CS105 – Computer Systems Fall 2020

Assignment 4: Bomb Lab

Due: Tuesday, September 22, 2020 at 11:59pm PT

The nefarious *Dr. Evil* has planted a slew of "binary bombs" on our course VM. 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 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 pair 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

Each group 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, one (and only one) of the group members should point their web browser to the bomb request daemon at

`http://pom-itb-cs2.campus.pomona.edu:15213/`

Note: you will need to be connected to the Pomona VPN to access this website and download your bomb. If you are unable to connect to the this server even when connected to the VPN (e.g., if your connection times out), let me know immediately and I will manually generate starter code for you.

When you visit the site, it will display a binary bomb request form for you to fill in.

- Where it says "User Name," enter your name and your partner's name. Note that you aren't allowed to use spaces, so use underscroes or hyphens.
- Enter the email address of just one team member.
- Then 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.

Copy the `bombk.tar` file to a (protected) directory on the VM in which you plan to do your work. You should be able to use something like:

```
scp ~/Downloads/bomb1.tar username@pom-itb-cs2.campus.pomona.edu:~
```

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 for some reason you request multiple bombs, just 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.

You must do the assignment on the course VM. 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 at the end of this document for some tips and ideas. You will no doubt use `gdb` to step through the disassembled binary.

Each time your bomb explodes it notifies the bomblab server. I'm feeling friendly today so you won't lose points, but the explosion will be recorded. So you might want to think about how to avoid exploding the bomb!

Phases 1 through 4 are each worth 10 points, and you will receive an additional 2 points for submitting a feedback file, for a total of 42 points.

Although phases get progressively harder to defuse, 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,

```
% ./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 do not have to keep retyping the solutions to phases you have already defused. Please take advantage of this vulnerability by creating a file named `solution.txt` and recording your solutions in this file.

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.

Practical Details

You must work with your assigned partner.

The bomb will notify me 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://pom-itb-cs2.campus.pomona.edu:15213/scoreboard`

This web page is updated continuously to show the progress for each bomb. Note: you will need to be connected to the VPN to access this website.

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 not always easy, and I don't particularly recommend it. 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:

- 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 come find you.
- We have not 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. Or the universe ends.

There are many tools which are designed to help you figure out both how programs work, and what is wrong when they do not 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`

  As you saw in the Debugger lab, the GNU debugger 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.

  You will likely find your notes and/or the gdb references from last week to be helpful. Here are some other tips for using `gdb`.

  - To keep the bomb from blowing up every time you type in a wrong input, you will want to remember how to set breakpoints.
  - 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 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 does not 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` (possibly after partially running the program).

- `strings`

  This utility will display the printable strings in your bomb.

Looking for a particular tool? How about documentation? Do not 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, keep in mind that you may not look for solutions on the web. And remember that the mentors (and I) are here to help you!

One other useful fact: you will find that the bomb has many functions with descriptive names. All functions do what their names say. Also remember that `sscanf` is a built-in library function. Do not try to reverse-engineer it unless you have several months to spend; instead read the manual page.

Step 3: Upload your files

Before you submit, please add your name and your partner's name to the end of your solution.txt file. Then please upload to submit.cs your file `solution.txt` and together with a file called `feedback.txt` that answers the following questions:

1. How long did each of you spend on this assignment?
2. Any comments on this assignment?

As always, how you answer these questions **will not affect your grade**, but whether you answer them will.