Computer Systems 2022
Lab Assignment 2: Debugging Lab

Assigned: Tuesday, September 15

Due: Thursday, September 22, **23:59**.

1 Introduction

The nefarious *Dr. Evil* has planted a slew of “binary bombs” on our 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 group 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 Logistics

Please read the complete assignment carefully before you start working.

For this assignment you are requested to work in pairs, with the same partner as for the previous lab. As part of this assignment, each pair is requested to submit the codes that defuse the entire bomb (all phases) and a ”lab report” (see section 5 for more details). Clarifications and corrections will be posted on the Canvas course page, if the need arises.

3 Get Your Bomb

Each group of students will attempt to defuse their own personalized bomb. Each bomb is a **Linux** binary executable file that has been compiled from a C program. To obtain your group’s bomb, ask the TA to assign a number for your team-of-2 (preferably the same number as for the previous assignment). As soon as you got a bomb number, you can download the bomb from Canvas.

To work locally (i.e., using a native or VM Linux on your own machine), you should just download the code from Canvas, store it in an appropriate folder (e.g., lab2 or DebugLab) and decompress it.

The zip file you receive contains two files:

- `bomb`: The executable binary bomb.
- `bomb.c`: Source file with the bomb's main routine.

Before you move on to defusing the bomb, please make sure it is executable. To set executable rights, in your bomb folder, type:

```
chmod u+x bomb
```

You should now be able to execute the code and you will see a ”friendly” message. Time to start defusing.

4 Defuse Your Bomb

You can use many tools to help you with defusing the 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.

A bomb has 6 phases, each worth 3 points, for a total of 18 points. Your lab report (where you document how you defused the bombs) will be graded separately.

The phases get progressively harder to defuse, but the expertise you gain as you move from phase to phase should offset this difficulty.

The bomb ignores blank input lines. If you run your bomb with a command line argument, for example,

```
linux>  ./bomb solution.txt
```

then it will read the input lines from `solution.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. In the `README` an example of the types of debug-commands you could use.

5 Lab Report

You will present your approach to solve this assignment, the challenges you encountered, and the lessons you learned into a lab report.

See the page 'Het labboek'[1] on the website on Academic Skills for some general pointers about what a lab journal/lab report should contain.

[1] `http://www.practicumav.nl/onderzoeken/labboek.html`

For this assignment, you will receive a template of such a lab report, and you will be requested to answer the questions in the provided text (a bit like ”fill in the blanks”, only with longer answers). You will be graded for these answers only, but make sure they are coherent, they make sense in the context of the report you submit, and they are readable. We do not correct grammar mistakes, but text that is not readable will not be graded. Correct answers need not be excessive in length (usually, one paragraph per (sub)question suffices), and can include images, data, or (psuedo)code snippets, if needed.

Each question has an allocated score. When a total score (i.e., the sum of all questions' scores) of a report exceeds 10, you simply have more opportunities to get a 10. Moreover, if your grade does exceed 10, excess points can be used to improve scores for other reports (your TA will manage these grades). Thus, it is in your overall advantage to try your best to answer all the questions.

Once you have filled in your answers, generate the PDF file of the report using your favorite Latex compiler (e.g., Overleaf). The PDF of your report is the file you hand in as your lab report.

Like most of the course material, the lab report is provided in English. If you have problems writing your answers in English please inform your TA or contact the coordinator (Ana) asap.

6 Submission instructions

When you have completed the lab, you will submit two files on Canvas: your solution (i.e., the codes to defuse the bomb) and your lab report.

The solution.txt is the file you used as argument for the bomb, and contains the ”codes” needed to defuse the bomb. Please rename this file as ”solution_BBB_XXXXXX_YYYYYY.txt”, where ”XXXXXX” and ”YYYYYY” stand for the student numbers in your pair, and ”BBB” stands for the bomb number. Then submit the file on Canvas.

Make sure that your report also clearly states the bomb you are solving (i.e., please include the bomb number and your team information). Please make sure you name your report clearly:
”Report_debuggingLab_bbb_xxxxx_yyyyy.pdf”, with the same naming convention as above.

7 Hints *(Do 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 the bomb.

We do make one request: *do not use brute force!* Even though you could write a program that will try every possible key to find the right one, this is not really feasible: 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 (wrong) 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. Moreover, you will not be able to correctly fill in the report questions, thus losing even more points on the report side.

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. Here are some 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.
  - The CS:APP Student Site at `http://csapp.cs.cmu.edu/public/students.html` has little longer single-page `gdb` summary.
  - Here's a `gdb` tutorial online: `http://www.unknownroad.com/rtfm/gdbtut/`
  - For other 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 the code in the bomb. You can also look at individual functions. Reading the assembler code can tell you how the bomb works.

  Although `objdump -d` gives a lot of information, it doesn't tell 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 need to disassemble within `gdb`.

- `objdump -s`

  Use to see the contents of all sections of the executable. For instance, the .rodata section contains the read-only data of the program.

- `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` and `man` are your friends. In particular, `man ascii` might come in useful.