CMSC 311, Fall 2010
Lab Assignment 2: Defusing a Binary Bomb
Due: Tuesday, October 20, 10:00pm

Post questions on this lab to the elms discussion board: `http://elms.umd.edu`.

1 Introduction

The nefarious *Dr. Evil* has planted a slew of "binary bombs" on our machines. A binary bomb is a program that consists of a sequence of phases. Each phase expects you to type a particular string on *stdin*. If you type the correct string, then the phase is *defused* and the bomb proceeds to the next phase. Otherwise, the bomb *explodes* by printing `"BOOM!!!"` and then terminating. The bomb is defused when every phase has been defused.

There are too many bombs for us to deal with, so we are giving each of you a bomb to defuse. Your mission, which you have no choice but to accept, is to defuse your bomb before the due date. Good luck, and welcome to the bomb squad!

Step 1: Get Your Bomb

Each student will attempt to defuse their own personalized bomb. Each bomb is a Linux binary executable file that has been compiled from a C program. To obtain your bomb, log onto linuxlab and check your mail. You should have a message with one attachment: the `bomb`! Save the bomb to a (protected) directory on linuxlab. You can do this using, for example, the pine e-mail reader: type `pine` on the command-line and get to the main screen; push *i* for your inbox, select the message with Enter, then *v* to view attachments, and *s* to save each one.

Step 2: Defuse Your Bomb

Your job is to defuse the bomb.

You can use many tools to help you with this; please look at the **hints** section for some tips and ideas. The best way is to use your favorite debugger to step through the disassembled binary.

There are six phases total, each phase worth 13 points, for a total of 78 points (but we'll grade it out of 75 like the other labs).

The phases get progressively harder to defuse, but the expertise you gain as you move from phase to phase should offset this difficulty. However, the last phase will challenge even the best students, so please don't wait until the last minute to start.

The bomb ignores blank input lines. If you run your bomb with a command line argument, for example,

```
linux>  ./bomb psol.txt
```

then it will read the input lines from `psol.txt` until it reaches EOF (end of file), and then switch over to `stdin`. In a moment of weakness, Dr. Evil added this feature so you don't have to keep retyping the solutions to phases you have already defused.

To avoid accidently detonating the bomb, you will need to learn how to single-step through the assembly code and how to set breakpoints. You will also need to learn how to inspect both the registers and the memory states. One of the nice side-effects of doing the lab is that you will get very good at using a debugger. This is a crucial skill that will pay big dividends the rest of your career.

Logistics

As usual pay close attention to the forum and the announcements on Blackboard.

Hand-In

You will not submit any code. You will submit a single textfile containing the solutions to your bomb. It must be named with your linuxlab username and a `.txt` extension. For example, if your linuxlab login is cs311001, then your solutions must be in a plain ASCII text file called `cs311001.txt`. You will submit only this file to the submit server. As usual, you can do via the submit server web interface.

Hints *(Please read this!)*

There are many ways of defusing your bomb. You can examine it in great detail without ever running the program, and figure out exactly what it does. This is a useful technique, but is not always easy to do. You can also run it under a debugger, watch what it does step by step, and use this information to defuse it. This is probably the fastest way of defusing it.

There are many tools which are designed to help you figure out both how programs work, and what is wrong when they don't work. Here is a list of some of the tools you may find useful in analyzing your bomb, and hints on how to use them.

- `gdb`

The GNU debugger, this is a command line debugger tool available on virtually every platform. You can trace through a program line by line, examine memory and registers, look at both the source code and assembly code (we are not giving you the source code for most of your bomb), set breakpoints, set memory watch points, and write scripts. Here are some tips for using `gdb`.

  - To keep the bomb from blowing up every time you type in a wrong input, you'll want to learn how to set breakpoints.
  - The CS:APP Student Site at `http://csapp.cs.cmu.edu/public/students.html` has a very handy single-page `gdb` summary.
  - For other documentation, type "`help`" at the `gdb` command prompt, or type "`man gdb`", or "`info gdb`" at a Unix prompt. Some people also like to run `gdb` under `gdb-mode` in `emacs`.

- `objdump -t`

  This will print out the bomb's symbol table. The symbol table includes the names of all functions and global variables in the bomb, the names of all the functions the bomb calls, and their addresses. You may learn something by looking at the function names!

- `objdump -d`

  Use this to disassemble all of the code in the bomb. You can also just look at individual functions. Reading the assembler code can tell you how the bomb works.

  Although `objdump -d` gives you a lot of information, it doesn't tell you the whole story. Calls to system-level functions are displayed in a cryptic form. For example, a call to `sscanf` might appear as:

  ```
  8048c36:  e8 99 fc ff ff  call   80488d4 <_init+0x1a0>
  ```

  To determine that the call was to `sscanf`, you would need to disassemble within `gdb`.

- `strings`

  This utility will display the printable strings in your bomb.

Looking for a particular tool? How about documentation? Don't forget, the commands `apropos` and `man` are your friends. In particular, `man ascii` might come in useful. Also, the web may also be a treasure trove of information. If you get stumped, feel free to ask your TA for help. Also, a *very helpful* tutorial is available on cmsc311 webpage `http://www.cs.umd.edu/users/meesh/cmsc311/lab2_tutorial.txt`.

Good luck!