CSE 351

ox86 Calling Conventions, Lab 2, & GDB
Calling Conventions Review

- This class uses x86-64 conventions, which differ from standard x86 conventions
- The first six arguments are passed in registers:
  - %rdi, %rsi, %rdx, %rcx, %r8, %r9
- Any additional arguments go on the stack
- Why?
  - Faster!
Caller- vs. Callee-Saved Registers

- Saving registers
  - Accomplished via the `push` command
  - Restored using `pop`

- Callee-saved registers
  - Must be saved by a function before changing its value, and then restored before returning
  - `%rbx, %rbp, %r12-%r15`

- Caller-saved registers
  - Must be saved by a function before calling any other subroutines if it wishes to preserve the value
  - All other registers (including all the parameter-passing registers)
GNU Project Debugger (GDB)

- Installed on CSE Home VM and attu
- Used to debug C programs (among other languages)
- It will be your best friend in this course
- Print out the GDB cheat sheet to use while debugging
  - This and other GDB resources can be found here
Lab 2

• Requires you to defuse “bombs” by entering a series of passcodes
  • Not real bombs/viruses/etc!

• Each passcode is validated by some function
  • You only have access to the assembly code

• It’s your job to determine what passcodes will prevent the program from ever calling the `explode_bomb()` function

• Each student has a different bomb
Lab 2 Files

- bomb
  - The executable bomb program
- bomb.c
  - This is the entry point for the bomb program, and it calls functions whose source code is not available to you
- defuser.txt
  - Contains passcodes, each separated by a newline
  - Place your passcodes here once you solve each phase
  - Can be passed as an argument to prevent you from entering the passcodes manually each time
• The bomb uses `sscanf`, which parses a string into values

• Example:

```c
int a, b;
sscanf("123, 456", "%d, %d", &a, &b);
```

• The first argument is parsed according to the format string

• After this code is run, a = 123 and b = 456
Lab 2 Tips

• Print out the disassembled phases
  • To disassemble a program, run `objdump -d bomb > bomb.s`
  • You can then print out `bomb.s`
  • Mark the printouts up with notes
• Try to work backwards from the “success” case of each phase
• Remember that some addresses are pointing to strings located elsewhere in memory
  • Print them out in GDB
• We will now go through Phase 1 of the bomb!
  • Pay close attention and ask questions