Assignment 3
Assigned: 10/11/2025
Due: 19/11/2025 - 23:55 (GMT+2)

Access Control Logging

Overview

In this assignment, you will develop an access control audit system in C that monitors and logs file access and modification events across the system. Your audit system will track every file open, write, and close operation, and record detailed information in a structured log. A separate monitoring tool will later analyze this log to detect suspicious behaviors. To intercept system-level file operations, you will use LD_PRELOAD to override specific functions from the standard C library.

Step 1: Audit Logging Library

You are required to develop a shared library, named audit_logger.so, which overrides standard I/O functions from the C library using the LD_PRELOAD mechanism. Specifically, you must override fopen(), fwrite(), and fclose() to add auditing capabilities before delegating to the original functions. Each event should be appended to a system-wide log file named access_audit.log.
The log file should be stored in a directory accessible to all users (e.g., /tmp/).

Each log entry must contain:

Field	Description
UID	The unique user ID (use **getuid**()).
PID	The process ID performing the action (use **getpid**()).
Filename	**Absolute** path of the accessed file.
Date	The date when the event occurred (**UTC**).
Time	The exact time (**UTC**).

Operation	Type of file operation: 0 = created, 1 = opened, 2 = written, 3 = closed.
Denied flag	1 if the operation was denied (no access privileges), 0 otherwise.
File hash	**SHA-256** hash of the file contents (use OpenSSL EVP).

Events to Log:

- File creation (fopen with mode "w", "w+", "a", etc. on a non-existent file)
- File opening (existing file opened)
- File modification (fwrite)
- File closing (fclose)

Hint: To distinguish creation vs. opening, check if the file existed before fopen().
Use stat() to determine file existence.

Implementation Constraints:

1. The log file must be opened in append mode to prevent overwriting.
2. If the user tries to access a file without permission, record the denied flag but continue execution gracefully.

Step 2: Audit Log Analyzer

Develop a command-line tool named audit_monitor.c, which will analyze the log generated by your audit_logger.so.

This tool should provide two main features based on command-line arguments:

1. Detect suspicious users

```
./audit_monitor -s
```

Parse the log file and print all users (UIDs) who attempted to access **more than 5 distinct files** where the action was denied.

2. Analyze file activity

```
./audit_monitor -i <filename>
```

For a given filename:

- Print all users who accessed it.
- For each user, count how many times they modified it (based on fwrite() or differing hash values).
- Display how many unique modifications occurred.

Step 3: Testing the Audit System

Create a test program named test_audit.c to demonstrate your implementation.

The program should:

1. Create several files.
2. Open and modify them under various conditions.
3. Attempt to open files without permissions to generate denied actions.

Run with the audit_logger.so preloaded:

```
LD_PRELOAD=./audit_logger.so ./test_audit
```

This should generate a populated access_audit.log.

Then, use your analyzer:

```
./audit_monitor -s
./audit_monitor -i example.txt
```

Tool Specifications

Tool	Description
audit_logger.so	Intercepts fopen, fwrite, fclose and logs events.
audit_monitor.c	Analyzes the generated log file.
test_audit.c	Generates file operations for testing.

Notes

1. This assignment is to be completed in pairs. Each student will be orally examined to demonstrate their understanding and implementation.
2. Do not skip this assignment. It may form the foundation for an upcoming exercise and will be required as a prerequisite.
3. If the Audit Log Monitoring tool is executed without a valid option or with incorrect arguments, it must display an appropriate help or error message.
4. You must provide a Makefile to compile your shared library and programs. You may either create your own or adapt the example provided.
5. A starter corpus is included to help you begin your implementation. You are free to modify or replace it entirely.
6. You must also include a README file containing:
    - Your full names
    - Your student IDs/logins
    - A short description summarizing your implementation approach

Penalty: A missing README file, or a README file without all the required information, will result in an automatic deduction of 1 point.

Submission Requirements

You must submit the following 6 files:

- README
- Makefile
- audit_logger.c
- audit_logger.so

- audit_monitor.c
- test_audit.c

All files should be placed in a single folder named:

```
<AM1>_<AM2>_assign3
```

where <AMX> corresponds to each person's student ID. Concatenate your IDs with an underscore.

Example:

```
2020123456_2020123666_assign3/
```

Compress this folder into a single .zip file:

```
2020123456_2020123666_assign3.zip
```

and submit that archive.

Implementation Notes

The provided code corpus serves only as an example. You are free to:

- Modify function definitions and add helper functions (except for the overridden fopen(), fwrite(), and fclose() signatures, which must remain intact).
- Ensure your implementation supports the command-line options and outputs defined in the Tool Specification section.

Questions and Support

For any questions or clarifications, please use the "Συζήτηση" (Discussion) tab on the course platform. Instructor responses will be posted on Monday, Wednesday, and Friday.