Phase 1

- First, check what are the functions of the bomb using: *info functions*
- Create a breakpoint at phase_1 function: *break phase_1*
- *run < input* and then, disas
  - *0x0000000000400ee4 <+4>:* *mov $0x402400,%esi* Moves "$0x402400" to esi
  - *0x0000000000400ee9 <+9>:* *callq 0x401338 <strings_not_equal>* Calls string_not_equal function which puts the result on %eax
  - *0x0000000000400eee <+14>:* *test %eax,%eax* Tests eax. If eax is 0 then ZF set to one, otherwise ZF is set to zero.
  - *0x0000000000400ef0 <+16>:* *je 0x400ef7 <phase_1+23>* If ZF is one jumps to <pahse_1+23>
  - *0x0000000000400ef2 <+18>:* *callq 0x40143a <explode_bomb>* Explodes the bomb
- We can see that it moves $0x402400 to %esi, which is a register used to store arguments for function calls. Since it calls a function called "string_not_equal", let's see what is this constant. To do this we step into instruction phase_1<+9> and see what's in %esi
- *p (char *) $esi*
  $3 = 0x402400 "Border relations with Canada have never been better."
- If we check now what is the other argument, in %rsi: *p (char *) $rsi*
  $4 = 0x402400 "The input I typed."
- At <phase_1+23>
  - *0x0000000000400ef7 <+23>:* *add $0x8,%rsp* Adds 8 to the stack pointer
  - *0x0000000000400efb <+27>:* *retq* returns
- Now, since the code for phase_1 only checks if the strings are equal and if they are the bomb is defused, the password for phase 1 is:

"Border relations with Canada have never been better."

Phase 2

- Create a breakpoint at phase_1 function: *break phase_2*
- *run < input* and then disas
  - *0x0000000000400f02 <+6>:* *mov %rsp,%rsi* Moves content of rsp to rsi, I.e rsp is an argument to the function called right after this line.
  - *0x0000000000400f05 <+9>:* *callq 0x40145c <read_six_numbers>* Calls read_six_numbers
  - *0x0000000000400f0a <+14>:* *cmpl $0x1,(%rsp)* Compares one to the element pointed by the first entry of the stack. If they are equal, ZF is set to one.
  - *0x0000000000400f0e <+18>:* *je 0x400f30 <phase_2+52>* If ZF is one we jump to <phase_2+52>
- In the first lines that the function read_six_numbers receives the stack pointer as an argument. I typed 6 numbers to see what happened after that to the stack, and:
  - (gdb) p *(int *) ($rsp)
  - $5 = <first typed number>
  - (gdb) p *(int *) ($rsp + 0x4)
  - $7 = <second typed number>
  - (gdb) p *(int *) ($rsp + 0x8)
  - $8 = <third typed number>
  - (gdb) p *(int *) ($rsp + 0xc)
  - $9 = <4th typed number>
  - (gdb) p *(int *) ($rsp + 0x10)
  - $11 = <5th typed number>
  - (gdb) p *(int *) ($rsp + 0x14)
  - $12 = <6th typed number>
- The function read six integers and put them in the stack. After that there is a comparison between $0x1 and the first read number. If they are not equal the bomb

explode. Then the 1st number must be 1.

- After that, there is a jump to <phase_2 + 52>, where:
  - *0x0000000000400f30 <+52>: lea 0x4(%rsp),%rbx* — Moves the address of 2nd element of the stack to rbx
  - *0x0000000000400f35 <+57>: lea 0x18(%rsp),%rbp* — Moves the address of 7th element of the stack to rbp
  - *0x0000000000400f3a <+62>: jmp 0x400f17 <phase_2+27>* — Jumps to <phase_2+27>

We copy the address of the second argument to %rbx and the 6th older element in the %rbp. Then we jump to <phase_2+27>

- In <phase_2+27>:
  - *0x0000000000400f17 <+27>: mov -0x4(%rbx),%eax* — Moves the element pointed right before rbx to eax
  - *0x0000000000400f1a <+30>: add %eax,%eax* — Makes eax = eax * 2
  - *0x0000000000400f1c <+32>: cmp %eax,(%rbx)* — Compares eax to the element pointed by rbx. If they are equal ZF is set to zero.
  - *0x0000000000400f1e <+34>: je 0x400f25 <phase_2+41>* — If ZF is one we jump to <phase_2+41>
  - *0x0000000000400f20 <+36>: callq 0x40143a <explode_bomb>* — Explodes the bomb

In this part we verify if the element that came after the current element pointed by %rbx is exactly twice the first one. If not the bomb explodes. If they are we jump to <phase_2+41>.

- In <phase_2+41>:
  - *0x0000000000400f25 <+41>: add $0x4,%rbx* — Adds 4 to rbx. Now rbx points to the next element in the stack.
  - *0x0000000000400f29 <+45>: cmp %rbp,%rbx* — Compares rbp to rbx. Right below there's an explanation of this line
  - *0x0000000000400f2c <+48>: jne 0x400f17 <phase_2+27>* — If they are not equal, jump back to <phase_2 + 27>
  - *0x0000000000400f2e <+50>: jmp 0x400f3c <phase_2+64>* — Jumps to <phase2+64>
  - *0x0000000000400f3c <+64>: add $0x28,%rsp* — Adds 0x28 to rsp
  - *0x0000000000400f40 <+68>: pop %rbx* — Pops the stack to rbx
  - *0x0000000000400f41 <+69>: pop %rbp* — Pops the stack to rbp
  - *0x0000000000400f42 <+70>: retq* — Returns

In this part, we update %rbx to point to the next entered number and we compare if %rbp and %rbx points to the same place. Remember that %rbx saves the address right after the 6th entered number, so this will be true when we add 0x4 to %rbx 6 times. If they are equal e end the function, otherwise we jump back to line 27 to verify the same thing, if the next number is alos twice the last one. Then, the answer to this phase is:

1 2 4 8 16 32

Other Phases

I also defused phase 3 and phase 4, which has, repectively, the passwords **3 256** and **1 0**.