Secure File System

ECE 422

Tobi Ogunleye (1636868)
Ren Wen Lim (1593841)

Abstract

This paper outlines a Python-based Secure File System (SFS) that incorporates encryption and tamper-detection to allow its users to store and retrieve data on an untrusted file server, whilst maintaining a level of confidentiality.

Introduction

With cloud-based storage becoming ever cheaper and more prevalent throughout the world, an increasing number of organizations are looking to store their data on the cloud. Storing data on external servers however, especially third-party ones, brings about a whole host of security and reliability issues pertaining to user access, data confidentiality and data integrity. Deploying an additional layer of software that handles user authentication and rights, as well as data encryption and verification, is therefore needed to resolve many of these issues. The SFS outlined in this paper is one such form of this software, and is meant to explore these concepts in an introductory fashion.

Design - Methodology and Technologies

For this project we decided to use Python, we chose this over an arguably better suited object oriented programming language like Java due to our familiarity with it. Python also works seamlessly with MongoDB, our database storage of choice, and also allows us the flexibility of explicitly defining types. Python is also fairly lenient with string and data structure manipulations which was an added bonus.

Encryption and Permissions

To ensure confidentiality in our design, the SFS encrypts all file and directory names, as well as file contents with the DES encryption algorithm, using Python's pyDES library. The names and contents are only decrypted when the SFS has verified that the requesting user has permissions to view the unencrypted contents. We acknowledge that, as was mentioned in the lectures, DES encryption is inferior to the AES standard and that an actual production file system would need to utilize stronger encryption standards. However, DES encryption was easier to implement and was sufficient for our purposes in this coursework setting. Besides this, the client also utilizes Python's

hashlib library to hash user credentials before transmission to the SFS, such an attacker would not be able to view any plaintext passwords, either in transmission or in the SFS's database in the event of a breach.

To ensure file integrity in our design, the SFS stores the DES-encrypted form of each file's SHA256 hash alongside the file; This hashing is carried out with Python's hashlib library. When a returning user logs in, the SFS recomputes each file's SHA256 hash and compares it to the decrypted form of their previously saved hash. A matching hash indicates the file is unchanged, whereas files with mismatched hashes will be flagged to the user. This method ensures that, without knowing the encryption key, an attacker would not be able to edit user data at rest without the user knowing nor would they be able to bypass the integrity checks by simply editing the saved SHA256 hash themselves.

File Storage and FileSystems

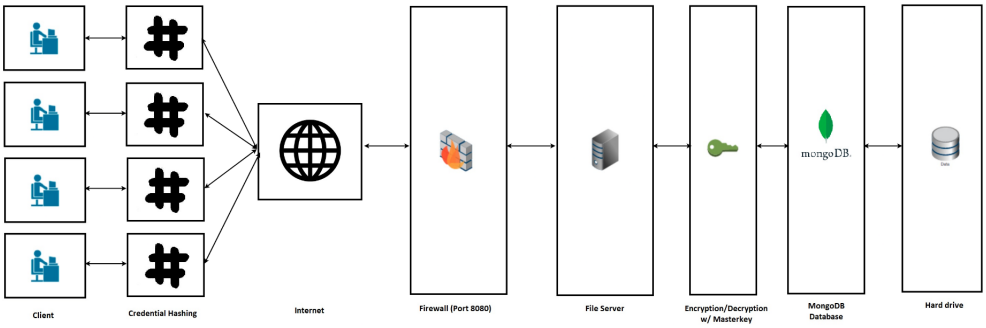
Every registered user has a filesystem, which is a class that contains all information, files and directories that belong to the user. Most of the terminal commands are methods in the file system; it acts as the center of the project as it connects to the other classes in use, such as the User, Component, Directory, File and AccessList classes. Once a user logs out of their account, the current state of their filesystem instance is saved, including all the variables associated with it, to a MongoDB database.

The decision to store the entire instance of the FileSystem class was made because rather than having to do individual database calls to insert and query for individual files or directories, we could save the entire state of the FileSystem and so everything (including the current working directory) the user had changed is preserved. This also makes it easy to check if the filesystem and the files within it had been tampered with.

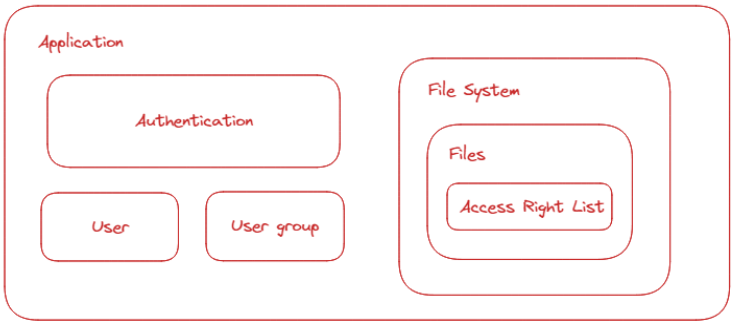
We decided to use mongodb as our backend as its python library provided simple and straightforward database queries and inserts. It also helped as it was able to store python objects as binary data, which became useful as we were able to store entire file systems instead of individual files.

Our pymongo database was also responsible for storing the usernames and the hashed passwords, which were never unhashed, so we compare the passwords hash-to-hash. It was also responsible for storing the group data, as that needed to persist outside the scope of any single user's filesystem.

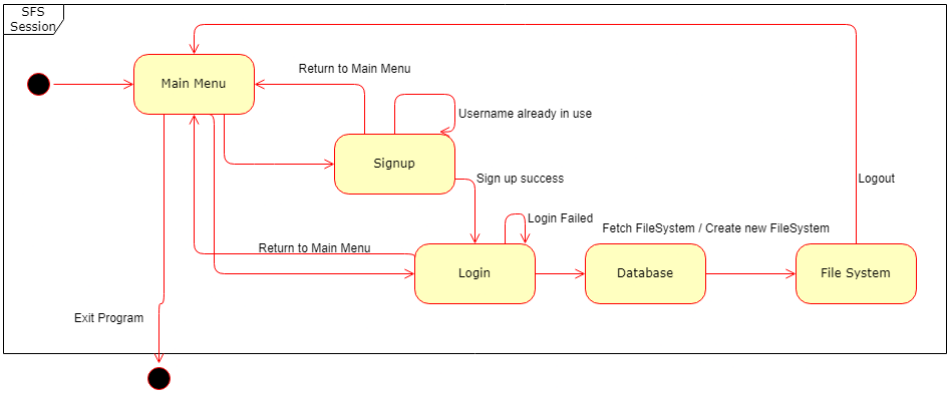
Design Artifacts



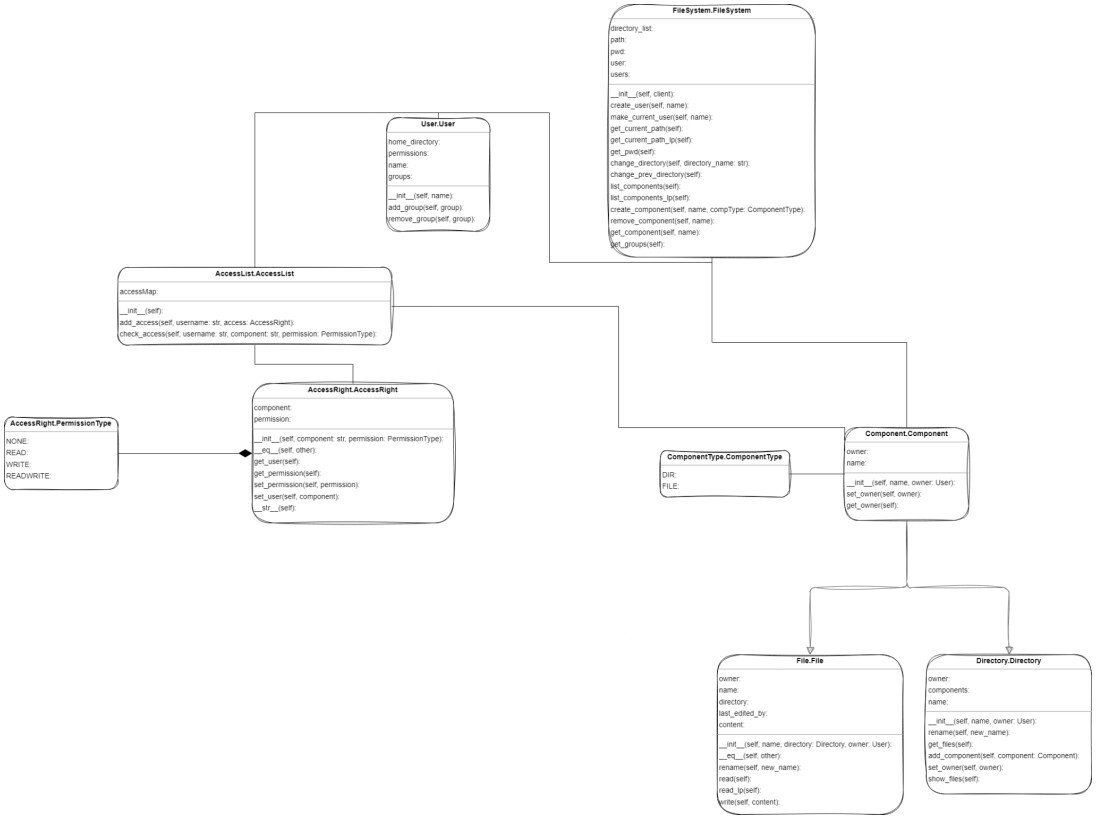
This is a high level architecture diagram that showcases the different segments of security and access in the project as well as the method of storage.



This is just a rough diagram we used initially to structure the overall project.

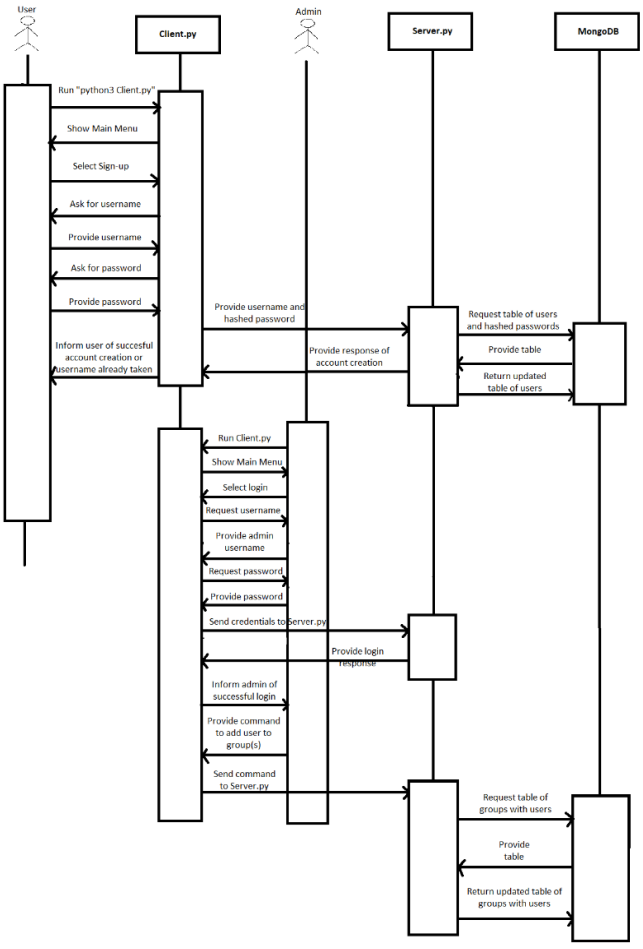


This is a simple state diagram that shows the high-level loop that a user will go through. Users are essentially looped through the main menu, as they are able to logout or return to the main menu at all stages.



The UML diagram shows that the FileSystem class is the center of all operations relating to a user's data. The FileSystem contains instances of the User, Component, Directory and File classes and makes use of their methods - hence the association relationship. The component class is the superclass for both files and directories, this

was done because the home directory, or any other directory for that matter is capable of creating either files or directories. So for simplicity, especially in functions that could take a file or directory as a parameter, we can just use a component, and then we can distinguish between the two by simply checking the ComponentType (which is instantiated in the Component class, hence the association). We also have the AccessList, which hosts the dictionary that stores the files each user has access to and their permissions, it also makes use of instances of the AccessRight class which is essentially a data type that holds a filename and a PermissionType. The PermissionType is dependent on the AccessRight class as without it it does not exist. The AccessList class is also associated with both the User and Component classes as it interfaces with instances of those classes.



This UML Sequence diagram outlines the steps taken for a user to create an account, and for an admin to then add that user to particular groups.

Deployment Instructions

The community edition of MongoDB should first be installed onto the Cybera Rapid Access Cloud (RAC)-based machine with these commands:

“wget -qO - https://www.mongodb.org/static/pgp/server-6.0.asc | sudo apt-key add -”

“echo "deb [arch=amd64,arm64] https://repo.mongodb.org/apt/ubuntu focal/mongodb-org/6.0 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-6.0.list”

“sudo apt-get update”

“sudo apt-get install -y mongodb-org”

“sudo systemctl start mongod”

The necessary Python libraries should then be installed with these commands:

“python3 -m pip install pymongo”

“pip install pyDES”

Afterwards, the SFS code should be transferred onto the computer.

With the necessary Python files:

-AccessList.py

-AccessRight.py

-Component.py

-ComponentType.py

-Directory.py

-File.py

-FileSystem.py

-Group.py

-Server.py

-User.py

-Main.py

In the same folder, the SFS can then be started with the command “python3 Server.py”

Alternatively, the previous two steps can be replaced by compiling the SFS code into an executable with the Pyinstaller library and the command “pyinstaller --onefile Server.py”. The output executable, Server, can then be transferred and executed on the file server.

Similarly, the client can then be executed on the user’s computer either using the command “python Client.py” or by building it into an executable, from which the user can begin accessing the SFS.

User Guide

Note: Users will need to have MongoDB installed on their machine, as well as pyDes.

From the client shell, the user will be prompted with a main menu, where they can log into an existing account, sign up as a new user or exit the program. Once a user is successfully logged into the SFS, they are given an input prompt to enter a command. Below is a list of valid commands:

“ls” - to list all of the folders and files in that directory.
“pwd” - print path of current working directory
“cd” <directory> to change the present working directory to the specified directory.
“mk” <directory> - to create a new directory, which will be a subdirectory of the present working directory.
“cr <filename>” - to create a new file in the current directory with the specified filename.
“rm <component>” - to delete a directory or file in the directory with the specified filename.
“rd <filename>” - to read the specified file in the directory.
“wr <filename> <data>” - to write to the specified file in the directory with the relevant data.
“rn <filename> <newfilename>” - to rename the specified file file in the directory to the newly given name.
“shg” - to list all of the groups the user is in
“lg” - to logout of the SFS.

Additionally, administrators will be able to use an additional set of commands:

“crg <newgroupname> <users>” - to create a new group with the specified list of users.
“dlg <groupname>” - to delete a specified group.
“upg <newgroupname> <users>” - to update an existing group with a new set of users.

Users in the same group will be allowed to access another user’s home directory and their filesystem. However, unless they have the appropriate permissions for specific files and directories, they are only able to see things in an encrypted format.

Additionally some notes on operation, writing to a pre-existing file will override all the previous contents of the file. Additionally, the external user would have to make malicious changes to the file through the mongodb database, which can be accessed through the host machine’s local instance of MongoDB.

Conclusion

This is a secure file system (SFS) written in python based on server-client connection, that allows its users to store data on an untrusted file server. External users can see internal users’ files and directories, but the SFS files’ name and content appear encrypted. The integrity of each user’s files and their content are checked upon login and the user will be alerted of any malicious modifications. Additionally, users of the same group are able to look through each other's file systems, while still maintaining confidentiality (shows as encrypted) unless permission for a specific file or directory is given by the owner.

References

1. https://realpython.com/python-sockets/
2. https://pymongo.readthedocs.io/en/stable/index.html
3. https://www.mongodb.com/docs/manual/tutorial/install-mongodb-on-ubuntu/
4. https://docs.python.org/3/library/hashlib.html