COMP.2030 **HW 8**

Assignment: Defusing a Binary Bomb – Worth 200 points.

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 completely defused when every phase has been defused.

Your job is to get your own bomb and defuse it before the due date. A secret phase does NOT get credit. Good luck, and welcome to the bomb squad!

Step 1: Download a bomb in your laptop

You can download a bomb in two ways.

a. If you have access to vlabs, enter http://mercury.cs.uml.edu:15213
b. Make a VPN connection (https://www.uml.edu/it/services/get-connected/remote-access/), and in a browser, enter http://mercury.cs.uml.edu:15213

Step 2: Transfer bomb.tar from your laptop to Mercury System

After downloading your bomb, you need to transfer the tar file to mercury.cs.uml.edu via winscp in Windows. In a Mac, you can use FileZilla or the command line instruction:

scp ./Documents/file_name yourCSusername@mercury.cs.uml.edu:~/file_name

Save the bomb*i*.tar file to a directory on your MERCURY system.

Step 3: Log in to Mercury to uncompress the bomb

Login to mercury by using Power Shell/putty in Windows or MAC's Terminal to run the command

ssh -l yourCSusername mercury.cs.uml.edu

Go to the folder that bomb*i*.tar is saved, and give the following command to uncompress it:

tar -xvf bomb*i*.tar

This will create a directory called ./bomb*i* 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.

Step 4: Pre-process the Executable 'bomb'

The executable 'bomb' is what you need to defuse (reverse engineer). The best practice is to get the assembly language version of 'bomb' by redirecting it to a text file in mercury:

objdump -d bomb > bomb.txt

Now, transfer the file bomb.txt back to your laptop/desktop, and you can add comments to describe the phase functions in a pseudo-C style to understand what each phase is about.

Step 5: Defuse the 'bomb'

Login to mercury system as a CS user. In windows, use putty. In Mac, open the terminal and enter the following command:

ssh -l yourCSusername mercury.cs.uml.edu.

Once you log in the mercury, you can use gdb tool to run the executable 'bomb' while stepping through the disassembled binary. Each time your bomb explodes it notifies the staff, and you lose 0.5 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!

Each phase is worth 10 points or more, for a total of 70 points.

The phases get progressively harder to defuse, but 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,

> ./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

You should do the assignment on *mercury*. 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 they say.

Hand-In

To make sure you get credit, please submit your username and bomb number to blackboard. There is no other hand-in for this assignment. The bomb will automatically report your score as you successfully defuse it. In vlabs or a VPN-connected browser, enter the following to check your score.

http://mercury.cs.uml.edu:15213/scoreboard

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 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 10 points) every time you guess incorrectly and blow up the bomb.

Every time you guess wrong, a message is sent to the staff. 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 (wrong) 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 that 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 bomb

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. Refer to Fig. 3.30 (page 255) for gdb commands.

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 other 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* and *man* are your friends. In particular, man ascii might come in handy. Of course, the web may hold a treasure trove of information.