Middle East Technical University | Department of Computer Engineering

CENG 331

Computer Organization

Fall '2016-2017

Hw 1: Defusing a Binary Bomb

Due date: 11 November 2016, Friday, 17:00

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 of you 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 Specifications

2.1 Step 1: Get Your Bomb

Each student will attempt to defuse his/her own personalized bomb. Each bomb is a Linux binary executable file that has been compiled from a C program. To obtain your bomb, you should point your web browser to the bomb request daemon at

`http://pulbiber.ceng.metu.edu.tr:15213/`

Note that, you should only access the daemon from ineks. **You are allowed to work with ONLY one bomb. After getting your bomb, you are strongly recommended to take copies of it to prevent it from any accidental deletion/corruption. No excuse will be accepted to work with multiple bombs. That means, the ones working with multiple bombs will get no credit from this lab.**

Fill out the HTML form with your Student ID and Name Surname, and then submit the form by clicking the *"Submit"* button. The request daemon will build your bomb and return it immediately 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 (protected) directory in which you plan to do your work. 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.

2.2 Step 2: Defuse Your Bomb

Your job is to defuse the bomb. **You must do the assignment on the class machines (ineks).** 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 they say.

You can use many tools to help you with this; 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. So there are consequences to exploding the bomb. You must be careful!

Maximum score you can get is 70 points. If you can solve all of the phases.

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. 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.

2.3 Logistics

This is an individual assignment. All handins are electronic. Any clarifications and revisions to the assignment will be posted on ceng331 newsgroup.

2.4 Handin

You will submit strings that defuse your bomb using COW System as a single file named **xxxxxxx.txt**, where **xxxxxxx is your 7 digit student id**. Also, the bomb will notify us automatically after you have successfully defused it. You can keep track of how you (and the other students) are doing by looking at

```
http://pulbiber.ceng.metu.edu.tr:15213/scoreboard
```

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

2.5 Hints

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. 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 TA for help.