Lab 02: Binary Bomb

CS 351-CUG Fall 2023

Due: 29 Oct 2023, 23:59 PM AOE

Objectives

- Learn to use GDB and other command line debugging utilities
- Practice debugging and tracing binary executables
- Learn to read and understand x86 assembly, including:
  - addressing modes
  - procedure call conventions
  - control structure implementation

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 standard input. 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

As with the previous lab, start by claiming your repository on GitHub via

`https://classroom.github.com/a/9L1tElWH`.

Then, clone your repository on Fourier with the command (replacing USER with your own username):

```
git clone git@github.com:CS351-CUG/lab-02-bomb-USER.git
```

You should now have a directory named "mp-bomb-USER". cd into it and obtain your bomb by running the command ./getbomb.sh.

This command will download a bomb from a remote server and create a directory named "bombN", where N is the unique number of your bomb. In that directory you'll find 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.

Step 2: Defuse Your Bomb

Your job for this lab is to defuse your bomb.

> For this assignment you must do the assignment on fourier.cs.iit.edu.

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 gdb to step through the disassembled binary.

Each time your bomb explodes it notifies the bomblab server, and you lose 1/4 point (up to a max of 10 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 6 points each. Phases 5 and 6 are a little more difficult, so they are worth 8 points each. So the maximum score you can get is 40 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,

```
> ./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 (and this class).

Part 3: Report

For this part of the lab, you will write a report.

Please consider using LaTeX! This document preparation system is widely-used in tech, and it's an excellent skill to have. I recommend using `overleaf.com` – this provides a GUI as well as *many* templates. Find one that suits your style, and get going making beautiful documents with ease!

Completing any lab report in LaTeXwill earn you 5% extra credit. Just include the `.tex` file in your BB submission, and indicate in your report that you've used LaTeX.

In this brief report, you should simply

1. Provide a title, your name, your CWID, and your IIT username
2. Document the location of your GitHub repo on Fourier
3. Describe your approach to this lab
4. Describe issues you experienced and how you addressed them
5. Export this report as a `.pdf` file named: `lab-02_<username>_cs351-cug-fall23.pdf` – all lower-case

Use PDF!

However you write this document (though again, I highly recommend LaTeX), you must submit it as a PDF. Any file type that is *not* a PDF will *not* be graded, and you will not receive credit for this portion of the lab.

Submission

Submit the lab report (PDF with the appropriate file name) to the BB assignment.

You do not need to commit or push any files to GitHub. The bomb will notify your instructor 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:

`https://labs.mseryn.com/scoreboard`

This web page is updated continuously 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:

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 1/4 point (up to a max of 10 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 2680 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 has a handy gdb summary that you can use as a reference and guide. 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. If you get stumped, feel free to ask for help.

Do not wait!

Since many of these are systems you will need to access and use, please *do not wait to begin this lab.* If you have not tried to use your accounts before (Fourier tracks login attempts), issues such as "I can't access the course server" will *not* earn you extra time. Of course, this excludes site-wide issues (server went down, VPN services went down, etc).

Put Simply:

Do not wait to begin this assignment!

Academic Dishonesty

You must do your own work. This does not so much apply to this assignment since it's just intended to confirm account setup and access, but this absolutely will apply to future assignments.

As per our syllabus, any copying (including from other peers, books, the internet, etc.) will earn you a zero on the assignment. Two such instances during the course will result in a report to the department.

Every assignment submission is passed through our course plagiarism detector, and suspected copying will be investigated. I *really* don't want this to happen to anyone. Please, do your own work! Even if you do *reference* something else, be sure the code you write is your own, and be sure you fully understand how it works.

Rubric

Points for this assignment will use the following rubric:

Points	Task
40	Points from completing lab (see scoreboard)
20	Report (PDF) includes name, UID, username, and required discussion points
6	Extra-credit for using LaTeXand including the `.tex` file in the assignment submission on BB
60	*Total possible points*

Reminder about FLDs:

All students have 6 flexible late days (FLDs) to use at their discretion this semester. For each assignment, up to 2 FLDs can be used, and each FLD provides a no-questions-asked 24-hour extension to the due date.

To use a FLD, you need to:

- Email the professor and TA with your name, UID, the assignment title, and how many FLDs you intend to take
- Include in your BlackBoard submission how many FLDs you have used for the submission

You do not need to explain why you are taking FLDs - no questions are asked.

Of course, if you have an extraordinary issue that prevents you completing your work on-time, please just reach out – you don't need to use FLDs for emergencies.

Last modified: Oct 19 2023