CMSC 626 PROJECT REPORT

TOPIC - DISTRIBUTED FILE SYSTEM WITH ENCRYPTION

Nishi Ajmera (JU61134), Saikrishna Dugyala (GU24496), Mansi Patel (BV96417), Kaushik Allu Suresh (KA41131), Shankar Sharma (QZ51136)

ABSTRACT:

In this project, a secure and encrypted distributed file system has been developed using the Python programming language. This file system allows us to share files with multiple users while making sure that all the performed operations are secure and done by authorized users. The file system is also encrypted, meaning that any data entered will be encrypted and stored in the files. Authorized users can create, delete, read, write, and rename files as well. The DFS also allows concurrent users to perform operations while guaranteeing the ACID (Atomicity, Consistency, Isolation, and Durability) properties.

INTRODUCTION:

Our distributed file system supports the same basic set of file I/O functionalities as any file system. Users can create, read, write, and delete files, and rename the files as well. They can also set permissions and share files and directories with other authorized users. To provide security and confidentiality, the file system encrypts all filenames, directory names and file content, shielding them from unauthorized users and attackers. Additionally, any unauthorized changes made to files or directories will be detected by the system and reported to the authorized users of the modified files and directories.

This distributed file system consists of one main server, and two other replication servers. Each of these servers can detect any of the changes mentioned above being made. The servers can only detect the changes made and cannot be used to perform any operations on a file (read [content and name], write, delete, rename) nor does it have any kind of access to the files. The servers also will not be able to take data from one file and supply it in response to a client reading a different file.

One or more users can log in to the file system at the same time and perform any authorized and non-conflicting operations. Each user will have to login using an ID and password, which liberates unauthorized access up until some level. The file names and the file contents are all encrypted using the cryptography library present in Python. This library is one of the many libraries and modules that Python offers for secure communication and encryption. The fernet module of the library provides a built-in key generator and provides encryption and decryption functions for a string of data and large files. The AES algorithm is used by this crypto library.

This prevents unauthorized eyes from reading the file names and contents. This also protects the files and their contents from the servers, and if a malicious server exists, it will not be privy to those contents.

MOTIVATION:

Recently, the trust placed by us in the government and in big tech companies has been shaken by privacy breaches that have been brought forth. Not only has this act increased the need for anonymity guaranteeing systems such as TOR but has also increased the need and requirement to encrypt data in places where having full control is not an option. One of these places is remote file storage. The use of storage services such as 'clouds' has become very popular in the last decade or so.

There are several advantages to using these cloud services. Additional storage space is granted, a data backup location is available, and easy file sharing is also a welcome and given bonus. Most of these cloud storage services are based on a centralized file system, i.e., Drive, SkyDrive, and Dropbox. However, according to the recent findings from National Security Agency (NSA), it is obvious that the entity which controls the central server does still hold the ability to decrypt the files which are being utilized and sorted by the users.

This was the main motivation for us to design this file system which through the use of unique permission settings and encryption features, omits the requirement of the users trusting the central server, as the server is not privy to any information being stored in the file system. This also lifts the probability of a malicious server trying to steal information or files and hand it over to a particular user, authorized or unauthorized.

THREAT MODEL:

Malicious file server -

We have assumed that the server can modify, read, and delete files that are stored on it. We are also assuming that attackers can send arbitrary messages to the server to carry out such malicious operations without being detected.

Malicious user -

An unauthorized user can attempt to access the files and perform changes to them, or an authorized user can attempt to perform actions on a file without having the required permissions to do so. The ID and password required login feature takes care of this to some extent.

Eavesdropping attackers -

We also assume that there are attackers eavesdropping on the network to which the servers and users are connected to. Such attackers can carry out replay attacks and can intercept any network package that users exchange. Under such conditions, it is important for the system not to reveal any private information over the network, and to always encrypt any data being sent or received.

FILE SYSTEM DESIGN -

The code files for the servers, clients, storage database, database checking, the file lock and unlock functions have all been coded in Python. The client and server communicate using a JSON-based protocol over regular TCP sockets. The server provides remote procedures to store and load data, and the client never communicates secret details (such as filenames or file contents) to the server in an unencrypted way.

The following modules have been created: three Servers, a directory service, and the locking service. Our DFS system can attend up to 10000 parallel requests at a time. Every file will have a total of 3 copies being stored, one on each of the servers. Permissions such as read-only, read-write and restricted have been added and based on the permissions of the file, the authorized clients can create, read, write, delete, and rename the files stored in the system. When files are created with the read_only and read_write permissions, the user who creates those files will be able to give other users permissions to access or edit the file, but when a file is create with the restricted permission, the file owner will not be able to grant any access to the file to another user. These permissions will be displayed in the client functions menu.

In order to run the system, the first thing to be done is to start the three servers. The next file to run would be the directory service file, and then the locking service file using the python command "python <filename>.py". After this, the client file needs to be run by different users on different machines. The user will be presented with a menu which will display the services offered by the file system and he/she can proceed with a choice of their own. The outline of this menu has been shown below:

- <list> - List all existing files
- <create> [filename] [permission]- Create the file
- <write> [Filename] - Write text to a file
- <read> [Filename] - Read from a file
- <rename> [oldfilename] [newfilename] - Rename the file
- <delete> [filename] - Delete the file
-<CDIR> [Dirname] – create a directory
-<CHDIR> [Dirname] – change to another directory
-<RDIR> [oldDirname] [newDirname] – rename a directory
- <quit> - Quit from the application

The file locking service has been implemented, using which we can manage the file versions when multiple clients are accessing the same file. For example, if two clients are requesting to write content to the same file, whichever user's request comes first will be considered and the system will lock that file until that user has finished writing content to that file. After completion, the system will unlock the file and then the second client can access it and perform the desired actions. The user ID and password required login feature has been added as well to prevent unauthorized access. To store the usernames and passwords for this feature, the csv file named "logins.csv" is being used.
The servers contained in our file system are the name server and the file servers. Here, three servers are the file servers, and the directory service is the name server. The directory service is used to keep track of the files and to also keep track of the servers in which the copies of those files are being stored, along with the version of the files. To store the data of the files, the "filemappings.txt" file is being used.

The directories and their features employed in this project are not user based. For example, two different users can create directories with the same name, and instead of creating two separate directories with the same name in the server, only one directory will be created, and both the user's files will be saved in that same directory. Also, as an obvious safety feature, one user will not be allowed to view the other user's files present within the same directory.

FUNCTIONS AVAILABLE TO USERS IN THE FILE SYSTEM:

1> List:

This function is used displaying the files currently being stored in the file database. This is a convenient feature for viewing the files present in the system and also reduces the risk of two files having the same name since the user can view the existing file names and not create another file with the same name.
Syntax - <list>

2> Create:

This function is used for creating a file to be stored in the database. While creating the file, the user needs to specify the file name and the permissions associated with the file.
Syntax - <create> [filename] [permission]

3> Write:

The user can write content to an existing file using this function, and this can be done only if that file is given the read-write permission.
Syntax - <write> [Filename]

4> Read:

This function is for reading the content present in an existing file in the database.
Syntax - <read> [Filename]

5> Rename:

This function can be used for renaming an existing file. This function can only be performed by users who have access to a particular file. For example, a user without having the appropriate permissions to access a file will not be allowed to rename it as well.

Syntax - <rename> [oldfilename] [newfilename]

6> Delete:

This function is for deleting a file stored in the database or a directory by authorized users.

Syntax - <delete> [filename]

7> Create directory:

This function is for allowing the user to create a directory in which they can create and store files.

Syntax - <cdir> [Dirname]

8> Change Directory:

This function is used for changing the working directory from one existing directory to another.

Syntax - <chdir> [Dirname]

9> Rename Directory:

This function is for allowing the user to change the name of an existing directory.

Syntax - <rdir> [oldDirname] [newDirname]

PERMISSIONS AVAILABLE TO USERS IN THE FILE SYSTEM:

1> Read_only:

This permission denotes that the file can be inserted with data only at the time of creation, and once the file has been created and saved with data, when it is accessed later, it can only be read from. The content of the file cannot be changed anytime in the future.

2> Read_write:

This means that a file with this permission can be read and edited even after the creation of the file. The file contents can be edited anytime in the future as well.

3> Restricted:

When a file is created with this permission, it means that only the person who can access or edit the file is the user who created the file itself, and none of the other users can read or write to that particular file.

MALICIOUS SERVER

The malicious server detection is being performed by running a daemon thread for every 45 seconds. What this means is that files present in the "filemappings.txt" file and the files present in the server do not match and this essentially shows that a malicious server is present. If any mismatches are existing, they will be reported to the master, which is the databse_checking file. A malicious server message will be displayed.

KEY ARCHITECTURE

In our system we are using one key to transmit the data, one key for encrypting the filenames and directory names. Along with this we are generating one key per user. It will be created when user credentials are created. While sharing the file with other users, we are not sharing the key with other users. if the user is permitted by the owner of the file, then system will allow him to make changes based on the permissions. We are encrypting and decrypting the data of files using the key to the owner.

CONCLUSION:

In this project, a salient and well defined encrypted distributed file system developed in the Python programming language was presented. The designed system covers all the basic functionalities of any distributed file system. The added encryption, directory and login-based features increase the security of the system, and the idea of a non-centralized file server offers protection from malicious server attacks. The encryption algorithms which encrypt file content and file names prevent attacks such as eavesdropping and network phishing by attackers and unauthorized users.