ITP 30002-02 Operating System, Spring 2020

Homework 1

Jeon, Yeo Hun / 21500630 / 21500630@handong.edu

1. Introduction

In this assignment I worked on building loadable kernel module (LKM) with two main functions. One function is to block specific user from opening files those have specific substring within their filename. Another function is to protect all processes created by certain user. The username and string will be given by system user, which means there would be a communication between the user space and kernel space. To implement the functions, I built a module which switches system call behavior to open file and kill process provided Linux kernel with my own function defined in the module. To support the communication between user space and kernel space, I built a command-line interface called 'jerry'. This interface gets input from user and pass the input to Mousehole to start the functions.

2. Approach

The abstract workflow of Mousehole and Jerry goes simple. After Mousehole set into kernel, Jerry gets commands from user. Jerry process the command into a form that Mousehole understands and pass the command to Mousehole through proc file system. Mousehole gets the command from proc file system and set its state. Whenever system call for opening file and killing process occurs, Mousehole made decision with based on the command it has.

Starts from the user-interface, Jerry, it gets command from user. Since it is designed as CLI, it gets command name and parameters to give Mousehole. I designed some command names and their parameters. The first command name is '-BlockOpen' with two parameters as username, and substring. This command indicates that Mousehole needs to work for blocking sys_open(). The other command is '-BlockKill' with username as parameter. This command will make Mousehole to make processes created by the user with given name immortal by catching sys_kill() system call. The last command is -ReleaseAll without any parameter that tells Mousehole to do normal system call routine for all situation.

To pass the command type and parameters, Jerry needs to process the data first. Since the username from user is given as string, Jerry needs to convert the username to uid, the form that kernel can understand. In Linux command, it is easy to get uid from username by giving 'id -u username' command, I used 'popen' function in stdio.h to make pipe of this command with shell. Jerry also converts the command names into single digit, so Mousehole easily manage them. I set '1' for -ReleaseAll, '2' for -BlockOpen, and '3' for -BlockKill. These command digits and given parameters will be combined by Jerry to make standard command form to give Mousehole: "<command> [uid] [substring]".

Since Jerry is in user space and Mousehole is in kernel space, Jerry needs to use proc, the virtual file system, of Mousehole to communicate with the module. When the command is ready, Jerry connects itself with the proc file system by opening '/proc/mousehole' and get file descriptor. With this file descriptor, Jerry writes the command. When the write occurs in the proc file system, the function defined in Mousehole for writing proc is invoked. In the function, Mousehole copy the written data from user space to kernel space. Then it reads the very first part of the data to see which command is given by Jerry. Mousehole will then decides how many arguments should be read from the data, and put the data into its global variables, command, username, and substring.

To dominate system call behavior, Mousehole needs to know when the system call of opening file and killing process occurs. Therefore, when Mousehole installed into kernel, the initializing function in Mousehole brings system call table and changes the system call routine contents to the function it has. Before changing, Mousehole saves the original system call routine in case Mousehole needs to call the original routine. The system call routine for opening file is indexed in __NR_open in the table, and __NR_kill for killing process.

When sys_open() system call occurs, Mousehole first checks the command indicator as its global variable. If commander does not want blocking, simply calls the backed-up of original sys_open() system call routine. Otherwise, the function checks if the filename passed to the system call contains the assigned substring. If it is true, now compare the assigned uid and uid of user who opens the file. This can be done by accessing 'task_struct' and get

uid of the opening user. When they have same uid, return '-1' as failure. Otherwise, return backed-up sys_open() to give normal behavior.

For sys_kill() behavior, if the module is not asked from user to do its behavior, return backed-up original sys_kill(). Otherwise, Mousehole will use for_each_process in linux/sched/signal.h. This function will bring all the running processes one by one and make task_struct pointer variable references the process. Since the task_struct contains both pid and uid, Mousehole traverse all the running processes and find the process matches with the pid passed to the system call. Then Mousehole checks if the process is created by the assigned uid. If it is created by assigned uid, Mousehole returns -1 value to send failure signal. By returning -1, system call to kill process will be rejected. Otherwise, return the backed-up sys_kill() system call to make normal behavior.

3. Evaluation

I set two additional users named 'guest1' and 'guest2'. Then I created directory located in root. This directory has authorization of read, write and execution for all users belong to the group named 'ShareUser'. The super user and two guests are added to the group to share files in the directory. In the directory, there are three different files named "he.c", "hello.c", and executable files of them. To observe the functionality of each options, I conducted two experiment. For BlockOpen, I checked if the guest1 and guest2 can open the files with certain substring when guest1's username and a string "he" are given to Mousehole through Jerry. For BlockKill, I checked if the super user can kill the process made by guest1.

The first evaluation was on BlockOpen. Super user gave Jerry BlockOpen command with "guest1" and "he" as its arguments. Then I checked if guest1 can open any files those contain "he" in the filename. To open these files, I used 'cat filename' command to print the contents of the file and also used 'vim filename' to modify the contents. As the result, I could observe that the 'cat' command only shows the filename, and 'vim' command shows empty editor screen with no permission message while guest2 was able to open them. After the super user gave -ReleaseAll option to jerry, guest1 could open the files successfully.

The second evaluation is on BlockKill. Super user gave Jerry BlockKill command with "guest1" as its argument. The guest1 runs the executable file in the shared directory which has infinite loop. After the process started and I could check the process information with command 'ps -al'. I got the pid of the process through the result. Then 'sudo kill -9 pid' command is made to kill the process. As the result, even though I used 'sudo' keyword to kill the process, it failed to kill the process. When super user gives -ReleaseAll command to Jerry, it was able to kill the process made by guest1. I uploaded this experiment on following link: https://youtu.be/4k3CrSp_s-c

4. Discussion

Even though the functions of Mousehole and Jerry worked properly, I found out some security issue on the LKM. After the LKM installed, it was able for guest1 and guest2 to change the behavior of Mousehole. If guest1 directly write a string into /proc/mousehole, even if the string is not a form corresponding to the one made by Jerry, proc_write function in Mousehole will be invoked, and destroys the current command given by Jerry. Therefore, it would be necessary to make authorization checker in Mousehole so no other process can give command to Mousehole besides through Jerry.

I spent a lot of time in setting exception handler and data process phase in Jerry. I thought that it would be simple to make user-interface, but I has a lot of things to consider especially as CLI. I need to care about user's wrong input, and if user gives username that is not existed, it could be disastrous because Mousehole does not know either the given user is really exists or not. I feel huge appreciation to developers who built standards of CLI and GUI, which make our life much easier.

5. Conclusion

I was asked to make LKM called Mousehole and user-interface Jerry. The LKM should have two main features: blocking specific user from opening file having certain substring, and protecting all processes made by specific user from kill system call. To implement these features, I switched the system call behavior routine in system call table of Linux kernel with my own function defined in Mousehole. During Mousehole is installed in the system, any 'open' system call and 'kill' system call will be hooked by Mousehole. To assign the parameters that Mousehole needs, I made CLI, Jerry, which gets arguments from user and process them into the form that can be understood by Mousehole. The communication between Jerry and Mousehole is made by proc file system, the virtual file system connects user and kernel space. Jerry writes the command into the file system and proc_write function will be invoked in Mousehole. In the function, Mousehole will get passed arguments and determines behavior.