CA BOMB LAB REPORT

This lab involves bombs which are diffused by giving correct input which we are deriving studying the assembly code written regarding bomb functioning that has been assembled from the given source file using gdb and accessing different memory slots

USAGE: Linux environment and gdb debugger working basics

PROCESS OF DIFFUSION .

- DOWNLOAD UR DEDICATED BOMB ZIPPED FILE,UNZIP AND EXTRACT THEM IN A FOLDER AND OPEN THE BOMB FOLDER IN TERMINAL.
- IF PERMISSION IS DENIED TO EXECUTE THE BOMB,TYPE chmod x+u bomb<file name> TO GIVE EXECUTABLE PERMISSIONS TO RUN THE FILE.
- ENTER THE GDB MODE BY TYPING gdb bomb<file name> IN TERMINAL
- Firstly open the source code provided and analyse that there are 6 functions for 6phases respectively phase_1,phase_2....

PHASE 1

.Enter the gdb mode and set break point at phase_1<shortcut~b phase_1>

.Run the command 'run' to execute the program bomb.exe<shortcut~r>

.Give a random input and enter to stop at the break point,currently accessing memory adress while running tthe program is displayed

.To dissasemble the code enter disas
.To move manually to next step enter ni
.To access information regarding registers at that point enter 'i r'
.To continue running the program until next break point enter c

since the input is wrong bomb blows up

.We see many instruction calls like mov,sub, call and test commands
our input in register %rsp is compared with string in adress 0x402490
which is moved to sub reg %esi

```
aditya@aditya-ThinkBook: ~/Downloads/S
aditya@aditya-ThinkBook:~/Downloads/SEM2/CA/labs/bomb81-20220627T090213Z-001/bomb81$ gdb bomb
GNU gdb (Ubuntu 11.1-0ubuntu2) 11.1
Copyright (C) 2021 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:
<https://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_1
Breakpoint 1 at 0x400e8d
(gdb) run
Starting program: /home/aditya/Downloads/SEM2/CA/labs/bomb81-20220627T090213Z-001/bomb81/bomb
[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!
greetings this  is aditya chakravarthy

Breakpoint 1, 0x0000000000400e8d in phase_1 ()
(gdb) disas
Dump of assembler code for function phase_1:
=> 0x0000000000400e8d <+0>:     sub    $0x8,%rsp
   0x0000000000400e91 <+4>:     mov    $0x402490,%esi
   0x0000000000400e96 <+9>:     call   0x4013f8 <strings_not_equal>
   0x0000000000400e9b <+14>:    test   %eax,%eax
   0x0000000000400e9d <+16>:    je     0x400ea4 <phase_1+23>
   0x0000000000400e9f <+18>:    call   0x4014f7 <explode_bomb>
   0x0000000000400ea4 <+23>:    add    $0x8,%rsp
   0x0000000000400ea8 <+27>:    ret
End of assembler dump.
(gdb) ni
0x0000000000400e91 in phase_1 ()
(gdb) ni
0x0000000000400e96 in phase_1 ()
(gdb) ni
0x0000000000400e9b in phase_1 ()
(gdb) ni
0x0000000000400e9d in phase_1 ()
(gdb) ni
0x0000000000400e9f in phase_1 ()
(gdb) ni

BOOM!!!
The bomb has blown up.
[Inferior 1 (process 5325) exited with code 010]
(gdb)
```

.ACCESSING MEMORY LOCATION

.x/s 0x402490 <to access string in that location>

```
Breakpoint 1, 0x0000000000400e8d in phase_1 ()
(gdb) disas
Dump of assembler code for function phase_1:
=> 0x0000000000400e8d <+0>:     sub    $0x8,%rsp
   0x0000000000400e91 <+4>:     mov    $0x402490,%esi
   0x0000000000400e96 <+9>:     call   0x4013f8 <strings_not_equal>
   0x0000000000400e9b <+14>:    test   %eax,%eax
   0x0000000000400e9d <+16>:    je     0x400ea4 <phase_1+23>
   0x0000000000400e9f <+18>:    call   0x4014f7 <explode_bomb>
   0x0000000000400ea4 <+23>:    add    $0x8,%rsp
   0x0000000000400ea8 <+27>:    ret
End of assembler dump.
(gdb) x/d $0x402490
Value can't be converted to integer.
(gdb) x/s $0x402490
Value can't be converted to integer.
(gdb) x/s 0x402490
0x402490:       "All your base are belong to us."
(gdb)
```

.So the correct input string is “All your base are belong to us”

.run the program again entering r and give the correct input this time to diffuse the bomb

```
(gdb) r
Starting program: /home/aditya/Downloads/SEM2/CA/labs/bomb81-20220627T090213Z-00
1/bomb81/bomb
[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!
All your base are belong to us.

Breakpoint 1, 0x0000000000400e8d in phase_1 ()
(gdb) c
Continuing.
Phase 1 defused. How about the next one?
```

.Enter the dedicated string as first line in a separate solution text file(Solutions.txt) in the same bomb folder and save it

PHASE 2

.Delete the break point at phase_1 using clear phase_1

.Set break point at phase_2(using shortcut as mentioned in phase_1)

.Run the program r Solutions.txt where first line is given as input to first phase and give some random input for the second phase,click enter to stop at break point phase_2

.Here on for every phase,we use disas to study the assembly code and make observations and identify which memory location while getting processed has call to function explode bomb where the bomb gets exploded

```
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 0x400ea9
(gdb) r Solutions.txt
Starting program: /home/aditya/Downloads/SEM2/CA/labs/bomb81-20220627T090213Z-001/bomb81/bomb Solutions.txt
[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!
Phase 1 defused. How about the next one?
1 2 3 4 5 6

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   0x401519 <read_six_numbers>
   0x0000000000400ec7 <+30>:    cmpl   $0x0,(%rsp)
   0x0000000000400ecb <+34>:    jns    0x400ed2 <phase_2+41>
   0x0000000000400ecd <+36>:    call   0x4014f7 <explode_bomb>
   0x0000000000400ed2 <+41>:    mov    %rsp,%rbp
   0x0000000000400ed5 <+44>:    mov    $0x1,%ebx
   0x0000000000400eda <+49>:    mov    %ebx,%eax
   0x0000000000400edc <+51>:    add    0x0(%rbp),%eax
   0x0000000000400edf <+54>:    cmp    %eax,0x4(%rbp)
   0x0000000000400ee2 <+57>:    je     0x400ee9 <phase_2+64>
   0x0000000000400ee4 <+59>:    call   0x4014f7 <explode_bomb>
   0x0000000000400ee9 <+64>:    add    $0x1,%ebx
   0x0000000000400eec <+67>:    add    $0x4,%rbp
   0x0000000000400ef0 <+71>:    cmp    $0x6,%ebx
   0x0000000000400ef3 <+74>:    jne    0x400eda <phase_2+49>
--Type <RET> for more, q to quit, c to continue without paging--c
   0x0000000000400ef5 <+76>:    mov    0x18(%rsp),%rax
   0x0000000000400efa <+81>:    xor    %fs:0x28,%rax
   0x0000000000400f03 <+90>:    je     0x400f0a <phase_2+97>
   0x0000000000400f05 <+92>:    call   0x400b00 <__stack_chk_fail@plt>
   0x0000000000400f0a <+97>:    add    $0x28,%rsp
   0x0000000000400f0e <+101>:   pop    %rbx
   0x0000000000400f0f <+102>:   pop    %rbp
   0x0000000000400f10 <+103>:   ret
End of assembler dump.
(gdb)
```

.our observations are <read six numbers> are scanning six integers where cmp command says its comparing two operands

.A loop is running as per assembly code where control jumps from <+74> to <+49> where cmp command executed again

.That states until all 6 digits compared,loop is being run

.J instruction is jump instruction based on condition

.jne is jump if not equal to,je is jump to if equal to

.so understanding the assembly code,where the input digit is compared to original digit in key,we can access the registers using i r to see the original digit which was accumulated for getting compared

.similar idea is followed in all phases here on

.so whenever a bomb function when je is not executed at cmp of digits <+54> is called,before it returns that is BOOM!!! ,access the registers to find correct the digit .Repeat the same by entering ni and running the program via loop and checking reg at time of bomb function call

SO THE CORRECT INPUT TURNS TO BE 0 1 3 6 10 15

```
0x0000000000400ee9 in phase_2 ()
(gdb)
0x0000000000400eec in phase_2 ()
(gdb)
0x0000000000400ef0 in phase_2 ()
(gdb)
0x0000000000400ef3 in phase_2 ()
(gdb)
0x0000000000400eda in phase_2 ()
(gdb)
0x0000000000400edc in phase_2 ()
(gdb)
0x0000000000400edf in phase_2 ()
(gdb)
0x0000000000400ee2 in phase_2 ()
(gdb)
0x0000000000400ee4 in phase_2 ()
(gdb) i r
rax            0x3                 3
rbx            0x2                 2
rcx            0x0                 0
rdx            0x5                 5
rsi            0x5                 5
rdi            0x7fffffffd870      140737488345200
rbp            0x7fffffffdee4      0x7fffffffdee4
rsp            0x7fffffffdee0      0x7fffffffdee0
r8             0x0                 0
r9             0x0                 0
r10            0x7ffff7f47ac0      140737353382592
r11            0x7ffff7f483c0      140737353384896
r12            0x7fffffffe048      140737488347208
r13            0x400d56            4197718
r14            0x0                 0
r15            0x7ffff7ffbc40      140737354120256
rip            0x400ee4            0x400ee4 <phase_2+59>
eflags         0x297               [ CF PF AF SF IF ]
cs             0x33                51
ss             0x2b                43
ds             0x0                 0
es             0x0                 0
fs             0x0                 0
gs             0x0                 0
k0             0xfffff800          4294965248
k1             0x880               2176
k2             0x0                 0
k3             0x0                 0
k4             0x0                 0
k5             0x0                 0
k6             0x0                 0
k7             0x0                 0
(gdb)
```

where ee4 is adress containing bomb call

```
(gdb) r
Starting program: /home/aditya/Downloads/SEM2/CA/labs/bomb81-20220627T090213Z-001/bomb81/bomb
[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!
All your base are belong to us.
Phase 1 defused. How about the next one?
0 1 3 6 10 15
That's number 2.  Keep going!
```

PHASE 3

.Here prev break point is cleared and now b phase_3 and run updated Solutions.txt with 2nd phase original key as second line

```
0000000000400f1a <+13>:    mov    %rax,0x18(%rsp)
0000000000400f1f <+18>:    xor    %eax,%eax
0000000000400f21 <+20>:    lea    0x14(%rsp),%r8
0000000000400f26 <+25>:    lea    0xf(%rsp),%rcx
0000000000400f2b <+30>:    lea    0x10(%rsp),%rdx
0000000000400f30 <+35>:    mov    $0x4024d6,%esi
0000000000400f35 <+40>:    call   0x400bb0 <__isoc99_sscanf@plt>
0000000000400f3a <+45>:    cmp    $0x2,%eax
0000000000400f3d <+48>:    jg     0x400f44 <phase_3+55>
0000000000400f3f <+50>:    call   0x4014f7 <explode_bomb>
0000000000400f44 <+55>:    cmpl   $0x7,0x18(%rsp)
0000000000400f49 <+60>:    ja     0x401000 <phase_3+305>
0000000000400f4f <+66>:    mov    0x10(%rsp),%eax
0000000000400f53 <+70>:    jmp    *0x4024e0(,%rax,8)
0000000000400f5a <+77>:    mov    $0x71,%eax
0000000000400f5f <+82>:    cmpl   $0x68,0x14(%rsp)
0000000000400f63 <+87>:    je     0x400fb4 <phase_3+315>
0000000000400f65 <+93>:    call   0x4014f7 <explode_bomb>
0000000000400f6a <+98>:    mov    $0x71,%eax
0000000000400f6f <+103>:   jmp    0x400fb4 <phase_3+315>
0000000000400f71 <+108>:   mov    $0x76,%eax
0000000000400f76 <+113>:   cmpl   $0x76,0x14(%rsp)
0000000000400f7a <+118>:   je     0x400fb4 <phase_3+315>
0000000000400f7c <+120>:   call   0x4014f7 <explode_bomb>
0000000000400f81 <+125>:   mov    $0x76,%eax
0000000000400f86 <+130>:   jmp    0x400fb4 <phase_3+315>
0000000000400f88 <+132>:   mov    $0x79,%eax
0000000000400f8d <+137>:   cmpl   $0xa5,0x14(%rsp)
0000000000400f91 <+141>:   je     0x400fb4 <phase_3+315>
0000000000400f93 <+143>:   call   0x4014f7 <explode_bomb>
0000000000400f98 <+148>:   mov    $0x79,%eax
0000000000400f9d <+153>:   jmp    0x400fb4 <phase_3+315>
0000000000400f9f <+155>:   mov    $0x79,%eax
0000000000400fa4 <+160>:   cmpl   $0xb3ad,0x14(%rsp)
0000000000400fa9 <+165>:   je     0x400fb4 <phase_3+315>
0000000000400fab <+167>:   call   0x4014f7 <explode_bomb>
0000000000400fb0 <+172>:   mov    $0x79,%eax
0000000000400fb5 <+177>:   jmp    0x400fb4 <phase_3+315>
0000000000400fb7 <+179>:   mov    $0x68,%eax
0000000000400fbc <+184>:   cmpl   $0x21c,0x14(%rsp)
0000000000400fc1 <+189>:   je     0x400fb4 <phase_3+315>
0000000000400fc3 <+191>:   call   0x4014f7 <explode_bomb>
0000000000400fc8 <+196>:   mov    $0x62,%eax
0000000000400fcd <+201>:   cmpl   $0x206,0x14(%rsp)
0000000000400fd4 <+208>:   je     0x400fb4 <phase_3+315>
--Type <RET> for more, q to quit, c to continue without paging--c
0000000000400fd6 <+210>:   call   0x4014f7 <explode_bomb>
0000000000400fdb <+215>:   mov    $0x62,%eax
0000000000400fe0 <+220>:   jmp    0x400fb4 <phase_3+315>
0000000000400fe2 <+222>:   mov    $0x79,%eax
0000000000400fe7 <+227>:   cmpl   $0xbc,0x14(%rsp)
0000000000400fee <+234>:   je     0x400fb4 <phase_3+315>
0000000000400ff0 <+236>:   call   0x4014f7 <explode_bomb>
0000000000400ff5 <+241>:   mov    $0x79,%eax
0000000000400ffa <+246>:   jmp    0x400fb4 <phase_3+315>
0000000000400ffc <+248>:   mov    $0x72,%eax
0000000000401001 <+253>:   cmpl   $0x69,0x14(%rsp)
0000000000401006 <+258>:   je     0x400fb4 <phase_3+315>
0000000000401008 <+260>:   call   0x4014f7 <explode_bomb>
0000000000401009 <+261>:   mov    $0x72,%eax
0000000000401010 <+268>:   jmp    0x400fb4 <phase_3+315>
0000000000401012 <+270>:   mov    $0x79,%eax
0000000000401017 <+275>:   cmp    0xf(%rsp),%al
0000000000401019 <+277>:   je     0x401020 <phase_3+326>
000000000040101b <+279>:   call   0x4014f7 <explode_bomb>
0000000000401020 <+284>:   mov    0x18(%rsp),%rax
0000000000401025 <+289>:   xor    %fs:0x28,%rax
000000000040102e <+298>:   je     0x401035 <phase_3+347>
0000000000401030 <+300>:   call   0x400b00 <__stack_chk_fail@plt>
0000000000401035 <+305>:   add    $0x28,%rsp
0000000000401039 <+309>:   ret
```

.Disas the code and observe

.Where the scan fn gets the slot to take input is moved in prev instruction,so we type x/s 0x4024d6 to know input type as %d %c %d so 3 inputs (integer character integer)

.This codE clearly states a switch statement based on first input and comparision (je) on further inputs.
.Similar to phase 2 give a random input(6 .. ..) which checks case 6 and corresponding inputs are to checked from registers at time of bomb call

.SO OUTPUT WILL BE 6 y 188 where y is ascii code of 121 which we got from code and register information

```
(gdb) b phase_3
Breakpoint 1 at 0x400f11
(gdb) r Solutions.txt
Starting program: /home/aditya/Downloads/SEM2/CA/labs/bomb81-20220627T090213Z-001/bomb81/bomb Solutions.txt
[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!
Phase 1 defused. How about the next one?
That's number 2.  Keep going!
6 y 188

Breakpoint 1, 0x0000000000400f11 in phase_3 ()
(gdb) c
Continuing.
Halfway there!
```

.Add the corresponding input to solutions.txt and save

PHASE 4

```
Breakpoint 1 at 0x4010a4
(gdb) r Solutions.txt
Starting program: /home/aditya/Downloads/SEM2/CA/labs/bomb81-20220627T090213Z-001/bomb81/bomb Solutions.txt
[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!
Phase 1 defused. How about the next one?
That's number 2.  Keep going!
Halfway there!
6 7

Breakpoint 1, 0x00000000004010a4 in phase_4 ()
(gdb) disas
Dump of assembler code for function phase_4:
=> 0x00000000004010a4 <+0>:     sub    $0x18,%rsp
   0x00000000004010a8 <+4>:     mov    %fs:0x28,%rax
   0x00000000004010b1 <+13>:    mov    %rax,0x8(%rsp)
   0x00000000004010b6 <+18>:    xor    %eax,%eax
   0x00000000004010b8 <+20>:    lea    0x4(%rsp),%rcx
   0x00000000004010bd <+25>:    mov    %rsp,%rdx
   0x00000000004010c0 <+28>:    mov    $0x40266f,%esi
   0x00000000004010c5 <+33>:    call   0x400bb0 <__isoc99_sscanf@plt>
   0x00000000004010ca <+38>:    cmp    $0x2,%eax
   0x00000000004010cd <+41>:    jne    0x4010d5 <phase_4+49>
   0x00000000004010cf <+43>:    cmpl   $0xe,(%rsp)
   0x00000000004010d3 <+47>:    jbe    0x4010da <phase_4+54>
   0x00000000004010d5 <+49>:    call   0x4014f7 <explode_bomb>
   0x00000000004010da <+54>:    mov    $0xe,%edx
   0x00000000004010df <+59>:    mov    $0x0,%esi
   0x00000000004010e4 <+64>:    mov    (%rsp),%edi
   0x00000000004010e7 <+67>:    call   0x401071 <func4>
   0x00000000004010ec <+72>:    cmp    $0x15,%eax
   0x00000000004010ef <+75>:    jne    0x4010f8 <phase_4+84>
   0x00000000004010f1 <+77>:    cmpl   $0x15,0x4(%rsp)
   0x00000000004010f6 <+82>:    je     0x4010fd <phase_4+89>
   0x00000000004010f8 <+84>:    call   0x4014f7 <explode_bomb>
   0x00000000004010fd <+89>:    mov    0x8(%rsp),%rax
   0x0000000000401102 <+94>:    xor    %fs:0x28,%rax
   0x000000000040110b <+103>:   je     0x401112 <phase_4+110>
   0x000000000040110d <+105>:   call   0x400b00 <__stack_chk_fail@plt>
   0x0000000000401112 <+110>:   add    $0x18,%rsp
   0x0000000000401116 <+114>:   ret
End of assembler dump.
(gdb) x/d 0x40266f
0x40266f:       622879781
(gdb) x/s 0x40266f
0x40266f:       "%d %d"
(gdb)
```

.disas the code and the input is 2 integers as type is %d %d

.cmp function for 2 iintegers test at<+38> is passed.

.we can enter ni to reach address <+67> and enter si to step into function 'func4' where its been recursively called

.value returned from func4 function stored in reg %eax so we can access the corresponding value 6

.One more observation is first input should be less than or equal to value at $0xe(immediate value) which corresponds to 14

```
0x00000000004010e7 in phase_4 ()
(gdb) si
0x0000000000401071 in func4 ()
(gdb) disas
Dump of assembler code for function func4:
=> 0x0000000000401071 <+0>:     push   %rbx
   0x0000000000401072 <+1>:     mov    %edx,%eax
   0x0000000000401074 <+3>:     sub    %esi,%eax
   0x0000000000401076 <+5>:     mov    %eax,%ebx
   0x0000000000401078 <+7>:     shr    $0x1f,%ebx
   0x000000000040107b <+10>:    add    %ebx,%eax
   0x000000000040107d <+12>:    sar    %eax
   0x000000000040107f <+14>:    lea    (%rax,%rsi,1),%ebx
   0x0000000000401082 <+17>:    cmp    %edi,%ebx
   0x0000000000401084 <+19>:    jle    0x401092 <func4+33>
   0x0000000000401086 <+21>:    lea    -0x1(%rbx),%edx
   0x0000000000401089 <+24>:    call   0x401071 <func4>
   0x000000000040108e <+29>:    add    %ebx,%eax
   0x0000000000401090 <+31>:    jmp    0x4010a2 <func4+49>
   0x0000000000401092 <+33>:    mov    %ebx,%eax
   0x0000000000401094 <+35>:    cmp    %edi,%ebx
   0x0000000000401096 <+37>:    jge    0x4010a2 <func4+49>
   0x0000000000401098 <+39>:    lea    0x1(%rbx),%esi
   0x000000000040109b <+42>:    call   0x401071 <func4>
   0x00000000004010a0 <+47>:    add    %ebx,%eax
   0x00000000004010a2 <+49>:    pop    %rbx
   0x00000000004010a3 <+50>:    ret
End of assembler dump.
(gdb)
```

.The second value comes to be 21
so input 6 21 to Solution.txt in 4th line and save it.

.clear break phases and set b phase_5

.Run Solution.txt to enter phase_5

```
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!
6 21

Breakpoint 1, 0x00000000004010a4 in phase_4 ()
(gdb) c
Continuing.
So you got that one.  Try this one.
```

PHASE 5

.Give random input and disassemble the code,check for input which is again %d %d

```
Breakpoint 1, 0x0000000000401117 in phase_5 ()
(gdb) disas
Dump of assembler code for function phase_5:
=> 0x0000000000401117 <+0>:     sub    $0x18,%rsp
   0x000000000040111b <+4>:     mov    %fs:0x28,%rax
   0x0000000000401124 <+13>:    mov    %rax,0x8(%rsp)
   0x0000000000401129 <+18>:    xor    %eax,%eax
   0x000000000040112b <+20>:    lea    0x4(%rsp),%rcx
   0x0000000000401130 <+25>:    mov    %rsp,%rdx
   0x0000000000401133 <+28>:    mov    $0x40266f,%esi
   0x0000000000401138 <+33>:    call   0x400b60 <__isoc99_sscanf@plt>
   0x000000000040113d <+38>:    cmp    $0x1,%eax
   0x0000000000401140 <+41>:    jg     0x401147 <phase_5+48>
   0x0000000000401142 <+43>:    call   0x4014f7 <explode_bomb>
   0x0000000000401147 <+48>:    mov    (%rsp),%eax
   0x000000000040114a <+51>:    and    $0xf,%eax
   0x000000000040114d <+54>:    mov    %eax,(%rsp)
   0x0000000000401150 <+57>:    cmp    $0xf,%eax
   0x0000000000401153 <+60>:    je     0x401184 <phase_5+109>
   0x0000000000401155 <+62>:    mov    $0x0,%ecx
   0x000000000040115a <+67>:    mov    $0x0,%edx
   0x000000000040115f <+72>:    add    $0x1,%edx
   0x0000000000401162 <+75>:    cltq
   0x0000000000401164 <+77>:    mov    0x402520(,%rax,4),%eax
   0x000000000040116b <+84>:    add    %eax,%ecx
   0x000000000040116d <+86>:    cmp    $0xf,%eax
   0x0000000000401170 <+89>:    jne    0x40115f <phase_5+72>
   0x0000000000401172 <+91>:    movl   $0xf,(%rsp)
   0x0000000000401179 <+98>:    cmp    $0xf,%edx
   0x000000000040117c <+101>:   jne    0x401184 <phase_5+109>
   0x000000000040117e <+103>:   cmp    0x4(%rsp),%ecx
   0x0000000000401182 <+107>:   je     0x401189 <phase_5+114>
   0x0000000000401184 <+109>:   call   0x4014f7 <explode_bomb>
   0x0000000000401189 <+114>:   mov    0x8(%rsp),%rax
   0x000000000040118e <+119>:   xor    %fs:0x28,%rax
   0x0000000000401197 <+128>:   je     0x40119e <phase_5+135>
   0x0000000000401199 <+130>:   call   0x400b00 <__stack_chk_fail@plt>
   0x000000000040119e <+135>:   add    $0x18,%rsp
   0x00000000004011a2 <+139>:   ret
End of assembler dump.
(gdb) x/d 0x40266f
0x40266f:       622879781
(gdb) x/s 0x40266f
0x40266f:       "%d %d"
```

.the next step is for first input,finding second input is when it comes to last bomb call<+103>
where comparision is done btw ecx register and input value

.so we access %rcx reg at bomb call for second value

```
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!
So you got that one.  Try this one.
5 21

Breakpoint 1, 0x0000000000401117 in phase_5 ()
(gdb) ni
0x000000000040111b in phase_5 ()
(gdb) until *Quit
(gdb) until *0x0000000000401184
0x0000000000401184 in phase_5 ()
(gdb) i r
rax            0xf                 15
rbx            0x7fffffffe028      140737488347176
rcx            0x73                115
rdx            0xf                 15
rsi            0x15                21
rdi            0x7fffffffd890      140737488345232
rbp            0x2                 0x2
rsp            0x7fffffffdee0      0x7fffffffdee0
r8             0x0                 0
r9             0x0                 0
r10            0x7ffff7f47ac0      140737353382592
r11            0x7ffff7f483c0      140737353384896
r12            0x7fffffffe028      140737488347176
r13            0x400d56            4197718
r14            0x0                 0
r15            0x7ffff7ffbc40      140737354120256
rip            0x401184            0x401184 <phase_5+109>
eflags         0x212               [ AF IF ]
cs             0x33                51
ss             0x2b                43
ds             0x0                 0
es             0x0                 0
fs             0x0                 0
gs             0x0                 0
k0             0xfffffff0          4294967280
k1             0x2040              8256
k2             0x0                 0
k3             0x0                 0
k4             0x0                 0
k5             0x0                 0
k6             0x0                 0
k7             0x0                 0
(gdb)
```

.give input 5 115 into Solutions.txt and run to diffuse phase_5

.clear other breaks and set b phase_6 to enter our last phase and give some random 6 inputs since function has call <read six integers>

. PHASE 6

```
Reading symbols from bomb...
(gdb) b phase_6
Breakpoint 1 at 0x4011a3
(gdb) r Solutions.txt
Starting program: /home/aditya/Downloads/SEM2/CA/labs/bomb81-20220627T090213Z-001/bomb81/bomb Solutions.txt
[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!
Phase 1 defused. How about the next one?
That's number 2.  Keep going!
Halfway there!
So you got that one.  Try this one.
Good work!  On to the next...
```

.As we can see below in that code that there are three loops running and each loop has own significance. From the address present in the code, we found the datatype of the structure used in this phase is linked list

.The code contains nested loops (see next slide for assembly code)

.From the assembly at the end contains loop

.So for the first time first integer will be checked where its index will be defined at some node.The second loop first fixes the input to following index then after re-ordering it passes the new linked list to the third loop and then the loop has an order in which after re-ordering all the inputs the linked list has to be sorted.

.Therefore its evident that in order of the nodes which %rbx stores data value of linked list nodes from 1 to 6 is the sorting order of the data arrangement itself

.Based on observation,Either the values of linked list traversed in decreasing or increasing order the sorting just is inverse of each other.

.In such way,we tested with 6 such random array inputs starting with 1,2,3,4,5,6 respectively to find node value stored in %rbx

.The output is observed as
3 1 2 6 5 4(descending w.r.t nodes adresses)

```
Breakpoint 1, 0x00000000004011a3 in phase_6 ()
(gdb) disas
Dump of assembler code for function phase_6:
=> 0x00000000004011a3 <+0>:     push   %r14
   0x00000000004011a5 <+2>:     push   %r13
   0x00000000004011a7 <+4>:     push   %r12
   0x00000000004011a9 <+6>:     push   %rbp
   0x00000000004011aa <+7>:     push   %rbx
   0x00000000004011ab <+8>:     sub    $0x60,%rsp
   0x00000000004011af <+12>:    mov    %fs:0x28,%rax
   0x00000000004011b8 <+21>:    mov    %rax,0x58(%rsp)
   0x00000000004011bd <+26>:    xor    %eax,%eax
   0x00000000004011bf <+28>:    mov    %rsp,%rsi
   0x00000000004011c2 <+31>:    call   0x401519 <read_six_numbers>
   0x00000000004011c7 <+36>:    mov    %rsp,%r12
   0x00000000004011ca <+39>:    mov    %rsp,%r13
   0x00000000004011cd <+42>:    mov    $0x0,%r14d
   0x00000000004011d3 <+48>:    mov    %r13,%rbp
   0x00000000004011d6 <+51>:    mov    0x0(%r13),%eax
   0x00000000004011da <+55>:    sub    $0x1,%eax
   0x00000000004011dd <+58>:    cmp    $0x5,%eax
   0x00000000004011e0 <+61>:    jbe    0x4011e7 <phase_6+68>
   0x00000000004011e2 <+63>:    call   0x4014f7 <explode_bomb>
   0x00000000004011e7 <+68>:    add    $0x1,%r14d
   0x00000000004011eb <+72>:    cmp    $0x6,%r14d
   0x00000000004011ef <+76>:    je     0x401712 <phase_6+111>
   0x00000000004011f1 <+78>:    mov    %r14d,%ebx
   0x00000000004011f4 <+81>:    movslq %ebx,%rax
   0x00000000004011f7 <+84>:    mov    (%rsp,%rax,4),%eax
   0x00000000004011fa <+87>:    cmp    %eax,0x0(%rbp)
   0x00000000004011fd <+90>:    jne    0x401204 <phase_6+97>
   0x00000000004011ff <+92>:    call   0x4014f7 <explode_bomb>
   0x0000000000401204 <+97>:    add    $0x1,%ebx
   0x0000000000401207 <+100>:   cmp    $0x5,%ebx
   0x000000000040120a <+103>:   jle    0x4011f1 <phase_6+81>
   0x000000000040120c <+105>:   add    $0x4,%r13
   0x0000000000401210 <+109>:   jmp    0x4011d3 <phase_6+48>
   0x0000000000401212 <+111>:   lea    0x18(%rsp),%rcx
   0x0000000000401217 <+116>:   mov    $0x7,%edx
   0x000000000040121c <+121>:   mov    %edx,%eax
   0x000000000040121e <+123>:   sub    (%r12),%eax
   0x0000000000401222 <+127>:   mov    %eax,(%r12)
   0x0000000000401226 <+131>:   add    $0x4,%r12
   0x000000000040122a <+135>:   cmp    %r12,%rcx
   0x000000000040122d <+138>:   jne    0x40121e <phase_6+121>
   0x000000000040122f <+140>:   mov    $0x0,%esi
   0x0000000000401234 <+145>:   jmp    0x401250 <phase_6+173>
   0x0000000000401236 <+147>:   mov    0x8(%rdx),%rdx
   0x000000000040123a <+151>:   add    $0x1,%eax
   0x000000000040123d <+154>:   cmp    %ecx,%eax
   0x000000000040123f <+156>:   jne    0x401236 <phase_6+147>
   0x0000000000401241 <+158>:   mov    %rdx,0x20(%rsp,%rsi,2)
--Type <RET> for more, q to quit, c to continue without paging--c
   0x0000000000401246 <+163>:   add    $0x4,%rsi
   0x000000000040124a <+167>:   cmp    $0x18,%rsi
   0x000000000040124e <+171>:   je     0x401254 <phase_6+193>
   0x0000000000401250 <+173>:   mov    (%rsp,%rsi,1),%ecx
   0x0000000000401253 <+176>:   mov    $0x1,%eax
   0x0000000000401258 <+181>:   mov    $0x6042f0,%edx
   0x000000000040125d <+186>:   cmp    $0x1,%ecx
   0x0000000000401260 <+189>:   jg     0x401236 <phase_6+147>
   0x0000000000401262 <+191>:   jmp    0x401241 <phase_6+158>
   0x0000000000401264 <+193>:   mov    0x20(%rsp),%rbx
   0x0000000000401269 <+198>:   lea    0x20(%rsp),%rax
   0x000000000040126e <+203>:   lea    0x48(%rsp),%rsi
   0x0000000000401273 <+208>:   mov    %rbx,%rcx
   0x0000000000401276 <+211>:   mov    0x8(%rax),%rdx
   0x000000000040127a <+215>:   mov    %rdx,0x8(%rcx)
   0x000000000040127e <+219>:   add    $0x8,%rax
   0x0000000000401282 <+223>:   mov    %rdx,%rcx
   0x0000000000401285 <+226>:   cmp    %rax,%rsi
   0x0000000000401288 <+229>:   jne    0x401276 <phase_6+211>
   0x000000000040128a <+231>:   movq   $0x0,0x8(%rdx)
   0x0000000000401292 <+239>:   mov    $0x5,%ebp
   0x0000000000401297 <+244>:   mov    0x8(%rbx),%rax
   0x000000000040129b <+248>:   mov    (%rax),%eax
   0x000000000040129d <+250>:   cmp    %eax,(%rbx)
   0x000000000040129f <+252>:   jge    0x4012a6 <phase_6+259>
   0x00000000004012a1 <+254>:   call   0x4014f7 <explode_bomb>
   0x00000000004012a6 <+259>:   mov    0x8(%rbx),%rbx
   0x00000000004012aa <+263>:   sub    $0x1,%ebp
   0x00000000004012ad <+266>:   jne    0x401297 <phase_6+244>
   0x00000000004012af <+268>:   mov    0x58(%rsp),%rax
   0x00000000004012b4 <+273>:   xor    %fs:0x28,%rax
   0x00000000004012bd <+282>:   je     0x4012c4 <phase_6+289>
   0x00000000004012bf <+284>:   call   0x400b00 <__stack_chk_fail@plt>
   0x00000000004012c4 <+289>:   add    $0x60,%rsp
   0x00000000004012c8 <+293>:   pop    %rbx
   0x00000000004012c9 <+294>:   pop    %rbp
   0x00000000004012ca <+295>:   pop    %r12
   0x00000000004012cc <+297>:   pop    %r13
   0x00000000004012ce <+299>:   pop    %r14
   0x00000000004012d0 <+301>:   ret
End of assembler dump.
(gdb)
```

.verify the input to see the bomb diffused

```
Start it from the beginning? (y or n) y
Starting program: /home/aditya/Downloads/SEM2/CA/labs/bomb81-20220627T090213Z-001/bomb81/bomb Solutions.txt
[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!
Phase 1 defused. How about the next one?
That's number 2.  Keep going!
Halfway there!
So you got that one.  Try this one.
Good work!  On to the next...
2 3 4 5 6 1

Breakpoint 1, 0x00000000004011a3 in phase_6 ()
(gdb) until *0x0000000000401276
0x0000000000401276 in phase_6 ()
(gdb) x/d $rdx
0x6042d0 <node4>:       818
(gdb) x/d $rbx
0x6042f0 <node5>:       684
(gdb) r Solutions.txt
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/aditya/Downloads/SEM2/CA/labs/bomb81-20220627T090213Z-001/bomb81/bomb Solutions.txt
[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!
Phase 1 defused. How about the next one?
That's number 2.  Keep going!
Halfway there!
So you got that one.  Try this one.
Good work!  On to the next...
3 4 5 6 1 2

Breakpoint 1, 0x00000000004011a3 in phase_6 ()
(gdb) x/d $rdx
0x6042d0 <input_strings+400>:   540287027
(gdb) until *0x0000000000401276
0x0000000000401276 in phase_6 ()
(gdb) x/d $rdx
0x6042f0 <node5>:       684
(gdb) x/d $rbx
0x6042e0 <node4>:       938
(gdb) r Solutions.txt
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/aditya/Downloads/SEM2/CA/labs/bomb81-20220627T090213Z-001/bomb81/bomb Solutions.txt
[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!
Phase 1 defused. How about the next one?
That's number 2.  Keep going!
Halfway there!
So you got that one.  Try this one.
Good work!  On to the next...
4 5 6 1 2 3

Breakpoint 1, 0x00000000004011a3 in phase_6 ()
(gdb) until *0x0000000000401276
0x0000000000401276 in phase_6 ()
(gdb) x/d $rdx
0x6042e0 <node4>:       938
(gdb) x/d $rbx
0x6042d0 <node3>:       99
(gdb) r Solutions.txt
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/aditya/Downloads/SEM2/CA/labs/bomb81-20220627T090213Z-001/bomb81/bomb Solutions.txt
[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!
Phase 1 defused. How about the next one?
That's number 2.  Keep going!
Halfway there!
So you got that one.  Try this one.
Good work!  On to the next...
5 6 1 2 3 4

Breakpoint 1, 0x00000000004011a3 in phase_6 ()
(gdb) until *0x0000000000401276
0x0000000000401276 in phase_6 ()
(gdb) x/d $rdx
0x6042d0 <node3>:       99
(gdb) x/d $rbx
0x6042c0 <node2>:       416
(gdb) r Solutions.txt
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/aditya/Downloads/SEM2/CA/labs/bomb81-20220627T090213Z-001/bomb81/bomb Solutions.txt
[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!
Phase 1 defused. How about the next one?
That's number 2.  Keep going!
Halfway there!
So you got that one.  Try this one.
Good work!  On to the next...
6 1 2 3 4 5

Breakpoint 1, 0x00000000004011a3 in phase_6 ()
(gdb) until *0x0000000000401276
0x0000000000401276 in phase_6 ()
(gdb) x/d $rdx
0x6042c0 <node3>:       416
(gdb) x/d $rbx
0x6042b0 <node1>:       450
(gdb)
```

completely

.enter this input into
Solutions.txt and save the final file

```
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/aditya/Downloads/SEM2/CA/labs/bomb81-20220627T090213Z-001/bomb81/bomb Solutions.txt
[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!
Phase 1 defused. How about the next one?
That's number 2.  Keep going!
Halfway there!
So you got that one.  Try this one.
Good work!  On to the next...

3 1 2 6 5 4

Breakpoint 1, 0x00000000004011a3 in phase_6 ()
(gdb) c
Continuing.
Congratulations! You've defused the bomb!
[Inferior 1 (process 8595) exited normally]
(gdb)
```

REPORT CONCLUSION

Open | Solutions.txt ~/Downloads/SEM2/CA/labs/bomb81-20220627T090213Z-001/bomb81 | Save

Solutions.txt | bomb.c

```
All your base are belong to us.
0 1 3 6 10 15
6 y 188
6 21
5 115
3 1 2 6 5 4
```

RUNNING THIS FILE Solution.txt IN BOMB FILE AND THE OUTPUT IS

```
aditya@aditya-ThinkBook:~/Downloads/SEM2/CA/labs/bomb81-20220627T090213Z-001/bomb81$ gdb bomb
GNU gdb (Ubuntu 11.1-0ubuntu2) 11.1
Copyright (C) 2021 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:
<https://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) r Solutions.txt
Starting program: /home/aditya/Downloads/SEM2/CA/labs/bomb81-20220627T090213Z-001/bomb81/bomb Solutions.txt
[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!
Phase 1 defused. How about the next one?
That's number 2.  Keep going!
Halfway there!
So you got that one.  Try this one.
Good work!  On to the next...
Congratulations! You've defused the bomb!
[Inferior 1 (process 8736) exited normally]
(gdb)
```

the end .