

CS140 Lab

Defusing The Bomb

Defusing A Binary Bomb

0 Acknowledgements

The idea for this lab comes from Dave O'Hallaron (droh@cs.cmu.edu) at Carnegie Mellon University. It's a great way for you to learn debugging by living on the edge. I must admit, when I first got a copy of the program, I stopped everything else and solved it. The version you have is a modification of that code.

1 Introduction

The nefarious *Dr. Evil* has planted a slew of "binary bombs" on our 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 person a bomb to defuse. Your mission, that 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

Each student will attempt to defuse her/his own personalized bomb. Each bomb is a Linux binary executable file that has been compiled from a C program. Your bomb should arrive by e-mail and should be either a tar file or a series of individual files.

If a tar file, give the command: `tar xvf bomb.tar`. This will create a directory called `./bomb` with the following files:

- _ bomb: The executable binary bomb.
- _ bomb.c: Source file for the bomb's main routine.
- _ sparkler: A practice executable you can all talk about and work on together
- IntelAssemblyLanguage.pdf Contains a brief overview of Intel code
- BombLab.pdf: This description file

Defusing A Binary Bomb

Step 2: Defuse Your Bomb

Your job is to defuse the bomb. You can use many tools to help you with this; please look at the hints section later in this document 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 staff, and you lose 1/4 point (up to a max of 10 points) in the final score for the lab. So there are consequences to exploding the bomb. You must be careful! Each phase is worth 20 points, for a total of 100 points. The phases get progressively harder to defuse, but 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 answers_so_far.txt
```

then it will read the input lines from `answers_so_far.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

You should do the assignment on the lab 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 they say.

Defusing A Binary Bomb

Hand-In

There is no explicit hand-in. The bomb will notify me automatically after you have successfully defused it. Make sure you understand the due date – start early!

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.

I 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 1/4 point (up to a max of 10 points) every time you guess incorrectly and the bomb explodes.

_ Every time you guess wrong, a message is sent to the staff. 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 (wrong) assumptions that they all are less than 80 characters long and only contain letters, then you will have ___ many many 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.

Defusing A Binary Bomb

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. Here are some 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.

The CS:APP Student Site at <http://csapp.cs.cmu.edu/public/students.html> has a very handy single-page gdb summary

For other documentation, type "help" at the gdb command prompt, or type "man gdb", or "info gdb" at a Unix prompt.

Defusing A Binary Bomb

What works AMAZINGLY well is defining a file with various commands that you want to use every time you run gdb. Then start gdb with a switch that specifies you should use that file. My file contained these (and other) commands:

```
b key_1
b *0x08048dd4
display/i $pc
display $eax
display $ebx
display $ecx
display $edx
r in
```

`_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!

Defusing A Binary Bomb

`_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 `scanf` might appear as:

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

To determine that the call was to `scanf`, 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` and `man` are your friends. In particular, `man ascii` might come in useful. Also, the web may also be a treasure trove of information. If you get stumped, feel free to ask.