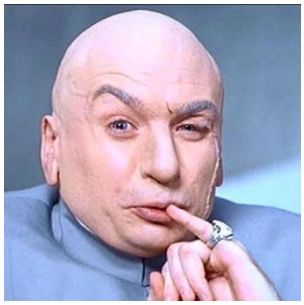
Defusing Bombs

Project Description



The nefarious Dr.Evil has posted a ``binary bomb'' for each student on the class web page. A binary bomb is a program that consists of a sequence of phases. Each phase reads a particular string from standard input, *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

Your bomb has been posted on the class web page in the Projects page, under Project 3. Find your section, and your name in the table, and next to your name is the hyperlink for the tar file that contains your bomb, called **bomb**<nnn>**.tar.gz**, where <nnn> is your unique ID number. Section A has numbers from 100-199, and Section B has numbers from 200-299.

Download this file and un-tar it on an LDAP machine. This will create a **bomb**<nnn> sub-directory that contains three files:

- **bomb** – The bomb executable. Note that the bomb has been compiled to run on LDAP machines and will not run on all hardware. You probably will not be able to run your bomb on your own laptop.
- **README** – A quick description that says who was assigned to work on this bomb.
- **bomb.c** – The C code that was used to create the bomb executable. Note that this file includes a main function and some lower level functions. Dr. Evil was being uncharacteristically kind when providing this code. However, he is truly evil because the bomb.c file does not contain *all* the source code… just the top level.

Step 2: Defusing Your Bomb

Your job for this project is to defuse your bomb.

You must do the assignment on one of the LDAP 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.

For instance, we have heard that it is possible for the bomb to report back to the professor every time it explodes. Some Professors team up with Dr. Evil, and actually subtract points every time your bomb explodes. We are not quite so evil here in upstate New York, but if we hear lots of explosions, we might have to turn this feature on.

Just to show how completely evil Dr. Evil is, he has given you a copy of the top level C file in **bomb.c**. Read it and weep --- it shows you how Dr. Evil can invoke bombs without revealing the detonation mechanism(s).

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.

Note that Dr. Evil likes to keep things challenging, and is not likely to use the same mechanism to guard each phase. 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 reading from *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 contents of memory. 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.

Hints and Suggestions

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.

Dr. Evil has given one piece of good advice: **do *not* try to use brute force**! You could write a program that will try every possible key to find the right one. But this is no good because 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 capital 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 some 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 (Dr. Evil has refused to give you the source code for most of your bomb), set breakpoints, set memory watch points, and write scripts.
- **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, 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.
- **strings** This utility will extract and display all the printable ASCII 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 be 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 ask your instructor or the TA's for help.

Also, keep your eye on the big picture. For instance, there are several calls to subroutines in the bomb code. Often you can make a pretty good guess at what the subroutine does, and do not need to step through the assembler code for the subroutine in detail. In these cases, you can save a lot of time by running *nexti* in gdb instead of *stepi*.

Academic Honesty

Feel free to discuss this project with your classmates, as well as asking for help from the Professor and TA's, and looking for examples on-line. However, you are responsible for your own solution. Since every bomb is different, this shouldn't be a problem

Submitting your Code

When you are done, create a text file that solves your bomb with your userid and bomb number in the file name, for instance: tbarten1.173.txt. Upload your text file on blackboard in the Project 3 area.

Project 3 Grading

The TA's will then check your solution against the correct answers. You will get 15 points for each phase which you answer correctly for a total of 100 points. Notice that 15 does not divide evenly into 100. However, it is possible to get 100 on this project. (Does Dr. Evil control the rules of Mathematics as well?) Even after you have gotten through all phases, you might want to poke around a little to see if you can find any secret hidden treasures that Dr. Evil may have up his sleeve.