THE UNIVERSITY of NEW ORLEANS
DEPARTMENT OF COMPUTER SCIENCE

CSCI 2467, Fall 2024

The Bomb Lab: Defusing a Binary Bomb

Assigned: Monday, September 23, Due: Monday, October 14, 11:59PM (CST)

2467 Instructor: Abdullah Naeem

anaeem@uno.edu

9/23/2024

Introduction

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

Login to your `systems-lab` virtual machine on the Web Terminal: ***https://webterminal.cs.uno.edu.***

You can obtain your bomb fromby connecting to this site with your Web browser: ***https://bomblab.cs.uno.edu/***

This will display a binary bomb request form for you to fill in. Enter your username and 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. **Note:** When you click "submit", the server may take a few seconds to prepare your bomb. Please wait, don't click submit repeatedly.

Save the `bomb`*k*`.tar` file to your downloads directory on your `systems-lab` virtual machine. You can move it to any convenient location, preferably your good old 2467. Then give the command: `tar -xvf bombk.tar`. This will create a directory called `./bomb`*k* 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.

If for some reason you request multiple bombs, this is not a problem. Choose one bomb to work on and delete the rest.

Step 2: Defuse Your Bomb

Your job for this lab is to defuse your bomb.
You must do the assignment on the class server, using either an HP terminal in Math 209, or through the Web Terminal. 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. Please look at the **hints** section for some tips and ideas. The best way is to use the `gdb` debugger to step through the disassembled binary.

Gaining and losing points

Each time your bomb explodes it notifies the bomb lab server, and you lose 1/2 point (up to a max of 20 points) in the final score for the lab. So, there are real consequences to exploding the bomb. **You must be careful!**
The first two phases are worth 5 points each. Phases 3, 4, and 5 are worth 10 points each. If you defuse phases 1 through 5, you will have your full 40 points for the lab.
Phase 6 is worth an additional 10 points, meaning you could get 50 out of 40 points by defusing phase 6.
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 strongest students!

Phase	Point value
1	5
2	5
3	10
4	10
5	10
Total	40
6	10 (*extra credit*)

Figure 1: Summary of bomb lab phases

How to "run" the bomb

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 for you in the future.

Handin

Each bomb is unique, so you will need to find your own solutions.

There is no explicit handin. The bomb will notify AutoLab automatically about your progress as you work on it. You can keep track of how you are doing by looking at the class scoreboard on AutoLab. This web page is updated continuously to show the progress for each bomb. Updates may take up to 30 seconds to appear.

Hints *(Please read this!)*

There are many ways of defusing your bomb. You can examine it in detail without ever running the program and figure out exactly what it does. This is a useful technique, but it is 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:

- You lose 1/2 point (up to a max of 20 points) every time you guess incorrectly, and the bomb explodes.
- 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 (incorrect) 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. On the next page we provide 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.
  The CS:APP web site
  `http://csapp.cs.cmu.edu/public/students.html`

  has a very handy single page `gdb` summary that you can print out and use as a reference. (Also Figure 3.39 on page 280 of CS:APP3e has a nice summary) 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.
  - For online documentation, type "`help`" at the `gdb` command prompt, or type "`man gdb`" at a Linux prompt.

  Note for students: `gdb` uses *AT&T* syntax for assembly instructions by default. If you wish to see the *Intel* syntax for assembly, issue the command `set disassembly-flavor intel`. The *Resources* section of the 2467 course website contains a `link to a page describing how to make this change permanent`, so you won't have to set the `disassembly-flavor` every time. The *Aside* on p. 177 of CS:APP3e discusses these "flavors" and their differences.

- `objdump -t bomb`

  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 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. 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`.

  Note for CSCI 2450 students: If you prefer the *Intel* syntax of assembly, use the following command: `objdump -M intel -d bomb`

- `strings`

  This utility will display the printable (ASCII) null-terminated strings in your bomb.

Looking for a more details about a particular tool? Manual pages (the `man` command) are a good reference for the command-line tools. Also `man ascii` might come in useful when looking at ASCII text. If you get stumped, please come to course help hours for assistance. Don't stay stuck!