Design

Confidentiality

PassHerd provides confidentiality in two main ways. First, all communications between client and server are done in the context of a TLS session. Second, stored credentials are encrypted on the server using a key only possessed by the client. Even if an attacker were able to breach the server, they would need to acquire the client's private key in order to recover the plaintext stored credentials. This private key is generated using the AES-128 algorithm and Java's built-in SecureRandom class. It is stored in a file on the client computer. The way we are generating and using this key is not ideal for a system with multiple clients. This is because the private key would need to be passed to these other clients somehow. In future releases, we plan to use a password to replace this key, so that users can decrypt properly on any client by entering that password.

To ensure the authenticity of the server when the client starts a session, we use certificates generated by the CS5430 certificate authority. The SSL socket on the server side uses a keystore to store private keys and the certificates with the corresponding public key, while the client uses a truststore to keep track of the certificates from the CA that we use to identify the server.

Integrity

Integrity is provided through the TLS protocol and by logging of all operations affecting the stored credentials. The TLS protocol implements MAC through HMAC. Any changes made by an attacker from an authenticated client will be logged and the changes can be reverted by someone with access to the server. Additionally, the use of a private key known only to the client for encryption of stored credentials enables user detection, as any password modified from what was originally sent by the client to the server will appear as garbage to the user, not having been encrypted with the same private key.

One conceivable attack on the integrity of PassHerd is for an attacker who manages to gain access to the server to switch ciphertext passwords around such that the user sees valid ciphertexts associated with incorrect accounts. We have provided for this by encrypting client-side the concatenation for any given credential of password and service name, then

upon retrieval of that credential ensuring that the names of the service from the decrypted concatenation and the name of the service requested are the same.

Audit

Audit is implemented by keeping track of all activities on accounts, master passwords, and credentials. We have a server log ("centerlog.txt") that records creation/deletion and authentication of accounts as well as account-specific logs ("username/log.txt") for user activities such as modifying credentials and changing master password. For each activity, username, IP address, type of actions performed, timestamp and server response are recorded.

Authentication

We implemented basic authentication by storing salted and hashed passwords with the corresponding salt in a file “master.txt” under the user directory (named after the user account for this registration). The hashed password and salt is consulted for authentication in future sessions.

Authorization

General Idea: Make modules for both client and server side to handle all operations acting on objects (files, sockets, etc.). Essentially we would like to implement a reference monitor that meets the NEAT properties. (Non-bypassable, Evaluable, Always invoked, Tamper-proof).