Bomb Lab – Assignment 2

COMP202 2016 Revised 29 August 2016

Len Hamey, len.hamey@mq.edu.au
Adapted from Bryant and O'Hallaron's Bomb Lab

This lab is the second of three assignments in COMP202.

Commences: Week 5.
Progress: Weeks 6, 7, 8
Due: 11:59pm, Tuesday Oct 11 (Week 9)

Value: 15% (12% for task, 3% weekly progress)

Introduction

The nefarious *Dr. Evil* has planted a slew of ``binary bombs'' on our servers. 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!!!" (or something similar) 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!

BREAKING NEWS!! Dr. Evil has evidently become aware of Internet sites that provide help for defusing his earlier binary bombs. These new binary bombs are very different.

Specific Learning Outcomes

The specific learning outcomes of this lab exercise include the following. You will encounter these learning outcomes at various phases of the lab.

- Using debugging tools is a major learning outcome. You cannot achieve anything worthwhile in this lab without using debugging tools.
- Interpret data values in memory and registers.
- Experience with ASCII character conversion.
- Interpret assembly code corresponding to numeric and logical operators.
- Interpret addressing modes including immediate, register, and register indirect modes.
- Recognise and interpret control constructs expressed in assembly including if-else, loop and switch.
- Recognise and interpret procedure calls, parameters, and returns expressed in assembly.
- Interpret information stored on the stack.
- Interpret simple data structures in memory.
- Experience with buffer overflow exploit.

Step 1: Get Your Bomb

You can obtain your bomb by using the lab command to get lab 2.1.

```
$ lab -g 2.1
```

A binary bomb has already been placed in the server for you, and the lab command will download it in a tar file. Save the tar file to a protected directory in which you plan to do your work. When you extract the tar file, it will create a directory called `bomb`*k* where *k* is your unique bomb ID. The directory contains 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 and a friendly greeting from Dr. Evil. We have verified that this source code is genuine. This file contains information that may be useful, although how much can you trust someone with the name `Dr. Evil'? In particular, we do not adhere to his licensing terms so you should feel free to use all the tools he mentions, and any other tools you like, to disassemble, dump and debug the bomb.

If for some reason you request the bomb again, this is not a problem. You will receive the same bomb.

Step 2: Defuse Your Bomb

Your job for this lab is to defuse your bomb.

You must do the assignment on one of the class machines iceberg or ash. In fact, there is a rumour 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, and there are/will be additional documents on iLearn providing further assistance. The best way to defuse the bomb is to use your favourite debugger to step through the disassembled binary. We suggest `gdb`.

Each time your bomb explodes it notifies the bomblab server, and you lose 0.1 points (up to 2.0 marks for each bomb) in the final score for the lab. There is only one way to recover lost points – to start again with a new bomb which we will offer to you later in the lab. So there are consequences to exploding the bomb. You must be careful!

There are six phases, each work 2.0 marks. There are three progress marks. So the maximum score you can get is 15 points. The minimum score is -2.0 if you explode the bombs 20 times each and do not solve any phases. Negative marks will count as zero in your final mark.

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. You can also use this feature when running your bomb in the debugger gdb.

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 for the rest of your career.

Logistics

This is an individual project. All handins are electronic and automatic – you do not need to use the lab command to submit anything because your bomb notifies the server of your successes and failures.

Clarifications and corrections will be posted in iLearn.

Achievement Marks

Achievement marks in this lab are earned for defusing stages of the bomb. Each of the six phases is worth 2.0 marks.

Phases 1, 2, 3 and 5 have been heavily influenced by Scott Evil. As you know, Scott is Dr Evil's son, and he has a good heart. Out of kindness to the world, Scott introduced a simpler solution to each of these bomb phases than Dr. Evil's solution. Scott Evil's solution successfully defuses the bomb, but is only worth 1.5 achievement marks. To obtain the full 2.0 marks for each phase you need to dig deeper into the assembly code. When your phase is defused using Dr. Evil's solution, the bomb will print a congratulatory message, as you can see by examining `bomb.c`.

Strategy

It is not possible to attempt to defuse a bomb phase until the preceding phases have been defused, so you have to work on the phases in sequence. However, once you have found Scott Evil's defuse string for a phase, you can then choose to work on the next phase or to continue working on the current phase in order to achieve Dr. Evil's solution. You can move on to the next phase and then return later, when you are more experienced, to try to find Dr. Evil's solution to the earlier phase.

Once you have a solution to a phase you should enter it into a text file that you use when you run the bomb, as explained above. You can name the text file as a parameter when you run the bomb inside the debugger.

Submission

There is nothing for you to submit explicitly, however you should keep any written notes and a copy of your solution file just in case there is a need to verify your results. 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 using the lab command to examine your marks for lab 2:

```
$ lab -m 2
```

The report is reasonably detailed but may take some time because it is generated automatically.

The bomb lab will close automatically on the due date and your bomb will not run. You can use the lab command for free extensions.

It will not be possible to run your bomb after the closing date of the lab. If individual students wish to explore further without the opportunity to earn additional marks, please contact Len.Hamey@mq.edu.au for special arrangements.

Progress Marks

Automatic progress marks are awarded for defusing each of the first three bomb phases. Scott Evil's solution is sufficient to achieve the progress mark. The progress mark schedule is lenient, but you should aim to be ahead of schedule, because the last three phases are considerably more difficult, and also more interesting!

The automatic progress mark schedule is

- Saturday of Week 6 (Sep 10): Phase 1 defused.
- Saturday of first week of break after week 7 (Sep 24): Phase 2 defused.
- Tuesday of Week 8 (Oct 4): Phase 3 defused.

The final deadline for this lab is Tuesday of week 9: 11th Oct 2016.

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.

Using a debugger you can step through the bomb. If you find a point at which it wants to explode the bomb, you can look at registers and memory and determine what is needed to avoid exploding the bomb at that point. Then, tracing through the program code, you work out what input string you need to provide in order to not explode the bomb at that point.

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 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. For more hints and help, see the Lab Notes section on iLearn.

gdb

The GNU debugger 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. Even better, learn about `.gdbinit` to protect yourself against typing mistakes.
- iLearn has/will have lab notes that provide more information about gdb and helpful hints for using it.
- For online documentation, type "`help`" at the `gdb` command prompt, or type "`man gdb`", or "`info gdb`" at a Unix prompt.

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 something like:

\begin{verbatim}

```
8048c36: e8 99 fc ff ff call 80488d4 <_init+0x1a0>
```

\end{verbatim}

To determine that the call was to `sscanf`, you would need to disassemble within `gdb`. Gdb recognises system calls, although the names appear somewhat strange in gdb.

strings

This utility will display the printable strings in your bomb.

Documentation

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 ask your instructor for help.