Operating System PA1
REPORT

21400240 Kim Hyeong Jun
21700646 Jeon Hye Won

1. Intro

A. Problem Analysis

a. Functionality 1

Linux Kernel Module(LKM) called dogdoor should record the names of files (up to 10) that the user recently opens. Specifically, bingo should retrieve the log lists and prints it to the user when the user requests. In order to do this, it is necessary to access the linux code that runs when the file is opened. And the user information is also needed for checking whether its user is currently tracking or not.

b. Functionality 2

Second functionality is preventing a certain process from killing. The process which has a specified pid only can be killed after user commands to release immortality. To prevent processes from being killed, it is necessary to first analyze what mechanism is executed when the kill command is executed. `sys_kill()` is the system call that is invoked when `kill()` is executed. So we will try to prevent process kill by intercepting this system call.

c. Functionality 3

A name of dogdoor module should not be seen from the module list(result of `lsmod()` command) once user gives a specified command. In addition, the name also makes itself appear again once user gives another command. To implement this functionality, it is necessary to first understand data structure of module list. We guess an instruction would access some data structure about the module which is currently running when the `lsmod()` command is executed. Therefore, if it is possible to delete the current module

directly from the module list, we might implement functionality in that way. And before deleting the module, it should be stored to return.

B. Solution Overview

a. Functionality 1.

When a user tries to open a file, all the files necessary to open it are opened together. And system call `sys_open()` must be run to open the file. At this time, by modifying the contents of the system call table, a function `sys_open` can be operated as we intended. And functionality requires not all file access log, but a specific user's file access log. So current user information is also needed. To figure out current user is the user that tracked or not, I use `current_id()` in `<linux/cred.h>`.

b. Functionality 2.

Similar to functionality 1, we can intercept the system call and modify its contents as we intend in `sys_kill()` function, which is invoked whenever a process called kill is executed. However, we should check to see whether preventing pid from user and process id from kill command are same. The difference from functionality 1 is that, in functionality 1, information from the current user had to be obtained through a separated other library and function, but in functionality 2, the process id is given to the `sys_kill()` itself in parameter, there is no need to bring the id of the process separately.

c. Functionality 3.

`lsmod` works by printing the contents of `/proc/modules` to the terminal. So if we run `cat /proc/modules`, we could get the same thing. Therefore, it would be best to remove the contents of the module from `/proc/modules` itself. However, the module should be stored so that it can be returned to its original state before being deleted.

2. Approach

A. Solution Design

a. Functionality 1.

we will modify the `read` part of example openhook. If the user executes a command that reads `/proc/dogdoor`, a list of the files that tracked user used will print out. But we don't need to take every log of the file. We only need to save a maximum of 10 logs. So, I take the concept of circular queue and implemented in the same way. If more than 10 logs were taken, the new logs were overwritten with the oldest logs.

```
if (tracking_user == cred_user_id.val) {
    strcpy(log_file[current_line], fname) ;
    current_line = (current_line + 1) % 10 ;
}
```

Fig 1. Code From Functionality 1

`tracking_user` was used as a flag to determine whether the current user should be tracked or not. It also tells us which users should be tracked if tracking. This variable is a global variable that continues to exist while the module is alive. We modified the contents of the table where the system call is stored. As a result, we could use `dogdoor_sys_open` function instead of the `original sys_open`. In this function, if the user to be tracked matches the current user, the name of the file would be saved in the array log_file.

b. Functionality 2.

We have compared the variable `no_kill_process` with the pid which is given to `sys_kill()` as parameter. Also, similar to functionality 2, `no_kill_process` used as a flag to determine whether or not to stop a process from killing.

c. Functionality 3.

The list of modules we want to access were made as a linked list. By viewing linux source code, we notice that `__this_module->list` is a struct, and its member variables are `prev` and `next`. Because data structure about module were made as a doubly linked list, it was possible to implement the functionality 3 using linked list's operation such as add and delete.

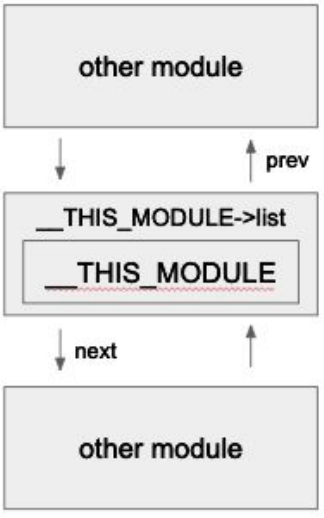

Figure 2. Structure of Module

d. bingo

We wanted to get a consistent command to provide a user-friendly interface. Also, we print the help screen if user do not give any arguments. Instead of writing directly to `/proc/dogdoor`, we write file pointer and interpret it using a huge switch statement.

3. Manual

A. Help screen when no argument is given.

```
os_test@CRA145:~/hyewon$ ./a.out
HI ~! I'm BINGO! Give the command number :)

====================================================
1. Give me a user name you want to track.
2. Retrieve the tracking list.
3. Specify a process ID.
4. Stop immortality.
5. Make module disapper.
6. Toggle.
====================================================
```

Figure 3. Help screen of bingo

Detailed manuals and methods of use will be covered in the video.

4. Discussion

A. In the given functionality 2, I intercepted `sys_kill()` to prevent certain process from being killed. I wondered if it would be possible to terminate the process in other ways in the situation where `sys_kill()` is intercepted.
   a. Even if we prevent vim process from being killed, it is confirmed that the `:q` command enable the program exit normally.
   b. Even though we block the current process of bash from being killed, it is confirmed that the program exits normally if we typed the `exit` command.
   c. Even if we prevent the kill in the executable file, pressing control + c will terminate normally.
   d. As a result, we confirmed that all three cases can be terminated normally. Therefore, we could guess that all the cases we've checked terminated by a different path than `sys_kill()`. We realized that when one process is terminated, there are various mechanisms not only that kill.

B. Killing Root Process
   a. When all current processes are identified by the `ps aux` command, the init process is the root of the process tree which is described in text book. I tried to

kill init process, but it was not killed. If we accidentally kill a process in root, all the processes under it will be zombie processes because there is no ancestor to connect the underlying processes. So we can see that the kill of the init process is blocked.

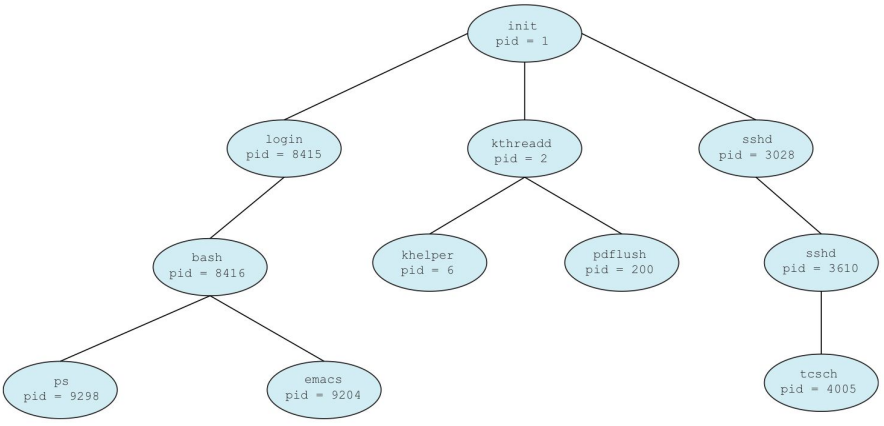

Figure 4. Process tree

C. Insert Duplicated Module

a. If we hide a module, it disappears from `lsmod` and is invisible. Because we removed it from the data structure at all, the computer can not identify the module that is currently running in any way. If so, Wouldn't it be possible to insert duplicate modules? It was not possible to insert a duplicate module even if a module was hidden. But I found that if the module is hidden, then the error message for `insmod`(ERROR: could not insert module dogdoor.ko: Invalid parameters) is slightly different when the module is visible(ERROR: could not insert module dogdoor.ko: File exists).

D. LIST_HEAD Structure

After searching `/proc/modules/`, I noticed that this file contains a list of currently loaded kernel modules. This list was a list of `struct list_head` types. We researched structure `list_head` and related contents from books and internet resources. In particular, the `<linux/list.h>` file clearly showed the definition of the function associated with the structure `list_head`.

```
#define LIST_HEAD_INIT(name) { &(name), &(name) }

#define LIST_HEAD(name) \
        struct list_head name = LIST_HEAD_INIT(name)
```

Figure 5. Definition of list_head structure