COSC 30203 Fall 2024
Defusing a Binary Bomb
Assigned: September 24, Due: 23:59, October 9

Introduction

The nefarious *Dr. Evil* has planted a slew of "binary bombs" on our class machine. 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.

Since there are so many bombs to deal with, each student will be given a unique bomb to defuse. Your mission, which you have no choice but to accept, is to defuse your bomb before the due date.

Step 1: Get Your Bomb

You can obtain your bomb by pointing your Web browser at:

```
http://babbage.cs.tcu.edu:30203/
```

This will display a binary bomb request form for you to fill in. Enter your TCU username and TCU email address and hit the `Submit` button. The server will build your bomb and return it to your browser in a `tar` file called `bomb<k>.tar`, where *k* is the unique number of your bomb.

Upload the `bomb<k>.tar` file to a (protected) directory on `babbage.cs.tcu.edu` in which you plan to do your work. Then give the command: `tar xvf bomb<k>.tar`. This will create a directory called `./bomb<k>` 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 and a friendly greeting from Dr. Evil.

You may only work with one bomb; the first one that you download.

Step 2: Defuse Your Bomb

Your job for this lab is to defuse your bomb.

You must do the assignment on our class machine: `babbage.cs.tcu.edu`. 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 it has been said.

You can use many tools to help you defuse your bomb. 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 2 points in the final score for the lab. So there are consequences to exploding the bomb. 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. So the maximum score you can get is 70 points.

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.

Logistics

This is an individual project. Each student will have their own unique bomb to defuse. All handins are electronic. Clarifications and corrections will be announced in class or via email.

Handin

When you have completely defused your bomb, submit your solution file `psol.txt` to the dropbox on TCU Online. The bomb will notify your instructor automatically about your progress as you work on it. You can keep track of your score and the rest of the class by viewing the scoreboard at:

```
http://babbage.cs.tcu.edu:30203/scoreboard
```

This web page is updated every 15 seconds to show the progress for each bomb.

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.

I 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 2 points (out of 70) 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 administrator (a.k.a. Dr. Scherger) to revoke your computer access.
- You are not told how long the strings are, nor do you know what characters are in them. Even if you made the (incorrect) assumptions that they all are less than 512 characters long and only contain letters, then you will have 26^{512} 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.

- `strings` This utility will display the printable strings in your bomb.
- `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`.

  You can also visualize jumps with the `--visualize-jumps=color` option. This is useful for finding conditionals and loops in your disassembled code.

- `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 (you are not provided with the source code for most of your bomb), set breakpoints, set memory watch points, and write scripts.

  The course web site under the resources tab has a couple of 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.
  - It useful to set breakpoints in `gdb`.
    - start gdb with the command `gdb ./bomb`
    - `layout asm`
    - `layout regs`
    - set any breakpoints
    - `run [psol.txt]`
  - Two new `gdb` commands: `nexti (ni)` and `stepi (si)`. These commands are similar to `next (n)` and `step (s)`, however they will advance by one assembly language instruction. You will be using these two new commands to trace and advance through your bomb almost exclusively.
  - It useful to learn how to "examine" memory. Read about the gdb command `x` and its variants to view memory.
  - 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`.

Looking for a particular tool? How about reading the 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 Dr. Scherger for help.