CSSE 332 -- Operating Systems

Introduction to Memory Organization and Safety

Name: Solution Key

Question 1. (5 points) Before running anything in the three sessions, and for the next lab, which command should you always run first?

> **Solution:** `./disable_aslr.sh`

Question 2. (5 points) After each session and after the lab, which command should you run?

> **Solution:** `./enable_aslr.sh`

Question 3. (5 points) Assume we make a function call to a function `foo`. How does `foo` know which instruction to go back to once it returns?

> **Solution:** It saves its *return address*, which the address of the *next instruction* after the `call` instruction.

Question 4. (5 points) In the `i386` architecture, where is this value stored? What about the `RISC-V` architecture?

> **Solution:** In `i386`, the return address is always stored on the stack. It is done by default by the `call` instruction.
>
> In `RISC-V`, the return address is saved in the `ra` register, which is register dedicated to store the return address. It is done by default using the *jump-and-link-register* instruction, or `jalr`.

Question 5. Assume we would like to make a call to the function `foo` with arguments `(1, 2, 3)`.

(a) (5 points) How are the arguments `1`, `2`, and `3` passed to `foo` in the `i386` architecture?

Solution: They are pushed onto the stack in reverse order.

(b) (5 points) Write down the `i386` instructions that correspond to the C statement `foo(1,2,3);`.

Solution:

```
push 3
push 2
push 1
call foo
```

Question 6. (5 points) Assume the function `foo` creates three **local** variables `x`, `y`, and `z`. Where are `x`, `y`, and `z` stored with respect to the `foo`'s arguments?

Solution: They are stored on the stack below the arguments and the return address (and some other nonsense needed by the compiler and the runtime).

Note that *below* is relative to the stack growing *downward*. In other words, they are at a *lower* address.

Question 7. (5 points) What do we call the area of memory that contains a function's return information, its arguments, and its local variables?

Solution: The function's *stack frame*.

Question 8. (10 points) Based on your previous answers, draw that area of memory for a call `foo(1,2,3)` to a function `foo` that defines two local variables, `x` and `y`.

Solution:

```
/*
 * | 3              |
 * | 2              |
 * | 1              |
 * | return address |
 * | (maybe stuff)  |
 * | x              |
 * | y              |
 */
```

Note that the order in which `x` and `y` are stored depends on how they are defined, but you get the point.