91.305 Computer Architecture, Fall 2002
Assignment 9: Defusing a Binary Bomb
Assigned: Dec. 4, Due: Wed. Dec. 11

Fred Martin (`fredm@cs.uml.edu`) is the lead person and bomb squad chief for this lab. The lab was created by Professors Randal E. Bryant and David O'Hallaron of Carnegie Mellon University.

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 I am giving each student 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

Students 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 personalized bomb, visit the 91.305 course home page after 2:00 pm today:

```
http://www.cs.uml.edu/~fredm/courses/91.305/
```

Look for the link to a bombs page at the top of the course home page. Then find the bomb for you, identified by the username you gave me at the beginning of the semester.

To play with the bomb, you will have to get the bomb onto an IA-32 Linux machine of your choosing (e.g., `mercury.cs.uml.edu`, `pollux.cs.uml.edu`).

Here is one way to do this. From a web browser, copy the link to your bomb. Then, from the Unix prompt of the Linux machine, say:

```
lynx pasted-link-to-your-bomb
```

The `lynx` web browser will then let you download and save your bomb.

Alternately, you can copy your bomb directly out of the directory from which it is stored. The bombs are located in:

```
/usr/cs/fac4/fredm/public_html/courses/91.305/bombs/
```

You can find your bomb in that directory and copy it to your own.

Step 2: Defuse Your Bomb

Once you have received your bomb from the bomb daemon, save it in a secure directory. 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.

In the CMU version of this lab, each time your bomb exploded, the staff were notified and you lost points:

> *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!*

But I have given you special bombs that just display a message when they explode. So go ahead, blow them up at will, there is no penalty for doing so.

You earn points for defusing the bomb, however. Each phase is worth 6 points, for a total of 36 points for defusing the whole bomb.

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.

In fact, no one should try to do this in one sitting. You'll have a lot more fun with the assignment if you give yourself several sessions to work on it.

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 accidently 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.

Logistics

You may work in a group of up to 2 people. If so, you must indicate in your write-up which individual's bomb you have defused.

Any clarifications and revisions to the assignment will be posted on the class home page and/or the ikonboard. **There will be hints on the ikonboard system, so look there.**

You should do the assignment on the class machines. 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. Well, if you have an Intel-based Linux machine at home, you can go ahead and defuse the bomb on that as well.

Hand-In

In the original CMU lab design, each personalized bomb notified the instructor automatically after you have successfully defused it, so no further write-up was required.

For us, notification is disabled, so I am requiring a written writeup, along with the input file you used to defuse your bomb.

For each phase that you worked on, your writeup must include:

- The input that you used to defuse it.
- A *detailed discussion of the process you used* to come to your solution. This should be at least three paragraphs long.

Even if you did not successfully solve a phase, turn in a description of what you tried and how you went about it.

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:

The first two reasons don't apply to us, but the third reason alone makes brute force effectively impossible.

- 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 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. 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.
  - The CS:APP Student Site at `http://csapp.cs.cmu.edu/public/students.html` has a very handy single-page `gdb` summary. *I have appended this document to this handout.*
  - 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 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` and `man` are your friends. In particular, `man ascii` might come in useful. Also, the web may also be a treasure trove of information. If you get stumped, feel free to ask for help on the ikonboard system.