FYREBOX

FyreBox - Encrypted File System

Natnael Getahun, Michelle Johnson, David Ogutu, Willy Vasquez
https://github.com/wrv/fyrebox
December 12, 2014

Introduction

The main problem that we aim to solve with our proof of concept is that we want to be able to store our data on the server, but we do not trust the server with both the filenames and content of the files. It is important that we have a high level of data confidentiality as we are assuming that the server exists compromised. With that in mind, the server cannot read any of the files it is holding. The server still retains knowledge of usernames, the files associated with those usernames and the permissions that username has on all the files in the system.

System Architecture

Our file system is made up of 3 components: the client, the server, and a trusted cloud service. Figure 1 gives a graphical representation of our architecture.

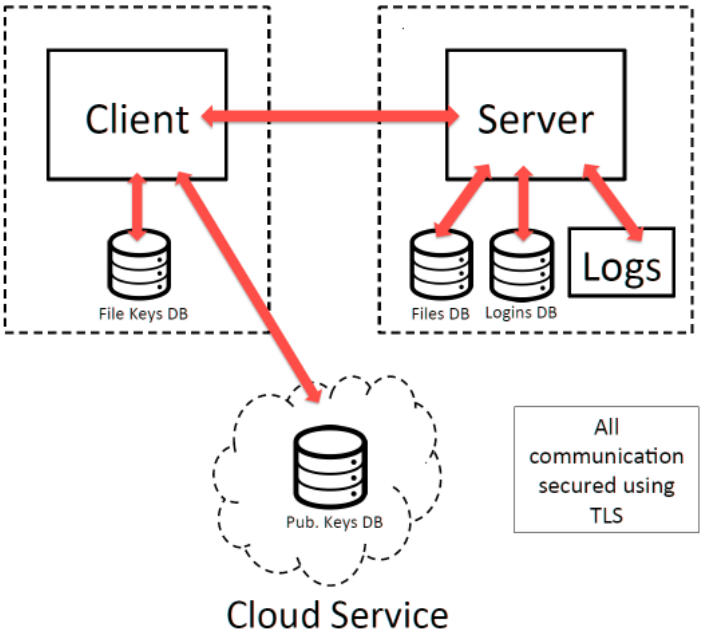

Figure 1: General representation of our file system.

Client Side

On the client side of the application there is a shell that allows the users to do basic operations such as creation, deletion, renaming, and adding of permissions to files and directories, uploading of files, and traversion of the file system. There is also a key database that stores a user's files' symmetric key, encrypted via the hash of the user's password. All encryption and decryption of files is done on the client side.

Upon user registration, a private and public key pair are generated using RSA encryption, where the public key is sent to the 3rd party cloud service. When a file or directory is created, the file or directory is first encrypted with a symmetric key and then sent to the server over a TLS connection.

Server Side

The server has a couple of different purposes. The first purpose is to sit there spinning, waiting for connections and messages. For each received message, it spins off a new thread to handle the connection. Every message that it receives and sends is logged so that no action is performed undetected.

The databases that reside on the server are the user credentials, the files database, and the permissions database. All data stored on the server side is encrypted and does not store any private keys so that in the event that the server is compromised, all user data is protected.

Cloud Service

The cloud service only stores the public keys of all the users of our application. When a user wants to share a file to other users, the other users' public keys are pulled from here.

Data Structures & Representation

Files & Directories

Both files and directories are both stored on the server's files database. What separates them from the database's point of view is the field in the table that serves as a flag for the entities. There is a parent ID field in the table to account for subdirectories and files nested within a directory. The server has no direct sense of a directory structure.

File Sharing

In order to assign read and write permissions to another user, one must obtain the other user's public key from the cloud service and use it to encrypt the symmetric key of the specified file. This gets stored in the permissions table on the server side.
When a user wants to read or write to a file, the application checks the permissions database to see if the user has the correct permissions to complete the action. In the table, there is a field that serves as a flag for which permissions are granted to the user. A value of "True" gives the user read and write permissions for a specific file. A value of "False" only gives the user read permissions. No entry in the table linking a user to a file means that the user has neither read nor write permissions to said file.

Conclusion

Challenges

Our biggest challenges were designing the database models and creating a design that made renaming files and directories easier. We wanted to limit the number of databases our application was using and keep all relationships between the tables as simple as possible. We did this by treating directories and files the same, differentiating them by a flag in one database. In order to make renaming files and directories easier, we linked the files and directories to the ID number of their parent directories so that in the event that a directory's name get's changed, we would not have to rename any of its nested files or subdirectories.

Future Work

In the future, we would like to add a freshness guarantee to all files so that no user sees remnants of old versions of files. A possible solution to guaranteeing freshness would be to use perfect forward secrecy for every write to a file. Another feature we would like to add to our file system would be portability so that users could log in from other machines. This could be done by creating a web application for our file system.