CS341: Fall 2024
Programming Project #2:
Defusing a Binary Bomb (a.k.a. ***The BombLab***)
Assigned: Oct. 3 **Due: October 28, 2024**

Soraya Abad-Mota (soraya@unm.edu) is the lead person for this lab.

Note: there is some generic "template" information in this document from the developers at Carnegie Mellon University who coded this assignment. HOWEVER, we have also included information specific to this class, including the grading rubric and what we expect you to turn in. PLEASE READ THIS DOCUMENT CAREFULLY! before you begin, really, no kidding.

1 Introduction

The nefarious *Dr. Evil* has planted a slew of "binary bombs" on the cs 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 0: Read this statement thoroughly

Before you begin your work on this project, you must read this complete statement, take notes, and make sure you know how to request your bomb from the server through one of the following methods:

- Connecting from my CS workstation (has a cs.unm.edu domain name and IP address).
- Connecting my laptop, using cs VPN over LoboWifi or from your own internet connection at home as long as you are using the cs VPN. To use this method, consult the link: `https://helpdesk.cs.unm.edu/help/en-us/17-vpn/16-computer-science-vpn`

- Connecting using NoMachine to a 2nd floor lab machine (e.g. `phobos.cs.unm.edu`)
- Connecting using NoMachine to a B146 lab machine (e.g. `b146-08.cs.unm.edu`)

To learn how to use the NoMachine option, please see:
`https://www.cs.unm.edu/computer-facilities/remote-login.html`.

You will request your bomb through a web browser, but you need to be connected to a machine in the cs domain for that, which is why we provide the options for doing it in the list above.

There is a lab session on `gdb` and there is supplementary material i.e. a couple of videos about it on canvas, in the `Videos for specific Topics` module. Please practice using gdb, since this is the main tool for doing this project. The idea is to use `gdb` to set breakpoints in the execution so that you don't produce unnecessary explosions on your bomb. You'll see why having very little explosions is better for your grade. If you make a mistake and do not want to get an unnecessary explosion, remember you can always use `ctrl^C`.

For this project please use any cs machine, especially the ones located in B146 lab in Centennial.

We'll see you back here in a couple of days.

Step 1: Get Your Bomb

You can obtain your bomb by pointing your Web browser at:

```
http://bomblab.cs.unm.edu:15213/
```

(**NOTE:** But don't try it until you have read this whole document and have attempted one of the accesses to the cs machines indicated in the previous section of this document.)

This will display a binary bomb request form for you to fill in. Enter your name in the format: lastname_firstname and your 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. Memorize `k` or write it down for future reference. You also need to answer a starter quiz on canvas, where you specify the number of the bomb you will be using.

If you are trying to acquire your bomb from outside of the cs network domain, it might be slow, so please be patient.

Log in to one of the cs machines in the b146 lab in Centennial by doing
`ssh yourUsername@b146-xx.cs.unm.edu`
where xx is a number between 01 and 70.

Create a directory for your work with the bomblab and protect it so that only you can read and write in that directory. Save the `bombk.tar` file in that (protected) directory. (You can do `chmod 700 bomblab` to protect it.)
Then give the command: `tar -xvf bombk.tar`, where k is the number of your bomb.
This will create a directory called `./bombk` with the following files:

- `README`: Identifies the bomb and its owner.
- `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. Make sure you write down the number of the bomb you'll be using.

Step 2: Defuse Your Bomb

Your job for this lab is to defuse your bomb.

You must do the assignment on any of the b146-xx.cs machines or any of the other cs machines available to you via `ssh`. 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 (`gdb`) 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 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. Therefore 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,

```
    soraya@b146-06.cs>  ./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 single-step through the assembly code and 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 Canvas.

Handin

The bomb will notify your instructor automatically about your progress as you work on it, as long as you work from a cs machine. You can keep track of how you are doing by looking at the class scoreboard at:

```
http://bomblab.cs.unm.edu:15213/scoreboard
```

This web page is updated continuously to show the progress for each bomb.

What to submit

Since the bomb may be invoked with a text file with the solutions to each phase as an argument, as you defuse phases, build the text file with the solutions. We suggest that you name this file: `solbk.txt`, where k is the number of your bomb. On **October 28, 2024** submit this text file to Canvas on the appropriate submission link for Project 2: The *bomblab*.

In order to validate your final score, it is necessary for you to be able to explain in detail how you defused each phase of the bomb, by describing which pieces of code contained the specific clues for each phase. You must submit this explanation in writing, in a pdf file called `justifyk.pdf` that contains documentation for how you solved each phase.

If you consulted any third-party (textbook, other people, websites, etc.) sources to help you defuse the bomb, you must submit a pdf called `creditsk.pdf` citing those sources and how you used them. In summary, your submission consists of three (3) files with the following names as described above:

1. `solbk.txt`,
2. `justifyk.pdf`, and
3. `creditsk.pdf`,

where `k` is the number of your bomb.

Rubric for 140 points total

Solving your bomb: (70) executing the bomb code with your submitted `solbk.txt` file.

- Phase 1 solved (10)
- Phase 2 solved (10)
- Phase 3 solved (10)
- Phase 4 solved (10)
- Phase 5 solved (15)

- Phase 6 solved (15)

Writeup: (70 points)
For each phase, please include the following in the `justifyk.pdf`.

- The ”disas” assembly dump from GDB.
- In the assembly dump, please annotate the significant/important assembly instructions you used to solve the phase. This means a brief comment after the instruction that describes what it does and how it helped you solve the phase.
- Prose that explains the procedure you used to solve the phase. You will be graded on how well we can follow what you did by reading this section.
- If you have not solved all the phases by the project deadline, please include what you have attempted so far to receive some credit.

Points breakdown for the writeup:

- Phase 1:
  - Your annotated assembly dump (5)
  - The procedure you used to solve the phase (5)
- Phase 2:
  - Your annotated assembly dump (5)
  - The procedure you used to solve the phase (5)
- Phase 3:
  - Your annotated assembly dump (5)
  - The procedure you used to solve the phase (5)
- Phase 4:
  - Your annotated assembly dump (5)
  - The procedure you used to solve the phase (5)
- Phase 5:
  - Your annotated assembly dump (7)
  - The procedure you used to solve the phase (8)
- Phase 6:
  - Your annotated assembly dump (7)
  - The procedure you used to solve the phase (8)

Other Notes

You will lose 50% off your final grade for this assignment if any of the following occur:

- You Canvas submission is missing the PDF justifyk.pdf
- Your PDF is not viewable. (i.e., corrupted)

We reserve the right to add additional penalties for excessive submission errors.

About plagiarism...

Bomblab is a common CS assignment in universities across the United States. Consequently, solutions for each phase are extremely easy to find online. **Please resist the temptation to search for solutions on the internet.** Each bomb has each phase generated randomly and it will be a big effort to find out those strings; without understanding what is going on you will learn nothing and will not be able to explain each phase properly. Instead, come to office hours or attend cs tutoring. You may also collaborate with your classmates on ideas of how to go about a specific phase. Each time your bomb explodes or you solve a phase it is logged on our internal server. At the end of the project, we use this log to calculate the average and standard deviation of the attempts it took the class to solve each phase. We consider this data and the quality of your writeup to determine whether your submission is academically honest.

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 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.

From the experience of other semesters, it is advisable that you try to write the C code that matches each phase of the bomb and try to figure out from there, what is the phase doing. This is specially relevant for phases 3 and beyond, but would definitely help the first two phases also.

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`

  With the GNU debugger, 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. With `stepi` you can run a program one instruction at a time.

  With both `gdb` and `objdump` you may show the assembly code for the program you are executing. An important resource for this lab is to review all those commands illustrated in the Lab 5 supplementary material on Week 5, available on Canvas. To avoid explosions on the lab sessions of week 9 you will have another `gdb` lab to refresh your memory on the commands inside `gdb` to avoid explosions. Also, section 3.10.2 p. 279 and figure 3.39 on p. 280 summarize the most important commands of `gdb`. The lab sessions of weeks 5 and 7 used the debugger extensively, review those labs for more information.

  An additional resource is the CS:APP web site

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

  which has a 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 must 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 be a treasure trove of information. If you get stumped, feel free to ask your instructor or your TA for help.