CS 211 Summer 2019
Assignment L: Defusing a Binary Bomb
Assigned: July 12th, Due: Sunday July 28th

David Domingo (djd240@cs.rutgers.edu) is the lead TA for this lab.

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 must obtain your bomb from inside the Rutgers network. If you are not connected to Rutgers network, you may log into iLab and then download the bomb.

There are two ways to download the bomb. The first method is the easiest: You can obtain your bomb by pointing your Web browser at:

```
http://prof2.cs.rutgers.edu:17200/
```

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 `bomb<id>.tar`, where `<id>` is the unique number of your bomb (i.e. `bomb25.tar`).

The other method is to use the terminal command `curl` to download the file. This method is quite volatile and sending a wrong request may shut the bomblab server down, so take caution. You want to use this command from the terminal:

```
curl -JLO
"http://prof2.cs.rutgers.edu:17200/?username=<N>&usermail=<E>&submit=Submit"
```

with everything in a single line. The above code is written in two separate lines because the command could not be contained in a single line in this pdf. Also, <N> is your NetID and <E> should be your email address. Make sure that you have typed in all the punctuations correctly including the quotation marks around the url. This command will download the bomb<id>.tar to the current directory that your terminal is at.

The extract the content using this command:

```
$ tar -xvf bomb<id>.tar
```

This will create a directory called ./bomb<ID> 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.

If for some reason you request multiple bombs, this is not a problem. Choose one bomb to work on and delete the rest.

Step 2: Defuse Your Bomb

Your job for this lab is to defuse your bomb.

You must do the assignment on ilab.cs.rutgers.edu or on any other official Rutgers CS server 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.

The first two phases are worth 10 points each. Phases 3, 4, 5, and 6 are a little more difficult, so they are worth 20 points each, to a combined total of 100 points. The last three phases (phase 7, 8, and 9) are extra credit and worth 5 points each. If you complete all nine phases, your score will be 115/100.

You can defuse the bomb by executing the bomb program,

```
$ ./bomb
```

and typing the correct string on stdin. You can also provide the correct string to the bomb by saving the string in a file and passing the file to a command line argument. For example,

```
$ ./bomb solution.txt
```

will read the input lines from `solution.txt` one line at a time until it reaches EOF (End of file), and then switch over to `stdin`. Each line in the file constitutes as an input string for each phase. 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.

Each time your bomb explodes it notifies the bomblab server, and you lose 1/2 point in the final score for the lab. So there are consequences to exploding the bomb. You must be careful!

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.

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 phases will challenge even the best students, so please don't wait until the last minute to start.

And as always, if you are having trouble, ask us or please go to the office hours. We will not know whether you're having trouble until you come and talk to us. If anything doesn't seem right, also ask us.

Grading

You can keep track of how you are doing by looking at the class scoreboard at:

`http://prof2.cs.rutgers.edu:17200/scoreboard`

This web page is updated continuously to show the progress for each bomb, the current score (out of 100 points) and how many times the bomb detonated.

Submission

You have to submit the assignment using Sakai. Your submission should be a tar file named `bomb<ID>.tar` that can be extracted using the command:

```
$ tar -xvf bomb<ID>.tar
```

Extracting your tar file must give a directory called `bomb<ID>`. This directory should contain the same files that you downloaded, along with the file `solution.txt` to defuse the bomb. In another words, your directory should look like this:

```
bomb<ID>
|- bomb
|- bomb.c
|- README
|- solution.txt
```

To create the tar file that you will submit after finishing your programming assignment, you should use the following command line, in the parent directory of `bomb<ID>`:

```
$  tar -cvf bomb<ID>.tar bomb<ID>
```

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.

We do make one request, *please do not use brute force!* You could write a program that will try every possible key to find the right one. But this is no good for several reasons:

- You lose 1/2 point every time you guess incorrectly and the bomb explodes.
- Every time you guess wrong, a message is sent to the bomblab server. You could very quickly saturate the network with these messages, and cause the system administrators to revoke your computer access.
- We haven't told you how long the strings are, nor have we told you what characters are in them. Even if you made the (incorrect) assumptions that they all are less than 80 characters long and only contain letters, then you will have 26^{80} guesses for each phase. This will take a very long time to run, and you will not get the answer before the assignment is due.

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.