**COMP.2030 HW #5 Due 4/25 (Wed) Midnight**

**Assignment: Defusing a Binary Bomb – Worth twice the weight of other programming homeworks.**

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.

You have to get a bomb of your own, and defuse your bomb before the due date. Good luck, and welcome to the bomb squad!

**Step 1: Get Your Bomb**

Each bomb is a Linux binary executable file that has been compiled from a C program. With the increased security around the system, you cannot obtain a bomb outside the department,. You should go to the CS lab in Olsen 3rd floor and go to the bomb request daemon at

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

Although it asks for an optional second member in a team, you have to work ALONE. Leave out the optional second member, fill out the html form with your email address and name and then submit the form by clicking the **Submit** button.

The request daemon will build your bomb and return it immediately to your browser in a tar file called bomb*i*.tar, where *i* is the unique number of your bomb (and serves as your personal ID). After downloading your bomb, you need to transfer it to mercury.cs.uml.edu via winscp or fugu.

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

Also, if you make any kind of mistake requesting a bomb (such as neglecting to save it or typing the wrong information), simply request another bomb.

**Step 2: Defuse Your Bomb**

You can use many tools to help you with this; 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 staff, 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!

Each phase is worth 10 points, for a total of 60 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,

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**

You should work on this by yourself. 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**

There is no explicit hand-in. The bomb will notify your instructor automatically after you have successfully defused it. You can keep track of how you (and the other groups) are doing by looking at

[http://www.cs.uml.edu/~bomb/bomb/Spring18Bomb.html](http://www.cs.uml.edu/~bomb/bomb/Fall17bomb.html)

This web page is updated continuously to show the progress of each group.

**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 1/4 point (up to a max of 10 points) every time you guess incorrectly and the bomb explodes**.

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 2680 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. Also, the web may also be a treasure trove of information.