CS20300 Fall 2025
Lab2: Defusing a Binary Bomb
Due: Thursday, 16th Oct. 23:59:59 KST

Please post your questions on Piazza:

- `http://piazza.com/kaist.ac.kr/fall2025/cs20300`
- Select folder: lab2

1 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 `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!

2 How to Get Your Bomb

You can obtain your bomb by pointing Web browser at:

```
http://143.248.188.41:40001/
```

This will display a binary bomb request form for you to fill in. Enter your **Student ID** and **your Portal email address (usually KAIST 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.

Save the `bombk.tar` file to your working directory on the your assigned server. 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.

If you request multiple bombs, this is not a problem. However, your grade for this assignment will take into account the highest number of phases you have successfully defused across your bombs, while **explosions from all bombs will be counted together** toward the penalty (see details in the **Grading** section).

3 How to Defuse Your Bomb

Your job for this lab is to defuse your bomb. You must do the assignment on **your assigned class server machine**. 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.

The bomb is divided into **6 phases**. You must defuse all of them to complete the assignment. If you enter an incorrect value, the bomb will explode. Each time this happens, the server is notified and your score will be reduced. So you must be careful when testing your solutions. 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.

3.1 Running the Bomb

Before running the bomb, you may need to make it executable:

```
chmod +x bomb
```

By default, if you simply run your bomb with:

```
./bomb
```

the program will wait for your input interactively. You must type each solution one line at a time, pressing Enter after each line. If the input is incorrect, the bomb will immediately explode.

Alternatively, you can store your solutions in a text file and run the bomb with that file as input:

```
./bomb psol.txt
```

In this mode, the bomb will read answers line by line from `psol.txt` until it reaches EOF (end of file), and then continue taking input from `stdin`. When preparing `psol.txt`, make sure to put each answer on its own line, and include a final newline at the end of the file. Without the final newline, the bomb may not recognize the last answer and will explode.

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 in the rest of your career.

3.2 Quiet Bomb

Before you start defusing with your real bomb, you can practice the first phase with quiet bomb TA uploaded on **klms and piazza**. Feel free to download `bomb-quiet.zip`, unzip, and practice with it. Explosion from `bomb-quiet` will not be graded. **Note that the answer you find for bomb-quiet may not work in your real bomb.** When you are defusing your real bomb, you need to find answer for it from the beginning. To run the quiet bomb, make it executable and then run it like below command. **We strongly recommend starting with the quiet bomb before attempting the real bomb.**

```
chmod +x bomb-quiet
```

```
./bomb-quiet
```

4 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. 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, 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. Following link gives you useful commands for gdb.

`http://csapp.cs.cmu.edu/2e/docs/gdbnotes-x86-64.pdf`

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.

5 Handin

There is no explicit handin. 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:

```
http://143.248.188.41:40001/scoreboard
```

This web page is updated continuously to show the progress for each bomb(not a final grade). Note that, GitLab is not required for this assignment.

6 Grading

The bomb lab consists of six phases, for a total of **70 points**.

- Phases 1–4: **10 points each**
- Phases 5–6: **15 points each** (slightly more difficult)

Each time your bomb explodes, you will lose **0.5 points**, up to a maximum deduction of 20 points. Scoring is handled automatically, and the score shown on the scoreboard will be your final grade for this assignment. While you may request **more than one bomb**, your final grade will be based on the highest number of phases you have successfully defused, with **explosion penalties accumulated** across all of your bombs.

6.1 Late Submission Policy

- Late submissions are penalized by **30% per day**.
- Only one day after the due date is allowed.