ECEN 324
Lab Assignment: Defusing a Binary Bomb

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 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!

Even though you have your **own individual bomb** to defuse, you will work in a team of 2 or 3 students to help each other. Your score on this lab is a combination of defusing **six** phases of your own individual bomb and helping teammates defusing up through phase five. (Hopefully, you will help them with phase six also.) Even though you will typically have two teammates to work with, you may discuss the lab with any of the students in the class. Each phase has two or three variations of what the phase does (algorithm) and your individual bomb may be significantly different from the bombs of your teammates.

There is a seventh phase in each of the bombs. This is the 'secret phase.' This is extra credit and you should work on this secret phase individually. To work on the secret phase, you first need to figure out how to get into it.

You are not to use Ghidra (released by the NSA in 2019), or any similar reverse engineering tool. You may use Internet resources to learn about gdb and answer general questions but not to find specific solutions for your bomb.

Step 1: Get Your Bomb

You will be given a bomb to work on. It will appear in your home directory with the name: "bombX.tar"

To unpack the tar file, give the command: `tar -xvf bombX.tar`. This will create a directory called `./bombk` (the "`k`" will be a unique number for your bomb) 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.

If you had already made a directory called '`ecen324`' in your home directory and wanted to put your bomb directory in a subdirectory named `lab4`, the commands to unpack and move things into the subdirectory could be like:

```
[username@jordanvm ~]$ tar -xvf bombX.tar
```

```
bomb2/README
bomb2/bomb.c
bomb2/bomb
[username@jordanvm ~]$ mv bombX.tar bomb2.tar
[username@jordanvm ~]$ mkdir ecen324/lab4
[username@jordanvm ~]$ mv bomb2.tar bomb2 ecen324/lab4
[username@jordanvm ~]$ cd ecen324/lab4
[username@jordanvm lab4]$ ls
bomb2  bomb2.tar
[username@jordanvm lab4]$ cd bomb2
[username@jordanvm bomb2]$ ls
bomb  bomb.c  README
```

Step 2: Defuse Your Bomb

Your job for this lab is to defuse your bomb.

You must do the assignment on the ECEN 324 Linux system. 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 (especially a debugger) to help you defuse (reverse engineer) 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 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! However, don't let this stress you out! Points are deducted in integer increments so you get three for free and even a 10-point deduction is only 1/16th of the grade.

The first five phases are worth 22 points each. Phase 6 is a little more difficult, so it is worth 30 points. So, the maximum score for the six phases you can get is 140 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,

```
    [adambyui@jordanvm bomb2] ./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

Clarifications and corrections will be sent in an email and/or posted to I-Learn.

You will have your own individual bomb but should work closely with two other students. A portion of your grade will be dependent on your teammates getting through five phases.

Handin

There is no explicit handin. All handins are electronic. The bomb will notify your instructor automatically about your progress as you work on it. You can keep track of how you are doing by looking at the class scoreboard in the Unit 05 (Chapter 3) module.

You may also see a more frequently updated scoreboard at:

```
http://157.201.194.253:15213/scoreboard
```

This web page is updated continuously to show the progress for each bomb. This will need to be accessed from on campus, or by using a Windows system in the VDI (`https://vdi.byui.edu`), or running Firefox on the ECEN 324 Linux VM.

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:

- You lose 1/4 point (up to a max of 10 points) every time you guess incorrectly and the bomb explodes. However, points are only deducted in whole point increments.
- 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 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.

If you want a binary bomb that you can blow up as often as you want, you may find one here: https://csapp.cs.cmu.edu/3e/labs.html.

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. You may wish to read section 3.10.2 in the textbook and see Figure 3.39.

  The CS:APP web site

  ```
  https://csapp.cs.cmu.edu/3e/students.html
  ```

  has a 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.

  – 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`.

  – Please note that typing '`dis`' is not the short form of the '`disassemble`' command. Using '`dis`' disables all breakpoints! Uses '`disas`' as the short form of the '`disassemble`' command.

  – The `gdb` command `layout regs` is a nice screen layout when debugging/reverse engineering at the assembly code level since it shows the assembly code and the registers. However, the screen sometimes gets messed up, a ctrl-l (control-ell) usually fixes things.

  – The `gdb` '`source`' command reads commands from a file (a `gdb` script). You might look at `/home/ecen324/gdbBombScript` that is a `gdb` script.

- `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`, `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. If you get stumped, talk to your teammates, other students in the class and your instructor for help. Also, the web may also be a treasure trove of information but sites that provide "too much" help are not to be used, and specifically, the NSA reverse engineering tool Ghidra (or similar tools) are not to be used.

This is not a lab that most people can sit down in one or two sittings and solve it. Work with others and help each other out. Use not only gdb/ddd, but also tools like the "strings" command, and a listing of the disassembled code. Don't try to figure out every little thing about the code and how it works, look at format strings, look at the general flow of what is happening and what happens at the decision points (compare and jump).

WARNING: A solution file created on a Windows system will have "cr lf" for the line terminations. The binary bomb does not like the "cr" and using a solution file with carriage returns in it will cause you to go to explode_bomb. It is recommended that you create your solution file on the ECEN 324 Linux server with the editor of your choice. Another option is to make sure your editor is using only line feeds (lf) as the line termination for your solution file.

Each bomb phase tests a different aspect of machine language programs:
Phase 1: string comparison
Phase 2: loops
Phase 3: conditionals/switches
Phase 4: recursive calls and the stack discipline
Phase 5: pointers
Phase 6: linked lists/pointers/structs