COMP201 ASSIGNMENT 3 – DEFUSING A BINARY BOMB
FARUK AKSOY-72090

- **PHASE 1:** I used to hints that were giving in the document at the beginning of my approach. “objdump -d” was good starting point to see all the functions. I didn’t explode the bomb thanks to using “b explode_bomb” in every trial. Phase_1 function was comparing input with string. At this point, I used “strings bomb” and looked for suitable strings as input near to the phase_1 and phase_2 strings. Then I realized the sentence “I was trying to give Tina Fey more material.”. Bomb was defused.

```
00000000000015b5 <phase_1>:
    15b5:       f3 0f 1e fa             repz nop %edx
    15b9:       48 83 ec 08             sub    $0x8,%rsp
    15bd:       48 8d 35 bc 1b 00 00    lea    0x1bbc(%rip),%rsi        # 3180 <_IO_stdin_used+0x180>
    15c4:       e8 f1 04 00 00          callq  1aba <strings_not_equal>
    15c9:       85 c0                   test   %eax,%eax
    15cb:       75 05                   jne    15d2 <phase_1+0x1d>
    15cd:       48 83 c4 08             add    $0x8,%rsp
    15d1:       c3                      retq
    15d2:       e8 8a 07 00 00          callq  1d61 <explode_bomb>
    15d7:       eb f4                   jmp    15cd <phase_1+0x18>
```

```
ders froH
m serverI
ATUSH
 []A\
Error: CH
lient unH
able to H
create sH
E ockef
Error: DH
NS is unH
able to H
resolve H
server aH
E(ddref
E,ss
APRL
[]A\A]A^A_
%s: Error: Couldn't open %s
Usage: %s [<input_file>]
That's number 2.  Keep going!
Greetings to COM201 bomb squad! :D
Welcome to my fiendish little bomb. You have 5 phases with
which to blow yourself up. Have a nice day!
Phase 1 defused. How about the next one?
So you got third one. Try this one.
You think you are smart, then check this one.
I was trying to give Tina Fey more material.
Wow! You've defused the secret stage!
flyers
maduiersnfotvbylSo you think you can stop the bomb with ctrl-c, do you?
Initialization error: Running on an illegal host [1]
Your instructor has been notified.
Curses, you've found the secret phase!
But finding it and solving it are quite different...
Congratulations! You've defused the bomb!
Your instructor has been notified and will verify your solution.
Well...
OK. :-)
Invalid phase%s
Initialization error:
defused
exploded
%d:%s:%d:%s
BOOM!!!
The bomb has blown up.
%d %d %d %d %d %d
Error: Premature EOF on stdin
GRADE_BOMB
Error: Input line too long
%d %d %s
DearDrEvil
```

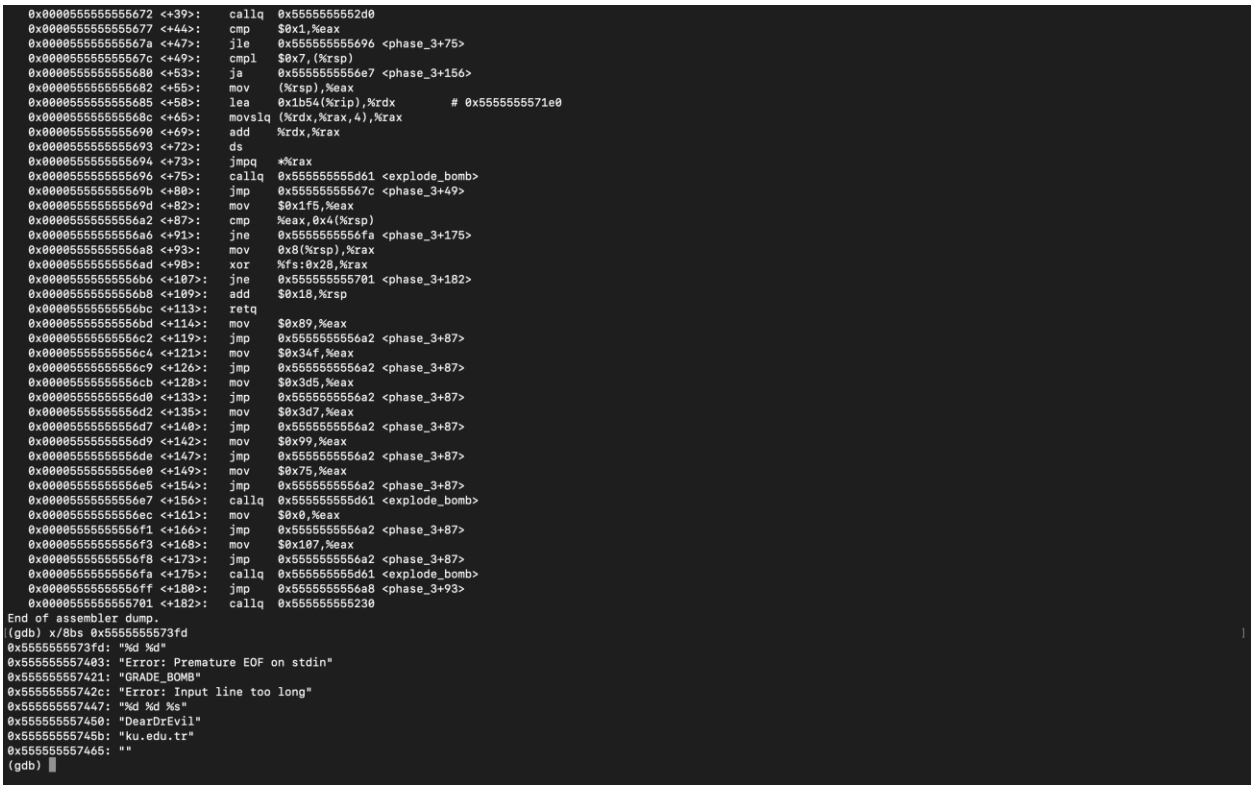
- **PHASE 2:** When I check the phase_2, I realized read_six_numbers function and dive into it by using “disas”. Then by checking the memory addresses I saw that we need to pass 6 numbers to this bomb to defuse. According to phase_2 function, every subsequent

number should be greater than or equal to the previous one plus its index which makes: “1 2 4 7 11 16” as a solution to this phase.

```
Breakpoint 2, 0x00005555555555d9 in phase_2 ()
(gdb) disas
Dump of assembler code for function phase_2:
=> 0x00005555555555d9 <+0>:     repz nop %edx
   0x00005555555555dd <+4>:     push   %rbp
   0x00005555555555de <+5>:     push   %rbx
   0x00005555555555df <+6>:     sub    $0x28,%rsp
   0x00005555555555e3 <+10>:    mov    %fs:0x28,%rax
   0x00005555555555ec <+19>:    mov    %rax,0x18(%rsp)
   0x00005555555555f1 <+24>:    xor    %eax,%eax
   0x00005555555555f3 <+26>:    mov    %rsp,%rsi
   0x00005555555555f6 <+29>:    callq  0x555555555da3 <read_six_numbers>
   0x00005555555555fb <+34>:    cmpl   $0x0,(%rsp)
   0x00005555555555ff <+38>:    js     0x55555555560b <phase_2+50>
   0x0000555555555601 <+40>:    mov    %rsp,%rbp
   0x0000555555555604 <+43>:    mov    $0x1,%ebx
   0x0000555555555609 <+48>:    jmp    0x555555555623 <phase_2+74>
   0x000055555555560b <+50>:    callq  0x555555555d61 <explode_bomb>
   0x0000555555555610 <+55>:    jmp    0x555555555601 <phase_2+40>
   0x0000555555555612 <+57>:    callq  0x555555555d61 <explode_bomb>
   0x0000555555555617 <+62>:    add    $0x1,%ebx
   0x000055555555561a <+65>:    add    $0x4,%rbp
   0x000055555555561e <+69>:    cmp    $0x6,%ebx
   0x0000555555555621 <+72>:    je     0x55555555562f <phase_2+86>
   0x0000555555555623 <+74>:    mov    %ebx,%eax
   0x0000555555555625 <+76>:    add    0x0(%rbp),%eax
   0x0000555555555628 <+79>:    cmp    %eax,0x4(%rbp)
   0x000055555555562b <+82>:    je     0x555555555617 <phase_2+62>
   0x000055555555562d <+84>:    jmp    0x555555555612 <phase_2+57>
   0x000055555555562f <+86>:    mov    0x18(%rsp),%rax
   0x0000555555555634 <+91>:    xor    %fs:0x28,%rax
   0x000055555555563d <+100>:   jne    0x555555555646 <phase_2+109>
   0x000055555555563f <+102>:   add    $0x28,%rsp
   0x0000555555555643 <+106>:   pop    %rbx
   0x0000555555555644 <+107>:   pop    %rbp
   0x0000555555555645 <+108>:   retq
   0x0000555555555646 <+109>:   callq  0x555555555230
End of assembler dump.
   0x0000555555555601 <+40>:    mov    %rsp,%rbp
   0x0000555555555604 <+43>:    mov    $0x1,%ebx
   0x0000555555555609 <+48>:    jmp    0x555555555623 <phase_2+74>
   0x000055555555560b <+50>:    callq  0x555555555d61 <explode_bomb>
   0x0000555555555610 <+55>:    jmp    0x555555555601 <phase_2+40>
   0x0000555555555612 <+57>:    callq  0x555555555d61 <explode_bomb>
   0x0000555555555617 <+62>:    add    $0x1,%ebx
   0x000055555555561a <+65>:    add    $0x4,%rbp
   0x000055555555561e <+69>:    cmp    $0x6,%ebx
   0x0000555555555621 <+72>:    je     0x55555555562f <phase_2+86>
   0x0000555555555623 <+74>:    mov    %ebx,%eax
   0x0000555555555625 <+76>:    add    0x0(%rbp),%eax
   0x0000555555555628 <+79>:    cmp    %eax,0x4(%rbp)
   0x000055555555562b <+82>:    je     0x555555555617 <phase_2+62>
   0x000055555555562d <+84>:    jmp    0x555555555612 <phase_2+57>
   0x000055555555562f <+86>:    mov    0x18(%rsp),%rax
   0x0000555555555634 <+91>:    xor    %fs:0x28,%rax
   0x000055555555563d <+100>:   jne    0x555555555646 <phase_2+109>
   0x000055555555563f <+102>:   add    $0x28,%rsp
   0x0000555555555643 <+106>:   pop    %rbx
   0x0000555555555644 <+107>:   pop    %rbp
   0x0000555555555645 <+108>:   retq
   0x0000555555555646 <+109>:   callq  0x555555555230
End of assembler dump.
(gdb) next
Single stepping until exit from function phase_2,
which has no line number information.

Breakpoint 3, 0x0000555555555da3 in read_six_numbers ()
(gdb) disas
Dump of assembler code for function read_six_numbers:
=> 0x0000555555555da3 <+0>:     repz nop %edx
   0x0000555555555da7 <+4>:     sub    $0x8,%rsp
   0x0000555555555dab <+8>:     mov    %rsi,%rdx
   0x0000555555555dae <+11>:    lea    0x4(%rsi),%rcx
   0x0000555555555db2 <+15>:    lea    0x14(%rsi),%rax
   0x0000555555555db6 <+19>:    push   %rax
   0x0000555555555db7 <+20>:    lea    0x10(%rsi),%rax
   0x0000555555555dbb <+24>:    push   %rax
   0x0000555555555dbc <+25>:    lea    0xc(%rsi),%r9
   0x0000555555555dc0 <+29>:    lea    0x8(%rsi),%r8
   0x0000555555555dc4 <+33>:    lea    0x1626(%rip),%rsi        # 0x5555555573f1
   0x0000555555555dcb <+40>:    mov    $0x0,%eax
   0x0000555555555dd0 <+45>:    callq  0x5555555552d0
   0x0000555555555dd5 <+50>:    add    $0x10,%rsp
   0x0000555555555dd9 <+54>:    cmp    $0x5,%eax
   0x0000555555555ddc <+57>:    jle    0x555555555de3 <read_six_numbers+64>
   0x0000555555555dde <+59>:    add    $0x8,%rsp
   0x0000555555555de2 <+63>:    retq
   0x0000555555555de3 <+64>:    callq  0x555555555d61 <explode_bomb>
End of assembler dump.
(gdb)
```

```
(gdb) x/16bs 0x5555555573f1
0x5555555573f1: "%d %d %d %d %d %d"
0x555555557403: "Error: Premature EOF on stdin"
0x555555557421: "GRADE_BOMB"
0x55555555742c: "Error: Input line too long"
0x555555557447: "%d %d %s"
0x555555557450: "DearDrEvil"
0x55555555745b: "ku.edu.tr"
0x555555557465: ""
0x555555557466: ""
0x555555557467: ""
0x555555557468: "Program timed out after %d seconds\n"
0x55555555748c: ""
0x55555555748d: ""
0x55555555748e: ""
0x55555555748f: ""
0x555555557490: "Error: HTTP request failed with error %d: %s"
(gdb)
```

- **PHASE 3:** When I entered to phase_3, I checked the memory addresses “0x5555555573fd” and “0x5555555571e0”. I realized that function that 2 numbers as inputs. After tha, in phase_3 function at instruction **<+49>**, the code compares if the provided index is between 0 and 7 (inclusive). The specific values for each index were hard-coded using the jmpq *%rax indirect jump. After calculating the value based on the index, it was checking against the target value. We convert hexadecimal values to decimal and the input values were: 0-137, 1-501, 2-847, 3-981, 4-983, 5-153, 6-117, 7-263. I used 1-501 pair.



```
   0x0000555555555672 <+39>:    callq  0x5555555552d0
   0x0000555555555677 <+44>:    cmp    $0x1,%eax
   0x000055555555567a <+47>:    jle    0x555555555696 <phase_3+75>
   0x000055555555567c <+49>:    cmpl   $0x7,(%rsp)
   0x0000555555555680 <+53>:    ja     0x5555555556e7 <phase_3+156>
   0x0000555555555682 <+55>:    mov    (%rsp),%eax
   0x0000555555555685 <+58>:    lea    0x1b54(%rip),%rdx        # 0x5555555571e0
   0x000055555555568c <+65>:    movslq (%rdx,%rax,4),%rax
   0x0000555555555690 <+69>:    add    %rdx,%rax
   0x0000555555555693 <+72>:    ds
   0x0000555555555694 <+73>:    jmpq   *%rax
   0x0000555555555696 <+75>:    callq  0x555555555d61 <explode_bomb>
   0x000055555555569b <+80>:    jmp    0x555555555567c <phase_3+49>
   0x000055555555569d <+82>:    mov    $0x1f5,%eax
   0x00005555555556a2 <+87>:    cmp    %eax,0x4(%rsp)
   0x00005555555556a6 <+91>:    jne    0x5555555556fa <phase_3+175>
   0x00005555555556a8 <+93>:    mov    0x8(%rsp),%rax
   0x00005555555556ad <+98>:    xor    %fs:0x28,%rax
   0x00005555555556b6 <+107>:   jne    0x555555555701 <phase_3+182>
   0x00005555555556b8 <+109>:   add    $0x18,%rsp
   0x00005555555556bc <+113>:   retq
   0x00005555555556bd <+114>:   mov    $0x89,%eax
   0x00005555555556c2 <+119>:   jmp    0x5555555556a2 <phase_3+87>
   0x00005555555556c4 <+121>:   mov    $0x34f,%eax
   0x00005555555556c9 <+126>:   jmp    0x5555555556a2 <phase_3+87>
   0x00005555555556cb <+128>:   mov    $0x3d5,%eax
   0x00005555555556d0 <+133>:   jmp    0x5555555556a2 <phase_3+87>
   0x00005555555556d2 <+135>:   mov    $0x3d7,%eax
   0x00005555555556d7 <+140>:   jmp    0x5555555556a2 <phase_3+87>
   0x00005555555556d9 <+142>:   mov    $0x99,%eax
   0x00005555555556de <+147>:   jmp    0x5555555556a2 <phase_3+87>
   0x00005555555556e0 <+149>:   mov    $0x75,%eax
   0x00005555555556e5 <+154>:   jmp    0x5555555556a2 <phase_3+87>
   0x00005555555556e7 <+156>:   callq  0x555555555d61 <explode_bomb>
   0x00005555555556ec <+161>:   mov    $0x0,%eax
   0x00005555555556f1 <+166>:   jmp    0x5555555556a2 <phase_3+87>
   0x00005555555556f3 <+168>:   mov    $0x107,%eax
   0x00005555555556f8 <+173>:   jmp    0x5555555556a2 <phase_3+87>
   0x00005555555556fa <+175>:   callq  0x555555555d61 <explode_bomb>
   0x00005555555556ff <+180>:   jmp    0x5555555556a8 <phase_3+93>
   0x0000555555555701 <+182>:   callq  0x555555555230
End of assembler dump.
(gdb) x/8bs 0x5555555573fd
0x5555555573fd: "%d %d"
0x555555557403: "Error: Premature EOF on stdin"
0x555555557421: "GRADE_BOMB"
0x55555555742c: "Error: Input line too long"
0x555555557447: "%d %d %s"
0x555555557450: "DearDrEvil"
0x55555555745b: "ku.edu.tr"
0x555555557465: ""
(gdb)
```

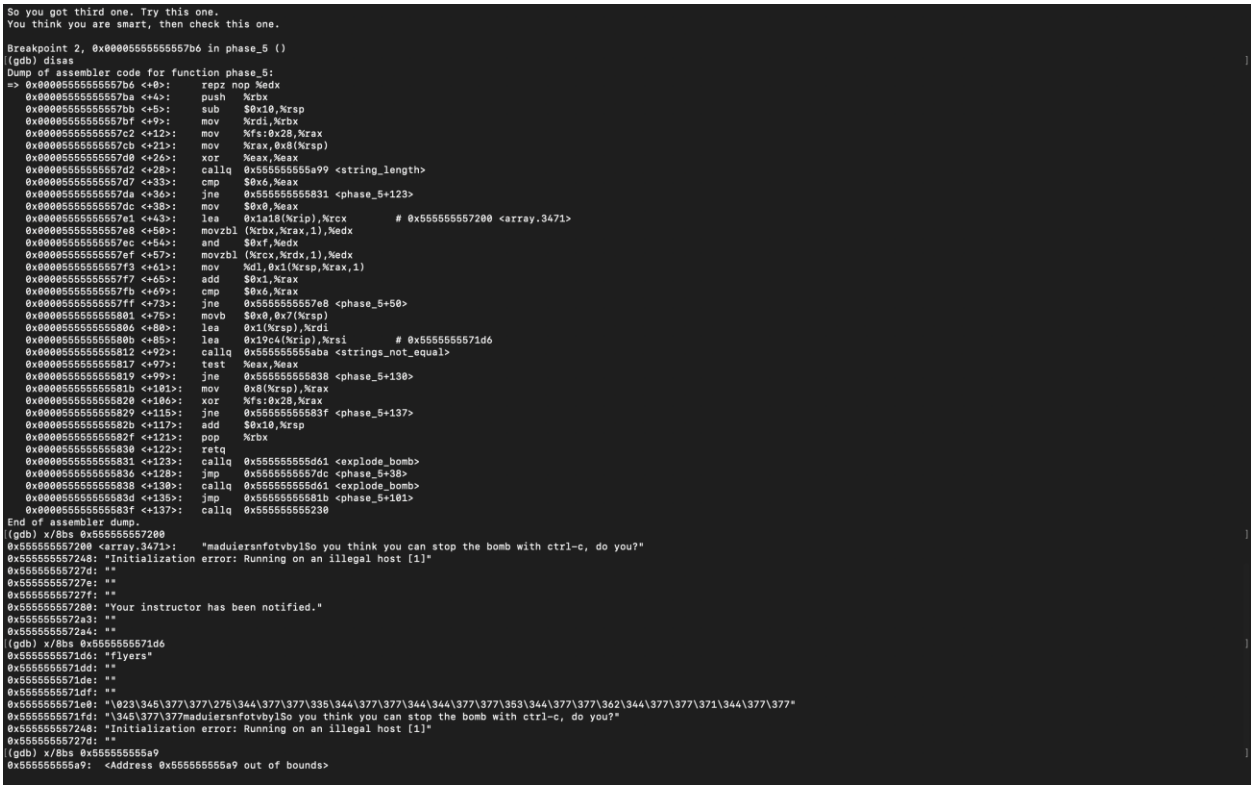
- **PHASE 4**: The phase_4 function is designed to test the input validation process by leveraging the recursive func4 function. In phase_4, the program accepts two numbers, checks that the first falls within a specified range (2 to 4), and then calls func4 with the first number and a base value of 9. func4 recursively computes values based on a binary-like structure, combining results from two subproblems to return an aggregated result. To solve this phase, I discovered that providing 352 as the first input and expecting 4 as the output involved finding a pattern that aligns 352 with 4 by manipulating func4's recursive pattern. After analyzing and reverse-engineering the assembly logic, the solution required identifying the combination that satisfied the recursive relationships, leading to the successful defusal of the bomb.

```
[[fsartik19@linux01 bomb38]$ gdb bomb
GNU gdb (GDB) Red Hat Enterprise Linux 7.6.1-120.el7
Copyright (C) 2013 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-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /Users/fsartik19/bomb38/bomb...done.
(gdb) b explode_bomb
Breakpoint 1 at 0x1d61
(gdb) b phase_4
Breakpoint 2 at 0x1741
(gdb) r psol.txt
Starting program: /Users/fsartik19/bomb38/bomb psol.txt
Greetings to COM201 bomb squad! :D
Welcome to my fiendish little bomb. You have 5 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!
So you got third one. Try this one.

Breakpoint 2, 0x0000555555555741 in phase_4 ()
(gdb) disas
Dump of assembler code for function phase_4:
=> 0x0000555555555741 <+0>:     repz nop %edx
   0x0000555555555745 <+4>:     sub    $0x18,%rsp
   0x0000555555555749 <+8>:     mov    %fs:0x28,%rax
   0x0000555555555752 <+17>:    mov    %rax,0x8(%rsp)
   0x0000555555555757 <+22>:    xor    %eax,%eax
   0x0000555555555759 <+24>:    mov    %rsp,%rcx
   0x000055555555575c <+27>:    lea    0x4(%rsp),%rdx
   0x0000555555555761 <+32>:    lea    0x1c95(%rip),%rsi        # 0x5555555573fd
   0x0000555555555768 <+39>:    callq  0x55555555552d0
   0x000055555555576d <+44>:    cmp    $0x2,%eax
   0x0000555555555770 <+47>:    jne    0x555555555577d <phase_4+60>
   0x0000555555555772 <+49>:    mov    (%rsp),%eax
   0x0000555555555775 <+52>:    sub    $0x2,%eax
   0x0000555555555778 <+55>:    cmp    $0x2,%eax
   0x000055555555577b <+58>:    jbe    0x5555555555782 <phase_4+65>
   0x000055555555577d <+60>:    callq  0x555555555d61 <explode_bomb>
   0x0000555555555782 <+65>:    mov    (%rsp),%esi
   0x0000555555555785 <+68>:    mov    $0x9,%edi
   0x000055555555578a <+73>:    callq  0x555555555706 <func4>
   0x000055555555578f <+78>:    cmp    %eax,0x4(%rsp)
   0x0000555555555793 <+82>:    jne    0x55555555557aa <phase_4+105>
   0x0000555555555795 <+84>:    mov    0x8(%rsp),%rax
   0x000055555555579a <+89>:    xor    %fs:0x28,%rax
   0x00005555555557a3 <+98>:    jne    0x55555555557b1 <phase_4+112>
   0x00005555555557a5 <+100>:   add    $0x18,%rsp
   0x00005555555557a9 <+104>:   retq
   0x00005555555557aa <+105>:   callq  0x555555555d61 <explode_bomb>
   0x00005555555557af <+110>:   jmp    0x5555555555795 <phase_4+84>
   0x00005555555557b1 <+112>:   callq  0x5555555555230
End of assembler dump.
(gdb) b func4
Breakpoint 3 at 0x555555555706
(gdb) next
Single stepping until exit from function phase_4,
which has no line number information.
```

```
   0x0000555555555793 <+82>:    jne    0x55555555557aa <phase_4+105>
   0x0000555555555795 <+84>:    mov    0x8(%rsp),%rax
   0x000055555555579a <+89>:    xor    %fs:0x28,%rax
   0x00005555555557a3 <+98>:    jne    0x55555555557b1 <phase_4+112>
   0x00005555555557a5 <+100>:   add    $0x18,%rsp
   0x00005555555557a9 <+104>:   retq
   0x00005555555557aa <+105>:   callq  0x555555555d61 <explode_bomb>
   0x00005555555557af <+110>:   jmp    0x5555555555795 <phase_4+84>
   0x00005555555557b1 <+112>:   callq  0x5555555555230
End of assembler dump.
(gdb) b func4
Breakpoint 3 at 0x555555555706
(gdb) next
Single stepping until exit from function phase_4,
which has no line number information.

Breakpoint 3, 0x0000555555555706 in func4 ()
(gdb) disas
Dump of assembler code for function func4:
=> 0x0000555555555706 <+0>:     repz nop %edx
   0x000055555555570a <+4>:     mov    $0x0,%eax
   0x000055555555570f <+9>:     test   %edi,%edi
   0x0000555555555711 <+11>:    jle    0x5555555555740 <func4+58>
   0x0000555555555713 <+13>:    push   %r12
   0x0000555555555715 <+15>:    push   %rbp
   0x0000555555555716 <+16>:    push   %rbx
   0x0000555555555717 <+17>:    mov    %edi,%ebx
   0x0000555555555719 <+19>:    mov    %esi,%ebp
   0x000055555555571b <+21>:    mov    %esi,%eax
   0x000055555555571d <+23>:    cmp    $0x1,%edi
   0x0000555555555720 <+26>:    je     0x555555555573b <func4+53>
   0x0000555555555722 <+28>:    lea    -0x1(%rdi),%edi
   0x0000555555555725 <+31>:    callq  0x555555555706 <func4>
   0x000055555555572a <+36>:    lea    (%rax,%rbp,1),%r12d
   0x000055555555572e <+40>:    lea    -0x2(%rbx),%edi
   0x0000555555555731 <+43>:    mov    %ebp,%esi
   0x0000555555555733 <+45>:    callq  0x555555555706 <func4>
   0x0000555555555738 <+50>:    add    %r12d,%eax
   0x000055555555573b <+53>:    pop    %rbx
   0x000055555555573c <+54>:    pop    %rbp
   0x000055555555573d <+55>:    pop    %r12
   0x000055555555573f <+57>:    retq
   0x0000555555555740 <+58>:    retq
End of assembler dump.
(gdb) x/8bs 0x555555555740
0x555555555740 <func4+58>:      "\303\363\017\036\372H\203\354\030dH\213\004%("
0x555555555750 <phase_4+15>:    ""
0x555555555751 <phase_4+16>:    ""
0x555555555752 <phase_4+17>:    "H\211D$\b1\300H\211\341H\215T$\004H\215\065\225\034"
0x555555555767 <phase_4+38>:    ""
0x555555555768 <phase_4+39>:    "\350c\373\377\377\203\370\002u\v\213\004$\203\350\002\203\370\002v\005\350\337\005"
0x555555555781 <phase_4+64>:    ""
0x555555555782 <phase_4+65>:    "\213\064$\277\t"
(gdb) x/8bs 0x55555555573b
0x55555555573b <func4+53>:      "[]A\\\303\303\363\017\036\372H\203\354\030dH\213\004%("
0x555555555750 <phase_4+15>:    ""
0x555555555751 <phase_4+16>:    ""
0x555555555752 <phase_4+17>:    "H\211D$\b1\300H\211\341H\215T$\004H\215\065\225\034"
0x555555555767 <phase_4+38>:    ""
0x555555555768 <phase_4+39>:    "\350c\373\377\377\203\370\002u\v\213\004$\203\350\002\203\370\002v\005\350\337\005"
0x555555555781 <phase_4+64>:    ""
0x555555555782 <phase_4+65>:    "\213\064$\277\t"
(gdb)
```

- **PHASE 5:** The pointer is stored in %rbx, and the string length is verified to be six characters by comparing %eax to 0x6. The function initializes %eax to zero and then translates each

character through a mapping table using %edx, applying a mask (%edx AND $0xf) to restrict values to four bits. It uses this masked value as an index into a predefined translation array referenced via %rcx. The mapped characters are stored sequentially in a stack buffer, with %rax used for indexing. The final string is compared against the predefined target string using the strings_not_equal function, with %rdi holding the newly constructed string and %rsi pointing to the target. If the comparison fails (%eax non-zero), the bomb explodes via explode_bomb. The correct input string "ionefg" produces the expected translated string that matches the target, passing the check and safely defusing the bomb.




- **SECRET PHASE:** Firstly I have used “objdump -d bomb” to see all functions that can be related to secret phase. Then I realized that it was in the phase_defused. While I was looking at the contents of memory addresses by using “x/8bs” in the phase_defused, I realized that there was a “%d %d %s” line for the input of phase_4 which makes us to trigger the secret_phase. Then I have entered by changing my input for phase_4 as “352 4 DearDrEvil”. After phase_5, when I entered to phase_defused I could access to inside of secret_phase. There was a fun7 recursive function which was creating a binary tree and returns some value. This value was “6” in my scenerio and I need to find the node that makes fun7 to return 6. I looked the node values by using “x/3x” and create the tree which was starting from 36 and goes on. After that, when we check the tree logic: left subtree was doubling the value of %eax and right subtree was doubling + adding 1 to the %eax.

By tracking this logic, going “left-right-right” was giving the node that’s value was “35” in my scenerio.

```
0000000000001f30 <phase_defused>:
    1f30:       f3 0f 1e fa             repz nop %edx
    1f34:       48 83 ec 78             sub    $0x78,%rsp
    1f38:       64 48 8b 04 25 28 00    mov    %fs:0x28,%rax
    1f3f:       00 00
    1f41:       48 89 44 24 68          mov    %rax,0x68(%rsp)
    1f46:       31 c0                   xor    %eax,%eax
    1f48:       bf 01 00 00 00          mov    $0x1,%edi
    1f4d:       e8 1c fd ff ff          callq  1c6e <send_msg>
    1f52:       83 3d 53 37 00 00 05    cmpl   $0x5,0x3753(%rip)        # 56ac <num_input_strings>
    1f59:       74 19                   je     1f74 <phase_defused+0x44>
    1f5b:       48 8b 44 24 68          mov    0x68(%rsp),%rax
    1f60:       64 48 33 04 25 28 00    xor    %fs:0x28,%rax
    1f67:       00 00
    1f69:       0f 85 84 00 00 00       jne    1ff3 <phase_defused+0xc3>
    1f6f:       48 83 c4 78             add    $0x78,%rsp
    1f73:       c3                      retq
    1f74:       48 8d 4c 24 0c          lea    0xc(%rsp),%rcx
    1f79:       48 8d 54 24 08          lea    0x8(%rsp),%rdx
    1f7e:       4c 8d 44 24 10          lea    0x10(%rsp),%r8
    1f83:       48 8d 35 bd 14 00 00    lea    0x14bd(%rip),%rsi        # 3447 <array.3471+0x247>
    1f8a:       48 8d 3d 1f 38 00 00    lea    0x381f(%rip),%rdi        # 57b0 <input_strings+0xf0>
    1f91:       b8 00 00 00 00          mov    $0x0,%eax
    1f96:       e8 35 f3 ff ff          callq  12d0 <.plt.got+0x100>
    1f9b:       83 f8 03                cmp    $0x3,%eax
    1f9e:       74 1a                   je     1fba <phase_defused+0x8a>
    1fa0:       48 8d 3d 61 13 00 00    lea    0x1361(%rip),%rdi        # 3308 <array.3471+0x108>
    1fa7:       e8 64 f2 ff ff          callq  1210 <.plt.got+0x40>
    1fac:       48 8d 3d 85 13 00 00    lea    0x1385(%rip),%rdi        # 3338 <array.3471+0x138>
    1fb3:       e8 58 f2 ff ff          callq  1210 <.plt.got+0x40>
    1fb8:       eb a1                   jmp    1f5b <phase_defused+0x2b>
    1fba:       48 8d 7c 24 10          lea    0x10(%rsp),%rdi
    1fbf:       48 8d 35 8a 14 00 00    lea    0x148a(%rip),%rsi        # 3450 <array.3471+0x250>
    1fc6:       e8 ef fa ff ff          callq  1aba <strings_not_equal>
    1fcb:       85 c0                   test   %eax,%eax
    1fcd:       75 d1                   jne    1fa0 <phase_defused+0x70>
    1fcf:       48 8d 3d d2 12 00 00    lea    0x12d2(%rip),%rdi        # 32a8 <array.3471+0xa8>
    1fd6:       e8 35 f2 ff ff          callq  1210 <.plt.got+0x40>
    1fdb:       48 8d 3d ee 12 00 00    lea    0x12ee(%rip),%rdi        # 32d0 <array.3471+0xd0>
    1fe2:       e8 29 f2 ff ff          callq  1210 <.plt.got+0x40>
    1fe7:       b8 00 00 00 00          mov    $0x0,%eax
    1fec:       e8 bc f9 ff ff          callq  19ad <secret_phase>
    1ff1:       eb ad                   jmp    1fa0 <phase_defused+0x70>
    1ff3:       e8 38 f2 ff ff          callq  1230 <.plt.got+0x60>

   0x0000555555555fec <+188>:   callq  0x5555555559ad <secret_phase>
   0x0000555555555ff1 <+193>:   jmp    0x5555555555fa0 <phase_defused+112>
   0x0000555555555ff3 <+195>:   callq  0x555555555230
End of assembler dump.
(gdb) x/8bs 0x555555557447
0x555555557447: "%d %d %s"
0x555555557450: "DearDrEvil"
0x55555555745b: "ku.edu.tr"
0x555555557465: ""
0x555555557466: ""
0x555555557467: ""
0x555555557468: "Program timed out after %d seconds\n"
0x55555555748c: ""
(gdb) x/8bs 0x5555555597b0
0x5555555597b0 <input_strings+240>:     ""
0x5555555597b1 <input_strings+241>:     ""
0x5555555597b2 <input_strings+242>:     ""
0x5555555597b3 <input_strings+243>:     ""
0x5555555597b4 <input_strings+244>:     ""
0x5555555597b5 <input_strings+245>:     ""
0x5555555597b6 <input_strings+246>:     ""
0x5555555597b7 <input_strings+247>:     ""
(gdb) x/8bs 0x555555557308
0x555555557308: "Congratulations! You've defused the bomb!"
0x555555557332: ""
0x555555557333: ""
0x555555557334: ""
0x555555557335: ""
0x555555557336: ""
0x555555557337: ""
0x555555557338: "Your instructor has been notified and will verify your solution."
(gdb) x/8bs 0x5555555596ac
0x5555555596ac <num_input_strings>:     "\001"
0x5555555596ae <num_input_strings+2>:   ""
0x5555555596af <num_input_strings+3>:   ""
0x5555555596b0 <infile>:        "\020\240UUUU"
0x5555555596b7 <infile+7>:      ""
0x5555555596b8: ""
0x5555555596b9: ""
0x5555555596ba: ""
(gdb) x/8bs 0x555555557338
0x555555557338: "Your instructor has been notified and will verify your solution."
0x555555557379: "Well..."
0x555555557381: "OK. :-)"
0x555555557389: "Invalid phase%s\n"
0x55555555739a: "Initialization error:\n%s\n"
0x5555555573b4: "defused"
0x5555555573bc: "exploded"
0x5555555573c5: "%d:%s:%d:%s"
(gdb) x/8bs 0x555555557450
0x555555557450: "DearDrEvil"
0x55555555745b: "ku.edu.tr"
```

```
    1967:       e8 c4 f8 ff ff          callq  1230 <.plt.got+0x60>

000000000000196c <fun7>:
    196c:       f3 0f 1e fa             repz nop %edx
    1970:       48 85 ff                test   %rdi,%rdi
    1973:       74 32                   je     19a7 <fun7+0x3b>
    1975:       48 83 ec 08             sub    $0x8,%rsp
    1979:       8b 17                   mov    (%rdi),%edx
    197b:       39 f2                   cmp    %esi,%edx
    197d:       7f 0c                   jg     198b <fun7+0x1f>
    197f:       b8 00 00 00 00          mov    $0x0,%eax
    1984:       75 12                   jne    1998 <fun7+0x2c>
    1986:       48 83 c4 08             add    $0x8,%rsp
    198a:       c3                      retq
    198b:       48 8b 7f 08             mov    0x8(%rdi),%rdi
    198f:       e8 d8 ff ff ff          callq  196c <fun7>
    1994:       01 c0                   add    %eax,%eax
    1996:       eb ee                   jmp    1986 <fun7+0x1a>
    1998:       48 8b 7f 10             mov    0x10(%rdi),%rdi
    199c:       e8 cb ff ff ff          callq  196c <fun7>
    19a1:       8d 44 00 01             lea    0x1(%rax,%rax,1),%eax
    19a5:       eb df                   jmp    1986 <fun7+0x1a>
    19a7:       b8 ff ff ff ff          mov    $0xffffffff,%eax
    19ac:       c3                      retq

00000000000019ad <secret_phase>:
    19ad:       f3 0f 1e fa             repz nop %edx
    19b1:       53                      push   %rbx
    19b2:       e8 31 04 00 00          callq  1de8 <read_line>
    19b7:       48 89 c7                mov    %rax,%rdi
    19ba:       ba 0a 00 00 00          mov    $0xa,%edx
    19bf:       be 00 00 00 00          mov    $0x0,%esi
    19c4:       e8 e7 f8 ff ff          callq  12b0 <.plt.got+0xe0>
    19c9:       48 89 c3                mov    %rax,%rbx
    19cc:       8d 40 ff                lea    -0x1(%rax),%eax
    19cf:       3d e8 03 00 00          cmp    $0x3e8,%eax
    19d4:       77 26                   ja     19fc <secret_phase+0x4f>
    19d6:       89 de                   mov    %ebx,%esi
    19d8:       48 8d 3d 71 37 00 00    lea    0x3771(%rip),%rdi        # 5150 <n1>
    19df:       e8 88 ff ff ff          callq  196c <fun7>
    19e4:       83 f8 06                cmp    $0x6,%eax
    19e7:       75 1a                   jne    1a03 <secret_phase+0x56>
    19e9:       48 8d 3d c0 17 00 00    lea    0x17c0(%rip),%rdi        # 31b0 <_IO_stdin_used+0x1b0>
    19f0:       e8 1b f8 ff ff          callq  1210 <.plt.got+0x40>
    19f5:       e8 36 05 00 00          callq  1f30 <phase_defused>
    19fa:       5b                      pop    %rbx
    19fb:       c3                      retq
    19fc:       e8 60 03 00 00          callq  1d61 <explode_bomb>
    1a01:       eb d3                   jmp    19d6 <secret_phase+0x29>
    1a03:       e8 59 03 00 00          callq  1d61 <explode_bomb>
    1a08:       eb df                   jmp    19e9 <secret_phase+0x3c>
```

```
fs             0x0      0
gs             0x0      0
(gdb) x/8dx 0x555555559150
0x555555559150 <n1>:    0x24    0x00    0x00    0x00    0x00    0x00    0x00    0x00
(gdb) x/16dx 0x555555559150
0x555555559150 <n1>:    0x24    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x555555559158 <n1+8>:  0x70    0x91    0x55    0x55    0x55    0x55    0x00    0x00
(gdb) x/60gx
0x555555559160 <n1+16>: 0x0000555555559190      0x0000000000000000
0x555555559170 <n21>:   0x0000000000000008      0x00005555555591f0
0x555555559180 <n21+16>:        0x00005555555591b0      0x0000000000000000
0x555555559190 <n22>:   0x0000000000000032      0x00005555555591d0
0x5555555591a0 <n22+16>:        0x0000555555559210      0x0000000000000000
0x5555555591b0 <n32>:   0x0000000000000016      0x00005555555590b0
0x5555555591c0 <n32+16>:        0x0000555555559070      0x0000000000000000
0x5555555591d0 <n33>:   0x000000000000002d      0x0000555555559010
0x5555555591e0 <n33+16>:        0x00005555555590d0      0x0000000000000000
0x5555555591f0 <n31>:   0x0000000000000006      0x0000555555559030
0x555555559200 <n31+16>:        0x0000555555559090      0x0000000000000000
0x555555559210 <n34>:   0x000000000000006b      0x0000555555559050
0x555555559220 <n34+16>:        0x00005555555590f0      0x0000000000000000
0x555555559230 <node1>: 0x00000001000003df      0x0000555555559240
0x555555559240 <node2>: 0x00000002000002aa      0x0000555555559250
0x555555559250 <node3>: 0x000000030000005d      0x0000555555559260
0x555555559260 <node4>: 0x0000000400000341      0x0000555555559270
0x555555559270 <node5>: 0x00000005000002f1      0x0000555555559110
0x555555559280 <host_table>:    0x000055555555745b      0x0000000000000000
0x555555559290 <host_table+16>: 0x0000000000000000      0x0000000000000000
0x5555555592a0 <host_table+32>: 0x0000000000000000      0x0000000000000000
0x5555555592b0 <host_table+48>: 0x0000000000000000      0x0000000000000000
0x5555555592c0 <host_table+64>: 0x0000000000000000      0x0000000000000000
0x5555555592d0 <host_table+80>: 0x0000000000000000      0x0000000000000000
0x5555555592e0 <host_table+96>: 0x0000000000000000      0x0000000000000000
0x5555555592f0 <host_table+112>:        0x0000000000000000      0x0000000000000000
0x555555559300 <host_table+128>:        0x0000000000000000      0x0000000000000000
0x555555559310 <host_table+144>:        0x0000000000000000      0x0000000000000000
0x555555559320 <host_table+160>:        0x0000000000000000      0x0000000000000000
0x555555559330 <host_table+176>:        0x0000000000000000      0x0000000000000000
(gdb) x/60gx 0x555555559150
0x555555559150 <n1>:    0x0000000000000024      0x0000555555559170
0x555555559160 <n1+16>: 0x0000555555559190      0x0000000000000000
0x555555559170 <n21>:   0x0000000000000008      0x00005555555591f0
0x555555559180 <n21+16>:        0x00005555555591b0      0x0000000000000000
0x555555559190 <n22>:   0x0000000000000032      0x00005555555591d0
0x5555555591a0 <n22+16>:        0x0000555555559210      0x0000000000000000
0x5555555591b0 <n32>:   0x0000000000000016      0x00005555555590b0
0x5555555591c0 <n32+16>:        0x0000555555559070      0x0000000000000000
0x5555555591d0 <n33>:   0x000000000000002d      0x0000555555559010
0x5555555591e0 <n33+16>:        0x00005555555590d0      0x0000000000000000
0x5555555591f0 <n31>:   0x0000000000000006      0x0000555555559030
0x555555559200 <n31+16>:        0x0000555555559090      0x0000000000000000
0x555555559210 <n34>:   0x000000000000006b      0x0000555555559050
```

```
   0x000055555555559c4 <+23>:    callq  0x5555555552b0
   0x000055555555559c9 <+28>:    mov    %rax,%rbx
   0x000055555555559cc <+31>:    lea    -0x1(%rax),%eax
   0x000055555555559cf <+34>:    cmp    $0x3e8,%eax
   0x000055555555559d4 <+39>:    ja     0x5555555559fc <secret_phase+79>
   0x000055555555559d6 <+41>:    mov    %ebx,%esi
   0x000055555555559d8 <+43>:    lea    0x3771(%rip),%rdi        # 0x555555559150 <n1>
   0x000055555555559df <+50>:    callq  0x555555555596c <fun7>
   0x000055555555559e4 <+55>:    cmp    $0x6,%eax
   0x000055555555559e7 <+58>:    jne    0x555555555a03 <secret_phase+86>
   0x000055555555559e9 <+60>:    lea    0x17c0(%rip),%rdi        # 0x5555555571b0
   0x000055555555559f0 <+67>:    callq  0x555555555210
   0x000055555555559f5 <+72>:    callq  0x555555555f30 <phase_defused>
   0x000055555555559fa <+77>:    pop    %rbx
   0x000055555555559fb <+78>:    retq
   0x000055555555559fc <+79>:    callq  0x555555555d61 <explode_bomb>
   0x00005555555555a01 <+84>:    jmp    0x5555555559d6 <secret_phase+41>
   0x00005555555555a03 <+86>:    callq  0x555555555d61 <explode_bomb>
   0x00005555555555a08 <+91>:    jmp    0x5555555559e9 <secret_phase+60>
End of assembler dump.
(gdb) next
Single stepping until exit from function secret_phase,
which has no line number information.
35

Breakpoint 4, 0x000055555555596c in fun7 ()
(gdb) next
Single stepping until exit from function fun7,
which has no line number information.

Breakpoint 4, 0x000055555555596c in fun7 ()
(gdb) next
Single stepping until exit from function fun7,
which has no line number information.

Breakpoint 4, 0x000055555555596c in fun7 ()
(gdb) next
Single stepping until exit from function fun7,
which has no line number information.

Breakpoint 4, 0x000055555555596c in fun7 ()
(gdb) next
Single stepping until exit from function fun7,
which has no line number information.
0x00005555555559e4 in secret_phase ()
(gdb) next
Single stepping until exit from function secret_phase,
which has no line number information.
Wow! You've defused the secret stage!

Breakpoint 2, 0x0000555555555f30 in phase_defused ()
(gdb) next
```

**NOTES:**

1) Screenshots have different colors because of the dark mode that depends on the time of the day.

```
[fsartik19@linux01 bomb38]$ gdb bomb
GNU gdb (GDB) Red Hat Enterprise Linux 7.6.1-120.el7
Copyright (C) 2013 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-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /Users/fsartik19/bomb38/bomb...done.
(gdb) b explode_bomb
Breakpoint 1 at 0x1d61
(gdb) r psol.txt
Starting program: /Users/fsartik19/bomb38/bomb psol.txt
Greetings to COM201 bomb squad! :D
Welcome to my fiendish little bomb. You have 5 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!
So you got third one. Try this one.
You think you are smart, then check this one.
Congratulations! You've defused the bomb!
Your instructor has been notified and will verify your solution.
[Inferior 1 (process 30140) exited normally]
(gdb)
```