Bomb **Lab**

The Adventures of
Bomb 229 and Bomb 322

Pre-Lab

- Before Starting:
  - Run **touch .gdbinit**
    - This creates a file of commands that will automatically be run upon starting gdb
  - emacs/vim into .gdbinit
  - Type **b explode_bomb** and exit
    - This makes it pretty much impossible to lose points
  - If a safe-path error appears when starting gdb, copy the authorization path it provides and paste it into ~/.gdbinit (different file)
  - Highly recommend updating this file with breakpoints (**b phase_1, b read_six_numbers,** etc.) as you progress
- Terminal Commands:
  - **echo "*string_input*" > *file_name*.txt**
    - Creates a new file called *file_name* with *string_input* as the first line
    - Run this with your phase 1 solution as soon as you get it
  - **echo "*string_input*" >> *file_name*.txt**
    - Appends an existing file called *file_name* with *string_input* on a new line
    - Run this with each additional solution
  - **objdump -d bomb**
    - Gives an overview of the entire assembly code for the program
- Basic GDB Commands:
  - **gdb bomb**

        - Begins to run debugging process
    - **r**
        - Runs the program from the start
    - **r *file_name*.txt**
        - Runs the program, using each line of *file_name* as an input for phases
        - Very helpful to skip past solved phases
    - **layout asm**
        - Gives a layout that allows you to debug and have assembly code in separate windows
    - **focus asm** or **cmd**
        - Switches the window your arrow keys operate in
    - **Ctrl + X + A**
        - Toggles the asm view, required when inputting strings
    - **q**
        - Quits GDB, deleting all breakpoints
    - **Ctrl + C**
        - Exits certain processes
- Breakpoint Commands:
    - **b *function_name***
        - Creates a breakpoint at *function_name*, which can be obtained from assembly or the C code
    - **b **memory_address***
        - Creates a breakpoint at *memory_address*, which can be obtained from assembly
    - **i b**
        - Outputs all active breakpoints and their corresponding memory addresses/function names
    - **c**

        - Moves on to next breakpoint/function end
    - **delete**
        - Deletes all breakpoints
    - **delete *breakpoint_number***
        - Deletes specifically breakpoint *breakpoint_number*
- Register Commands:
    - **i r**
        - Outputs the contents of all registers
    - **i r *register_name***
        - Outputs the contents of ***register_name***
- Other Useful Commands:
    - **disas**
        - Creates an object dump of the current function
    - **disas *function_name***
        - Creates an object dump of *function_name*
    - **x/x $*register_name*** or ***memory_address***
        - Displays the contents at the given location in hex format
    - **x/d $*register_name*** or ***memory_address***
        - Displays the contents at the given location in decimal format
    - **x/s $*register_name*** or ***memory_address***
        - Displays the contents at the given location in string format
    - **x/*num_bytes* $*register_name*** or ***memory_address***
        - Displays the contents of the next *num_bytes* from the given location

Phase 1: String Compare

```
phase_1 Dump =>
   0x0000000000400efe <+0>:      sub    $0x8,%rsp
   0x0000000000400f02 <+4>:      mov    $0x4023f0,%esi
   0x0000000000400f07 <+9>:      callq  0x4012d6 <strings_not_equal>
   0x0000000000400f0c <+14>:     test   %eax,%eax
   0x0000000000400f0e <+16>:     jne    0x400f15 <phase_1+23>
   0x0000000000400f10 <+18>:     add    $0x8,%rsp
   0x0000000000400f14 <+22>:     retq
   0x0000000000400f15 <+23>:     callq  0x4014f1 <explode_bomb>
   0x0000000000400f1a <+28>:     jmp    0x400f10 <phase_1+18>
strings_not_equal Dump =>
   0x00000000004012d6 <+0>:      push   %r12
   0x00000000004012d8 <+2>:      push   %rbp
   0x00000000004012d9 <+3>:      push   %rbx
   0x00000000004012da <+4>:      mov    %rdi,%rbx
   0x00000000004012dd <+7>:      mov    %rsi,%rbp
   0x00000000004012e0 <+10>:     callq  0x4012b9 <string_length>
   0x00000000004012e5 <+15>:     mov    %eax,%r12d
   0x00000000004012e8 <+18>:     mov    %rbp,%rdi
   0x00000000004012eb <+21>:     callq  0x4012b9 <string_length>
   0x00000000004012f0 <+26>:     mov    %eax,%edx // comparing to given string
   0x00000000004012f2 <+28>:     mov    $0x1,%eax
   0x00000000004012f7 <+33>:     cmp    %edx,%r12d
   0x00000000004012fa <+36>:     jne    0x40132d <strings_not_equal+87>
   0x00000000004012fc <+38>:     movzbl (%rbx),%edx
   0x00000000004012ff <+41>:     test   %dl,%dl
   0x0000000000401301 <+43>:     je     0x401321 <strings_not_equal+75>
   0x0000000000401303 <+45>:     mov    $0x0,%eax
   0x0000000000401308 <+50>:     cmp    %dl,0x0(%rbp,%rax,1)
   0x000000000040130c <+54>:     jne    0x401328 <strings_not_equal+82>
   0x000000000040130e <+56>:     add    $0x1,%rax
   0x0000000000401312 <+60>:     movzbl (%rbx,%rax,1),%edx
   0x0000000000401316 <+64>:     test   %dl,%dl
   0x0000000000401318 <+66>:     jne    0x401308 <strings_not_equal+50>
   0x000000000040131a <+68>:     mov    $0x0,%eax
   0x000000000040131f <+73>:     jmp    0x40132d <strings_not_equal+87>
   0x0000000000401321 <+75>:     mov    $0x0,%eax
   0x0000000000401326 <+80>:     jmp    0x40132d <strings_not_equal+87>
   0x0000000000401328 <+82>:     mov    $0x1,%eax
   0x000000000040132d <+87>:     pop    %rbx
   0x000000000040132e <+88>:     pop    %rbp
   0x000000000040132f <+89>:     pop    %r12
   0x0000000000401331 <+91>:     retq
```

Phase 2: Six Numbers

```
phase_2 Dump =>
   0x0000000000400f1c <+0>:      push   %rbx
   0x0000000000400f1d <+1>:      sub    $0x20,%rsp
   0x0000000000400f21 <+5>:      mov    %rsp,%rsi
   0x0000000000400f24 <+8>:      callq  0x401527 <read_six_numbers>
   0x0000000000400f29 <+13>:     cmpl   $0x0,(%rsp) // rax is the # of inputs
   0x0000000000400f2d <+17>:     js     0x400f36 <phase_2+26>
   0x0000000000400f2f <+19>:     mov    $0x1,%ebx // ebx rewritten to 1
   0x0000000000400f34 <+24>:     jmp    0x400f4c <phase_2+48> // jump to +48
   0x0000000000400f36 <+26>:     callq  0x4014f1 <explode_bomb>
   0x0000000000400f3b <+31>:     jmp    0x400f2f <phase_2+19>
   0x0000000000400f3d <+33>:     callq  0x4014f1 <explode_bomb>
   0x0000000000400f42 <+38>:     add    $0x1,%rbx
   0x0000000000400f46 <+42>:     cmp    $0x6,%rbx // this loop must repeat 5 times
   0x0000000000400f4a <+46>:     je     0x400f59 <phase_2+61>
   0x0000000000400f4c <+48>:     mov    %ebx,%eax // eax rewritten to rbx
   0x0000000000400f4e <+50>:     add    -0x4(%rsp,%rbx,4),%eax // rbx indexes array
   0x0000000000400f52 <+54>:     cmp    %eax,(%rsp,%rbx,4) // comp. cur. input to next
   0x0000000000400f55 <+57>:     je     0x400f42 <phase_2+38> // loops if equal
   0x0000000000400f57 <+59>:     jmp    0x400f3d <phase_2+33>
   0x0000000000400f59 <+61>:     add    $0x20,%rsp
   0x0000000000400f5d <+65>:     pop    %rbx
   0x0000000000400f5e <+66>:     retq
read_six_numbers Dump =>
   0x0000000000401527 <+0>:      sub    $0x8,%rsp
   0x000000000040152b <+4>:      mov    %rsi,%rdx // rsi holds value 0
   0x000000000040152e <+7>:      lea    0x4(%rsi),%rcx
   0x0000000000401532 <+11>:     lea    0x14(%rsi),%rax
   0x0000000000401536 <+15>:     push   %rax // -1 pushed onto stack
   0x0000000000401537 <+16>:     lea    0x10(%rsi),%rax
   0x000000000040153b <+20>:     push   %rax // 72 pushed onto stack
   0x000000000040153c <+21>:     lea    0xc(%rsi),%r9
   0x0000000000401540 <+25>:     lea    0x8(%rsi),%r8
   0x0000000000401544 <+29>:     mov    $0x4026f9,%esi // esi is now 37
   0x0000000000401549 <+34>:     mov    $0x0,%eax // zeroes out eax
   0x000000000040154e <+39>:     callq  0x400c50 <__isoc99_sscanf@plt> // check length
   0x0000000000401553 <+44>:     add    $0x10,%rsp
   0x0000000000401557 <+48>:     cmp    $0x5,%eax // checks if input is 6+ digits
   0x000000000040155a <+51>:     jle    0x401561 <read_six_numbers+58>
   0x000000000040155c <+53>:     add    $0x8,%rsp
   0x0000000000401560 <+57>:     retq
   0x0000000000401561 <+58>:     callq  0x4014f1 <explode_bomb>
```

Phase 3: Jump Table

```
phase_3 Dump =>
   0x0000000000400f5f <+0>:     sub    $0x18,%rsp // irrelevant
   0x0000000000400f63 <+4>:     lea    0x8(%rsp),%rcx // stores esp in rcx
   0x0000000000400f68 <+9>:     lea    0xc(%rsp),%rdx // stores sp in rdx
   0x0000000000400f6d <+14>:    mov    $0x402705,%esi // answer is in form %d %d
   0x0000000000400f72 <+19>:    mov    $0x0,%eax // irrelevant
   0x0000000000400f77 <+24>:    callq  0x400c50 <__isoc99_sscanf@plt>
   0x0000000000400f7c <+29>:    cmp    $0x1,%eax // input is length >1
   0x0000000000400f7f <+32>:    jle    0x400f93 <phase_3+52>
   0x0000000000400f81 <+34>:    cmpl   $0x7,0xc(%rsp) // 1st element is below 7
   0x0000000000400f86 <+39>:    ja     0x400fd4 <phase_3+117>
   0x0000000000400f88 <+41>:    mov    0xc(%rsp),%eax // eax holds the 1st element
   0x0000000000400f8c <+45>:    jmpq   *0x402460(,%rax,8) // jump table w/ rax
   0x0000000000400f93 <+52>:    callq  0x4014f1 <explode_bomb>
   0x0000000000400f98 <+57>:    jmp    0x400f81 <phase_3+34>
   0x0000000000400f9a <+59>:    mov    $0x133,%eax // 1st = 1 => 2nd = 307
   0x0000000000400f9f <+64>:    cmp    %eax,0x8(%rsp) // comparing to 2nd input
   0x0000000000400fa3 <+68>:    jne    0x400fe7 <phase_3+136>
   0x0000000000400fa5 <+70>:    add    $0x18,%rsp
   0x0000000000400fa9 <+74>:    retq
   0x0000000000400faa <+75>:    mov    $0x3bc,%eax // 1st = 2 => 2nd = 956
   0x0000000000400faf <+80>:    jmp    0x400f9f <phase_3+64>
   0x0000000000400fb1 <+82>:    mov    $0x2e0,%eax // 1st = 3 => 2nd = 752
   0x0000000000400fb6 <+87>:    jmp    0x400f9f <phase_3+64>
   0x0000000000400fb8 <+89>:    mov    $0x2d4,%eax // 1st = 4 => 2nd = 740
   0x0000000000400fbd <+94>:    jmp    0x400f9f <phase_3+64>
   0x0000000000400fbf <+96>:    mov    $0x291,%eax // 1st = 5 => 2nd = 657
   0x0000000000400fc4 <+101>:   jmp    0x400f9f <phase_3+64>
   0x0000000000400fc6 <+103>:   mov    $0x2d3,%eax // 1st = 6 => 2nd = 723
   0x0000000000400fcb <+108>:   jmp    0x400f9f <phase_3+64>
   0x0000000000400fcd <+110>:   mov    $0x1aa,%eax // 1st = 7 => 2nd = 426
   0x0000000000400fd2 <+115>:   jmp    0x400f9f <phase_3+64>
   0x0000000000400fd4 <+117>:   callq  0x4014f1 <explode_bomb>
   0x0000000000400fd9 <+122>:   mov    $0x0,%eax
   0x0000000000400fde <+127>:   jmp    0x400f9f <phase_3+64>
   0x0000000000400fe0 <+129>:   mov    $0x2b3,%eax // 1st = 0 => 2nd = 691
   0x0000000000400fe5 <+134>:   jmp    0x400f9f <phase_3+64>
   0x0000000000400fe7 <+136>:   callq  0x4014f1 <explode_bomb>
   0x0000000000400fec <+141>:   jmp    0x400fa5 <phase_3+70>
```

Phase 4: Recursion

```
phase_4 Dump =>
   0x0000000000401025 <+0>:      sub    $0x18,%rsp
   0x0000000000401029 <+4>:      lea    0xc(%rsp),%rcx
   0x000000000040102e <+9>:      lea    0x8(%rsp),%rdx
   0x0000000000401033 <+14>:     mov    $0x402705,%esi // answer is in form %d %d
   0x0000000000401038 <+19>:     mov    $0x0,%eax
   0x000000000040103d <+24>:     callq  0x400c50 <__isoc99_sscanf@plt>
   0x0000000000401042 <+29>:     cmp    $0x2,%eax // answer is length 2
   0x0000000000401045 <+32>:     jne    0x401053 <phase_4+46>
   0x0000000000401047 <+34>:     mov    0xc(%rsp),%eax // 2nd input into eax
   0x000000000040104b <+38>:     sub    $0x2,%eax
   0x000000000040104e <+41>:     cmp    $0x2,%eax // 2nd input - 2 <= 2 (2-4)
   0x0000000000401051 <+44>:     jbe    0x401058 <phase_4+51>
   0x0000000000401053 <+46>:     callq  0x4014f1 <explode_bomb>
   0x0000000000401058 <+51>:     mov    0xc(%rsp),%esi // 2nd input into esi
   0x000000000040105c <+55>:     mov    $0x8,%edi
   0x0000000000401061 <+60>:     callq  0x400fee <func4>
   0x0000000000401066 <+65>:     cmp    %eax,0x8(%rsp) // return value = 1st input
   0x000000000040106a <+69>:     jne    0x401071 <phase_4+76>
   0x000000000040106c <+71>:     add    $0x18,%rsp
   0x0000000000401070 <+75>:     retq
   0x0000000000401071 <+76>:     callq  0x4014f1 <explode_bomb>
   0x0000000000401076 <+81>:     jmp    0x40106c <phase_4+71>
func4 Dump =>
   0x0000000000400fee <+0>:      mov    $0x0,%eax // irrelevant
   0x0000000000400ff3 <+5>:      test   %edi,%edi // base case
   0x0000000000400ff5 <+7>:      jle    0x401024 <func4+54> // return when edi is <= 0
   0x0000000000400ff7 <+9>:      push   %r12
   0x0000000000400ff9 <+11>:     push   %rbp
   0x0000000000400ffa <+12>:     push   %rbx
   0x0000000000400ffb <+13>:     mov    %edi,%ebx // edi is 8 on 1st loop
   0x0000000000400ffd <+15>:     mov    %esi,%ebp // esi is the 2nd input
   0x0000000000400fff <+17>:     mov    %esi,%eax
   0x0000000000401001 <+19>:     cmp    $0x1,%edi // edi always 8 on 1st loop
   0x0000000000401004 <+22>:     je     0x40101f <func4+49> // return case
   0x0000000000401006 <+24>:     lea    -0x1(%rdi),%edi // rdi--
   0x0000000000401009 <+27>:     callq  0x400fee <func4> // recursive call w/ rdi--
   0x000000000040100e <+32>:     lea    (%rax,%rbp,1),%r12d
   0x0000000000401012 <+36>:     lea    -0x2(%rbx),%edi
   0x0000000000401015 <+39>:     mov    %ebp,%esi
   0x0000000000401017 <+41>:     callq  0x400fee <func4> // recursive call
   0x000000000040101c <+46>:     add    %r12d,%eax // eax modified by recursive calls
   0x000000000040101f <+49>:     pop    %rbx
```

```
0x0000000000401020 <+50>:     pop    %rbp
0x0000000000401021 <+51>:     pop    %r12
0x0000000000401023 <+53>:     retq
0x0000000000401024 <+54>:     retq
```

Phase 5: C-Strings

phase_5 Dump =>

```
   0x0000000000401078 <+0>:      push   %rbx // holds input
   0x0000000000401079 <+1>:      mov    %rdi,%rbx
   0x000000000040107c <+4>:      callq  0x4012b9 <string_length>
   0x0000000000401081 <+9>:      cmp    $0x6,%eax // answer is string of length 6
   0x0000000000401084 <+12>:     jne    0x4010af <phase_5+55>
   0x0000000000401086 <+14>:     mov    %rbx,%rax
   0x0000000000401089 <+17>:     lea    0x6(%rbx),%rdi // end of string into rdi
   0x000000000040108d <+21>:     mov    $0x0,%ecx
   0x0000000000401092 <+26>:     movzbl (%rax),%edx // first char of input into edx
   0x0000000000401095 <+29>:     and    $0xf,%edx // edx reduced to lower 4 bytes
   0x0000000000401098 <+32>:     add    0x4024a0(,%rdx,4),%ecx // maps chars to ints
   0x000000000040109f <+39>:     add    $0x1,%rax // iteration thru string
   0x00000000004010a3 <+43>:     cmp    %rdi,%rax // loop thru each char of string
   0x00000000004010a6 <+46>:     jne    0x401092 <phase_5+26> // loop structure
   0x00000000004010a8 <+48>:     cmp    $0x2e,%ecx // char->int map adds to 46
   0x00000000004010ab <+51>:     jne    0x4010b6 <phase_5+62>
   0x00000000004010ad <+53>:     pop    %rbx
   0x00000000004010ae <+54>:     retq
   0x00000000004010af <+55>:     callq  0x4014f1 <explode_bomb>
   0x00000000004010b4 <+60>:     jmp    0x401086 <phase_5+14>
   0x00000000004010b6 <+62>:     callq  0x4014f1 <explode_bomb>
   0x00000000004010bb <+67>:     jmp    0x4010ad <phase_5+53>
```

Phase 6: Linked List

---

phase_6 Dump =>

```
   0x00000000004010bd <+0>:      push   %r13
   0x00000000004010bf <+2>:      push   %r12
   0x00000000004010c1 <+4>:      push   %rbp
   0x00000000004010c2 <+5>:      push   %rbx
   0x00000000004010c3 <+6>:      sub    $0x58,%rsp
   0x00000000004010c7 <+10>:     lea    0x30(%rsp),%rsi
   0x00000000004010cc <+15>:     callq  0x401527 <read_six_numbers> // answer is 6 #s
   0x00000000004010d1 <+20>:     lea    0x30(%rsp),%r12 // input array begins at r12
   0x00000000004010d6 <+25>:     mov    $0x1,%r13d
   0x00000000004010dc <+31>:     jmp    0x401106 <phase_6+73>
   0x00000000004010de <+33>:     callq  0x4014f1 <explode_bomb>
   0x00000000004010e3 <+38>:     jmp    0x401115 <phase_6+88>
   0x00000000004010e5 <+40>:     add    $0x1,%rbx // rbx is the index
   0x00000000004010e9 <+44>:     cmp    $0x5,%ebx // if index > 5, jump
   0x00000000004010ec <+47>:     jg     0x4010fe <phase_6+65>
   0x00000000004010ee <+49>:     mov    0x30(%rsp,%rbx,4),%eax // indexing of array
   0x00000000004010f2 <+53>:     cmp    %eax,0x0(%rbp) // rbp is prev. input
   0x00000000004010f5 <+56>:     jne    0x4010e5 <phase_6+40> // no duplicate inputs
   0x00000000004010f7 <+58>:     callq  0x4014f1 <explode_bomb>
   0x00000000004010fc <+63>:     jmp    0x4010e5 <phase_6+40>
   0x00000000004010fe <+65>:     add    $0x1,%r13
   0x0000000000401102 <+69>:     add    $0x4,%r12 // increments input
   0x0000000000401106 <+73>:     mov    %r12,%rbp // rbp set to current input
   0x0000000000401109 <+76>:     mov    (%r12),%eax // 1st input into eax
   0x000000000040110d <+80>:     sub    $0x1,%eax // eax is 1st input - 1
   0x0000000000401110 <+83>:     cmp    $0x5,%eax // input - 1 must be less than 5
   0x0000000000401113 <+86>:     ja     0x4010de <phase_6+33>
   0x0000000000401115 <+88>:     cmp    $0x5,%r13d // r13 is 1 on 1st loop
   0x0000000000401119 <+92>:     jg     0x401120 <phase_6+99> // all inputs < 6
   0x000000000040111b <+94>:     mov    %r13,%rbx
   0x000000000040111e <+97>:     jmp    0x4010ee <phase_6+49> // loop structure
   0x0000000000401120 <+99>:     mov    $0x0,%esi
   0x0000000000401125 <+104>:    mov    0x30(%rsp,%rsi,4),%ecx // ecx is indexed
   0x0000000000401129 <+108>:    mov    $0x1,%eax
   0x000000000040112e <+113>:    mov    $0x6042f0,%edx // linked list head into edx
   0x0000000000401133 <+118>:    cmp    $0x1,%ecx
   0x0000000000401136 <+121>:    jle    0x401143 <phase_6+134> // if input is <= 1
   0x0000000000401138 <+123>:    mov    0x8(%rdx),%rdx // next node
   0x000000000040113c <+127>:    add    $0x1,%eax
   0x000000000040113f <+130>:    cmp    %ecx,%eax
   0x0000000000401141 <+132>:    jne    0x401138 <phase_6+123>
   0x0000000000401143 <+134>:    mov    %rdx,(%rsp,%rsi,8)
   0x0000000000401147 <+138>:    add    $0x1,%rsi
```

```
0x000000000040114b <+142>:     cmp    $0x6,%rsi
0x000000000040114f <+146>:     jne    0x401125 <phase_6+104>
0x0000000000401151 <+148>:     mov    (%rsp),%rbx
0x0000000000401155 <+152>:     mov    0x8(%rsp),%rax
0x000000000040115a <+157>:     mov    %rax,0x8(%rbx)
0x000000000040115e <+161>:     mov    0x10(%rsp),%rdx
0x0000000000401163 <+166>:     mov    %rdx,0x8(%rax) // orders nodes by input
0x0000000000401167 <+170>:     mov    0x18(%rsp),%rax
0x000000000040116c <+175>:     mov    %rax,0x8(%rdx)
0x0000000000401170 <+179>:     mov    0x20(%rsp),%rdx
0x0000000000401175 <+184>:     mov    %rdx,0x8(%rax)
0x0000000000401179 <+188>:     mov    0x28(%rsp),%rax
0x000000000040117e <+193>:     mov    %rax,0x8(%rdx)
0x0000000000401182 <+197>:     movq   $0x0,0x8(%rax)
0x000000000040118a <+205>:     mov    $0x5,%ebp
0x000000000040118f <+210>:     jmp    0x40119a <phase_6+221>
0x0000000000401191 <+212>:     mov    0x8(%rbx),%rbx
0x0000000000401195 <+216>:     sub    $0x1,%ebp
0x0000000000401198 <+219>:     je     0x4011ab <phase_6+238> // defused
0x000000000040119a <+221>:     mov    0x8(%rbx),%rax // rax is next node
0x000000000040119e <+225>:     mov    (%rax),%eax // eax is val of next node
0x00000000004011a0 <+227>:     cmp    %eax,(%rbx) // cur node val is <= to next
0x00000000004011a2 <+229>:     jle    0x401191 <phase_6+212> // must be <= to eax
0x00000000004011a4 <+231>:     callq  0x4014f1 <explode_bomb>
0x00000000004011a9 <+236>:     jmp    0x401191 <phase_6+212>
0x00000000004011ab <+238>:     add    $0x58,%rsp
0x00000000004011af <+242>:     pop    %rbx
0x00000000004011b0 <+243>:     pop    %rbp
0x00000000004011b1 <+244>:     pop    %r12
0x00000000004011b3 <+246>:     pop    %r13
0x00000000004011b5 <+248>:     retq
```

Process

1. Analyze the dump, marking any obvious loops, jumps, explode_bombs, etc.
2. Search for the format of the answer to narrow down test cases
3. Look for the registers that store your input
4. Follow these registers and take note of how your input is modified/broken apart
5. Search for the phase’s defuse condition
6. Backtrace through the register modifications to arrive at a correct input