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!

For this lab, you must work on the CS machines. The instructions and commands mentioned here are meant to be issued at the command line while you are logged into a CS machine. That is because the bomb executable communicates with a CS server which tallies progress.

As with project 1, you can work in pairs. Indeed highly encouraged! If you work in pairs, you will submit via Canvas both names, and the highest scoring of the created bombs by the two people (phases solved minus fewest deductions) will be the score given to both students.

Stage 1: Get Your Bomb(s)

While ssh connected to your CS account, create a directory (perhaps call is *proj2*), and from within that directory, issue the following:

```
curl -G http://systems.cs.wwu.edu:24701/?username=X\&usermail=X\@wwu.edu\&submit=Submit > bomb.tar
```

where the X is your CS username. In effect what this does is go to the URL specified (which is ON a machine inside the CS firewall), through port 24701, and provide credentials as if you were accessing that page via a browser. The output (download) is then piped to (sent to) *bomb.tar*

If done correctly, you should now have the file *bomb.tar* in your directory.

Then give the command: `tar -xvf bomb.tar`. This will untar (which is similar to unzip) and create a directory called `bombk`, where `k` is the bomb number that YOU have been given. The directory will have 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.

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 stage bomb explodes it will notify the server, and you will lose 1/2 point (up to a max of 15 points) in the final score. This is for the purpose of preventing a brute force approach (**see the hints**). The first explosion doesn't count towards your tally. 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. Thus the maximum score you can get is 70 points (assuming no deductions for bomb explosions).

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,

```
$./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 this project is that you will get very good at using a debugger. This is a crucial skill that will pay big dividends in the rest of your career.

Submission

There is no explicit handin. The bomb will notify your instructor automatically about your progress as you work on it. Progress stats will be updated and posted periodically to Canvas and/or announcements during lectures.

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 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 15 points) every time you guess incorrectly and the bomb explodes.

- Every time you guess wrong, a message is sent to the bomb server. You could very quickly saturate the network with these messages, and cause the system administrators to revoke your computer access. If you tally far more than 20 explosions (in other words, you are doing a brute force approach) the server will lock you out, and you will get a zero for the assignment. **In other words, do not do a brute force approach.**

- 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. As you've done in previous assignments, 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. The following CS:APP web site includes good resources:

  http://csapp.cs.cmu.edu/public/students.html

  It includes 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.
– Consider the use of the `layout` command. For example, `layout asm` and `layout regs` will show the assembly and registers in real-time.

- `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`:

  ```
  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, while `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 ask your instructor for help.

Phase 1 Hints

- Before even trying to solve the puzzle, figure out where the program blows up the bomb and set a breakpoint there, to prevent explosions.
- Definitely try the `strings` command in gdb
- Invest time in learning a lot of different gdb commands...it will pay off!

Rubric

Item	Points
Phases 1-4	4 x 10 = 40 points
Phase 5	15 points
Phase 6	15 points

Deductions	Points
explosions 2 onward	0.5 points per explosion, capped at 15 points deduction