BOMBLAB Fall 2021
Phase 1 - due 10/04 11:59 PM
Phase 2 - due 10/08 11:59 PM
Phase 3 - due 10/13 11:59 PM
Phase 4 - due 10/17 11:59 PM
Phase 5 and (Extra Credit Phases 6 and 7) - due 10/23 11:59 PM

1 Points Breakdown

Phase 1: 20 Pts
Phase 2: 20 Pts
Phase 3: 20 Pts
Phase 4: 20 Pts
Phase 5: 20 Pts
Phase 6: 5 Pts
Secret Phase (Phase 7): 5 Pts

Phases 6 and 7 are for extra credit, and there is no penalty for skipping them. Phases 1 through 5 are required. You may still get credit for phases completed late, but only up to 24hrs after the deadline. You can only receive 85 percent of the points for a phase you submitted late. There is no official policy for obtaining partial credit for phases beyond this.

If after the late submission period you have still not solved a phase you can request the phase solution by posting a private post to instructors on piazza and list the bomb number and phase you would like the solution for. Your score will be sum of the points for all the phases you complete, before the respective deadlines, for a single bomb. As an example of what not to do, if you do phase 1 with bomb 10, and phase 2 with bomb 125, and get both in before your deadline you will only get 20 points.

2 Introduction

The nefarious *Dr. Evil* has planted a slew of “binary bombs” on our class 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!

Step 1: Get Your Bomb

You can obtain your bomb, while on the UIC network, by pointing your Web browser at:

```
http://systems3.cs.uic.edu:15213/
```

This will display a binary bomb request form for you to fill in. Enter your netid 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 `bombk.tar`, where *k* is the unique number of your bomb. If you wish to download the bomb while off campus, you will need to connect to the UIC VPN first. See https://accc.uic.edu/service/vpn for more details about using the UIC VPN.

Save the `bombk.tar` file to a (protected) directory in which you plan to do your work on: systems1.cs.uic.edu systems2.cs.uic.edu systems3.cs.uic.edu systems4.cs.uic.edu 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 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. You can have multiple bombs, but downloading more bombs after phases have been completed, means you will miss out on points for the previous phases.

Step 2: Defuse Your Bomb

Your job is to defuse your bomb.

You must do the assignment on one of the class machines. 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 20 points) in the final score for the lab. So there are consequences to exploding the bomb. You must be careful!

The first five phases are worth 20 points each. Phases 6 and 7 are a little more difficult, so they are extra credit and worth 5 points each. So the maximum score you can get is 110 points. (And the project is out of 100 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,

```
    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 project 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

This is an individual project. All handins are electronic. Clarifications and corrections will be posted on piazza.

Handin

There is no explicit handin. The bomb will notify us of your progress as you work on it, assuming you entered your uic email address when downloading the bomb. You can keep track of how you are doing by looking at the class scoreboard at (while on the UIC network, or connected via the VPN):

```
http://systems3.cs.uic.edu:15213/scoreboard
```

This web page is updated every 10 minutes to show the progress for each bomb. You should check to make sure that your scores are recorded correctly on the server. The server has occasionally crashed, so checking that your scores were received is always a good idea, but remember it may take more than 10 minutes for the score to be updated.

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 20 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.

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 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`.

- `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.

- `.gdbinit gdb config file` The file .gdbinit holds gdb commands so everytime you start gdb certain commands are executed. The file should be in the same directory as the executable you are debugging.

  For example, including the line

  ```
  break main
  ```

  in the file .gdbinit will insert a breakpoint at main everytime you start gdb.

  On the school machine you need to create a .gdbinit file in your root directory (the directory you first login to when you login to the machine) that contains

  ```
  set auto-load safe-path /
  ```

  Then you can create a .gdbinit file in the directory with your bomb that has whatever breakpoints or settings you want loaded everytime you start gdb.

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. Also, the web may also be a treasure trove of information. If you get stumped, feel free to post a question to piazza.