CS 220: Fall 2020
Project 3: Defusing a Binary Bomb
Assigned: Oct. 19, Due: Sat Oct. 31 before midnight

Adapted from material provided from csapp3e textbook web site.

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:

```
http://zdu.binghamton.edu:15213
```

[This URL can only be accessed from one of the `remote.cs` machines.

- See the GUI remote access document in the `misc` section of the course web site for some possibilities if you want to use a GUI browser like firefox (unfortunately, the text-mode browser lynx does not seem download files correctly).
- If nothing else works, simply use curl:

```
curl -s 'URL?username=USER&usermail=EMAIL&submit=Submit' \
  > bomb.tar
```

where URL is the above url, USER is the portion of your BU-email before the @ sign and EMAIL is your BU-email. This should grab your bomb download in bomb.tar in your current directory.]

This will display a binary bomb request form for you to fill in. Enter your user name or remote.cs and Binghamton University 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. Note that the tar file will be downloaded directly and there is no success feedback; you should find the downloaded tar file in your Downloads directory.

Save the bombk.tar file to a (protected) directory in which you plan to do your work, like your ~/i220X/ directory. Note that you will not be submitting this project (see below), but you should still save your work on github. 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.

If for some reason you request multiple bombs, this is not a problem. Choose one bomb to work on and delete the rest. However, if you cause explosions in multiple bombs, then **all** those explosions will be used to determine your grade. Also, explosions in **all phases** will be used to determine your grade as there is no way to distinguish which phase an explosion occcured.

Step 2: Defuse Your Bomb

Your job for this project is to defuse your bomb.

You must do the assignment on one of the remote.cs 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.

Note that when the bomb requires you to type in an input string, it silently pauses without any prompt.

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 objdump to disassemble your binary combined with using your favorite debugger to step through the disassembled binary.

Each time your bomb explodes it notifies the bomblab server, and you lose 1/2 point (up to a max of 20 points) in the final score for the project. So there are consequences to exploding the bomb. You must be careful!

The bomb requires a 6-phase defusing. However, in this project we will defuse only the first 4 steps. Phase 1 will be worth 19 points and the remaining 3 phases are worth 27 points each. Hence if you correctly defuse all 4 phases but have the bomb go off 4 times, you will obtain a grade of 98.

Although phases get progressively harder to defuse, the expertise you gain as you move from phase to phase should offset this difficulty.

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

```
$  ./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 project 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.

Handin

There is no explicit handin. The bomb will automatically report your progress to us as you work on it. You can keep track of how you are doing by looking at the class scoreboard at:

```
http://zdu.binghamton.edu:15213/scoreboard
```

[This URL can only be accessed from one of the `remote.cs` machines.]

This web page is updated continuously to show the progress for each bomb. Ignore the *Score* shown on this scoreboard as we are using a different scoring system for this project.

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 (up to a max of 20 points) 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/2e/docs/gdbnotes-x86-64.pdf`

  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.

- `objdump -d -j .text -j .rodata -j .data`

  This will disassemble the code in the file as well as dump out all the data.

- `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.

More hints:

- Since this is a very popular project offered by many schools, you are likely to find a lot of help online. Hopefully, this help will not constitute a complete solution, but could get you pointed in the right direction.

  To get the maximum utility from this lab, force yourself to defuse a phase entirely by yourself; look for online information only after you have succeeded; you should do so only to compare your approach with those used by others.

- When using gdb, `layout asm` and `layout regs` are invaluable. See

  ```
  https://sourceware.org/gdb/onlinedocs/gdb/TUI-Commands.html
  ```

  Note that the TUI is a bit rough around the edges, use the `focus` command to switch between the windows within your terminal.

- Dr. Evil has been nice enough to provide correct names for functions in the object file; i.e., the functions do what their names imply. So `sscanf()` does indeed refer to the standard library function which reads input from a string.

- It is possible to set things up so that it is impossible for the bomb to detonate. Using the debugger, you should be able to see when your bomb is about to detonate and stop or restart the program at that point.