I put my testing all over the report, so the blue text describes testing/known bugs.

To begin this project, I realized that the securitylevel folder must be in the userspace (a.k.a. /home/reptilian/… (for the most part how I used it). I went to /usr/ originally, but realized my error when I had permissions denied, and so the folder was put in the directory /home/reptilian. From there, I knew that I needed securitylevel.h, securitylevel.c (where the main functionality would go for managing get & set for processes' securitylevel using syscalls), and a Makefile (with the appropriate compilation commands).

It is important to note that with only these three files (.c, .h, & Makefile), the securitylevel.o and libsecuritylevel.a files would be autogenerated with "make". It is also essential to understand that the two library functions explained in the specification document – particularly set_security_level and get_security_level – are actually implemented within the kernel, and the securitylevel.c file in the user space actually tests out both of those syscalls. The four harness functions are also implemented in securitylevel.c (this is where the bulk of the coded testing I have made lies), as the harness functions aim to verify the security of the system calls from the system library (using library as an intermediary in the syscall process). My first approach was a bit meek for facilitation purposes: I made a syscall that would print to log and be able to check that the input var passed efficaciously (in this case, the parameter is in the user space). Going further, I made a function to make the syscall. I tried to read various parameters within the growing process table; at first, it did not work, as my code was faulty. Most of the issues lied in the number of parameters, the spacing issue in the table of the syscalls, & permissions denied (rarely).

Aside from that, since the library functions were to be in the kernel mode, I navigated to /usr/rep/src/reptilian-kernel and made my new directory called "proyecto_1." For the Makefile, I had to be confident that the kernel src had my new dir & I had to compile the one source file I made (fortunately, only one .c needed); the .c consists of calling the syscalls to get & set processes' security levels via the two library functions. To accomplish this feat, later on, I had to include my changes in the kernel's Makefile, as well. Navigating over to there, I made sure that it included my new directory (a.k.a. letting the compiler know my new folder holds new syscalls to be maneuvered). Moreover, I had to initialize my security_level var of type int. The goal was to put the var in a place that ideally produced the process table so that each new process will have the property & own version of var. To do this, I navigated towards /include/linux/sched.h from the kernel and initialized my var under a certain struct that accomplished that; it proved to be a logical place to put the var as it fulfills the goal. I used that same struct style as a pointer within my getlevel and set level methods in securitylevel dir. This made things consistent and sensible. This part proved very tricky for me, as StackOverFlow suggested modifying similar files like sudmac and rtc (which led to my kernel crashing). It took a long time of testing with crashing and rebooting (once even deleting a snapshot that just messed up) to find the right file to modify. I recreated my p1.diff several times due to this problem.

In addition, I had to add my new syscalls to the kernel's understanding, and so I navigated into /include /linux/syscalls.h from the kernel & (using asmlinkage) made sure that my new syscalls' signatures were defined, making them understood and clear. I added my two lines at the bottom of the file (I used approach of asm linkages in proyecto_1.c). I also had to make my syscall's names and numbers defined, and navigating to /arch/x86/entry/syscalls/syscall_64.tbl proved to do the trick. I put my new syscalls under the existent table of x64 syscalls (from 0 to 331) & continued their trend. It was a colossal issue placing the new syscalls (directly affecting their numbers) & tables' spacing for declaring syscalls (the solution that allowed calls from securitylevel.c & securitytest.c to work was using solely tabs & no spaces in my other files). I was struggling to understand relationships between call numbers, entry vectors, and their format, too. Eventually, a blogger under a Linux Forum helped me. I broke my machine a couple of times over these small issues.
Furthermore, the project specifications state that "all processes should be initialized with a security level of zero (0)" and, since kernel/fork.c is operated during the generation of a child (which correlates with the definition of forking), then kernel/fork.c should be the place to do this. Right under "nr = pid_vnr(pid);" in _do_fork(…), I added a line to make the security_level var of every new process equivalent to the value zero.