Michael Gordon P1 System Calls Report

In order to implement this project first I needed to make changes to the task struct. From given resources, I knew that I could accomplish this by making changes to **./include/linux/sche.d**. Inside the declaration for the **struct task_struct** I added the following line. **int securitylevel;**

Next I had to initialize **securityleve**l to 0. I accomplished this by finding the **init_task.h** file and inserting the following line inside of the **INIT_TASK** definition. **.securitylevel = 0, \**

This would conclude the first part of the project. Successfully modifying the process table.

The second part of the project required me to create system call and could approximately modify the **securitylevel** based on certain conditions. To accomplish this, I had to add my system call to the system call table located in **/usr/rep/src/reptilian-kernel/arch/x86/entry/syscalls/syscall_64.tbl.**

| | | | |
|---|---|---|---|
| **332** | **common** | **set_level** | **sys_set_level** |
| **333** | **common** | **get_level** | **sys_get_level** |

Afterwards, function protypes were defined in the **./include/linux/syscalls.**h file

**asmlinkage long sys_set_level(int pid, int level);**

**asmlinkage long sys_get_level(int pid);**

These changes must specifically be made to link my syscalls to the kernel.

Now my functions were ready to be implemented**. sys_get_level** implantation was straightforward as I used the **pid.h** library functions **find_get_pid()** to get a **pid_struct()** for the PID which would ultimately help me find the correct **task_struct** using **pid_task()**. If the task struct wasn't found -1 would be return. If it was found then I was able to access the **securityleve**l of the **task struct** and return it.

**sys_set_level's** implantation also used the same implantation to find the current process's task_struct(using **sys_getpid()**) to find current process's pid) and the target process's **task_struct**. Afterwards I implemented the appropriate logic to check whether the calling process had the ability to change the target process's **securitylevel**.

This was done by first checking if the target process existed. Afterwards if the calling process was a super user(checked by using **sys_geteuid()** and seeing if that equaled zero), it was allowed to change the process to whatever **securitylevel** it wanted. The rest was implemented using simple Boolean logic.

I also had to create a Makefile including my syscall. I also had to include my syscall folder in the main Makefile by adding inside under the **core-y = ..dirs**

After this I implemented the appropriate library functions. For the main get_level and set_level fcts, I simply passed the parameters to the syscall. The retrieve_params fcts store the syscall number, and parameters into a pointer to int. The interpret functions just had to return the passed **ret_value** as my syscalls appropriately implemented the interpretation already.

Testing was straight forward as I compiled the included testing library with library linked. During my first test I passed all except the test case where the child process with **securitylevel = 3** tries to change the parent process with **securitylevel= 3.** I fixed this bug by changing my logic.