Assignment 2: Binary Bomb

- Due Sep 23, 2022 by 11:59pm
- Points 100
- Available until Sep 26, 2022 at 11:59pm

This assignment was locked Sep 26, 2022 at 11:59pm.

The nefarious Dr. Evil has planted a slew of “binary bombs” on our CADE lab1 machines. A binary bomb is a program that consists of a sequence of phases, each one expecting 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, 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

Obtain your bomb by pointing your web browser to:

> **http://cs4400.eng.utah.edu:4400 (http://cs4400.eng.utah.edu:4400)**

This page displays a binary bomb request form for you to fill in. Enter your uID, email address, and hit the Submit button. **IMPORTANT**: Enter your exact uID with a **lowercase 'u'** followed by 7 digits.

The server builds your bomb and returns it to your browser in a tar file called *bomb**k**.tar*, where ***k*** is the unique number of your bomb.

You should request only one bomb and save it. If you request a second bomb with the same uID, the second bomb will be different. If you have an issue with a bomb and need a new one, alert the course staff. (Points may be deducted for requesting more than one bomb.) **This means it is YOUR duty to download your bomb early and make sure it works.** No deadline extensions are granted due to server troubles because you procrastinated in downloading 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
```

which creates a directory called *bombk* with the following files:

- *README* — Identifies the bomb and its owner.
- *bomb* — The executable binary bomb.

- *bomb.c* — Source file with the bomb's main routine and a friendly greeting from Dr.Evil.

Step 2: Defuse your bomb

You must do the assignment on one of the lab1-xx.eng.utah.edu 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 we hear.

You can use many tools to help you defuse your bomb. Look at the Hints section below 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 deducts 1 point from your final score for the assignment. There are consequences to exploding the bomb, so you must be careful!

The first four phases are worth 10 points each. Phases 5 and 6 are a little more difficult, so they are worth 15 points each. The maximum score you can achieve is 70 points. (Note that this score will be scaled and recorded out of 100 points on Canvas.)

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 challenges even the best students — DO NOT 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 sol.txt
```

then it reads the input lines from *sol.txt* until it reaches EOF (end of file) and then switches over to stdin. In a moment of weakness, Dr. Evil added this feature so that you do not have to keep retyping the solutions to phases you have already defused.

To avoid accidentally detonating the bomb, you must single-step through the assembly code and set breakpoints. You should also inspect the registers and memory states. One of the nice side-effects of doing this assignment is that you will get very good at using a debugger, which is a crucial skill that pays big dividends the rest of your career.

Submission

There is no explicit file submission for this assignment. The bomb notifies the course staff 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://cs4400.eng.utah.edu:4400/scoreboard (http://cs4400.eng.utah.edu:4400/scoreboard)

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

Hints

Run your bomb under a debugger, watch what it does step by step, and use this information to defuse it.

Write down everything! Stepping through assembly and keeping all the relevant registers and memory values straight in your head is nearly impossible. **If you are working on this assignment without a piece of paper in front of you, you are doing it wrong.**

Do not use brute force! You could write a program that tries every possible key to find the right one. But this is not a good idea for several reasons:

- You lose 1 point 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 have not told you how long the strings are, nor have we told you what characters they contain. Even if you made the (incorrect) assumptions that they all are less than 80 characters long and only contain letters, you would not guess the answer before the due date.

There are many tools that are designed to help you figure out both how programs work, and what is wrong when they do not work. Below 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 every Linux platform. You can trace through a program line by line, examine memory and registers, look at both the source code and assembly code, set breakpoints, set memory watchpoints, and write scripts. (Note: You are not given the source code for most of your bomb.)

  The **textbook website (http://csapp.cs.cmu.edu/public/students.html)** includes a handy single-page gdb summary for your 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, set a breakpoint before the explosion routine.
  - For online documentation, type "help" at the gdb command prompt; or type "man gdb" or "info gdb" at a Unix prompt. You may also consider running gdb under the gdb-mode in emacs.

- objdump -t

  This command displays an executable's symbol table, which includes the names of all functions and global variables, the names of all the functions called, and their addresses. You may learn something by looking at the function names of your bomb!

- objdump -d

This command disassembles the code of an executable. You can also look at individual functions. Reading the assembly 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 is to a function called *sscanf*, you must to disassemble within gdb.

- strings

  This utility displays the printable strings in an executable.

Looking for a particular tool or documentation? Commands "apropos", "man", and "info" are your friends. In particular, "man ascii" might be useful, and "info gas" gives you more than you ever wanted to know about the GNU Assembler.

The bomb assignment was created by Randal E. Bryant and David R. O'Hallaron for Computer Systems, A Programmer's Perspective.