T-107-TOLH Computer Architecture
Lab Assignment 2: Bomb Lab
Assigned: Okt. 13th., Due: Thursday Oct. 30th.

Teacher responsible is Gylfi Th. Gudmundsson (`gylfig@ru.is`).

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. 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

You can obtain your bomb simply by logging in to our precious server `skel.ru.is` using your ru-loggin (as usual) and you will find that a directory will have been added to your home folder, **bombXXX/** where the XXX has nothing to do with the awesome Vin Diesel movie but is in fact the id of your particular bomb.

In this folder you will find 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.

The bomb, should you need to re-extract it, can be found in the `tar` file called `tolh25_bomb.tar`.

Please make sure you have received a working bomb, i.e. the files are there and the tar file is not of zero bytes in size. If there is anything wrong you MUST notify your teacher(s) that will assist you in obtaining a working bomb. If for some reason you lose your bomb or obtain multiple bombs, this is not a problem. Choose one bomb to work on and delete the rest.

Step 2: Defuse Your Bomb

Your mission, should you choose to accept it, is to defuse your bomb.

You must do the assignment on our beloved `skel.ru.is` server. 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, who knows.

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 bomb-lab-server, and you lose 1/2 point (up to a maximum 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. So 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,

```
linux]$  ./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` (i.e. reading what you type in). 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.
You really should take note (and remember) that the gdb debugger we use for stepping the assembly is essentially the same as the debugger you will use to step code written in a higher-level language.

To properly configure your gdb to work with assembly, so that gdb shows you all the things you need to see, please copy a configuration file for gdb to your home directory like so:

```
linux]$  cp /labs/tolh25/tools/gdbinit ~/.gdbinit
```

Please note that the filename in the home folder starts with a '.', so you need to type in the command EXACTLY as above. To get rid of the warning in gdb about *.gdbini.local* you can also create the following file:

```
linux]$ touch ~/.gdbini.local
```

Logistics

This is an individual project. All handins are electronic. Clarifications and corrections will be posted in Canvas and Discord and questions can be posted to Piazza. Please note that discussion channels are available for each phase on Discord where you can feel free vent your frustration or boast about your progress.

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://skel.ru.is/tolh25-bomblab.html
```

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

AND! Don't forget to check out the Bomb Lab song (link on http://skel.ru.is).

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.
- 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. In Canvas you will find a *gdb cheat sheet* and **do not forget to copy the gdb config file** (see above).

The CS:APP web site also has useful hints/tools as well as the single-page cheat sheet summary (same as on Canvas).

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

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.

If you don't know what a function in C does we recommend looking at the reference page rather than something like Stack Overflow. The reference pages have clear structure of what the function does, what parameters it takes and what it returns. Here is an example of unsing cplusplus.com to look at how *scanf()* works:

```
https://cplusplus.com/reference/cstdio/scanf/?kw=scanf+
```