CSE 351

x86 Calling Conventions, Control Flow, & Lab 2

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)

Callee/Caller Explained

- Why not have all registers caller, or all registers callee?
  - We want to minimize the number of pushes/pops (they're slow!)
- Think of callee registers are non-volatile
  - You must save them to use them, BUT…
  - You can be guaranteed that when you call a function, it will have the same value when it returns
  - Good for values that must be preserved for a long time, across many calls
- Think of caller registers as volatile
  - You don't have to save them in order to use them, SO…
  - If you ever have values that you need before a function call but don't need afterward (i.e. arguments) then they should be in caller-saved registers
  - This way, no values are pushed onto the stack unnecessarily

Control Flow

- 1-bit condition code registers [CF, SF, ZF, OF]
- Set as side effect by arithmetic instructions or by cmp, test
- CF – Carry Flag
  - Set if addition causes a carry out of the most significant (leftmost) bit.
- SF – Sign Flag
  - Set if the result had its most significant bit set (negative in two's complement)
- ZF – Zero Flag
  - Set if the result was zero
- OF – Overflow Flag
  - If the addition with the sign bits off yields a result number with the sign bit on or vice versa

Control Flow Examples

x86:

```
test %rax, %rax    ; set ZF to 1 if rax == 0
je <location>      ; jump if ZF == 1
```

```
cmp %rax, %rbx
jg <location> (hint: jg checks if ZF = 0 and SF = OF)
```

```
cmp %rax, %rbx
xor %rbx, %rbx
jo <location> (hint: js checks if OF = 1)
```

Result:

Jumps to <location> if rax == 0

rax and rbx are interpreted as signed then compared, if rbx > rax we jump to <location>

Jumps to <location> if rbx is negative (or if rbx is a very large unsigned value)

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
  - To do this, you can run `set args defuser.txt` from within GDB and then whenever you run your program, it will automatically read its input from defuser.txt

Lab 2 Notes

- The bomb uses sscanf, which parses a string into values
- Example:

```
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

Lab 2 Demo

- We will now go through Phase 1 of the bomb!
  - Pay close attention and ask questions