Bomb Lab – Assignment 2

COMP2100

Richard Han, richard.han@mq.edu.au
Adapted by Len Hamey from Bryant and O'Hallaron's Bomb Lab

This lab is the second assignment in COMP2100.

Commences: See iLearn for notification when it is available.
Progress: approximately weekly. For specific dates use: **`$ lab -d`**
Due: Week 13. For specific due date use: **`$ lab -d`**

Value: 25%

- 4% weekly progress
- 11% for the task
- 10% for the end-of-assignment mastery quiz

Welcome to the binary bomb lab!

This lab is a game in which you try to solve a series of puzzles, working out what input your "bomb" will accept for each bomb phase. As with any game, you will benefit the most (and learn the most) if you play by the rules.

See the section "Rules of the Game (Academic Integrity)" below.

Videos

There are introductory videos available for this assignment. They provide an overview of the assignment and helpful discussion and demonstrations.

How to get started:

1. Watch the introduction video.
2. Read this assignment specification.
3. Watch the demonstration videos using is_yes.
4. Practice the gdb skills that you need using the is_yes example that you can download from ilearn (yes.tar).
5. Download your bomb.
6. Carefully start exploring your bomb in gdb.

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!

Your assignment is to defuse the phases of a single bomb. In order to maximise your progress marks, you should defuse a different phase of the bomb each week. In order to maximise your achievement marks, you will want to be ahead of the progress mark deadlines because there are only four progress marks, but the bomb has six phases.

> BREAKING NEWS!! Dr. Evil has evidently become aware of Internet sites that provide help for defusing his earlier binary bombs. The most recent 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 and loops.
- 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.
- Convert assembly code to pseudo-C and interpret the code.

Rules of the Game (Academic Integrity)

The principles of academic integrity apply to this assignment. The work you do in this assignment must be your own individual work. Any help that you receive from others should be of a general nature and not specific to the bomb lab assignment, to your bomb or any other person's bomb.

We want the binary bomb lab to be an effective learning environment for you and for your fellow students. Therefore, please keep within the following rules.

Rules.

- Don't share your bomb, disassembled code from your bomb, or your bomb solutions with any other person except teaching staff in the Computing School.
- Don't share the assignment specification outside Macquarie University.
- Don't store or post the assignment specification or your assignment work on any website or forum.

- Don't put anything on github or similar sites.
- Don't seek specific help from any person except teaching staff in the Computing Department. Any person is permitted to give you general help, but if you are showing them your bomb, even just one assembly line, then you should be seeking help from Computing staff.
- Don't ask questions specific to your bomb on the ilearn forums. Find a more general way to ask what you need help with, or ask Computing staff.
- Don't post code from your bomb into any forum, not even the ilearn forums.
- Don't use GHIDRA or similar tools. Use gdb instead and learn how to interpret assembly as pseudo-C, because you will be tested on your ability to do that.
- Don't try to find information on the internet about how to solve the bomb lab phases – the information you find will not be relevant to the bomb lab that we are running at Macquarie University.
- Do not share key ideas or specifics of what you did to solve the phases. Instead, let your fellow students having the same learning experience that you did, solving their bomb phases for themselves.
- Don't try to solve the bomb by brute force – you will not learn what we intend you to learn and you may be locked out of the server.

Seeking Help?

The right kind of help is valuable for your learning. The right kind of help supports you to meet the challenge through helping you to understand the problem and how to go about solving it. With the wrong kind of help, you may get the answer but you still don't know why it works.

You are encouraged to discuss general issues with your fellow students and on the iLearn forums. This includes topics such as:

- Understanding the assignment specification or support notes documents on iLearn.
- Understanding the examples discussed in lectures and video recordings.
- How to use the lab command.
- Gdb commands and how to use `gdb` to set breakpoints, step through code, disassemble, etc.
- Useful Unix commands (see the Hints section below).
- Useful information on the unit's ilearn site.
- Useful general information on the internet, such as assembly language and gdb information.
- Information about machine instructions, such as discussing "what does cmpb do?"

If you need help that is specific to your bomb, please approach your workshop tutor in person, on zoom, or via email; or contact the super tutors; or approach the unit convenor by email richard.han@mq.edu.au. If you are considering seeking help in some other way but are not sure whether it is allowed, please ask the unit convenor by email richard.han@mq.edu.au.

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.

Note: You can earn all the progress marks and all the achievement marks by defusing the six phases of the one bomb. Unlike the first assignment, this assignment does not require you to download something different for each stage of the assignment. Instead, you will work through the phases of your 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 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 Unix system tools to help you defuse your bomb. Please look at the **hints** section below for some tips and ideas, and there are additional documents on iLearn providing further assistance. The best way to defuse the bomb is to use the Unix debugger `gdb` to disassemble the code and interpret it, then step through the disassembled binary to check your understanding. We have provided a separate document giving you some help to defuse your bomb with `gdb`, and a video demonstrating some of the useful `gdb` features on an example module called `is_yes`.

Each time your bomb explodes it notifies the bomblab server. The first explosion does not cost you marks, but after that you lose 0.1 points (up to 2.0 marks for each bomb) in the final score for defusing that bomb. There is only one way to recover lost points – to start again with a **second bomb** which we will offer to you later in the lab. So there are consequences to exploding the bomb. You must be careful!

The command to fetch your **second bomb** is

```
$ lab -g 2.2
```

However, the second bomb is not available when the lab first opens – we have to wait for Dr Evil to put more bombs on our server. Also, you have to defuse at least two stages of the first bomb before you can fetch a second bomb.

There are six phases. The first four phases are worth 2.0 marks each and the last two phases are worth 1.5 marks each. There are also four progress marks, and 10 marks for a final quiz testing your mastery of the concepts learned in the assignment. The maximum score you can get is 25 marks. The minimum score is -2.0 if you explode your bombs 20 times each and do not solve any phases. Negative marks in this assignment will count as zero in your final grade for the unit.

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 asks for your input with a prompt. It ignores blank input lines.

It is not wise to run your bomb from the command line, although you can do that. It is recommended to run it inside `gdb` using gdb's `run` command. When you run the bomb, it will display a prompt and read the input line that you type. It will then decide whether it accepts the input line and defuses the phase, or explodes.

After you enter your input line, the bomb typically takes **1-2 seconds** to decide whether it accepts your input. This gives you time to press control-C if you make a mistake in your input. However, it is strongly recommended to use breakpoints to prevent the bomb from exploding.

You can type the name of a text file as the command-line argument to run the bomb, for example:

```
$ gdb bomb
(gdb) run psol.txt
```

The bomb will then 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 – you can store your earlier phase solutions in the text file.

When the bomb is reading from a file, it displays the prompt and each line that it reads from the file.

To avoid accidentally detonating the bomb, you will need to learn how to single-step through the assembly code and how to set breakpoints in `gdb`. 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 submissions 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 phases of the bomb. The first four phases are worth 2.0 marks each and the last two phases are worth 1.5 marks each.

Phases 1, 2, 3, 4 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 in phases 1 through 4, and 1.0 in phase 5. To obtain the full 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 an additional 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 defusion string for a particular phase, then you can choose whether to work on the next phase or to continue working on the current phase in order to achieve Dr. Evil's solution. You might find it helpful to 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. This will move through the phases that you have already solved, leaving you ready to work on your current phase.

Submission

There is nothing for you to submit explicitly. Every time you defuse the bomb, and every time the bomb explodes, the event is recorded. These defusions and explosions are used to automatically compute your mark. 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.

You can keep track of your achievements 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 to appear because it is generated automatically.

The bomb lab will close automatically on the due date. You can use the lab command for free extensions.

Progress Marks

Automatic progress marks are awarded for defusing each of the first four 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 phases are considerably more difficult, and also more interesting!

Remember that you only need to work through the phases of a single bomb – you don't need to fetch a new bomb for each stage of this assignment.

The automatic progress mark schedule is shown by the command **`$ lab -d`**

The final deadline for this lab is in week 13, and is shown by the command **`$ lab -d`**

If you wish to claim a **free extension** for a **progress** mark, you must claim the extension before you defuse the relevant bomb phase. Once the phase is defused, the server will register the progress mark that is due in the future, and it will not be possible to extend the progress mark that is overdue.

If you have grounds for special consideration and request an extension of a progress mark due date, please also email the unit convenor. Extensions of progress marks can be tricky but we will do our best to give you an appropriate extension if the disruption justifies one.

Progress marks are not linked to the particular bomb that you are working on, so if you request your **second bomb** you don't lose the progress marks that you previously earned. However, to gain further progress marks you must solve later phases of either the first or second bomb. Before you can work on the later phases of the second bomb, you have to solve its early phases, but you should find it a lot easier to solve those phases in the second bomb because of all you have learned about assembly and gdb when you were working on your first bomb.

Mastery Quiz

In week 13, we will offer a mastery quiz work 10 marks towards this assignment. The quiz will test what you have learned about assembly and `gdb`. It will specifically test knowledge that is required to complete the phases of the binary bomb. More details will be provided on iLearn closer to the time of the quiz.

After the Lab Closes

After the lab closes, it will still be possible to run your bomb. Every defusion and explosion that you cause after the lab closes will still be recorded, but it will not affect your mark unless you are given an extension. If you claim a free extension, or receive an extension due to special consideration, then **all** your bomb activity up to the new closing date will be counted towards your mark. For this reason we recommend that you be as careful with your bomb after the closing date as you were beforehand.

If you request an **extension** of the closing date, please also email the unit convenor. You can continue to work on the lab as you are able, even before an extension is granted. As explained above, every defusion and explosion will be recorded by the automarker, although your marking report will not show your new work until an extension is granted, so you should keep your own notes.

If you wish to use a **free extension** for the closing date, you can minimise your risk by working on the bomb lab after closing without claiming the extension. This only applies to an extension of the **closing date**. You will need to keep your own notes of your defusions and explosions. If you achieve a worthwhile improvement, then you can claim the free extension and then be rewarded with the marks for the new work. However, **be warned**: Explosions take away marks, so if you lose more through explosions than you gain through improved defusion results, your total mark could possibly go down when you claim a free extension. Also, remember that it is not possible to claim or change free extensions that would end at a time in the past – the end of your free extension must always be in the future when you are claiming or changing it.

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 – rewriting the assembly as pseudo-C will be helpful. This is a useful technique, but it not always easy to do. You can also run it under `gdb`, watch what it does step by step, and use this information to defuse it. This is probably the fastest way of defusing it.

> Using `gdb` 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 will avoid learning about assembly and will miss out on learning debugging skills.
- You lose points every time you guess incorrectly and the bomb explodes.
- Every time you guess wrong, a message is sent to the bomb lab 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. Below 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.

The Phases

The phases are in order of increasing difficulty. The file bomb.c contains some comments, which even though they are written by Dr Evil, still provide some useful hints. Here are some additional notes to help you get started with the early phases.

Phase 1 involves identifying particular characters that are required to be placed in particular locations in the input string. Dr Evil has added a twist for his solution. The `is_ok` example given in lectures and the `is_yes` example in the bomblab videos are designed to help you with this stage in particular.

Phase 2 involves understanding the parameters of certain procedure calls. The procedure names are similar to standard library routines and the procedures are equivalent to those routines. Again, Dr Evil has added a twist to this puzzle.

Phase 3 is a puzzle about ALU (Arithmetic Logic Unit) instructions. For this puzzle, we suggest you convert the assembly to pseudo-C, then simplify until you have a tidy C expression to solve.

Phase 4 is a demonstration of buffer overflow. Think carefully about how to achieve the desired effect.

Tools

Gdb

The document "Bomb Defusing with gdb" provides a lot more information about how to use gdb to defuse your bomb. Please read it! Here is a very brief introduction.

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:APP3e web site http://csapp.cs.cmu.edu/3e/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 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 Unix command 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 Unix command 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:

```
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`. 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? iLearn contains some useful information in the Support Notes section. Don't forget, the Unix commands `apropos`, `man`, and `info` are your friends. In particular, `man ascii` might come in useful and `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 general questions on the forum, or ask your instructor for specific help.

Index