UF UNIVERSITY of FLORIDA
Herbert Wertheim College of Engineering

COP4600

P1: System Calls

Overview

You work for a top-secret shadow government organization dedicated to the rise of the Silurian overlords. You, as a faithful member of the Lizard Legion, are part of the team charged with improving data storage and handling, particularly tracking ***metadata*** – that is, data about data – within the organization's computer systems. You have been tasked adding a security level attribute to the process metadata for kernels running in "Sky Skink", the cloud computing system. Naturally, the Legion uses the superior Reptilian operating system distribution.

In this project, you will implement a system call in Reptilian along with two static library functions that allow the system call to be invoked from a C API. These custom system calls will *get* and *set* a custom process table attribute you will create, called *security level*, to track the permissions level of running processes. We, as your benevolent lizard overlords, will provide a program that exercises and demonstrates the new calls. You create a short video to demonstrate your code. (Our masters will be most pleased.) You'll submit the project via Canvas so as not to invite suspicion.

NOTE: Take Snapshots in VirtualBox! You will most likely brick your machine at some point during this or other projects, and you will not want to start from scratch. <u>No, seriously – take snapshots!</u>

Structure

The project is broken into four main parts:

1) Create a custom *security level* attribute for all processes.
2) Create system calls that allows a process to *get* or *set* the *security level* of a process.
3) Create static library functions that allow the system calls to be invoked via a C API.

Figure 1: A system call invoked from a user program

While exact implementation may vary, the library functions must match the signatures laid out in this document, and the system calls must apply the security model properly.

System Call

Each process in the modified system will have a *security level*. The rules for this system will be based on a write restricted access model, namely:

1) All processes should be initialized with a *security level* of zero (0).
2) A process running as the superuser may read and write the *security level* of any process.
3) A user process can read the *security level* of any process.
4) A user process can write the *security level* of a process at a lower level.
5) A user process may only raise the level of a process to its own level (not above it).
6) A user process can lower its own *security level* (but not that of other processes with the same level).
7) Security levels can have values between 0 and 2,147,483,647 (32-bit signed integer).

The *security level* of each process must be stored in the process table; as such, you will need to modify the process table to add an entry for it. They are called via `syscall(call_number, param1, param2)`. ***Your system call must be limited to no more than two parameters!***

Static Library

You will create a static library to invoke the system calls in a directory named **`securitylevel`**. This will be composed of a header with name **`securitylevel.h`** and a static library file named **`libsecuritylevel.a`**. You will also need to provide a Makefile for this library in the directory. All other sources must be contained within the **`securitylevel`** directory. Please note, *these names of the files must match exactly!*

You will create a tarred gzip file of the **`securitylevel`** directory with name **`securitylevel.tar.gz`**. When testing your code, we will decompress the archive, enter the **`securitylevel`** directory, and build. All functions enumerated below must be made available by including **`"securitylevel.h"`**. See *Submission* for details.

In addition to the standard library functions, you will implement testing harness functions. The testing harness functions are used to verify security of the system calls from the system library (and are required for full credit on this assignment). We will call these functions to retrieve the information needed to make a system call. We will then call the system call within our own program. This ensures that no security checks are being done in the user-level library.

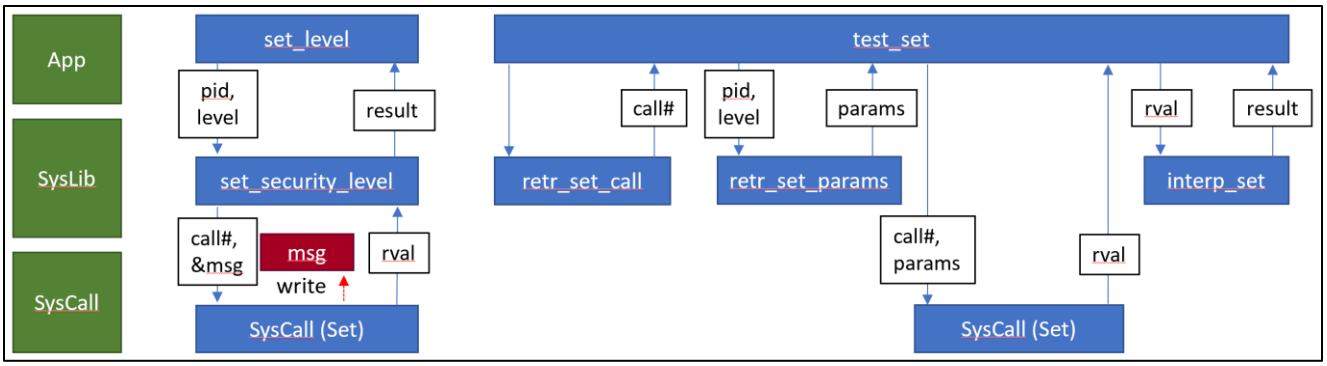


Figure 2: The harness functions can be used to simulate the system calls

Library Functions

These functions are to be used by programs.

`int` **`set_security_level`**`(int pid, int new_level)`
Invokes system call which attempts to change the process identified by **`pid`** to security level **`new_level`**. Returns **`new_level`** on success, and **`-1`** otherwise.

`int` **`get_security_level`**`(int pid)`
Invokes system call which reads the security level of the process identified by **`pid`**. Returns the security level on success, and **`-1`** otherwise.

Harness Functions

System call parameter retrieval data should be returned as a pointer to an **`int`** array of 2-4 values that can be used to make the system call. It has this format:

```
{ call_number, num_parameters [, parameter1] [, parameter2] }
```

e.g.: `{ 42, 2, 867, 5309 }` → `syscall(42, 867, 5309)`

`int*` **`retrieve_set_security_params`**`(int pid, int new_level)`
Returns an **`int`** array of 2-4 values that can be used to make the set-security system call.

`int*` **`retrieve_get_security_params`**`(int pid)`
Returns an **`int`** array of 2-4 values that can be used to make the get-security system call.

`int` **`interpret_set_security_result`**`(int ret_value)`
After making the system call, we will pass the syscall return value to this function call. It should return **`set_security_level`**'s interpretation of the system call completing with return value **`ret_value`**.

`int` **`interpret_get_security_result`**`(int ret_value)`
After making the system call, we will pass the syscall return value to this function call. It should return **`get_security_level`**'s interpretation of the system call completing with return value **`ret_value`**.

Submissions

You will submit the following at the end of this project (3 separate files):

- Report (**`p1.txt`**) in man page format on Canvas, including link to unlisted screencast video
- Kernel Patch File (**`p1.diff`**) on Canvas
- Compressed tar archive (**`securitylevel.tar.gz`**) for **securitylevel** library on Canvas

Report

Your report will explain how you implemented the new system call in the kernel, including what changes were made to which files and why for each. It will describe how testing was performed and any known bugs. The report should be created using **`man`** format saved as a .txt. The report should be no more than 500 words (about two pages in man format), cover all relevant aspects of the project, and be organized and formatted professionally – ***this is not a memo!***

Screencast

In addition to the written text report, you should submit a screencast (with audio) walking through the changes you make to the operating system to enable the system calls. Additionally, the screencast should include you showing/demoing your changes in action. (no more than 5 minutes).

Patch File

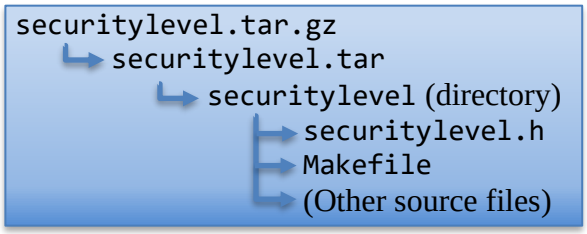
The patch file will include all changes to all files in a single patch. Applying the patches and remaking the necessary parts of Reptilian, then rebooting and then building the test code (which we will also copy over) should compile the test program.

Your project will be tested by applying the patch by switching to **`/usr/rep/src/kernel`** and running:

```
$ git apply p1.diff
$ make && sudo make install && sudo make modules_install
```

Compressed Archive (securitylevel.tar.gz)

Your compressed tar file should have the following directory/file structure:



To build the library, we will execute these commands:

```
$ tar zxvf securitylevel.tar.gz
$ cd securitylevel
$ make
$ cd ..
```

To link against the library, we will execute this command:

```
$ cc -o program_name sourcefile.c -L ./securitylevel -lsecuritylevel
```

Please test your library build and linking before submission! If your library does not compile it will result in **zero credit** (0, none, goose-egg) for the library portion of the project.