Using objdump -d bomb > bomb.s
chmod 777 bomb
gdb bomb

1. Set up break point:
(gdb) b explode_bomb
Breakpoint 1 at 0x40142a
(gdb) b phase_1
Breakpoint 1 at 0x400e8d
(gdb) r
Starting program: /home/Desktop/bomb003/bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
try

//when doing ni in the gdb terminal until the output we come to see that the break point at explode_bomb function is being executed which means the user input string that we gave is not equal to the system input string so we search for the system input stored in some address

Breakpoint 1, 0x000000000040142a in explode_bomb()
(gdb) disas phase_1
Dump of assembler code for function phase_1:
=> 0x0000000000400e8d <+0>: sub $0x8,%rsp //building stack frame with 8 more bytes
0x0000000000400e91 <+4>: mov $0x4023b0,%esi //check what does this address stores
0x0000000000400e96 <+9>: call 0x40132b <strings_not_equal> //it compares the user input string with stored string
0x0000000000400e9b <+14>: test %eax,%eax
0x0000000000400e9d <+16>: je 0x400ea4 <phase_1+23>
0x0000000000400e9f <+18>: call 0x40142a <explode_bomb>
0x0000000000400ea4 <+23>: add $0x8,%rsp
0x0000000000400ea8 <+27>: ret
End of assembler dump.

since the address 0x4023b0 stores something, to check the value the following command is used to print the value in string formate:

(gdb) x/s 0x4023b0
0x4023b0: "Border relations with Canada have never been better."

So the address stores this string and moves into %esi, and will be passed to the function called <strings_not_equal>
To see what <strings_not_equal> does:

(gdb) disas strings_not_equal

Dump of assembler code for function strings_not_equal:
=> 0x000000000040132b <+0>: push %r12
0x000000000040132d <+2>: push %rbp
0x000000000040132e <+3>: push %rbx
0x000000000040132f <+4>: mov %rdi,%rbx //user input moved
0x0000000000401332 <+7>: mov %rsi,%rbp //system input moved

```
   0x0000000000401335 <+10>:    call   0x40130d <string_length>// checking the length of the
user input

    Dump of assembler code for function string_length:
      => 0x000000000040130d <+0>:   cmpb   $0x0,(%rdi) // compare to user input in bite
        0x0000000000401310 <+3>:   je     0x401325 <string_length+24> //jump if equal else skip
        0x0000000000401312 <+5>:   mov    $0x0,%eax
        0x0000000000401317 <+10>:  add    $0x1,%rdi //removing char by 1 byte
        0x000000000040131b <+14>:  add    $0x1,%eax  //keeping count of char in user input
        0x000000000040131e <+17>:  cmpb   $0x0,(%rdi) //loop until all the character is removed
and count
        0x0000000000401321 <+20>:  jne    0x401317 <string_length+10>
        0x0000000000401323 <+22>:  repz ret // return to the previous function
        0x0000000000401325 <+24>:  mov    $0x0,%eax
        0x000000000040132a <+29>:  ret
    End of assembler dump.

  0x000000000040133a <+15>:     mov    %eax,%r12d //length of the user input is moved
  0x000000000040133d <+18>:     mov    %rbp,%rdi //system input
  0x0000000000401340 <+21>:     call   0x40130d <string_length> // checking the length of the
user input
  0x0000000000401345 <+26>:     mov    $0x1,%edx //move flag 1 if not same
  0x000000000040134a <+31>:     cmp    %eax,%r12d // comparing length of system input ans
user input
  0x000000000040134d <+34>:     jne    0x40138b <strings_not_equal+96> //jump if not equal
  0x000000000040134f <+36>:     movzbl (%rbx),%eax
  0x0000000000401352 <+39>:     test   %al,%al //comparing bit wise
  0x0000000000401354 <+41>:     je     0x401378 <strings_not_equal+77> jump if equal
  0x0000000000401356 <+43>:     cmp    0x0(%rbp),%al
  0x0000000000401359 <+46>:     je     0x401362 <strings_not_equal+55>
  0x000000000040135b <+48>:     jmp    0x40137f <strings_not_equal+84>
  0x000000000040135d <+50>:     cmp    0x0(%rbp),%al
  0x0000000000401360 <+53>:     jne    0x401386 <strings_not_equal+91>
  0x0000000000401362 <+55>:     add    $0x1,%rbx
  0x0000000000401366 <+59>:     add    $0x1,%rbp
--Type <RET> for more, q to quit, c to continue without paging--c
  0x000000000040136a <+63>:     movzbl (%rbx),%eax
  0x000000000040136d <+66>:     test   %al,%al
  0x000000000040136f <+68>:     jne    0x40135d <strings_not_equal+50>
  0x0000000000401371 <+70>:     mov    $0x0,%edx
  0x0000000000401376 <+75>:     jmp    0x40138b <strings_not_equal+96>
  0x0000000000401378 <+77>:     mov    $0x0,%edx //move flag 1 if not same
  0x000000000040137d <+82>:     jmp    0x40138b <strings_not_equal+96>
  0x000000000040137f <+84>:     mov    $0x1,%edx
  0x0000000000401384 <+89>:     jmp    0x40138b <strings_not_equal+96>
  0x0000000000401386 <+91>:     mov    $0x1,%edx
  0x000000000040138b <+96>:     mov    %edx,%eax
  0x000000000040138d <+98>:     pop    %rbx
  0x000000000040138e <+99>:     pop    %rbp
  0x000000000040138f <+100>:    pop    %r12
  0x0000000000401391 <+102>:    ret
End of assembler dump.
```

<Strings_not_equal> doesn’t have a call to bomb so it's okay to execute. Looking at %eax, we see it is = 1, so it will call the bomb.

```
Dump of assembler code for function phase_1:
    0x0000000000400e8d <+0>:     sub    $0x8,%rsp
    0x0000000000400e91 <+4>:     mov    $0x4023b0,%esi
    0x0000000000400e96 <+9>:     call   0x40132b <strings_not_equal>
=>0x0000000000400e9b <+14>:    test   %eax,%eax //compare bit wise
    0x0000000000400e9d <+16>:    je     0x400ea4 <phase_1+23> //jump if equal else skip
    0x0000000000400e9f <+18>:    call   0x40142a <explode_bomb>
    0x0000000000400ea4 <+23>:    add    $0x8,%rsp
    0x0000000000400ea8 <+27>:    ret
End of assembler dump.

(gdb) i r
rax            0x1                 1      //will call the bomb as 1&1 will not give back zero
rbx            0x4021e0        4202976
rcx            0x3             3
rdx            0x1             1
```

Lets try using the string we found in the disassembler code and see the value of %eax for it:
"Border relations with Canada have never been better."

```
  (gdb) b explode_bomb
  Breakpoint 1 at 0x40142a
  (gdb) r
  Starting program: /home/chh/bomb8/bomb
  Welcome to my fiendish little bomb. You have 6 phases with
  which to blow yourself up. Have a nice day!
  Border relations with Canada have never been better.

  Breakpoint 1, 0x000000000040142a in explode_bomb ()
  gdb) disas
  Dump of assembler code for function phase_1:
=> 0x0000000000400e8d <+0>:     sub    $0x8,%rsp
    0x0000000000400e91 <+4>:     mov    $0x4023b0,%esi
    0x0000000000400e96 <+9>:     call   0x40132b <strings_not_equal>
    0x0000000000400e9b <+14>:    test   %eax,%eax
    0x0000000000400e9d <+16>:    je     0x400ea4 <phase_1+23>
    0x0000000000400e9f <+18>:    call   0x40142a <explode_bomb>
    0x0000000000400ea4 <+23>:    add    $0x8,%rsp
    0x0000000000400ea8 <+27>:    ret
  End of assembler dump.

  (gdb) ni 3
  0x0000000000400e9b in phase_1 ()

  (gdb) disas
  Dump of assembler code for function phase_1:
    0x0000000000400e8d <+0>:     sub    $0x8,%rsp
```

```
   0x0000000000400e91 <+4>:     mov    $0x4023b0,%esi
   0x0000000000400e96 <+9>:     call   0x40132b <strings_not_equal>
=> 0x0000000000400e9b <+14>:    test   %eax,%eax
   0x0000000000400e9d <+16>:    je     0x400ea4 <phase_1+23>
   0x0000000000400e9f <+18>:    call   0x40142a <explode_bomb>
   0x0000000000400ea4 <+23>:    add    $0x8,%rsp
   0x0000000000400ea8 <+27>:    ret
 End of assembler dump.

 (gdb) i r
 rax            0x0         0       //%rax is equal to 0! which means it will jump pass the explode_bomb.
 rbx            0x4021e0    4202976
 rcx            0x0                 0
```

So solution is Border relations with Canada have never been better.