Assignment 3

CS/ECE 354, Spring 2016
Due Friday, March 11 before 9 AM

1. Collaboration Policy

This assignment is an individual project. Furthermore, each student gets different problems to solve and the answers are different.

2. Learning Goals

There are two main objectives of this project. The first is to quickly become familiar with x86 assembly language. This is a hugely useful skill! In real life, one is often faced with the task of trying to figure out why some code is not working as planned, and sometimes this leads to staring at the instructions that are executing on the processor to gain understanding. The second relates to the first: to gain some familiarity with powerful tools that help with this process, namely gdb (the debugger) and objdump (the disassembler). These tools will also serve you well in your future endeavors.

3. Defusing Binary Bombs

In this assignment, you will be defusing four binary bombs. The idea is simple: each bomb is just an executable program, which expects five inputs from you, the bomb defusing expert. If you type in the right values, you successfully defuse the bomb. If not, the bomb explodes! (Don't worry, it just prints that the bomb explodes; no real harm is done to you or your computer.)

The four bombs are located in your assignment 3 directory (/p/course/cs354-common/public/spring16.handin/login/p3, substituting login with your own CS login), named b1 through b4. Your task is to create four files, **b1.solution** through **b4.solution**. Each file contains the five lines of input demanded by its associated bomb. For example, if the solution to b1 was 1, 2, 3, 4, and 5, you would create a file called b1.solution with the following entries:

```
1
2
3
4
5
```

You can create this file with a text editor (vim/gedit/nano) on a Linux Machine. If you are using Windows or Mac, editing locally followed by uploading will fail for b2. Use remote accessing tools like ssh or putty instead. Make sure your solution file contains five non-empty lines. Remember to press

enter or return after the last line. The bomb will be trapped into an infinite loop if the solution file contains less than 5 lines. If that happens, press Ctrl-C to break.

The challenge is to figure out the inputs expected by each of the four bombs. To do this, use two tools: **gdb** and **objdump**. Both are incredibly useful for this type of reverse engineering work.

Please copy your executable bombs to your own private directory, and work towards finding your solutions in your own directory. That way, you will have the original executables in your handin directory if you accidentally overwrite an executable.

To test whether you have figured out a bomb's defusing code correctly, run the bomb with its input file

```
./b1 < b1.solution
```

This is how we will grade your assignment. It will print the following success message if you have figured everything out; otherwise it will tell you the bomb exploded. You can also run each bomb interactively, and type in your guesses, one at a time. This will be useful in defusing each bomb with the debugger, as described below.

guess 1 (of 5)? guess 2 (of 5)? guess 3 (of 5)? guess 4 (of 5)? guess 5 (of 5)? success!

This testing also uses another shell skill that you have already used: redirection of input or output. This command redirects stdin to come from a file. The first part of Chapter 7 in K&R has an introduction, as does this TLDP web page.

4. The Tools: gdb and objdump

To figure out how to defuse your binary bombs, use two powerful tools: gdb and objdump. Both are critical in understanding what each binary bomb does.

4.1. objdump

With objdump, two important command line options are

-d, which disassembles a binary
-s, which displays the full binary contents of the executable

For example, to see the assembly code of bomb b1, you might type:

```
objdump -d b1
```

This will show an assembly listing of each function in the bomb. Your first task then might be to look at main() and figure out what the code is doing.

The -s flag is also quite useful, as it shows the contents of each segment of the executable. This may be needed when looking for the initial value of a given variable.

By redirecting stdout, you can capture the output of objdump in a file, such that you can look at this output without having to regenerate it every time. And, you can use both command line options at the same time to create a full dump of the contents of the executable as well as the disassembled contents. (Hint, hint!)

4.2. gdb

The debugger, gdb, is an even more powerful ally in your search for clues as to how to defuse each binary bomb. To run gdb on a particular bomb (say b1),

```
gdb b1
```

which will launch the debugger and ready you for a debugging session. The command **`run`** causes the debugger to run the program, in this case prompting you for input.

However, before running the debugger, you likely need to first set some breakpoints. Breakpoints are places in the code where the debugger will stop running and let you take control of the debugging session. For example, a common thing to do will be to fire up the debugger, and then

```
break main
```

to set a breakpoint at the main() routine of the program, and then type

```
run
```

to run the program. When the debugger enters the main() routine, it will then stop running the program and pass control back to you, the user. (Ignore the line saying 'Missing separate debuginfos, use: debuginfo-install glibc-2.12-1.166.el6_7.7.i686'. This is intended.)

At this point, you will need to do some work. One likely command you will use is **`stepi`**, which steps through the code one instruction at a time. Another useful command is **`info registers`**, which shows you the contents of all of the registers of the system. Another is **`x/x 0xADDRESS`**, which is the examine command. It shows you the contents at the address ADDRESS, and does so in hexadecimal. The second x determines the format, whereas the first x is the examine command. You can also have gdb disassemble the code by typing the **`disassemble`** command. Finally, **`break *0xADDRESS`** sets up another breakpoint at address ADDRESS and **`continue`** resumes the execution until any breakpoint is reached again.

A former TA for the course wrote an excellent tutorial that can help you learn to use gdb. Questions for this assignment go to Piazza; please do not email Chris Feilbach, the author of this tutorial. prog1.c and prog2.c mentioned in the gdb tutorial are available at /p/course/cs354-common/public/spring16/html/projects/p3aux/. And, you might like to refer to Cherin's quick list of useful gdb commands. A lengthy tutorial that contains way more than you need for this assignment is the Peter Jay Salzman gdb tutorial. And, for those that like to watch videos, here is an excellent YouTube video tutorial.

Getting good with gdb will make this project go smoothly, so spend the time and learn! One thing to notice: using the keyboard's up and down arrows (or control-p and control-n for previous and next, respectively) allows you to go through your gdb history and easily re-execute old commands; getting good at using your history, whether in gdb or more generally in the shell you use, is a good idea.

5. Hints

- x86 cheat sheet
- Every C program has a main() function. Figure out how to locate it.
- A loop in main() iterates five times. Remember that each bomb requires five inputs.
- On a wrong input, function bomb() is called. This results in an explosion.
- If all five inputs are correct, function success() is called.
- Function strtol() corresponds to the use of atoi() in C source code.
- Function arguments are set up in the call stack just prior to the function call.
- The two parameters to strcmp() are addresses to 2 C strings.

6. Grading

Grading will be based entirely on whether you defuse the bombs. Each bomb is thus worth 25 percent of the grade for this project. If you have only part of the five inputs correct, the bomb still explodes.

7. Handing In the Programs

Turn in your **4 files** b1.solution through b4.solution by copying them to /p/course/cs354-common/public/spring16.handin/login/p3 where login is your CS login.

Double check your filenames! They end with '.solution', not '.solutions' or '.solution.txt'