CSE 238/2038/2138 Systems Programming
Project 2: Defusing a Binary Bomb
Due: 29.04.2019 23:59

Introduction

The purpose of this project is to become more familiar with machine level programming.

Each of you will work with a special “binary bomb”. 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 different bomb to defuse. Your mission is to defuse your bomb before the due date. Below, the two steps of the project are explained in detail.

Step 1: Get Your Bomb

Each “bomb” is a Linux binary executable file that has been compiled from a C program. To obtain your bomb, you should download all the bombs from the Canvas service.

Each bomb for the class will be in a `tar` file called `bombs.tar`. In this tar file, you will find your special bomb `bombk.tar,` where k is the unique number of your bomb indicating your student ID. You will only need your file for the project. Inside your file `bombk.tar,` you will find the following files:

- `bomb`: The executable binary bomb.
- `bomb.c`: Source file with the bomb’s main routine.

Step 2: Defuse Your Bomb

Your job for this lab is to defuse your bomb.

You can use many tools to help you defuse your bomb. 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.

Although phases get progressively harder to defuse, 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`.

To avoid accidentally exploding 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 this project is that you will get very good at using a debugger. This is a crucial skill that will pay big bonuses the rest of your career.

The first four phases are worth 10 points each. Phases 5 and 6 are a little more difficult, so they are worth 15 points each. For the remaining **30** points, you will have a project quiz. The project quiz will be in lab sessions on April 30 - May 2.

Handin

This is an individual project. You will handin a `txt` file containing the input lines to defuse the bomb. Although there are 6 inputs to defuse the bomb totally, you can submit the `txt` file with as many input lines as you can find. Recall that you should not submit a tar/zip file.

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.

  In the CS:APP web site `http://csapp.cs.cmu.edu/public/students.html`, you can find detailed information about `gdb` that you can use as a reference. Here are some other 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.

  – For online 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`, `man`, and `info` are your friends. In particular, `man ascii` might come in useful. `info gas` will give you more than you ever wanted to know about the GNU Assembler. Also, the web may be very useful. If you get lost, feel free to ask teaching staff for help.