CSCI 2021-010 Fall 2018
HA3: Defusing a Binary Bomb
Assigned: Oct. 17, Due: Friday Oct. 26, @11:55pm

Welcome to bomblab.

1 Introduction

The nefarious *Dr. Evil* has planted a slew of "binary bombs" on our class 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 student 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

You can obtain your bomb by pointing your Web browser at (This will only work on a lab computer):

```
http://apollo.cselabs.umn.edu:2021/
```

Note: Make sure you download it first on a CSELabs computer, your personal computer will not be able to connect to the website that distributes the bombs. This will display a binary bomb request form for you to fill in. Enter your user name and email address and hit the Submit button. The server will build your bomb and return it to your browser in a `tar` file called `bombk.tar`, where k is the unique number of your bomb. With this version of the bomblab, you are allowed to run your bomb on your own personal computers (running Linux) to experiment with them. However, you will not be given credit for any defuses on a personal computer. In the case that you are not on a CSELabs computer, the bomb should notify you at the beginning of running the bomb, and also once you have defused the bomb. To get credit for the bomblab the bomb must be run on a CSELabs machine. This will notify the server that you have defused the bomb and will earn the points for the phases you have defused. So, before turning this project in, make sure to run the bomblab on a CSELabs computer.

Save the `bombk.tar` file to a (protected) directory in which you plan to do your work. Then give the command: `tar -xvf bombk.tar`. This will create a directory called `./bombk` with the following files:

- `README`: Identifies the bomb and its owners.
- `bomb`: The executable binary bomb.
- `bomb.c`: Source file with the bomb's main routine and a friendly greeting from Dr. Evil.
- `writeup.{pdf,ps}`: The lab writeup.

Please refrain from working with multiple bombs.

Step 2: Defuse Your Bomb

Your job for this lab is to defuse your bomb.

You must do the assignment on one of the CSELabs machines. In fact, there is a rumor that Dr. Evil really is evil, and the bomb will always blow up if run elsewhere. There are several other tamper-proofing devices built into the bomb as well, or so we hear.

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.

Each time your bomb explodes it notifies the bomblab server. Unlike previous years, if your bomb explodes you will not lose any points. If you can successfully defuse the bomb, this will earn full credit for the assignment.

The first four phases are worth 14 points each. Phases 5 and 6 are a little more difficult, so they are worth 22 points each. So the maximum score you can get is 100 points.

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`. 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 accidentally 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

This is an individual project. All handins are electronic. Clarifications and corrections will be posted on the course message board.

Handin

For this assignment, you will have to make a submission to moodle in case of any errors with the bomblab server. You will have to turn in a .tar file of the directory that contains your bomb executable, input file, etc... The bomb will also notify your instructor automatically about your progress as you work on it. You can keep track of how you are doing by looking at the class scoreboard at:

```
http://apollo.cselabs.umn.edu:2021/scoreboard
```

This web page is updated continuously to show the progress for each bomb. (Note: If you want to work on a personal computer, progress will not be tracked).

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 it 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.

  The CS:APP web site

  `http://csapp.cs.cmu.edu/public/students.html`

  has a very handy single-page `gdb` summary that you can print out and 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 also be a treasure trove of information. If you get stumped, feel free to ask your instructor for help.