_bomb>
0x00000000400f9e <+141>: mov    0x8(%rsp),%rax
0x00000000400fa3 <+146>: xor    %fs:0x28,%rax
0x00000000400fac <+155>: je     0x400f93 <phase_3+162>
0x00000000400fae <+157>: callq  0x400000 <__stack_chk_fail@plt>
0x00000000400fb3 <+162>: add    $0x18,%rsp
0x00000000400fb7 <+166>: retq

End of assembler dump.
(gdb) c
Continuing.
Halfway there!
```

Phase 4 :

I have done phase 4 also in a similar way as phase 3 I took 2 random number as input and ran them through the code and found the correct replacement for one of them in the registers in the final comparison step after the fun4 function in phase 4 was executed.

The key I got for phase 4 is :
“132 , 4”



```
0x00000000400f00 <+111>: jmp     0x400f93 <phase_3+130>
0x00000000400f02 <+113>: callq   0x40141d <explode_bomb>
0x00000000400f07 <+118>: mov     $0x0,%eax
0x00000000400f0c <+123>: jmp     0x400f93 <phase_3+130>
0x00000000400f0e <+125>: mov     $0x1cf,%eax
0x00000000400f13 <+130>: cmp     0x4(%rsp),%eax
0x00000000400f17 <+134>: je      0x400f9e <phase_3+141>
0x00000000400f19 <+136>: callq   0x40141d <explode_bomb>
0x00000000400f1e <+141>: mov     0x8(%rsp),%rax
0x00000000400f23 <+146>: xor     %fsi,%rax
0x00000000400f2c <+155>: je      0x400f93 <phase_3+162>
0x00000000400f2e <+157>: callq   0x400b00 <__stack_chk_fail@plt>
0x00000000400f33 <+162>: add     $0x18,%rsp
0x00000000400f37 <+166>: retq

End of assembler dump.
(gdb) c
Continuing.
Halfway there!
132 4

Breakpoint 3, 0x00000000400f93 in phase_4 ()
(gdb) disass
Dump of assembler code for function phase_4:
=> 0x00000000400f73 <+8>: sub     $0x18,%rsp
0x00000000400f77 <+4>: mov     %fsi,%rax
0x00000000400f80 <+13>: mov     %rax,0x8(%rsp)
0x00000000400f85 <+18>: xor     %eax,%eax
0x00000000400f87 <+20>: mov     %rax,%rcx
0x00000000400f8a <+23>: lea     0x4(%rsp),%rdx
0x00000000400f8f <+28>: mov     $0x4025af,%esi
0x00000000400f94 <+33>: callq   0x400b00 <__tsc99_5scanf@plt>
0x00000000400f99 <+38>: cmp     $0x2,%eax
0x00000000400fa0 <+41>: jne     0x401029 <phase_4+54>
0x00000000400fa3 <+43>: mov     (%rsp),%eax
0x00000000400fa6 <+46>: sub     $0x2,%eax
0x00000000400fa9 <+49>: cmp     $0x2,%eax
0x00000000400fab <+52>: jbe     0x40102e <phase_4+59>
0x00000000400fac <+54>: callq   0x40141d <explode_bomb>
0x00000000400fae <+59>: mov     (%rsp),%esi
0x00000000400fb1 <+62>: mov     $0x7,%edi
0x00000000400fb4 <+67>: callq   0x400b00 <func0>
0x00000000400fb7 <+72>: cmp     0x4(%rsp),%eax
0x00000000400fb9 <+76>: je      0x401046 <phase_4+83>
0x00000000400fbc <+78>: callq   0x40141d <explode_bomb>
0x00000000400fbd <+83>: mov     0x8(%rsp),%rax
0x00000000400fbf <+88>: xor     %fsi,%rax
0x00000000400fc4 <+97>: je      0x40105b <phase_4+104>
0x00000000400fc6 <+99>: callq   0x400b00 <__stack_chk_fail@plt>
0x00000000400fc9 <+104>: add     $0x18,%rsp
0x00000000400fd7 <+108>: retq

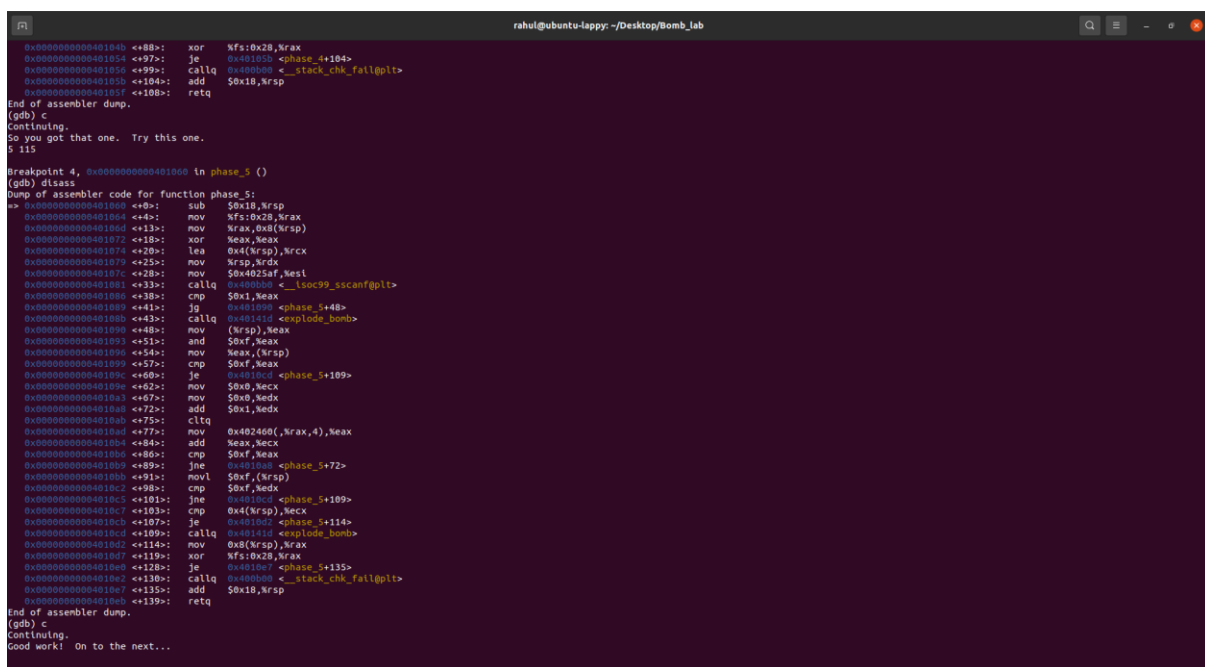
End of assembler dump.
(gdb) c
Continuing.
So you got that one. Try this one.
```

Phase 5 :

In phase 5 there is an hardcoded array which we traverse it in such a way that the element in the current index is taken to be

the next index to be searched. The 1st number of the key is the index at which we start searching for 15 such that it is found on the 15th iteration and the 2nd number is the sum of all the elements of the array which we visited in the process.

The key I found for phase 5 is :
“ 5 , 115 ”



```
raahul@ubuntu-lappy: ~/Desktop/Bomb_lab
0x00000000401000: xor     %fs:0x28,%rax
0x00000000401004: je      0x40100b <phase_4+104>
0x00000000401006: callq   0x400000 <__stack_chk_fail@plt>
0x00000000401008: add     $0x18,%rsp
0x0000000040100a: retq

End of assembler dump.
(gdb) c
Continuing.
So you got that one. Try this one.
5 115

Breakpoint 4, 0x00000000401000 in phase_5 ()
(gdb) disass
Dump of assembler code for function phase_5:
0x00000000401000: sub     $0x18,%rsp
0x00000000401004: mov     %fs:0x28,%rax
0x00000000401006: mov     %rax,0x8(%rsp)
0x00000000401008: xor     %eax,%eax
0x0000000040100a: lea     0x4(%rsp),%rcx
0x0000000040100c: mov     %rsp,%rdx
0x0000000040100e: mov     $0x4025af,%esi
0x00000000401010: callq   0x400000 <__isoc99_sscanf@plt>
0x00000000401012: cmp     $0x1,%eax
0x00000000401014: jg      0x401009 <phase_5+48>
0x00000000401016: callq   0x40101d <explode_bomb>
0x00000000401018: mov     (%rsp),%eax
0x0000000040101a: and     $0xf,%eax
0x0000000040101c: mov     %eax,(%rsp)
0x0000000040101e: cmp     $0xf,%eax
0x00000000401020: je      0x4010cd <phase_5+109>
0x00000000401022: mov     $0x0,%ecx
0x00000000401024: mov     $0x0,%edx
0x00000000401026: add     $0x1,%edx
0x00000000401028: cltq
0x0000000040102a: mov     0x402460(%rax,4),%eax
0x0000000040102c: add     %eax,%ecx
0x0000000040102e: cmp     $0xf,%eax
0x00000000401030: jne     0x401008 <phase_5+72>
0x00000000401032: movl    $0xf,(%rsp)
0x00000000401034: cmpl    $0xf,%edx
0x00000000401036: jne     0x4010cd <phase_5+109>
0x00000000401038: cmpl    0x4(%rsp),%ecx
0x0000000040103a: je      0x4010cd <phase_5+114>
0x0000000040103c: callq   0x40101d <explode_bomb>
0x0000000040103e: mov     0x4(%rsp),%rax
0x00000000401040: xor     %fs:0x28,%rax
0x00000000401042: je      0x4010e7 <phase_5+135>
0x00000000401044: callq   0x400000 <__stack_chk_fail@plt>
0x00000000401046: add     $0x18,%rsp
0x00000000401048: retq

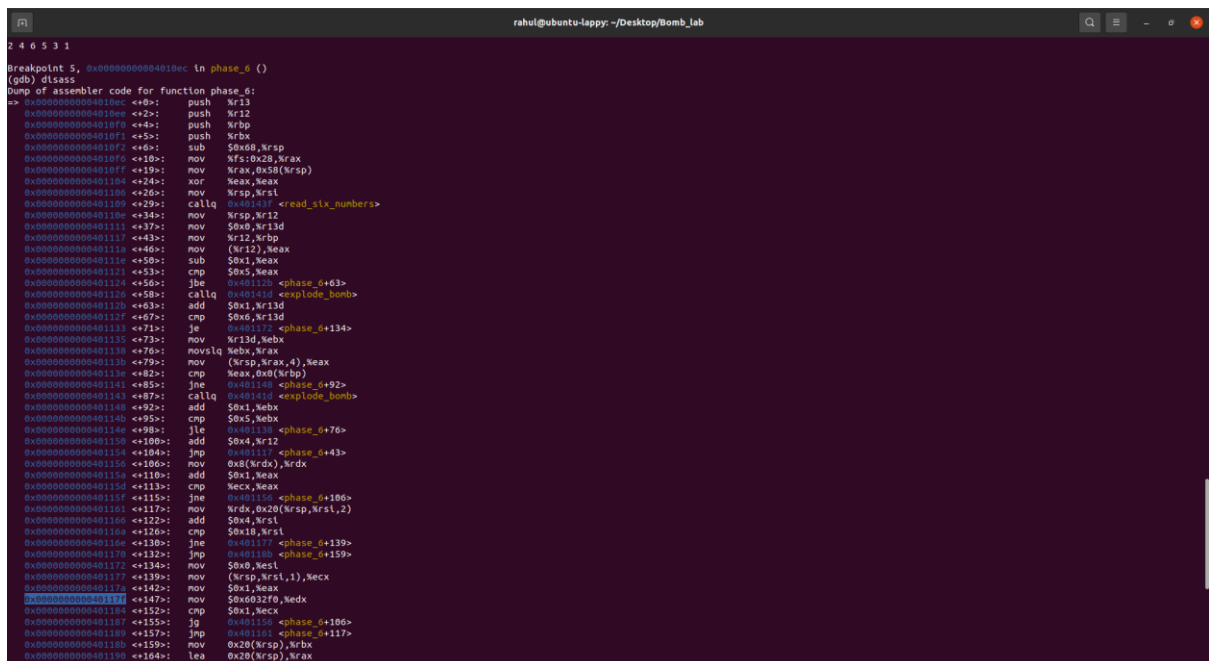
End of assembler dump.
(gdb) c
Continuing.
Good work! On to the next...
```

Phase 6 :
In phase 6 we enter the node numbers of a linked list in ascending order which is hardcoded, the bomb 1st checks that all the

number are between 1 and 6 and that all are different using nested loops. Then the bomb sorts the linked list to be in ascending order.

The key I found for phase 6 is :

“ 2 , 4 , 6 , 5 , 3 , 1 ”



```
2 4 6 5 3 1
Breakpoint 5, 0x00000000004010ec in phase_6 ()
(gdb) dsass
Dump of assembler code for function phase_6:
0x00000000004010ec <+0>: push    %r13
0x00000000004010ee <+2>: push    %r12
0x00000000004010f0 <+4>: push    %rbp
0x00000000004010f1 <+5>: push    %rbx
0x00000000004010f2 <+6>: sub     $0x68,%rsp
0x00000000004010f5 <+10>: mov     %fs:0x28,%rax
0x00000000004010f7 <+19>: mov     %rax,0x50(%rsp)
0x0000000000401104 <+24>: xor     %eax,%eax
0x0000000000401106 <+26>: mov     %rsp,%rsi
0x0000000000401109 <+29>: callq   0x40103f <read_six_numbers>
0x000000000040110e <+34>: mov     %rsp,%r12
0x0000000000401111 <+37>: mov     $0x0,%r13d
0x0000000000401117 <+43>: mov     %r12,%rbp
0x000000000040111a <+46>: mov     (%r12),%eax
0x000000000040111c <+50>: sub     $0x1,%eax
0x0000000000401121 <+53>: cmp     $0x5,%eax
0x0000000000401124 <+56>: jbe     0x40112b <phase_6+63>
0x0000000000401126 <+58>: callq   0x40114d <explode_bomb>
0x000000000040112b <+63>: add     $0x1,%r13d
0x000000000040112f <+67>: cmp     $0x6,%r13d
0x0000000000401133 <+71>: je      0x401137 <phase_6+134>
0x0000000000401135 <+73>: mov     %r13d,%ebx
0x0000000000401138 <+76>: movslq  %ebx,%rax
0x000000000040113b <+79>: mov     (%rsp,%rax,4),%eax
0x000000000040113e <+82>: cmp     %eax,0x0(%rsp)
0x0000000000401141 <+85>: jne     0x401148 <phase_6+92>
0x0000000000401143 <+87>: callq   0x40114d <explode_bomb>
0x0000000000401148 <+92>: add     $0x1,%ebx
0x000000000040114b <+95>: cmp     $0x5,%ebx
0x000000000040114e <+98>: jle     0x401158 <phase_6+76>
0x0000000000401150 <+100>: add     $0x4,%r12
0x0000000000401154 <+104>: jmp     0x401117 <phase_6+43>
0x0000000000401156 <+106>: mov     0x8(%rdx),%rdx
0x000000000040115a <+110>: add     $0x1,%eax
0x000000000040115d <+113>: cmp     %ecx,%eax
0x000000000040115f <+115>: jne     0x401158 <phase_6+106>
0x0000000000401161 <+117>: mov     %rdx,0x20(%rsp,%rsi,2)
0x0000000000401166 <+122>: add     $0x4,%rsi
0x000000000040116a <+126>: cmp     $0x10,%rsi
0x000000000040116e <+130>: jne     0x401177 <phase_6+139>
0x0000000000401170 <+132>: jmp     0x401180 <phase_6+159>
0x0000000000401172 <+134>: mov     $0x0,%esi
0x0000000000401177 <+139>: mov     (%rsp,%rsi,1),%ecx
0x000000000040117a <+142>: mov     $0x1,%eax
0x000000000040117b <+147>: mov     $0x403270,%edx
0x0000000000401184 <+152>: cmp     $0x1,%ecx
0x0000000000401187 <+155>: jg      0x401158 <phase_6+106>
0x0000000000401189 <+157>: jmp     0x401161 <phase_6+117>
0x000000000040118b <+159>: mov     0x20(%rsp),%rbx
0x0000000000401190 <+164>: lea     0x20(%rsi-1),%rax
0x0000000000401193 <+168>: lea     0x40(%rsi-1),%rax
```

```
rahu@ubuntu-lappy: ~/Desktop/Bomb_lab
0x0000000000401156 <+106>: mov    0x8(%rdx),%rdx
0x0000000000401158 <+110>: add    $0x1,%eax
0x000000000040115d <+113>: cmp    %ecx,%eax
0x000000000040115f <+115>: jne     0x401158 <phase_6+106>
0x0000000000401161 <+117>: mov    %rdx,0x20(%rsp,%rsl,2)
0x0000000000401166 <+122>: add    $0x4,%rsl
0x000000000040116a <+126>: cmp    $0x10,%rsl
0x000000000040116e <+130>: jne     0x401177 <phase_6+119>
0x0000000000401170 <+132>: jmp     0x401180 <phase_6+129>
0x0000000000401172 <+134>: mov    $0x0,%esi
0x0000000000401177 <+139>: mov    (%rsp,%rsl,1),%ecx
0x0000000000401179 <+142>: mov    $0x1,%eax
0x000000000040117f <+147>: mov    $0x6032f0,%edx
0x0000000000401184 <+152>: cmp    $0x1,%ecx
0x0000000000401187 <+155>: jg      0x401180 <phase_6+106>
0x0000000000401189 <+157>: jmp     0x401161 <phase_6+117>
0x000000000040118b <+159>: mov    0x20(%rsp),%rbx
0x0000000000401190 <+164>: lea     0x20(%rsp),%rax
0x0000000000401195 <+169>: lea     0x40(%rsp),%rsl
0x000000000040119a <+174>: mov    %rbx,%rcx
0x000000000040119d <+177>: mov    0x8(%rax),%rdx
--Type <RET> for more, q to quit, c to continue without paging.--c
0x00000000004011a1 <+181>: mov    %rdx,0x8(%rcx)
0x00000000004011a5 <+185>: add    $0x8,%rax
0x00000000004011a9 <+189>: mov    %rdx,%rcx
0x00000000004011ac <+192>: cmp    %rsl,%rax
0x00000000004011af <+195>: jne     0x40119d <phase_6+177>
0x00000000004011b1 <+197>: movq    $0x0,%rdi
0x00000000004011b9 <+205>: mov    $0x5,%ebp
0x00000000004011be <+210>: mov    0x8(%rbx),%rax
0x00000000004011c2 <+214>: mov    (%rax),%eax
0x00000000004011c4 <+216>: cmp    %eax,(%rbx)
0x00000000004011c6 <+218>: jle     0x4011c0 <phase_6+225>
0x00000000004011c8 <+220>: callq   0x4011d4 <explode_bomb>
0x00000000004011c0 <+225>: mov    0x8(%rbx),%rbx
0x00000000004011d1 <+229>: sub     $0x1,%ebp
0x00000000004011d4 <+232>: jne     0x4011be <phase_6+210>
0x00000000004011d6 <+234>: mov    0x58(%rsp),%rax
0x00000000004011d8 <+239>: xor     %fs,%rax
0x00000000004011e4 <+248>: je      0x4011e0 <phase_6+255>
0x00000000004011e6 <+250>: callq   0x400b00 <__stack_chk_fail@plt>
0x00000000004011e8 <+255>: add     $0x68,%rsp
0x00000000004011ef <+259>: pop     %rbx
0x00000000004011f0 <+260>: pop     %rbp
0x00000000004011f1 <+261>: pop     %r12
0x00000000004011f3 <+263>: pop     %r13
0x00000000004011f5 <+265>: retq
End of assembler dump.
(gdb) b *0x000000000040117f
Breakpoint 6 at 0x40117f
(gdb) c
Continuing.
Breakpoint 6, 0x000000000040117f in phase_6 ()
(gdb)
```

```
rahu@ubuntu-lappy: ~/Desktop/Bomb_lab
0x00000000004011e4 <+248>: je      0x4011e0 <phase_6+255>
0x00000000004011e6 <+250>: callq   0x400b00 <__stack_chk_fail@plt>
0x00000000004011e8 <+255>: add     $0x68,%rsp
0x00000000004011ef <+259>: pop     %rbx
0x00000000004011f0 <+260>: pop     %rbp
0x00000000004011f1 <+261>: pop     %r12
0x00000000004011f3 <+263>: pop     %r13
0x00000000004011f5 <+265>: retq
End of assembler dump.
(gdb) b *0x000000000040117f
Breakpoint 6 at 0x40117f
(gdb) c
Continuing.
Breakpoint 6, 0x000000000040117f in phase_6 ()
(gdb) nt
0x0000000000401184 in phase_6 ()
(gdb) p $edx
$1 = 6304496
(gdb) p /24gw 6304496
Size letters are meaningless in "print" command.
(gdb) x/24gw 6304496
0x6032f0 <node1>: 933 1 6304512 0
0x603300 <node2>: 220 2 6304528 0
0x603310 <node3>: 706 3 6304544 0
0x603320 <node4>: 368 4 6304560 0
0x603330 <node5>: 438 5 6304576 0
0x603340 <node6>: 380 6 0 0
(gdb)
```

After finding the keys of all the phases I defused the bomb successfully.


```
rahul@ubuntu-lappy: ~/Desktop/Bomb_lab
rahul@ubuntu-lappy:~/Desktop/Bomb_lab$ ./bomb
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?
1 2 4 7 11 16
That's number 2. Keep going!
3 79
Halfway there!
132 4
So you got that one. Try this one.
5 115
Good work! On to the next...
2 4 6 5 3 1
Congratulations! You've defused the bomb!
rahul@ubuntu-lappy:~/Desktop/Bomb_lab$
```