File System Project

Overview

You will implement system calls and system library functions that allow a classification level to be set or accessed from the C API. We will provide a program that exercises and demonstrates the new calls. You'll then create a short video to demonstrate your code. You'll submit the project via Canvas.

NOTE: Modifying the file system is a dangerous operation. You will almost certainly "brick" your VM at some point. Back up your source! Take snapshots! **You have been warned.**

Structure

The project is broken into five main parts:

1) Create a routines to read and change the classification level of a file in the Minix File System (MFS)
2) Create routines to "get" or "set" the classification level for a file in the Virtual File System (VFS).
3) Create one or more system calls to execute the routines if the Virtual File System (VFS).
4) Create system library functions that allow the system call(s) to be invoked via a C API.
5) Add manual page entries for the system library calls.

$$\begin{tikzcd}\boxed{\text{System Call}} \arrow[r] & \boxed{\text{VFS Routine}} \arrow[r] & \boxed{\text{MFS Routine}}\end{tikzcd}$$

How the Minix File System (MFS) is read/modified via a Minix system call

While exact implementation may vary the system 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 system will have a classification level. The rules for this system will be based on the standard Unix file access model, namely:

1) All files should have a default classification level of zero (0).
2) A process running as the superuser may read and write the classification level of any file.
3) A user process's read/write access are determined user, group, and world access file attributes.

The classification level of each process must be stored in the file system; as such, you will need to identify a location in the file's meta data to add an entry for it.

System Library

You must add system library functions to the standard library for your system as follows.

Library Functions

These functions provide primary functionality. They should be made available by including `<unistd.h>`.

`int set_classification(FILE descriptor, int new_level)`

Invokes system call which attempts to change the file identified by **`descriptor`** to classification level **`new_level`**. Returns **`new_level`** on success, and **`-1`** otherwise.

`int get_classification(FILE descriptor)`

Invokes system call which reads the classification level of the file identified by **`descriptor`**. Returns the access level on success, and **`-1`** otherwise.

Test Functions

These functions serve as a testing harness to verify security of the system calls from the system library (and are required for full credit on this assignment). They should be made available by including `<harness.h>`.

`int retrieve_set_class_callnum()`

Returns the call number of the system call that would be used to set the classification level of a file.

`int retrieve_get_class_callnum()`

Returns the call number of the system call that would be used to get the access level of a process.

`message retrieve_set_class_message(FILE descriptor, int new_level)`

Returns a **`message`** structure that could be used to invoke a system call to set the classification level of a file.

`message retrieve_get_class_message(FILE descriptor)`

Returns a **`message`** structure that could be used to invoke a system call to get the classification level of a file.

`int interpret_set_class_result(int retval, message msg)`

Returns **`set_classification`**'s interpretation of the system call completing with return value **`retval`** and with final message state **`msg`**. (Depending on implementation, this may just pass-through the return value.)

`int interpret_get_class_result(int retval, message msg)`

Returns **`set_classification`**'s interpretation of the system call completing with return value **`retval`** and with final message state **`msg`**. (Depending on implementation, this may just pass-through the return value.)

Manual Page

You are required to create a manual page detailing the system library function calls. The manual page for your library procedures should be in man page format; you may copy and modify an existing man page for this purpose. You must place the new man page in the proper location so that entering the command man set_classification or man get_classification will return the man page for these calls. You may check environment variable MANPATH to see where the man pages are, and type "man man" to see how the sections are arranged. At a minimum, the man page must have proper header/footer, name, synopsis, description, and errors.

Testing

You should test your code using the provided test code.

Submissions

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

- Report on Canvas
- Screencast on Canvas
- Unified patch file 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 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). The report should be no more than a page and should cover all relevant aspects of the project.

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 (~5-10 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 Minix, then rebooting and then building the test code (which we will also copy over) should compile the test program and link in the new library object code. Typing "man set_classification" should display the man page for set_classification, and typing "man get_classification" should display the man page for get_classification.

The patch file must ONLY change files in the /usr/src directory. We will run the patch file with the following command from the ROOT DIRECTORY (“**`/`**”):

```
patch -p0 -i PATCHFILENAME
```

You project will be tested by applying the patch and then, from within the “**`/usr/src`**” directory, executing “**`make build`**”.