CSCI 247 Winter 2017
Project 1: Defusing a Binary Bomb
Assigned: January 29th
Due: February 17th 5:30 p.m.

Michael Meehan

Your lab TA is the primary contact for this project but as always take advantage of the tutoring service and my office hours. As always, if you can't make my office hours, email to arrange an appointment.

1 Introduction

The nefarious *Dr. Evil* has planted a slew of "binary bombs" on our lab 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://scofield.cs.wwu.edu:24703/
```

This will display a binary bomb request form for you to fill in. Enter your WWU CS department user name and WWU email address and hit the Submit button. The server will build your custom unique bomb and return it to your browser in a `tar` file called `bombk.tar`, where k is the unique number of your bomb.

Save the `bombk.tar` file to a (protected) directory in which you plan to do your work in the file space under your home directory somewhere on the CS department lab machines running Linux. 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, i.e. this file.

If for some reason you mess up and request multiple bombs (why I'm not sure?), this is not an unsolvable problem. Just work on the first one you got and delete all the rest.

Step 2: Defuse Your Bomb

Your job for this lab is to defuse your bomb.

You must do the assignment on one of the CS department lab machines. This includes the machines in CF414, CF416, CF418, CF162, CF164, CF405, and the "linux pool." 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, and you lose 1/2 point (up to a max of 20 points (40 explosions)) in the final score for the lab. So there are consequences to exploding the bomb. You must be careful!

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.

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 Canvas pages if needed.

Handin

There is no explicit handin. The bomb will notify my server 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://scofield.cs.wwu.edu:24703/scoreboard
```

This web page is updated continuously (every 40 seconds) to show the progress for each bomb.

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 scofield, a computer in my office. Brute force attempts would saturate the network with many millions of these messages, and cause the system administrators to become irritated and 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 (unfounded) 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 Canvas pages for this course

```
http://canvas.wwu.edu
```

has a very handy single-page `gdb` summary that you can print out and use as a reference. The bombs are designed for 64 bit machines since all the cs department lab machines are 64 bit, so you will want to get the 64 bit version of the single page reference. The 32 bit version is provided simply for completeness. 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 shell prompt. Some people also like to run `gdb` under `gdb-mode` in `emacs`, if you're an emacs type like me, give it a try.

- `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 it finds embedded 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 lab TA or the tutors or me for help.