4190.308 Computer Architecture, Spring 2019
Bomb Lab: Defusing a Binary Bomb
Assigned: Wed, Mar 27th. Due: Wed, Apr 10th, 23:59 AoE

1 Introduction

Someone has planted a slew of "binary bombs" on our university's machines. A binary bomb is a program that consists of a sequence of phases. Each phase prompts 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 cleared.

The bombs were compiled and planted in binary form. There are too many bombs for SNU's network security team to deal with, so we are giving each of you a bomb to defuse. Your mission is to defuse your bomb before the due date. Good luck!

Step 1: Get Your Bomb

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

```
https://csap.snu.ac.kr/comparch/bomblab/
```

This will display a binary bomb request form for you to fill in. Enter your name and student ID, then hit the Submit button. The server will return your bomb 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 directory in which you plan to do your work; the shared folder of your VM is a good location. 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.

Important notes: (1) we use the student ID to match your bomb(s) to you. If you provide a wrong student ID, you cannot get credit for your work. (2) you can download and work on several bombs, your score will be equal to the score of your best bomb.

Step 2: Defuse Your Bomb

Your job for this lab is to defuse your bomb. We are not sure what exactly the bomb is doing apart from printing `"BOOM!!!"`, so it is a good idea to consider it to be untrusted code. We suggest that you defuse the bomb inside the virtual machine provided on eTL to avoid damaging your computer or operating system.

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) in the final score for the lab.

Each bomb consists of at least six phases. 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. You are also required to submit a report explaining how you solved the lab. This report is worth 30 points, i.e., the maximum score you can get is 100 points.

Although phases get progressively harder to defuse, the expertise you gain as you move from phase to phase should offset this difficulty. The last phase is likely to challenge even the best students, so please don't wait until the last minute to start.

As you solve the individual phases, the bomb keeps asking you for the solutions to the different phases. Blank input lines are ignored. To avoid repeatedly retyping the solutions to phases you have already defused, you can saved them in a file that is given to the bomb as a command-line argument as follows:

```
linux>  ./bomb solutions.txt
```

The bomb reads the lines from `solutions.txt` until it reaches EOF (end of file), and then switch over to manual input (`stdin`).

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 are posted on eTL.

Handin

The bomb notifies 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://csap.snu.ac.kr/comparch/bomblab/scoreboard
```

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

For the report, sign up for an account at our CSAP lab's GitLab server, then clone the Computer Architecture repository at the following URL:

```
https://git.csap.snu.ac.kr/comparchTA/comparch
```

In the `/labs/bomblab` directory you will find a template for the report. Modify and save your report there, then commit and push it to your repository. Please follow the following naming convention: `20xx-xxxxx_bomblab_report.pdf`.

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 in 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 `gdb` 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 (note, however, that the source code for most of your bomb is not available), set breakpoints, set memory watch points, and write scripts.

  Starting your bomb in GDB is as easy as prepending the bomb command with the GNU debugger:

  ```
  linux> gdb ./bomb
  ```

Running and stepping through a program in GDB is easy once you know how. The board 'Additional Material and Resources' board on eTL contains useful information on GDB and the Intel Architecture

`http://etl.snu.ac.kr/mod/ubboard/article.php?id=806856&bwid=1670435`

Here are some other tips for using `gdb`.

- To start your bomb with a file containing the solutions to (some of) the phases, run `gdb` with the `--args` parameter

  ```
  linux> gdb --args ./bomb solutions.txt
  ```

- 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. Also, the web may also be a treasure trove of information. If you get stumped, ask your TAs for help.

Good luck!