BombLab Cheat Sheet

Created By: Joseph Turcotte

CS2011 D'18

What is BombLab?

BombLab is the second assignment of CS2011. In this assignment, you must defuse a bomb in six phases without blowing it up. You will have to examine assembly code and supply correct passwords in each phase to move to the next phase. This is a very important mission, so be very careful (or you'll lose points!).

Example (Phase 1)

```
0000000000400ee0 <phase_1>:
        400ee0:     48 83 ec 08          sub     $0x8, %rsp
        400ee4:     be 00 24 40 00       mov     $0x402400,%esi
        400ee9:     e8 4a 04 00 00       callq   401338 <strings_not_equal>
        400eee:     85 c0                test    %eax,%eax
        400ef0:     74 05                je      400ef7 <phase_1+0x17>
        400ef2:     e8 43 05 00 00       callq   40143a <explode_bomb>
        400ef7:     48 83 c4 08          add     $0x8,%rsp
        400efb:     c3                   retq
```

Phase 1 is relatively simple. The first line of this assembly code snippet reserves space on the stack; you can ignore this line for the purposes of this assignment, as well as the line towards the end that frees the space on the stack. The important lines are highlighted in green. First, we move whatever value is stored at 0x402400 into the register esi, and then we call a function called "strings_not_equal." Well, that implies we're comparing two strings. What are those two strings?

Why don't you try printing out the value stored at 0x402400? If you want to print a value as a string, use "x/s [address]", such as "x/s $0x402400". It turns out that if you want to clear this phase, you need to provide that string you just printed to the console as your passcode. Then the strings will be equal, and the code will jump over the explode_bomb call! Then the function phase1 will return, and you can move on to phase 2.

Helpful Strategies

Reverse Engineering	Consider this line of assembly code: `cmp $0xfff, $eax` eax is almost always used to store the return value of a function. You will likely see weird functions like "fun4" or "evil_function"; the truth is that it doesn't always matter what these functions do. It matters what they return. If you can get the return value of the function to be 0xfff, then the compare statement will evaluate to true. This strategy is called reverse engineering because you have the desired output, but you need the input that produces that output.
GDB Commands	• b explode_bomb → sets a breakpoint at explode_bomb to prevent an explosion • b phasex → stops execution at the beginning of the phase • layout asm → provides a nice layout with gdb (highlights current line, etc.) • p $register_name → prints the value in the given register (Note: this only works for data registers like eax, rax, ebx, rbx, etc.) • x/x $register_name → prints the value in the given register as a hexadecimal value (Note: this only works for pointer registers like rsi, rsp, rbp, etc.)

General Advice

- Guess and check is a valid strategy to begin. See how far you can get with an input before the bomb blows up (make sure to set a breakpoint at the explode bomb function!). Then modify your input and see how far you get.
- Not all instructions are important; focus on jumps and compare statements.
- Keep track of register values as you progress through the code. What kinds of values could those registers be holding? Counters? Accumulators? Garbage?
- Come to office hours if you don't understand something! The TAs and SAs are here to help you. Good luck! :)