CS5231 – Systems Security

Homework 1: Warm Up

Due date and time: 23:59pm, September 9, 2023. This is a firm deadline. This homework MUST be finished independently.

1. Introduction

In this homework, you will explore how to debug binary programs using the GNU Debugger (GDB) in Linux systems and how to develop loadable kernel modules in Linux kernel.

To finish this homework, you need to have access to a Linux system. Since we have planned to use Ubuntu 20.04 (x86_64) as our testing and grading platform, you are required to use the same Ubuntu version when doing your homework. You are encouraged to set up your programming environment (make sure the Linux distribution is Ubuntu 20.04, x86_64), using VirtualBox on Intel machines and UTM on Apple Silicon machines. We also have created virtual machine (VM) images for different host architectures.

If you use an Intel machine

We have created a VM image with Ubuntu 20.04 (x86_64) installed. The VM image is in OVA format, which is supported by popular virtual machine monitors like VirtualBox. You can download this file via the following Google Driver link:
https://drive.google.com/file/d/1c5ojnqxFCgRdUC59CpR9iWZbHtO3LjQF/view
You can refer to this guide for importing OVA files into VirtualBox:
https://www.maketecheasier.com/import-export-ova-files-in-virtualbox/.
The login password for the `root` user is `student`.

If you use an Apple Silicon MacBook

You are recommended to use UTM (https://mac.getutm.app/) to run the VM. You can download the UTM file via the following Google driver link:
https://drive.google.com/file/d/1-sd87jfkaVLsEhilvGnnqtzPJviu8j9h/view
The login password for the user `user` (with sudo privilege) is `user`.

Note that the provided VM image is installed with Ubuntu Server 20.04 (x86_64), without any GUI. We do so because of the considerable performance degradation when simulating the x86_64 architecture on ARM hosts. In addition, this homework does not need to access any GUI; all the tasks can be done in a terminal. If you'd like

to start a GUI, we recommend to try LXDE desktop. You can also set up SSH server in the VM to access it over the network.

2. Tasks

2.1. Task 1 (5 marks) Debugging Programs using GDB

In this task, you are asked to debug the following program using GDB. Write down the GDB commands to perform the following debugging tasks.

1) Put a breakpoint at the entry to the `main` function and run the program to this breakpoint
2) Print the address of variable `a`
3) Print the address of function `f`
4) Add a watchpoint to the variable `c`
5) Print the assembly instructions of function `g`
6) Put a breakpoint at the entry to function `g`, run to this breakpoint, and print the backtrace of all stack frames
7) Print the return address at the breakpoint set in the previous step
8) Print the process ID of the current process

```
// Filename: gdb_exploration.c
#include <stdio.h>
int f(int x, int y);
int g(int x, int y);
int main( ) {
        int a = 1, b = 2, c = 0;
        c = f(a, b);
        printf("Result = %d\n", c);
        return 0;
}
int f(int x, int y) {
      return (x+y) * g(x,y);
}
int g(int x, int y) {
      return x*y;
}
```

To familiarize yourself with GDB, you can refer to this tutorial:
https://www.tutorialspoint.com/gnu_debugger/index.htm

The official documentation for GDB can be found at this link:
https://sourceware.org/gdb/current/onlinedocs/gdb.pdf

First, you compile the above source code using the following command:

```
$ gcc -g gdb_exploration.c -o gdb_exploration
```

Then, you start the debugging session using the following command:
`$ gdb ./gdb_exploration`

What to submit:
Please submit a file named ***your-matric-id*_gdb_exploration.txt**.
You are required to place all the commands for each debugging task on a single line, separated by semicolons:
`gdb_command_1; gdb_command_2; gdb_command_3`
In addition, please use the full name for command names, e.g., `break` instead of `b`.

2.2 Task 2 (5 marks) Loadable Kernel Module Development in Linux Kernel

In this task, you are asked to develop a loadable kernel module that performs the following tasks when being loaded:

- print how many processes are running in the system
- print current time in the form of seconds

In Linux, the current time is maintained by keeping the number of seconds elapsed since midnight of January 01, 1970 (called `epoch`).

To familiarize you with the development of loadable kernel modules in Linux, we give a guided tour here. Through a `hello-world` kernel module, we demonstrate 1) how to set up the development environment, 2) how to compile the source code, 3) how to insert the module into a running kernel, and 4) how to remove a module from the running kernel. The source code (`hello_world_lkm.zip`) for this kernel module comprises two files: `Makefile` and `hello_world_lkm.c`.

Let's begin our tour.
Step 1) Installing the prerequisite packages. To build and use a loadable kernel module in a Linux distribution, we need to install the following two packages:

- Toolchain: comprising compiler, assembler, linker/loader, C library, etc.

```
$ sudo apt install gcc
```

- Linux kernel headers: these headers will be used during compilation

```
$ sudo apt install linux-headers-`uname -r`
```

Step 2) Compiling the source code of the hello-world kernel module into a kernel module object (`.ko`). Here, `hello_world_lkm.ko` is the resulting `.ko` file.

Under the directory `hello_world_lkm`, issue the following command.
`$ make`

Step 3) Inserting the kernel module file (the `.ko` file) into the live kernel at runtime, effectively making it a part of the kernel.
`$ sudo insmod ./hello_world_lkm.ko`

To check that our module is loaded properly, we can use the following command.

```
$ lsmod | head
```

Step 4) Unloading the module from kernel using the following command.
```
$ sudo rmmod hello_world_lkm
```

To check that our module is unloaded successfully, we can use the following command.
```
$ dmesg | tails
```

For those who want to get more in-depth knowledge on loadable kernel module, you are recommended to refer to this guide: https://sysprog21.github.io/lkmpg/

Instructions: If you successfully went through the above tour, congratulations! First, change the directory name "`hello_world_lkm`" to "*`your_mactric_id`*`_lkm`". Then, you can fill in your code in the function `hello_world_lkm_init(void)` in file `hello_world_lkm.c`. Finally, compress this directory into an archive file named *`your_matric_id`*`_lkm.zip` .

What to submit: Please submit two files: ***your-matric-id*_lkm.zip** and a report (PDF, **up to 3 pages**) named ***your_matric_id*_lkm.pdf** . In the report, you should include the code of the function `hello_world_lkm_init(void)` and screenshots for successful loading and unloading of your kernel module, as demonstrated in Step 3) and Step 4) above. Also, please **INCLUDE** your name and matric ID in the report.

3. Submission

Your submission into Canvas is a single .zip file. Please name this file with the following format: ***your_matric_id*-hw1.zip**, e.g. A19930314-hw1.zip. The .zip file should include three files: ***your-matric-id*_gdb_exploration.txt**, ***your-matric-id*_lkm.zip**, and ***your_matric_id*_lkm.pdf** .

Note that we only accept the following formats for compressed file: .zip, .tar.gz, .tar.bz, and .tar.bz2 . Also, please make sure your report does not **EXCEED 3 pages**, excluding references and appendices.

For any questions, please contact the teaching assistants via cs5231ta@googlegroups.com. Hope you can have fun in exploring the basics in systems security. This homework also helps you to choose your final project topic.