UNIVERSITY of FLORIDA
Herbert Wertheim
College of Engineering

COP4600

P3: File Systems

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 system. You have been tasked adding a classification attribute to the file system used by the organization. Naturally, the organization uses the superior Reptilian operating system distribution.

In this project, you will implement two system calls in Reptilian along with two static library functions that allow the system calls to be invoked from a C API. These custom system calls will retrieve and set a custom file attribute you will create to track a file's *classification* for use within the Lizard Legion organization. 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 three main parts:

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

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 file in the modified file system will have a *classification*. The rules for this system will be as follows:

1) All files should be initialized with a classification level of zero (0).
2) Files existing before addition of classification should be interpreted as having classification of zero (0).
3) A process running as the superuser may read and write the *classification* of any file.
4) A user's read and write access for a file is determined by the standard Linux file permissions.

NOTE: The system calls will be called using `syscall(call_number, parameter1, parameter2)`. ***Your system call must be limited to no more than two parameters!*** In addition, the *classification* of files must be stored persistently; as such, you will need to modify the file system to add it.

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

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

Library Functions
These functions are to be used by programs.

`int set_classification (const char *filename, int new_class)`
Invokes system call which attempts to change the file identified by **`filename`** to classification **`new_class`**. Returns **`new_class`** on success, and **`-1`** otherwise.

`int get_classification (const char *filename)`
Invokes system call which reads the classification of the process identified by **`filename`**. Returns the access level on success, and **`-1`** otherwise.

Submissions
You will submit the following at the end of this project:

- Report on Canvas
- Screencast on Canvas
- Kernel Patch File (`lastname_firstname-p3.diff`) on Canvas
- Compressed tar archive (`classification.tar.gz`) for **`classification`** library on Canvas

Report
Your report will explain how you implemented the new system calls, including what changes were made to which files and why each change was made. It must include an explanation of the structures that were changed. It will include description of how testing was performed along with any known bugs. The report may be in Portable Document Format (pdf) or plain-text (txt) and should be no more than a page. It should 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 (~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 lastname_firstname-p3.diff
$ make && sudo make install && sudo make modules_install
```

Compressed Archive (classification.tar.gz)

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

```
classification.tar.gz
  └→ classification.tar
       └→ classification (directory)
            ├→ classification.h
            ├→ Makefile
            └→ (Other source files)
```

To build the library, we will execute these commands:

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

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

```
$ cc -o program_name sourcefile.c -L ./classification -lclassification
```

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/system call portion of the project.

Helpful Links

You may find the following resources helpful when reading about how to add the classification system:

http://www.linfo.org/filesystem.html
http://www.linfo.org/inode.html
http://www.linfo.org/ext4fs.html