CSCI 356 Fall 2018
Project 3: Defusing a Binary Bomb
Due: Mon. Oct. 1, 2018

1 Introduction

The evil *Bruin Nation* 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

Your bomb should already be in your git repository in the `proj3` folder. You should only need to do a pull to get this on your machine.

Step 2: Defuse Your Bomb

Your job for this lab is to defuse your bomb.

You must do the assignment on the virtual machine. Your virtual machine *must* be connected to the internet when you are running it as the bomb will connect with our server when it explodes or a phase is defused.

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, and you lose some points in the final score for the lab. You do not necessarily lose the points immediately. Once you complete a phase you will not lose points in that phase if you explode the bomb in that phase at a later time (i.e. accidentally as you try to get to a later phase). You also will always GAIN points for completing a phase regardless of how many times the bomb has exploded as you worked on that phase.

So there are consequences to exploding the bomb. You must be careful! But remember solving a phase ALWAYS adds points to your score.

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. So the maximum score you can get is 70 points.

Each explosion potentially costs you one-half point. We will be tracking which phase your explosion occurred, subject to the rules above. If you finish a phase, only then are the point costs applied. First, we will grant you **1 free explosion for phases 1-4** and **3 free explosions for phases 5-6**. Beyond that you gain the larger of 40% of the maximum points for the phase OR 100% of the maximum points for the phase minus the explosion points (i.e. the sum of the one-half point deductions). For example, if you incur thirteen or more explosions before you solve phase one, but later solve it successfully, you would gain 4 points. If you only had five explosions as you solved phase 1 but then solve it you would earn eight points for that phase.

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, Bruin Nation added this feature so you don't have to keep retyping the solutions to phases you have already defused. You will notice that your repository does not come with this text file. It will be automatically generated after you complete the first phase, and appended to as you complete subsequent phases.

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 and automatically logged by the software. Clarifications and corrections will be posted on the course Piazza page.

Handin

There is no explicit handin. The bomb will notify our server automatically each time it explodes or you defuse a phase.

We have included the ability for you to check your score and last completed phase. In your repo, there should be a script `CheckScore.sh`. Running `./CheckScore.sh` will output your attempts at each

phase, as well as the one you last completed. Make sure you are connected to the Internet when running the script.

Since our server logs your successful phase completions or bomb explosions we will handle grace day usage by just looking at the time stamps. You do not need to do anything special to use grace days. However, don't attempt to pass any more phases after the due date as we will count those as a grace day usage.

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 could lose 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 our server's 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.