# CS 367 (Sections 001/002) Spring 2017
## Lab 1: Defusing a Binary Bomb
## Due: Thursday March 23, 2017 (11:59:59 PM EST)

## 1   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

To obtain your bomb, you need to be able to be on a machine that can connect to the machine zeus.vse.gmu.edu. Logon to zeus.vse.gmu.edu using ssh and your GMU userid/password. Remember that you need to be on a VSE lab machine, on zeus or mason servers, or be connecting remotely through the VPN software. The bottom line is that the technique youve been using to connect to zeus in the previous assignments should also work for this lab.

You can obtain your bomb by pointing your Web browser at:

```
http://zeus.vse.gmu.edu:15223/
```

This will display a binary bomb request form for you to fill in. Enter your user name 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.

**Important:** Due to the heavy load on zeus, sometimes it may take up to 60-90 seconds before the bomb file is loaded on your browser. Please be patient. If you are appropriately connected (VSE lab or via VPN) and the site seems to be down, please send a message on Piazza immediately so others can chime in or an instructor can restart the server.

Save the bombk.tar file to a (protected) directory in which you plan to do your work. 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.

## Step 2: Defuse Your Bomb

Your job for this lab is to defuse your bomb.

You must do the assignment only on zeus. 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 four phases are worth 15 points each. Phases 5 and 6 are a little more difficult, so they are worth 20 points each. So the maximum score you can get is 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 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

This is an individual project. All handins are electronic. Clarifications and corrections will be posted on the course message board.

## Handin

There is no explicit handin. The bomb will notify your instructor automatically about your progress as you work on it. You can keep track of how you are doing by looking at the class scoreboard at:

`http://zeus.vse.gmu.edu:15223/scoreboard.html`

This web page is updated continuously to show the progress for each bomb.

## Hints

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.

  The CS:APP web site

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

  has 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`.

- `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`, `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.

## More Advice

Based on our experience with defusing bombs, we have the following advice:

- Consider printing out all of the code associated with a particular phase when you start to diffuse it. As you work, you can annotate this code with information as you figure out what is going on.

- As you will notice from examining bomb.c, the user input is sent into the phase as a single string. One of the first things the phase will do is try to 'parse' it into the form it is expecting.

- When you are trying to figure out what values are needed for a phase (particularly if the values are integers), try 'interesting' numbers like 42, -17, ... first. The reason for this is that if you see one of these numbers appear in a register or in a memory location, changes are pretty good that was your input.

- Don't bother trying to step through system functions. This is generally a waste of time. If you don't know what the function does, use 'man' to investigate the parameters and return information. Then examine the parameters that are sent and the return value.

- If you end up inside a function that you don't want to step through, the GDB command 'finish' will take you to the end.

- The bomb functions typically have a generic name like fn7 or phase2. It has been our experience that if a function has a descriptive name, you can assume that the function behaves as expected and that they are NOT trying to trick you.

- Understanding parameter passing is critical to this assignment. You should remember what registers are used to pass parameters to functions, and in which order.

- It may not be important to understand exactly what every statement is doing - the goal is to avoid the explode_bomb calls. You can, for example, step through the code till just before one of these calls and then focus on what conditions are not being met. You can also set a breakpoint at explode_bomb to ensure you dont accidently execute it.

The server will be disabled at the specified deadline, i.e., Thursday March 23, 2017 11:59:59 PM EST. In other words, late submissions will not be allowed. Please plan accordingly.

The GTA Nasrin Akhter (nakhter3@masonlive.gmu.edu) as well as the UTAs Nabil Darwich (ndarwich@masonlive.gmu.edu) and Thi Nguyen (tnguy138@masonlive.gmu.edu) can help you with the use of tools such as gdb and objdump. You are encouraged to use the Piazza forum for your questions.