ITP 30002-02 Operating System, Spring 2020

Homework 1

Tae Eun Kim 21400217, 21400217@handong.edu

1. Introduction

The ultimate goal of this homework was to block a specific user from opening a specific file, and to give immortality to a specific user's process to keep it from being killed.

To do so, I built a Linux Kernel Module and a user-level interface program to control it. This module work as a backdoor in the system, like a mousehole, which is manipulated by an interface provided by a user-level program, a mouse (jerry). The details are discussed in section 2.

By loading this module in the kernel, we can use jerry to set whom to block, what to block and whom to give immortal processes.

2. Approach

2.2 Overview

Here is the simple overview of my approach. In order to achieve our goal, I changed the system call table so that it will point to my own handlers. Then, inside my handlers, I checked some conditions, regarding user ID or the file name. According to the condition, I either denied the request, or returned the original handler. This all happened in the kernel level, thus implemented in the LKM part. In order to access and control this module, I built a user-level interface program. It reads and writes on the proc file that provides a virtual file path to access the module.

2.2 LKM

First, I will explain the LKM part. In order to accomplish the goals of the given task, I needed to alter the default system call handling routines. To block the opening of a file and to prevent killing of a process, system call for opening and killing was changed.

Here is how I changed the handling routine. I first accessed the system call handling table. Then I looked up for the element for open and kill by their index, which are each '__NR_open' and '__NR_kill'. I first backed up the pointer for the original handler, because I still need them. Then I replaced the elements of the table with my own handlers. This all happened at the initiation of this module. By doing this, whenever a user tries to open a file or kill a process, my handler snatches the request. Then my handler checks if this request is valid. If there is no problem, it returns the original handler, which will do the given job, or else, it will let down the request by returning an error.

2.2.1 Handler for opening a file

In order to block a specific user from opening a specific file, we need to know the current user, and the file that he/she is trying to open. The file name is given as an argument of system call for open. But the user ID, I had to access the current "cred" data structure, which holds the security context of the task(process). From there, I extracted the user ID. Once the current user ID and the file name was extracted, all that was left was comparing them to what I have. If they match, I returned -1, which basically just denies the request. Otherwise, I called the original opening handler with the same arguments that I was given.

2.2.2 Handler for killing a process

To prevent the killing of a specific user's process, only thing we need to know is the user ID of the process being killed. I used the Process ID given as an argument to search through the current processes represented in task_struct. If I find the target process, and if the user ID of that process matches the one I have, I denied the request the same way I did in section 2.2.1. Otherwise, I also did the same as in section 2.2.1.

2.3 User-level interface program

It is not easy to access the kernel-level module from user-level. So, I made a program named jerry, to take user's input and perform given operations, which are read status, block some user, immortalize some user, set to default. To do so, jerry reads or writes on the proc file of the module. I set a protocol between jerry and the LKM. When jerry receives an operation from the user, it asks for required inputs, then writes on proc file in this format. "op_code given_inputs". Additionally, jerry translates the given username into an ID before passing it on. To ease the retrieval of the user ID, and accessing the proc file, I implemented these by forking the process and delivering the data through piping. Examples of using this interface are shown in section 3.

3. Evaluation

Evaluation of this program is straightforward, because its functionalities are clear. To test on a multi-user situation, I had my sudo-user connected to the system via VMware machine, and the target user via the PuTTY. I provide interface examples of jerry.

```
goodtaeeun@ubuntu:~/Workspace/OS_2020$ ./jerry
***Choose operation***
b : to block a certain user from opening some files
i : to give immortality to processes owned by certain user
s : to see the status
d : to set to default settings.
q : to quit and exit

Your choice is :
```

There are three users, the sudo user “goodtaeeun” (1000), “mouse” (1001), “mouse2” (1002).

3.1 Blocking

First, I need to check if I can block, and unblock a user to test the blocking functionality. To do so, while running the jerry, I typed operation “b” and gave “mouse” and “hello”, which blocks “mouse” from opening any file with “hello” in its name.

```
Your choice is : b
Enter the name of user you want to block : mouse
Enter part of the file name you want to block : hello
```

The result was successful, mouse couldn’t read, write, utilize files like “hello.c”. However, when I set it free with choosing the option “d”, it restored its access to the files.

```
mouse@ubuntu:~$ cat hello.txt
cat: hello.txt
mouse@ubuntu:~$
```

3.2 Killing

I can split this case into prevent killing the process of oneself, and prevent killing another user’s process. For the first case, I chose operation “i” and gave my username, “goodtaeeun”. After opening another pane with tmux and running a program on it, I couldn’t kill the process. However, when I set the module back to default, I could finally kill the program. So, it was successful.

For the second case, I set the immortal user to “mouse2”. Since killing other user’s process requires sudo authority, I tried to kill the process of “mouse2” as sudo user. However, I could not kill it. Only after setting the module back to default, I could kill the process. So, this was also proven to be successful.

3.3 Interface

To see that the interface is delivering the user inputs to the Module successfully, we can check the status of the module after giving inputs. By choosing the option “s”, jerry gives the current status information of proc file about blocking and killing functionalities.

```
Your choice is : i
Enter the name of user you want to give immortality : mouse2
```

```
Your choice is : s
blocked uid: -1
blocked file: ''
immortal uid: 1002
```

4. Discussion

The most fundamental thing I learned by this assignment is that I can now understand the big picture of how system calls are managed. Although I learned in theory how it works, I didn’t deeply understand it in a big picture. But by actually receiving a system call and handling it with my own handler, I can understand the process. Now I can even change the behaviour of system calls by changing the handler routine.

4.1 Tackling difficulties

Despite the abundancy of given resources and example codes, at the first glance, it was hard to picture how the program would work as a whole. The reason was that I don’t have experience in kernel programming. But I could build up my understanding by carefully reading the given codes. It indeed took a lot of time because of this knowledge building process, but once the understanding was accomplished, it didn’t seem that difficult anymore.

4.2 Applicable points

As professor said, making an LKM can actually be used in hacking. If I want to attack the system, I can take away almost every authority given to a user by installing and loading a single module in the kernel. The user may not be able to kill a process, open a file, or even delete a file. The only solution may be just turning off the machine.

It can be applied in managing multiple users too. With this assignment, I only received one user at a time, but if I change it to manage multiple users, I can set their access to files separately to provide privacy and security, protect their process to be not killed.

To a personal use, I can change the handler of system call for deleting a file to ask me again for some important files that I have chosen. This way, I may not accidently erase my important data.

5. Conclusion

I built a program that controls the opening of a file and killing of a process. It was achieved by making and loading an LKM that changes the handling routines of those system calls to my own. It also has an interface on user-level that communicates with the LKM via proc file by IPC through piping and forking.

They were all novel skills for me. I learned them through video lectures, but that didn’t guarantee that I can make a use of them. However, this assignment put me in a place to actually use them and assemble the knowledge into one piece. I am grateful to accomplish this assignment.

***video link:** https://youtu.be/LtNQnJPapmA