Project 1 Report – Timothy Hinds

In order to implement the security level attribute, I had to figure out how to find the existing process attributes and where they were initialized in the kernel. I found a helpful document which described the process descriptor, the structure which contains all the information about one task, including process id and other attributes. The attributes are declared in a function “struct task_struct” which is implemented in the file /include/linux/sched.h. I added the line “int securitylevel;” to declare my new process attribute. I then needed to initialize the variable for all new processes, and I found that the /include/linux/init_task.h file was the one that initialized the attributes for processes when they are created. Inside init_task.h I initialized securitylevel to 0.

I declared the system call functions in a header file, then initialized the functions in securitycalls.c. I used the task task_struct to create a task variable which would help me search for my desired process to change. Then used the for_each_process() function to search through all active processes for the desired one to edit. I used this for the get and set functions. I used pointers to point to the securitylevel attribute and the pid attribute. In order to check for user permissions I used “current” which is a task_struct which points to the current process being run, and I used current to point to the current user id and process id.

Once I was done writing the functions, I created my makefile for my syscall with the line, “obj-y+=securitycalls.c”. Then I edited the kernel’s makefile found at /reptilian-kernel/Makefile, and included my new library by editing this line to be: “core-y += kernel/ certs/ ….. securitylevel/. Following that, I edited the syscalls headerfile found in /include/linux/syscalls.h, and at the bottom of the file I included the syscall functions. Finally, I then had to edit the system calls table found at /arch/x86/entry/syscalls/syscall_64.tbl, and added my system calls.

After some tweaks to the syscalls I was able to get them to function properly. Then I had the task of trying to get the permissions to function properly when called. I found on stackoverflow someone who suggested using current->cred->(gid or pid).val to be able to point to the current process pid and user id. After figuring this out I then started formulating my logic statements to ensure that the permissions were upheld correctly.

Creating the harness functions caused some challenges when attempting to compile sudo ./harnesstest. I kept getting errors which wouldn’t complete the running of the program. I figured out that I needed to use mallo() to allocate space for the arrays in my function that returned arrays.

I had the hardest time figuring out how to get my system calls to work while accounting for all of the parameters checked in the security level test. It felt like no matter how many times I rewrote my logical statements either every test case was successful, or unsuccessful but never all correct. I spent probably 70% of the time on this project just working with the set_security_level function’s parameters for user processes.

Sometimes when I recompiled the kernel, I would have to restart the machine multiple times before it would load in, and I ran into a lot of snapshot recoveries from having whitespace in my code.

https://tampub.uta.fi/bitstream/handle/10024/96864/GRADU-1428493916.pdf

https://medium.com/@ssreehari/implementing-a-system-call-in-linux-kernel-4-7-1-6f98250a8c38