Audit Framework for System Monitoring

[Your Name]

June 18, 2025

Abstract

This project presents an Audit Framework developed in C for monitoring system activities in a Linux-based environment (WSL). The framework tracks user sessions, file system changes, running processes, and executed commands, logging all activities to timestamped files. Using Linux system calls like `fork`, `execl`, `inotify`, and the `/proc` filesystem, the framework provides real-time auditing capabilities. This report outlines the projects objectives, methodology, implementation, results, challenges, and future improvements.

1 Introduction

The Audit Framework is a lightweight, modular tool designed to monitor and log system activities for auditing purposes. Implemented in C, it leverages Linux kernel features to track user sessions, file changes, running processes, and bash commands. The framework is particularly suited for system administrators needing to audit user behavior in a Linux environment, such as WSL. This report details the projects objectives, implementation, results, and potential enhancements.

2 Objectives

The project aims to:

- Develop a framework that spawns independent processes for monitoring system activities.
- Log user session start and end events with timestamps and usernames.
- Monitor file creation, modification, and deletion in a specified directory.
- Track running processes using the `/proc` filesystem.
- Log commands executed in the bash shell by monitoring the bash history file.

- Ensure robust error handling and continuous operation until interrupted.

3 Methodology

The framework is implemented using C and Linux system programming techniques. Key methods include:

- **Process Management**: Using `fork()` and `execl()` for creating and executing child processes.
- **File Monitoring**: Employing the `inotify` API for real-time filesystem event detection.
- **Process Monitoring**: Traversing the `/proc` filesystem to extract process information.
- **Command Tracking**: Monitoring the bash history file for new commands.
- **Signal Handling**: Catching termination signals for graceful logging.
- **Logging**: Writing events to files in a `logs/` directory with timestamps.

The framework comprises five C programs: `main.c`, `session.c`, `file_watch.c`, `process_watch.c` and `command_logger.c`.

4 Implementation

4.1 System Architecture

The framework uses a parent-child process model:

- The main process (`main.c`) forks four child processes, each running a separate monitoring program.
- Each child operates independently, writing logs to dedicated files in `logs/`.
- The parent process remains active to keep the framework running.

4.2 Component Details

1. **Main Program (`main.c`)**:
    - Creates child processes for `session`, `file_watch`, `process_watch`, and `command_logger` using `fork()`.
    - Executes programs with `execl()` and prints their PIDs.
    - Maintains an infinite loop to keep the framework alive.

2. **Session Logger (`session.c`)**:
    - Logs session start and end events to `logs/session.log`.
    - Uses `getenv("USER")` for usernames and `signal()` for handling `SIGINT` and `SIGTERM`.
    - Suspends with `pause()` until interrupted.
3. **File Monitor (`file_watch.c`)**:
    - Uses `inotify` to monitor a directory for `IN_CREATE`, `IN_MODIFY`, and `IN_DELETE` events.
    - Logs events to `logs/file_access.log`.
4. **Process Monitor (`process_watch.c`)**:
    - Scans `/proc` every 10 seconds for numeric directories (PIDs).
    - Reads `/proc/<PID>/comm` for process names and logs to `logs/process.log`.
5. **Command Logger (`command_logger.c`)**:
    - Monitors `.bash_history` every 2 seconds for new commands.
    - Logs commands to `logs/command.log` based on file size changes.

4.3 Development Environment

- **OS**: WSL (Ubuntu).
- **Compiler**: GCC.
- **Libraries**: Standard C libraries (`stdio.h`, `stdlib.h`, `unistd.h`, `sys/inotify.h`, etc.).
- **Log Directory**: `logs/` created manually.

5 Results

The framework was compiled and tested successfully:

- **Compilation**:

```
gcc -o main main.c
gcc -o session session.c
gcc -o file_watch file_watch.c
```

```
gcc -o process_watch process_watch.c
gcc -o command_logger command_logger.c
```

- **Execution**: Ran `./main` with a directory path for `file_watch`.
- **Logs Generated**:
  - `session.log`: Session start and end events.
  - `file_access.log`: File creation, modification, deletion events.
  - `process.log`: Running processes with PIDs and names.
  - `command.log`: Bash commands from `.bash_history`.

Sample Log Entries:

- `session.log`: `[Wed Jun 18 00:10:00 2025] User session started: mohammed`
- `file_access.log`: `Created: test.txt`
- `process.log`: `[Wed Jun 18 00:10:00 2025] Process running: bash (PID 1234)`
- `command.log`: `[Wed Jun 18 00:10:02 2025] Command executed: ls -la`

The framework operated continuously until interrupted, with logs written in real-time.

6 Challenges and Solutions

- **Challenge**: Ensuring immediate log writes.
  - **Solution**: Used `fflush()` after log writes.
- **Challenge**: Handling file watch directory input.
  - **Solution**: Required command-line argument, but a default path could improve usability.
- **Challenge**: Detecting new commands in `.bash_history`.
  - **Solution**: Tracked file size changes to read new lines.
- **Challenge**: Graceful process termination.

- **Solution**: Implemented signal handlers in `session.c`, but `main.c` needs cleanup logic.

7 Future Improvements

- Add a configuration file for log paths and intervals.
- Implement `logger.c` for centralized logging.
- Enhance error handling for log directory and permissions.
- Enable `main.c` to terminate child processes on exit.
- Support cross-platform paths for broader compatibility.

8 Conclusion

The Audit Framework successfully monitors system activities, providing a robust auditing tool. It demonstrates proficiency in Linux system programming, including process management, filesystem monitoring, and signal handling. Future enhancements could improve usability and portability. This project enhanced my understanding of system-level programming and prepared me for advanced systems development.

9 References

- Linux Man Pages: `fork(2)`, `execl(3)`, `inotify(7)`, `signal(2)`, `proc(5)`.
- Beejs Guide to Unix IPC.
- Love, R. (2013). *Linux System Programming*.