Timothy Hinds

COP4600

Professor Blanchard

9 November 2018

Project 3 Report

For starting off, I read the given resources which put me off to a good start. I wanted to create the classification attribute, so I looked for where the inode structure was created. It is created under the file system header file 'fs.h', and so I added the 'i_classification' attribute to be another attribute associated with the inode. Afterwards, I went into 'fs/inode.c' to initialize the classification variable to 0 under the init inode function.

The next step of this project was to create the system calls. Luckily for me this was familiar from project 1 so I already knew to declare my system calls in the system call table 'arch/x86/entry/syscalls/ syscall_64.tbl' and also add their function headers to the 'linux/syscalls.h' system call header file. Writing the system calls was a bit trickier than I hoped it would be though. I was finding it difficult to find any resources which would allow me to get the inode of a file only given its filename. However, I was finding that it was possible to get the inode of a file given its file descriptor. And so, I decided to change my systemcalls to take the file descriptor as a parameter instead of the filename.

From here, I was able to create an inode and set it equal to the inode of the file that is inputted to be modified. So now that I had the inode as I wanted it, I still needed a way to check the permissions of the file. Luckily since I had the inode I was able to do just that by checking the inode->i_mode variable. The i_mode variable returns the permissions of the file. The trickiest part of this project was piecing together the information about the inode to realize that all you really needed to figure out for the systemcalls was that if you could get the inode of the file, you can check and alter any of the file's attributes. However, I think the given filename parameter in the library function threw me off and made me look in the wrong places.

Now that my system calls are in place, in my library functions I needed to obtain the file descriptor before I could use my systemcalls. In order to do this I used the open() system call, which returns the file descriptor of a file given its name. I made sure to test multiple different permissions of a file when running the securitytagtest. I think I accounted for all permission possibilities in the systemcalls.

I ran into some troubles during this project when I was trying to figure out how to obtain information about a file only given its filename in kernel space. I kept trying to run system calls in kernel space, only after about 3 hours of trying to get different system calls to work in kernel space did, I find out that it was impossible to run a system call within a system call. I felt dumb, but I learned it the hard way. I did not encounter any bugs while running my test cases.

https://youtu.be/hnizQXhHxmw

https://linuxhint.com/stat-system-call-linux/