Contents

Lab 2: Disassembling and Defusing a Binary Bomb

Overview

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 everyone 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!

Instructions

The bombs were constructed specifically for 64-bit machines. You should do this assignment on the 64-bit Linux VM and be sure it works there (or at least test your solution there before submitting it!) so that you can be sure it works when we grade it. In fact, there is a rumor that Dr. Evil has ensured 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.

Perform an update to your course-materials directory on the VM by running the `update-course` command from a terminal window. On the script's success, you should find the provided code for lab2 in your course-materials directory. As a convenience, here is an archive of the course-materials directory as of this lab assignment: lab2.tar.gz. The lab2 directory should then contain the following files:

- `bomb`: The executable binary bomb
- `bomb.c`: Source file with the bomb's main routine
- `defuser.txt`: File in which you write your defusing solution

Your job is to defuse the bomb. You can use many tools to help you with this; please look at the tools section for some tips and ideas. The best way is to use a debugger to step through the disassembled binary.

The bomb has 5 regular phases. The 6th phase is extra credit (worth half as much as a regular phase), and rumor has it that a secret 7th phase exists. If it does and you can find and defuse it, you will receive

additional extra credit points. The phases get progressively harder to defuse, but the expertise you gain as you move from phase to phase should offset this difficulty. Nonetheless, the latter phases are not easy, so please don't wait until the last minute to start. (If you're stumped, check the hints section at the end of this document.)

The bomb ignores blank input lines. If you run your bomb with a command line argument, for example,

```
./bomb defuser.txt
```

then it will read the input lines from defuser.txt until it reaches EOF (end of file), and **then switch over to stdin** (standard input from the terminal). 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.

Resources

There are a number of online resources that will help you understand any assembly instructions you may encounter while examining the bomb. In particular, the programming manuals for x86-64 processors distributed by Intel and AMD are exceptionally valuable. They both describe the same ISA, but sometimes one may be easier to understand than the other.

Useful for this Lab

- Intel Instruction Reference
- AMD Instruction Reference

Not Directly Useful, but Good Brainfood Nonetheless

- Intel 64 and IA-32 Architectures Software Developer's Manual Volume 1: Basic Architecture
- Intel 64 and IA-32 Architectures Software Developer's Manual Combined Volumes 3A and 3B: System Programming Guide, Parts 1 and 2
- AMD64 Architecture Programmer's Manual Volume 1: Application Programming
- AMD64 Architecture Programmer's Manual Volume 2: System Programming
- AMD64 Architecture Programmer's Manual Volume 4: 128-bit and 256 bit media instructions
- AMD64 Architecture Programmer's Manual Volume 5: 64-Bit Media and x87 Floating-Point Instructions

x86-64 Calling Conventions

As a reminder, the x86-64 ISA passes the first six arguments to a function in registers. Registers are used in the following order: `rdi`, `rsi`, `rdx`, `rcx`, `r8`, `r9`. The return value for functions is passed in `rax`.

Tools (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.

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 the number of possibilities is so large that you won't be able to try them all in time.

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 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 has a very handy gdb summary (there is also a more extensive tutorial).
  - For other 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 bomb`: 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 bomb`: 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 may look cryptic. 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 -t x bomb`: This utility will display the printable strings in your bomb and their offset within the bomb.

Looking for a particular tool? How about documentation? Don't forget, the commands `apropos` and `man` are your friends. In particular, `man ascii` is more useful than you'd think. If you get stumped, use the course's discussion board.

Hints

If you're still having trouble figuring out what your bomb is doing, here are some hints for what to think about at each stage: (1) comparison, (2) loops, (3) switch statements, (4) recursion, (5) pointers and arrays, (6) sorting linked lists.

Submitting Your Work

You will be able to submit your assignment with the `submit-hw` script that is bundled with the course-materials. Using the script should be straight-forward, but it does expect you to not move/rename any files from the "course-materials" directory. Open a terminal and type the following command:

```
submit-hw lab2
```

After calling the `submit-hw` script, you will be prompted for your Coursera username and then your submission password. **Your submission password is NOT the same as your regular Coursera account password!!!!** You may locate your submission password at the top of the assignments list page.

After providing your credentials, the `submit-hw` script will run some basic checks on your code. For lab 2, this means that it checks that you have passed the first 5 phases correctly.

Once confirming that you wish to submit, with a working Internet connection, the script should submit your code properly. You can go to the assignments list page to double-check that it has been submitted and check your score as well as any feedback the auto-grader gives you. You may also download your submission code from this page, if you wish.