Security Functionality of the System

Authentication

A user can create an account and log in to our system. The password is hashed using SHA-256 on client side to ensure that the password is not sent in plain to the server side since that will allow the admin to decrypt the private keys of the user (which is encrypted using a hash of the password using the PBKDF2 algorithm). Given that it is hard to invert a hash of the password, this ensures that the private key will not be revealed to the server admins. The SHA-256 hashed password is then further hashed and salted for 3,000 iterations and then verified against the database. Password updating is also currently supported by the system. The user is required to be first authenticated using his/her old password before he/she is able to update the password. The private key of the user, which is encrypted using the password of the user is also updated.

In the implementation of TLS for our product, we used the cipher suite "TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256" that specifies the use of RSA as the signing algorithm to authenticate the key exchange. Asymmetric key cryptography for the key exchange is performed using the ECDHE (elliptic curve diffie-hellman key exchange). The keys sent by the server to the client are signed by the server using the server's private key. The client verifies that the signature received is valid using the server's public key that was sent with the certificate when the client starts the program. This was done in advance where the client has to manually determine whether the certificate sent by the server can be trusted by examining the certificate. In this case, the server is authenticated by its certificate. The premaster secret is also encrypted and sent by the client to the server, and it will be used to generate all the keys and IVs needed for encryption and data integrity.

Authorization

Permission is always checked before a request to the database can be processed. For example, the download of files will always first check if the user performing the action has at least view privilege to the file. The checks are performed on server side by querying the database, which ensures no access is accidentally given to a user who had his/her privileges revoked. Privileges are updated simultaneously with the creation of files/folders. This ensures that the owner of the file/folder is given edit access. Although the sql queries are done for the addition of privileges, this feature has yet to be fully integrated with the front end, and hence will only be rolled out during the next milestone.

Audit

Audit is implemented on server side by keeping track of file logs and user logs. File log currently keeps track of: (i) uploading of files; (ii) creation of folders; (iii) renaming of files/folders; and (iv) adding/removal of privileges. The log of the deletion of folders, overwriting of files, and rollback will be implemented in the following milestones. User log keeps track of the creation of new accounts, change of passwords, and failed login attempts to a valid username. However, the file logs are not yet accessible by the client through the GUI, and the admin does not have a direct means of accessing the user logs.

Confidentiality

All files/folders and file/folder names are encrypted using a symmetric secret key. Specifically, AES-CBC is used with a PKCS7 Padding and a randomly generated 128 bit key. An initialization vector is generated on the client side along with the secret key such that the encryption is unique each time. The secret keys of the files are encrypted using the user's private key before being stored into the server. This prevents the server admin from getting access to the contents of the file or even know the name of the file. We do assume, however, that the size, date modified, and viewers/editors of the files are not confidential to the server administrators as this can easily be seen from the files stored on the system as well as through SQL logs.

Passwords are hashed using SHA-256 before being sent to the server so that it is never sent in the plain to the server. It is then further hashed and salted using PBKDF2 algorithm to defend against rainbow table attacks.

Prepared statements are used to prevent SQL injection which further protects the confidentiality of the presence of files, sizes and dates modified to other users.

A commonly used secure channel protocol Transport Layer Security (TLS) is used to maintain the confidentiality of information transmitted between the server and client. The symmetric cipher suite "TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256" is used for both the client and the server side. We chose this cipher suite because it is the same cipher suite that Google web server uses and also similar to the cipher suite(TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384) that the CS 5431 System Practicum web server uses. Furthermore, the cipher suite chosen is in compliance with NIST recommendations.

AES is the symmetric encryption algorithm used to ensure that data transmitted is confidential. The algorithm used has a 128 bit key size and GCM(Galois/Counter Mode) is the mode of operation. GCM maintains a counter. For each block of data, it sends the current value of the counter through the block cipher. Then, it XOR the output of the block cipher with the plaintext to form the ciphertext.

Integrity

Principals must be authorized before they are able to make any requests to the database. This ensures that no unauthorized modifications can occur. The server admin is trusted to not modify either the files or the database, so integrity protections after actions have been committed to the database are unnecessary.

The integrity of information transmitted between the server and client is also maintained with TLS. As explained in the confidentiality section, the symmetric cipher suite "TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256" is used for both the client and the server side. SHA 256 is used for hashing to ensure that data integrity is maintained.