Design

Jackie Law (jjl284) *jjl284@cornell.edu*
Nishad Mathur (nm594) *nm594@cornell.edu*
Ning Wang (nw265) *nw265@cornell.edu*
Zhan Zhao (zz444) *zz444@cornell.edu*

March 9, 2017

1 Confidentiality

In general the confidentiality is the most complete and relevant part of security for this milestone, as the majority of the stories handled this week were focused on the user facing client and single user synchronization.

The general design currently adopted by the client is to use a master password for the system, which is used to access the per keychain passwords and the server login password. This master password is used to derive a 256bit key using Scrypt (currently with placeholder parameters) and this password then decrypts the directory.

The directory its self is encrypted using AESWrap (RFC3394) and the intent is to switch to a more modern variant of AESWrap (RFC5649) provided by the BouncyCastle libraries due to its improved padding support. AESWrap is a variant of AES designed specifically for offline key-storage and as a consequence is secure but fairly slow, but as all encryption and decryption of the directory is client side this is an acceptable trade-off.

The directory directly stores the server authentication key as well as the encryption keys and nonces for each of the keychains. The keychains are encrypted using 128bit AES-GCM (this will likely increase when we switch to bouncy-castle) and the encryption keys and nonces are rotated each time a keychain is saved to disk. Shared keychains may exhibit slightly different behavior. Each keychain key and nonce is generated using Javas strong secure random provider and is used to encrypt a serialized JSON object containing the passwords and their associated metadata. All metadata not related to a password is stored in the directory as this minimizes the amount of time that the system need keep the decrypted keychain in memory.

The server stores a combined directory entry-keychain object which stores the metadata for a keychain and the keychain its self, but there is currently no encryption employed on the server end, that will be pursued for the next milestone.

As for miscellaneous other issues, currently all passwords are stored in Java SecretKey objects, which support zeroing out and the intent is to automatically zero them out after a time has elapsed and the system requires that the user renter their login details to decrypt the password once again. Currently the system uses the Sun crypto provider (but as above) intends to switch to the bouncy castle provider or light weight API for the crypto primitives.

2 Integrity

This is maintained primarily by the two ciphers used for encryption, AES wrap and AES-GCM both of which are AEAD ciphers and integrate authentication with encryption. The system will automatically fail if the keychains have been altered.

3 Availability

This is primarily achieved through the offline first approach taken by the system, allowing read/write access to the user's keychains at all times. Server synchronization is partially implemented and will ensure that (for non-compromised servers) online clients are synchronized (the intent is eventual consistency).

4 Authentication

Currently users are authorized using their master password (the details how this is used are above). Server side authentication is not yet incorporated and is planned to be done in the next milestone.

5 Authorization

There is no authorization scheme implemented at the moment, as data is stored offline for a single user with access to the application. Strict authorization will be implemented in the next milestone, guaranteeing that one user does not get access to another user's stored information. In addition to that, we will be implementing sharing and permission in the next milestone, which require a thorough authorization scheme.