CSCI 5573: Advanced Operating Systems
Fall 2018

Programming Assignment Two

Due Date and Time: 11:55 PM, Monday, October 29, 2018

The goal of this programming assignment is to get familiar with Linux Kernel Modules as well as some kernel debugging tools.

You must work in a virtual machine for this assignment, otherwise you may corrupt your operating system. You will need super user privileges to do this assignment.

You may work in teams of size two students.

1. Inserting a new system call using LKM

   Write a kernel module to add a new system call *getprocinfo(pid_t pid, struct procinfo *inf)* that returns the following data about the specified process and prints it on the standard output:

```
struct procinfo {
        pid_t pid;                  /* Process ID */
        pit_t ppid;                 /* Parent process ID */
        struct timespec start_time;   /* Process start time */
        int num_sib;                /* Number of siblings */
};
```

   You can find all of the information you need in *task_struct*, which is defined in *include/linux/sched.h*. If *pid* is greater than 0, it represents a processID whose information is sought. If it's zero, return the information for the current process. If it's less than zero, return the information for the parent of the current process. As always, check inputs for validity. Return error EINVAL if necessary.

   sys_call_table is a fixed size array and its size is determined at compile time by the number of registered syscalls. This means there is no space for another one. You can explore two approaches to add a new system call:

   Approach 1: Although *sys_call_table* is of fixed size, in some cases there may be free positions in the table. The kernel fills all positions of *sys_call_table* with a pointer to *sys_ni_syscall* ("not implemented" system call, which does nothing except return -ENOSYS, the error corresponding to an invalid system call). Your task is then to identify these positions and "register" the new system call there.

   Approach 2: Build a module that creates a new block device and implements your "system call" as an *ioctl* on that device.

   In either case, measure the time it takes to execute your system call.

2. Experience with rootkits and countermeasures

   The goal of this exercise is to gain experience with rootkits and how to counter them. A rootkit is a collection of computer software, typically malicious, designed to enable access to a computer or areas of its software that is not otherwise allowed (for example, to an unauthorized user) and often masks its existence or the existence of other software. A common way rootkits work is by manipulating systems calls.

CSCI 5573: Advanced Operating Systems
Fall 2018

(a) Your first task is to write a very simple rootkit module that modifies *ls* so that files/folders whose name begins with prefix *abc* are not listed when *ls* is run. These files/folders should still be available and you should be able to access them, e.g. by using an editor.

To write this rootkit, you will first identify the system call that *ls* uses to find the file listings. Use a tool called *strace* to do this. Next, you will write an LKM to intercept that system call. Use the methodology discussed in class to do this. In the modified system call function, invoke the original system call function and then filter out any filenames that start with prefix *abc*.

Show your work about how you identified the target system call. Further, identify at least one other command that is affected by this rootkit? How is that command affected?

(b) While this rootkit hides all files whose names start with *abc*, the name of the rootkit (the LKM module you built) is still visible, e.g. by running *lsmod* command. Your second task is to extend your rookit so that it is no longer visible from *lsmod*.

(c) Since system call manipulation is a common techniques utilized by rootkits, one useful tool would be to identify every system call that is being intercepted. So, your third task is to build a kernel module that can identify all system calls that are being intercepted as well as all new system calls introduced in the kernel after your new kernel module has been installed in the kernel. When invoked from the user space (as a new system call), this tool should print out a list of all such system calls.

A key concern with this kernel module is that a rootkit could intercept it and modify it. Explain how this could be done.

(d) Your final task is to address the concern of part (c) by building a kernel module and a utility function *check_sys_calls*. The utility function simply loads the kernel module in the kernel and then removes the module from the kernel. When the kernel module is loaded for the first time, it records the current state of all system calls. Afterwards, whenever this kernel module is loaded, it reports all system calls that are being intercepted as well as all new system calls introduced in the kernel after the kernel module was first loaded in the kernel.