Secure File System (SFS)
ECE 422 LEC B1 - Winter 2023
Reliable Secure Systems Design

Authors:
Chengxuan Li
Ajiitesh Gupta

Abstract

This project focuses on the design and implementation of a secure file system (SFS) that enables internal users to store their data on an untrusted file server while ensuring confidentiality and integrity. The SFS must support multiple users and home directories and allow internal users to control read and write permissions on files and directories. Additionally, the system must permit users to create groups and users like Unix file systems and enable users to share files with one another.

To achieve these goals, the project required a high-level architectural view, UML class diagram, UML sequence diagram, and a functional prototype implemented in any language, and we used Java. The SFS's design would aim to provide data confidentiality and integrity, preventing external users from reading file content or file and directory names in plain mode. Furthermore, the system was designed to notify the file owner immediately after logging in if the data integrity was compromised.

Overall, the project's result is a functional and secure file system that meets the project requirements as mentioned in the PDF. The SFS can provide a reliable and safe storage solution for internal users to store their sensitive data, protecting it from unauthorized access, tampering, or theft.

Introduction

In today's digital age, data storage and management have become crucial for individuals and organizations alike. With the increasing reliance on cloud-based storage solutions, the need for secure file storage (SFS) has become more important than ever. In particular, there is a need for a file system that allows multiple internal users to store their data on an untrusted file server while maintaining confidentiality and integrity.

To address this need, we have designed and implemented a Secure File System (SFS) that provides encryption of file names and contents, access control through permissions, and notifications to users when files have been modified or corrupted. The SFS is designed to provide a high level of security to internal users while still allowing them to easily access and share their files.

Our SFS allows for the creation of groups and users, supports directories and home directories, and enables internal users to set permissions on files and directories. Additionally, external users are prevented from modifying or corrupting files without detection and are unable to access file content, names, or directory names in plain mode.

In this project, we present the high-level architectural view, UML class diagram, and UML sequence diagram of our SFS. We have implemented our SFS using a combination of server

and client-side programming, with the client providing a minimal shell environment that allows users to perform the operations described in the requirements.

Overall, our SFS provides a functional and secure file system implementation that meets the requirements of secure file storage in a cloud-based environment. We believe that our SFS can be useful for individuals and organizations that require a secure file storage solution while still maintaining the ease of access and sharing of files.

Technologies

The language which we used to program the secure file system (SFS) was Java. We used GitHub to collaborate and to keep track of the progress we made during the development. Discord was used as a mode of communication between the group members. Finally, diagrams.net was used to create the UML Class Diagram, UML Sequence Diagram, Use Case Sequence, a high level architectural view diagram. Finally, we also made sure we had an outline and a detailed diagram using a notepad.

We used Java because it had a lot of resources available online, so it made it easier for us to work with that language. GitHub was used because it is a very convenient tool to collaborate, making us very efficient. Discord was used by us as we could call, and even share our screens during those calls, which was very convenient and the best mode of communication. We used diagrams.net to make all the diagrams, because it could be linked with GitHub, thus preferring it over Lucidchart.

Design Artifacts

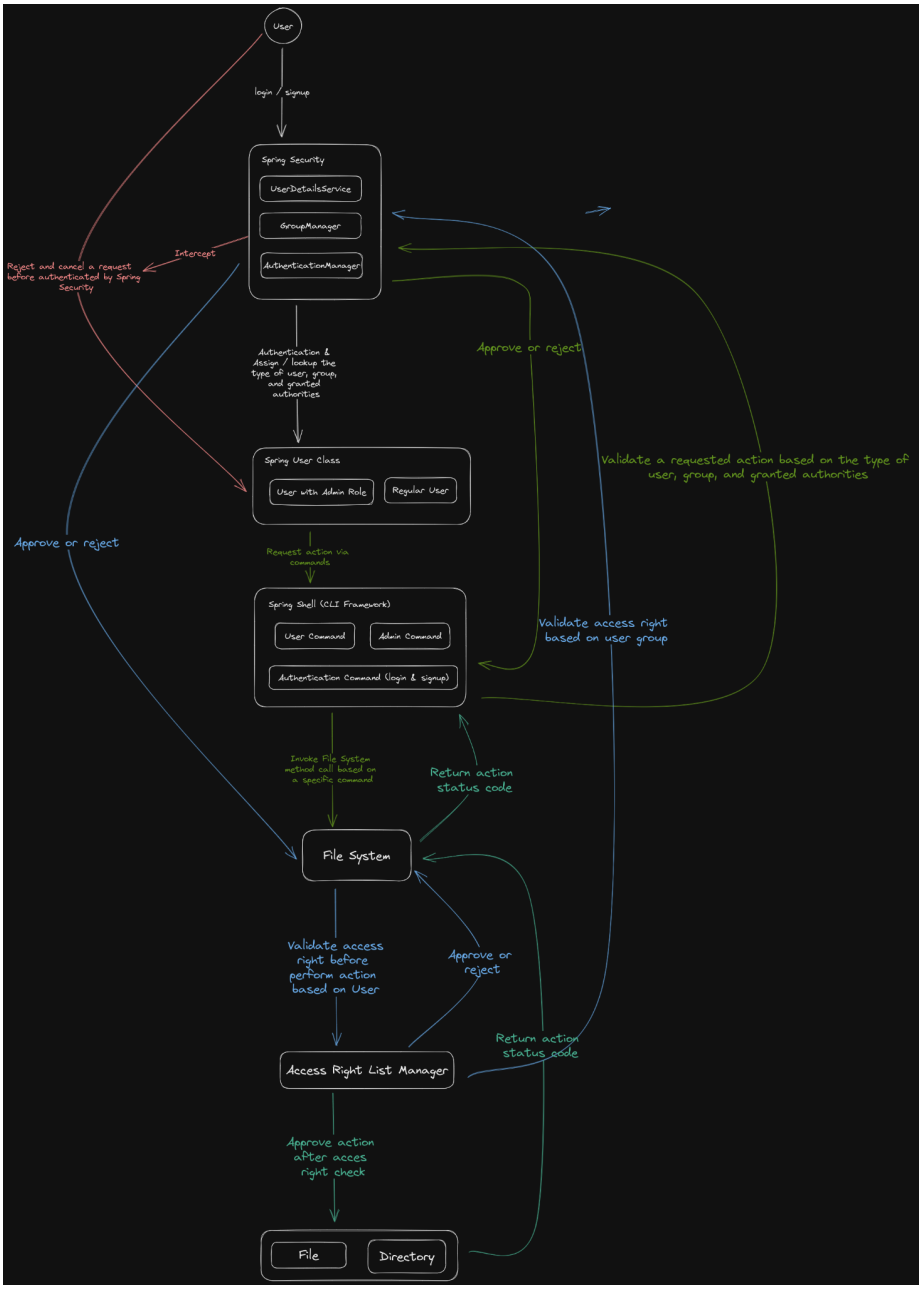

Figure 1: High Level Diagram

As can be seen in the image above (Figure 1), which represents the High Level Diagram of the secure file system (SFS), we can see what the basic structure of the SFS is going to look like.

The design of the SFS is pretty straightforward with the first step being the user signing up or logging in, depending on their account already being present or not, or by rejecting and canceling a request before being authenticated by Spring User class. Let's say the user logs in or signs up for the SFS, then the user interacts with the Spring Security class, following which

the user gets authenticated and is either assigned to a group, or looks up the kind of groups, and the users, and the authorities which are granted to the users, which again takes them to the Spring User class, where the user has either got an admin role, or is just a regular user.

Once the user has interacted with the Spring User class, the user requests an action via commands, such as reading/writing, etc, and interacts with the Spring Shell (CLI Framework), which contains User Command, Admin Command, or Authentication Command (login & signup).

Following this, we go to the File System by invoking the File System method call based on a specific command, and then the access is validated right before performing action based on the user, which takes us to the Access Right List Manager. Once the action after access right is checked, we go to check for the file and/or directory, which returns the action status code, and goes back to the File System.

There is a lot of backtracking happening, where there is a backward cycle from Access Right List Manager to the Spring Security class, and the Spring Shell to the Spring Security class as well. Based on the fact if the Access Right List Manager is either approved or rejected, it is also backtracked to the File System.

We made a lot of diagrams, such as the UML Class Diagram, UML Sequence Diagram, a high level architectural view diagram, and a Use Case Diagram, which are also included in the submission.

Deployment Instructions/ Setting up SFS

There are quite a few steps which need to be followed to set up our SFS.

1. We need to choose a suitable Operating System and an environment for the secure file server (SFS) to be hosted, and for that, we are choosing Cybera.
2. We also need to install the required software components on the file server.
3. We need to configure the file server to produce secure access to the SFS.
4. We need to create user accounts and set up the permissions on the SFS.
5. Following all these steps, we need to test out the SFS and see if it is working in the desired manner, and then regularly update and maintain the SFS to make sure our server remains secure and functional over time.

User Guide for SFS

Here is the user guys for the secure file system (SFS):

1. **Creating an Account**
    a. An account with a unique username and password must be created to use the secure file system (SFS).
    b. The system administrator can also create groups, and add users to the groups, such as adding 2 users to group 1, and a third user to group 2.

    c. Once an account is created, the user must be able to log in to the file system using the credentials they created.
2. **Uploading Files**
    a. Once the user logs in, they can start uploading files to the system.
    b. The user also has the option to create directories and subdirectories to organize their files.
    c. The user should set the correct permissions for each file or directory so that only authorized users can access or modify them.
3. **Sharing Files**
    a. These users can share files with other users in the system.
    b. To share a file, the user needs to grant read and/or write permissions to the user or group.
    c. Only users with the appropriate permissions will be able to access the file.
    d. Users from one group will not be able to access the other group's directories because they belong to another group.
4. **Modifying Files**
    a. To modify a file, a user has to have write permissions to change the contents of that file.
    b. The user can then make the necessary changes to the file and save it.
    c. External users may however be able to access the user's files and directories, but they will be encrypted.
    d. If an external user makes changes to any of the user's files, the user is notified once they log back in, notifying them about the changes made to their files.
5. **Deleting Files**
    a. To delete a file, a user must navigate to the directory where the file is located and delete the file.
    b. The user needs to have the appropriate permissions to delete a file, and confirm deleting it.
6. **Logging Out**
    a. When you are finished using the secure file system, the user needs to log out of their account.
    b. This is going to help in securing your account in the SFS, thus making sure that no one else can access or modify your files.

Conclusion

The design and implementation of a secure file system (SFS) that meets the security requirements of internal users who need to store their data on an untrusted file server is a very important requirement and is in great demand in today's world. This project has successfully designed and implemented an SFS that provides confidentiality for file names and contents, checks file integrity to detect any corruption attempts, and allows sharing of files among internal users while notifying the internal users if any changes were made to their files by external users. The SFS is easy to use and deploy, making it a practical solution for organizations and individuals who need to store their data securely.

References

- Secure File System Project.pdf from eclass
- https://docs.oracle.com/en/java/
- Demo Marking Guide.pdf from eclass