CS 211: Computer Architecture

Homework 4: Bomb lab

1 Introduction

The purpose of this assignment is for you to become familiar with the x86 Instruction Set Architecture (ISA). 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 everyone a bomb to defuse. Your mission is to defuse your bomb before the due date. Good luck, and welcome to the bomb squad!

2 Instruction

To download your bomb, go to:
http://cray1.cs.rutgers.edu:20211

Fill the form up with your NetID and your email address to get your bomb package. The file that you will get is in the format bombN.tar, where N is your bomb ID. If you havent downloaded it in the ilab machines, copy the file there and untar your bomb into your home directory.

You must do this assignment on one of the ilab machines.

You can then untar the bomb with:

```
tar -xvf bombID.tar
```

It will create a directory bombID that should contain the following files:

1. bomb: The executable binary bomb
2. bomb.c: (partial) source file with the bombs main routine

Your job is to defuse the bomb. You can use many tools to help you with this; please look at the tools section for some tips and ideas. The best way is to use a debugger to step through the disassembled binary. The bomb has multiple phases. The phases get progressively harder to defuse, but the expertise you gain as you move from phase to phase should offset this difficulty. Nonetheless, the latter phases are not easy, so please dont 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 mysolution.txt
```

then it will read the input lines from `mysolution.txt` until it reaches EOF (end of file), and then switch over to stdin (standard input from the terminal). In a moment of weakness, Dr. Evil added this feature so you dont 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.

IMPORTANT: Every time that the bomb explodes, you will lose 0.5 points. It is important that you use breakpoints and avoid those unnecessary explosions.

3 Checking your Work

We provided a webpage where you can check your work. Here you can access the scoreboard to verify how many points you have, up to which phase you have defused the bomb, and so on.

http://cray1.cs.rutgers.edu:20211/scoreboard

4 Tools

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 2will try every possible key to find the right one, but the number of possibilities is so large that you wont be able to try them all in time. There are many tools which are designed to help you figure out both how programs work, and what is wrong when they dont 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 is a command line debugger tool available on virtually ev- ery 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, youll 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.
  - The CS:APP Student Site has a very handy gdb summary (there is also a more extensive tutorial)
- `objdump -t bomb`: This will print out the bombs 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 bomb`: 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.
- `strings -t x bomb`: This utility will display the printable strings in your bomb and their offset within the bomb. Dont forget, the commands apropos and man are your friends.

5 Submission

You have to submit the assignment using Canvas. Your submission should be a tar file named `bombID.tar` that can be extracted using the command:

```
tar -xf bombID.tar
```

Extracting your tar file must give a directory called bombID. This directory should contain the same files that you downloaded, along with the file mysolution.txt to defuse the bomb. To create the tar file that you will submit after finishing your programming assignment, you will use the following command line, in the parent directory of bombID:

```
tar -cvf bombID.tar bombID
```