**1. Introduction**

ITP 30002-02 Operating System, Spring 2020

**Homework 1**

Eunjun Jang (21800633), jangej1031@naver.com

The mousehole can be summarized into two functions. The first function is blocking the file opening. With the received uid and file name from Jerry, the mousehole prevents a certain user from opening the file. The second function is to protect the process kill. Mousehole protects to kill process executed by uid which is provided by jerry before the kill system call routine is executed.

User interface (UI) part of mousehole is jerry. Jerry provides to user more fancy and safe command-line interface to use mousehole’s functions. To implement this communication between jerry and mousehole, it constructs a communication through the proc filesystem.

**2. Approach**

In this section, I will describe details on the mousehole and jerry.

**2.1. Jerry Command-line Interface (CLI)**

Jerry provides five commands: -bf, -uf, -pk, -uk, and -status. If given command-line is not valid format, jerry prints its usage. Also, if given command-line has some error, jerry prints error message. For example: jerry -bf ubuntu hello.txt.

-bf stands for block file open. This command takes two arguments which are username and file name. -pk stands for protect killing process. This command takes only one argument which is username. Jerry converts username to uid using getpwnam() function. Because with username, jerry can get passwd struct which has its uid. -uf and -uk indicate -bf command and -pk command cancel, respectively. -status command shows mousehole’s current status. -uf, -uk and -status have no additional arguments.

**2.2. Communication through Proc Filesystem**

Jerry and mousehole are establish the communication using proc filesystem. When mousehole initiates, it creates mousehole file into /proc directory and register its own callback functions like proc\_read. In jerry, if the command is -status, Mousehole provides its status through reading /proc/mousehole, so jerry reads /proc/mousehole

**2.3. Blocking file opening**

If the sys\_open routine that was originally in the system call table at \_\_NR\_open is replaced with a mousehole’s sys\_open function, the mousehole can be listened to for all sys\_open system calls.

If the mousehole’s sys\_open does not call an original sys\_open, the file is not opened.

In hooked sys\_open, mousehole tokenizes the file path received when the system call is called using strsep with ‘/’ delimiter. Mousehole uses the strstr function to compare whether the string received from Jerry exists as a substring of the last string of strsep. If the return value of strstr is not null, mousehole compares uid. Using current → cred → uid, mousehole compares the uid of the user who requested this system call and the uid received from Jerry. If it is the same, it means mousehole have to block this system call, -EACCES is returned. Otherwise, original sys\_open is called.

**2.4. Protect process killing**

Protecting process kills is similar to preventing file opening. As the original sys\_open was in \_\_NR\_open in the system call table, sys\_kill is at \_\_NR\_kill in system call table. Mousehole replaces it with own sys\_kill. Then, like sys\_open, mousehole can listen for all process kill system calls. Mousehole needs to compare the uid received from Jerry with the uid that creator of the process being killed. To find the process, which is candidate of killing, use for\_each\_process to search all currently open processes and compare pid. The process is implemented as a task\_struct, which has a cred among its members and a uid in cred. Mousehole compares this uid with the uid received from Jerry. If it is equal, it returns -EACCES. Otherwise, original sys\_kill is called.

**3. Evaluation**

I will describe the requirements that must be satisfied to ensure that they are implemented properly.

In the function one, only the user given by Jerry should not be able to open the file. To show this, I give a user through Jerry and try to open the file as a user and a root user respectively. Also, I try to open another file which is not contain jerry provided file name with a user. Later, I undo the file open prevention command through jerry and tries. To proof successfully file opening or failing, I made a simple file open program with open() function. If the file descriptor (fd) returned by open() is negative value, it prints fd and error string. In my experiments, It shows that the user which is provided to mousehole cannot open the file. In contrast, root user can open the file.

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

Figure 1. The function one experiment with test.txt, test\_a.txt hello.txt.

In the function two, in order to check if the second function has been properly implemented, the process created by the user passed through Jerry must be prevented from being killed. To show this, after running a simple blocking program written by C, we will give a command through Jerry and check whether the process is killed with the privileges of various users.

스크린샷, 모니터, 전화, 노트북이(가) 표시된 사진

자동 생성된 설명

Figure 2. The function two experiment with dontdie process.

In my second experiment, I successfully showed it is working very well.

**4. Discussion**

In this discussion section, I will describe that I noticed information and some future works.

**4.1. About Error Throwing**

As you can see in Figure 2, kill said that is successfully killed. But it did not kill at all. In my mousehole code, I simply return -EACCES value. But it really not working. I searched about kernel module error throwing mechanism but I can’t find another method. Many people said that returning negative value is automatically set errno by OS. I need to figure out this problem in future.

**4.2. Process Blocking Product for Kids**

In iOS, Apple showed a "screen time" feature that forces a process to shut down when it's on. Like this, many parents need process management program for kids to manage usage of digital devices. I think this kind of program made by this kernel module. After this homework, I want to make my own process management program to save my time.

**4.3. About Rootkit**

At the first homework describing video, professor said that this homework is making kind of rootkit. After making mousehole, I really have a question, how to make rootkit? I googled and found nice site describing about rootkit. Like hook sys\_open system call, rootkit hooks setreuid system call and change privilege to root. So I can run with root user. It is amazing!

**5. Conclusion**

Mousehole can be summarize by two protection functions: file open by specific user and being killed process which creator is specific user. In this homework, I really enjoyed and had some funs to explorer linux kernel site, forums and stack overflow.

Demo: <https://youtu.be/P2aP4WJnnek>

Drive: <https://drive.google.com/file/d/155oqILtNL1KX5e0Qoil-47jSv7TgJa_N/view?usp=sharing>