Report – Star Power Team

Roi Atalla
Justin Buchanan
Mohit Tandon

Overall Design

Jprobes were used instead of kprobes because the handler function could be written to take in the same arguments as the actual command. For example the chmod command can be linked to a C function that has two inputs: “filename” and “mode”. This allows us to get access to the arguments very effectively and comfortably.

In order to store the log data, we used a circular buffer of char pointers that point to a char array (also can be referred to as a string). This allows us to kmalloc a fixed sized buffer and then kmalloc strings of the required length at each location in the buffer. The benefit of this is that it uses less memory since some log entries will require lesser characters than others. The benefit of using the circular buffer is that it allows you to keep the latest log entries and overwrite the oldest entry in the situation where the buffer size is exceeded.

For the proc entries, 3 separate “files” are created. The sysmon_uid and sysmon_toggle proc files are set to be accessible only by root. The sysmon_log proc file only supports read by a root user. The sysmon_uid and sysmon_toggle proc entries have write handlers that would validate the input and then store the data if valid. If the data is not valid, it prints the error message to dmesg and then returns -EINVAL. The proc entries also have read handlers that return the current value of that file. For reading the log, since the return user buffer has a fixed length, the read function is called multiple times until a 0 is returned which states that all the required data has been read.

Experimental Configuration

In order to test our kernel module, we made a test script that loads and starts the module, then runs it to monitor a specially-created test user account. We then run commands as that user and examine the resulting log. To test the overhead of the system, we ran a syscall-intensive script with and without the system call interposer enabled and measured the timing differences. With the logger on, it took 0.457s system time and with it off, it took 0.22s of system time, showing that there is definitely a decent amount of overhead associated with using our logger.

Challenges Faced

1) Figuring out the correct implementation for the proc entries. Most of the tutorials available online are for kernels below version 3.10. In kernel version 3.10, the proc entry APIs were changed.

2) Designing a data structure to use for storing the log. We initially planned to store excess log entries to a file on the disk. We realized that this was not a good idea and the time taken for file I/O would be quite large. Therefore we took the route of using a data structure in the kernel (in this case a circular buffer).
3) Testing the functionality for reading/writing to the proc file. After setting the permissions to 0600, the read and write could only be done via root. So using sudo to read worked fine, however using sudo with echo did not work. This was because the “>” (redirect) uses the regular shell, not sudo privileges. Hence the way to solve it was to sudo su into the root user and then perform the same commands.

Possible Improvements

1) Figure out a way to make the stored log file persistent even after a reboot. This could potentially be done by having the kernel module call a userspace application to write to a file.
2) Better formatting of the log outputs so it seems more reasonable for the log reader.
3) Designing a UI application to change the settings instead of using the proc files since it would be much more intuitive for the user.

What did we enjoy?

1) Teamwork
2) Learning kernel development from scratch (it’s fascinating)

What did we not enjoy?

1) Loading a janky kernel module into the kernel and crashing the machine.

What took the most time?

1) Making design decisions.
2) Figuring out proc entries (since we all ended up finding the same pre-3.10 kernel tutorials)