`bomblab` Activity - Handout

Your TAs

September 2024

1 Introduction

In this recitation, we'll be defusing three phases in a recitation-specific bomb! The format will be very similar to the real `bomblab`, explosions and all. Note that the bomb we'll be working on is not connected to Autolab in any way, so there is no penalty for explosions.

The goal of this activity is to learn how to reason about more complex phases by translating the assembly you are given to pseudocode. Please try to solve the phases with this in mind, since this approach will help you with the later phases of `bomblab`!

2 Getting Started

Please copy the tarfile from our AFS directory to an appropriate folder, then untar it:

```
$ cp /afs/cs.cmu.edu/academic/class/15213-f24/bomb_reci.tar .
$ tar xvpf bomb_reci.tar
```

You should be left with an executable called `bomb`, which you can open with `gdb`:

```
$ gdb bomb
```

You may also need to mark the bomb as executable, by typing `chmod +x bomb`.

3 Hints

Even though there is no penalty for exploding the recitation bomb, you will probably want to start by setting some breakpoints to build good habits for `bomblab`. For example, you can enter `break phase_a` into the `gdb` prompt to set a breakpoint at the first phase. Having a breakpoint at `explode_bomb` is probably a good idea, too.

3.1 Phase A

- How can we find out what kind of input the phase expects? What property (or properties) does the outer function enforce on the input string?
- Try translating the assembly you're given for the helper function to pseudocode - this will make it a lot easier to understand what's going on.
- Watch out for *loops* as you're translating. What might these look like in assembly?
- Once you're happy with your pseudocode, try reasoning about what property (or properties) the function is enforcing on the input string. Can you craft an input string which satisfies the properties enforced by the outer function and the helper function?

3.2 Phase B

- What kind of input does the phase expect?
- This phase calls a mystery function on the (converted) input. What do we need the mystery function to return to prevent the bomb from exploding?
- Like in Phase A, see if you can translate the assembly code for the mystery function into pseudocode. It'll help - we promise!
- Once you have your pseudocode, see if you can deduce how the function calculates its return value from its input. It may help to write down a sequence of input/output pairs to see if you can get to the desired output.

3.3 Phase C

- What kind of input does the phase expect? How does the phase transform the input before using it?
- The control flow for this phase seems pretty unconventional at first glance. The assembly instructions seem to pull some constant(s) from some memory region. The constant(s) is then used to calculate the address of the next instruction to jump to. What control flow technique from Lecture does this remind you of? You may find the Lecture slides for *Machine Programming II: Control* helpful here.
- Once you know what the control flow is doing, you will probably find it helpful to inspect some of the addresses used in the phase with `gdb`. We can use `gdb`'s `x` command for this. Here are some example commands you can build on:
  - (`x /5gx <address>`) - print five 8-byte quantities in hex format, starting at `address`. Useful for printing blocks of contiguous addresses.
  - (`x /5wx <address>`) - print five 4-byte quantities in hex format, starting at `address`. Useful for printing blocks of contiguous integers.
  - (`x /5s <address>`) - print five strings, starting at `address`.
- See if you can find where the different paths through the program join up. How does the phase decide whether or not to explode the bomb? What does this tell you about what you need to input?