ICS Lab2: Defusing a Binary Bomb

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 student a bomb to defuse. Your mission, whom 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

You should work individually in solving the problems for this assignment. Any clarifications and revisions to the assignment will be posted on our website and forum.

3. Get Your Bomb

Each student will attempt to defuse his/her own personalized bomb. The bomb is generated for each student such that everybody does a different job. Each bomb is a Linux binary executable file that has been compiled from a C program. To obtain your bomb like lab1, except that the name of lab is “lab2”. You will get the following three files in the directory “lab2”:

- ✧ **README**: Identifies the bomb and its owners.
- ✧ **bomb**: The executable binary bomb.
- ✧ **bomb.c**: Source file with the bomb’s main routine.

After getting your own bomb, you must _**FIRSTLY**_ check README that the bomb owner is not anybody else. If README says that the bomb owner is somebody else, please contact your TA. Or else you could not receive your credits from the auto-grader.

4. Defuse Your Bomb

Your job is to defuse the bomb.

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 staff, and you lose 1/4 point (up to a max of 10 points) in the final score for the lab. So there are consequences to exploding the bomb. You must be careful!

Each phase is worth 10 points, for a total of 60 points.

The phases get progressively harder to defuse, but 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,

```
$./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. For example, you have defused the first three phases whose inputs are "hello world.", "1 2 3 and four" and "I don't like ICS labs...", you could write them in file psol.txt as the following format and use it as input to the bomb.

```
hello world.
1 2 3 and four
I don't like ICS labs...
```

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.

5. Hand-In

There is **no explicit hand-in** (**no need to "svn ci ..." !**). The bomb will notify your instructor automatically after you have successfully defused it, so please ensure the machine you used to defuse bomb links to our network (ex: you can browse the web page http://ipads.se.sjtu.edu.cn/).

You can keep track of how you (and the other groups) are doing by looking at the link published on https://ipads.se.sjtu.edu.cn/courses/ics/index.shtml.
This web page is updated continuously to show the progress of each bomb.

6. 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.

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 1/4 point (up to a max of 10 points) every time you guess incorrectly and the bomb explodes.
- ✧ Every time you guess wrong, a message is sent to the staff. 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 (wrong) assumptions that they all are less than 80 characters long and only contain letters, then you will have 2680 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. 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.
  - ✧ 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.
  - ✧ http://ipads.se.sjtu.edu.cn/courses/ics/tutorials/gdb-ref.txt

- ➢ **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 printf might appear as:

  ```
  171d:  e8 6e f8 ff ff    callq  f90 <printf@plt>
  ```

  To determine that the call was to printf, you would need to disassemble within gdb.

- ➢ **strings**

  This utility will display the printable strings in your bomb.