Lab 6

Binary Bomb Defusing Lab

Due: Week of March 28, before the start of your lab section

This is an individual-effort project. You may discuss concepts and syntax with other students, but you may discuss solutions only with the professor and the TAs. Sharing code with or copying code from another student or the internet is prohibited.

The instructions are written assuming you will perform the lab on your account on the *csce.unl.edu* Linux server. There is a "practice bomb" that, if you wish, you may run on any x86-64 Linux environment, but the "live bomb" will only run on your account on your account on the *csce.unl.edu* Linux server.

Learning Objectives

After successful completion of this assignment, students will be able to:

- Disassemble machine code
- Recognize program structures in assembly code
- Trace the execution of assembly language programs

Continuing Forward

Most immediately, your ability to use a debugger to study a process's state will be very useful in AttackLab. Your improved familiarity with assembly code in general, and with program structures specifically, will benefit you greatly on the next exam.

During Lab Time

During your lab period, the TAs will demonstrate how to disassemble a program and how to use gdb to examine the state of a process and to step through its execution. The TAs will also demonstrate solving Phase 1 using the practice bomb. During the remaining time, the TAs will be available to answer questions.

Scenario

In a jarring collision of movie franchises, the CEO of Virtucon makes a Zoom call to the Pleistocene Petting Zoo. For some reason that nobody really explains, you're the only person available to handle the situation. The guy, who sounds kind of like an animated ogre, demands that the Pleistocene Petting Zoo deliver to him a megalodon shark with a head-mounted laser capable of emitting a beam of pure antimatter.

You blurt out, "Then it's not a laser," and then try to explain to him that megalodons are from the Miocene epoch, and expecting to find them at the Pleistocene Petting Zoo would be as ridiculous as a Cretaceous-period tyrannosaur at a Jurassic-themed park.

"Zip it!" commands the guy who kind of looks like the host of a public-access show you used to watch. "Since you won't meet my demand, my minions have placed a 'binary bomb' under your zoo. Because I like really convoluted plans, we put software on your Linux server that controls the bomb. If you do nothing, the bomb will explode. If you turn off the Linux server, the bomb will explode. If you go slower than 50mph, the bomb will – no, never mind that last part.

"The bomb software 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*. The bomb is defused when every phase has been defused.

"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!"

1 Get Your Bomb

Each student will attempt to defuse their own personalized bomb.[1] Each bomb is a Linux binary executable file that has been compiled from a C program. To obtain your bomb, you need to visit the bomb request daemon webpage[2,3] at

```
http://csce.unl.edu:15213/
```

Fill out the HTML form with the your *CSE username* (do *not* use your Canvas username) and email address, and then submit the form by clicking the "Submit" button. The request daemon will build your bomb and return it to your browser in a `tar` file called `bombk.tar`, where k is the unique number of your bomb.

The request daemon needs a few seconds to build your bomb, so it might seem to be non-responsive, and you might be tempted to click "Submit" again. If you accidentally request multiple bombs, that's okay as long as you only defuse one of them.

[1]This lab borrowed from Bryant & O'Halloron and modified by Bohn.

[2]You might be able to visit this webpage only if connected to a UNL network.

[3]Chrome doesn't seem to be willing to make a non- secure (*i.e.* "http" instead of "https") connection. Firefox seems to work fine, as does Lynx.

If you attempt to defuse more than one bomb, we will use the grade for the first bomb that gets an entry on the Scoreboard.

Because your bomb only runs on `csce.unl.edu` (will be referred to as `CSCE` from this point onward), your next step is to move your bomb to `CSCE`. To login to `CSCE`, you will use the same login information as logging in to `cse.unl.edu`. Create a directory to store your bomb on `CSCE` then give the command:

```
tar xvf bombk.tar
```

This will create a directory called `./bombk` 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.

2 Defuse Your Bomb

Your job for this lab is to defuse your bomb.

You must do the assignment on `CSCE`. 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 your favorite debugger to step through the disassembled binary.

Each time your bomb explodes it notifies the bomblab server, and you lose 1/2 point (up to a max of 25 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.[4]

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,

```
csce>  ./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

[4]There is also extra credit, and you will discover how to obtain the extra credit only by thoroughly studying the bomb code to glean its secrets.

inspect both the registers and the memory states. ***The TAs will demonstrate how to do these during lab time.*** 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.

A sample bomb is provided as part of this assignment. You can obtain the practice bomb through Canvas.

Scoreboard

The bomb will notify the 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[5] at:

```
http://csce.unl.edu:15213/scoreboard
```

This web page is updated continuously 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.

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 25 points) 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 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.

[5]You might only be able to visit this webpage only if connected to a UNL network. If you want to check your progress from home, then while you have a terminal connected to csce.unl.edu, type `lynx csce.unl.edu:15213/scoreboard`, *or* connect to the School of Computing's Windows Terminal Server or NoMachine service (see `https://computing.unl.edu/faq`) and use a graphical web browser.

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 course website has useful documents on gdb available on this lab's assignment page.
  - 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`.

- `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, feel free to ask your TA for help.

Turn-in and Grading

You do *not* need to turn anything in for this lab. The BombLab service will automatically record your progress and generate a score.

This assignment is worth 70 points.

_____ **+10** Complete Phase 1

_____ **+10** Complete Phase 2

_____ **+10** Complete Phase 3

_____ **+10** Complete Phase 4

_____ **+15** Complete Phase 5

_____ **+15** Complete Phase 6

_____ **Bonus +10** Complete Secret Phase

Penalties

_____ $-\frac{1}{2}$ For each explosion, not to exceed -25

Epilogue

Having saved the Zoo from Dr. Evil's binary bomb, you relax back in your chair and think about taking a break. Maybe an entire week in which you don't have solve any problems or meet any deadlines – that'd be real nice.

Another Zoom call comes in. *What now!?* you wonder as you take your feet off of the desk to answer the call. An uncomfortable-looking animal handler says, "We can't unlock the food lockers. It's the animals' feeding time, and we can't open the food lockers! It's feeding time, we can't get to the animals' food, and," his eyes dart nervously toward the animal enclosures, "and many of them have sharp, pointy claws and others have big, stompy feet."

To be continued...