CS 161 Project 2 Design Document

Haiming Xu and Ru Pei

Users

Initializing a User

The first step would be to efficiently store all the users of the system. Each user has a password, so we can store the user's UUID along with their hashed, salted password along with a signature or MAC in the vulnerable store.

Logging In

To log in, the user needs to enter a password, which we will hash and see if the entered password's hash is the same as the password in the vulnerable datastore. We can detect whether or not the user's password has been compromised by comparing the hashed value in the datastore with its associated MAC.

Structure

Each user needs more information than just their password. Thus, we create a class that holds on to metainformation:

1. A map that associates the encrypted filenames the user has access to and the encrypted file.
2. A map that associates the encrypted filenames the user has access to and the decryption keys.
3. A shared files pointers map which associates encrypted filenames with pointers to the shared file (note that encrypted filenames could be different than the shared filename)
4. A shared files map that associates the shared filenames with the keys needed to decrypt them.
5. A shared files map that associates the shared filenames with sets of people approved to access.

Note that every map has their values signed with a digital signature. Given the datastore is a key-value store, we can use json.Marshall to compress this data as a value and associate it with the user; we also must add a signature or MAC to ensure the contents of each user isn't changed.

Single-User File Storage

Storing Files

When a user wants to store their own file, they can create access keys. Then, the user encrypts the file and the filename and gives the files associated UUIDs. To ensure integrity, we also attach a MAC or digital signature to the file contents and filename. Finally, the user will append both pieces of data to maps 1 and 2 in the user class.

Loading Files

We take in the filename and encrypt it using the same hashing algorithm we used for storing. Then, we can retrieve the encrypted file and the decryption keys from the maps found in our user class. Then, we can simply decrypt using the decryption keys.

Appending to Files

This function can be abstracted using the store files and loading files functions. We first load the file and append the data to the end of the plaintext. We then call the store file function with the same file name, but with the updated file contents. The same security protocols used in the two functions used are also used for this function.

Sharing and Revocation

We can use public key encryption to share files. There is a trusted keystore which we can use to store the public keys of every user.

Share File

When a user wants to share a file, this function can send to the recipient the UUID of the file, the pointer to the file, and the required keys for decryption and authentication. Since the sending channel is insecure, we can use RSA encryption, taking advantage of the public keystore, as well as a signature or MAC. Finally, append the recipient to map 5.

When a person A wants to share a file they themselves received from someone else, the owner of the file will append person A concatenated with the recipients, separated by a delimited. For example, if Amy shared a file with Bob, Amy's map 5 contains Bob. If Bob shares that file with Joe and Tom, Amy's map contains Bob, Bob-Joe, and Bob-Tom. Something similar can be done for every layer of sharing. This way, when Amy revokes access to Bob, she can revoke access to every name with Bob as a prefix.

Receive File

When the recipient receives a file, they will first check if the data is accurate by confirming with the signature or MAC. If legitimate, the user can then add a name (may be different from the original filename) and file pointer to map 3 and new filename and shared file keys to map 4. Since there is a file pointer, the recipient will be editing directly to the top level copy of the file. When the recipient wants to write or view a file, the system will go to the owner's access list and checks if the recipient is on it, and if not, terminates.

Revocation

To revoke, first remove the person from map 5 along with everyone with the person's name as a prefix (see share file for details regarding when this will happen). Future file access attempts will fail the checking phase.

Security

Confidentiality and Integrity - Users

We apply digital signatures and MAC the condensed user structure so that users are aware if the malicious datastore has altered their contents.

Confidentiality and Integrity - Files

We encrypt both files and filenames to preserve confidentiality. We attach digital signatures to ensure that we are notified if the files or filenames have been altered.

Sharing Files

We have ensured that malicious attackers or revoked users cannot store or load to a shared file. We prevent Mallory-type attackers by forcing those wishing to store or load to enter their unique key before accessing. We prevent those revoked from accessing because although their unique key can decrypt the files, the final checking step (verifying the user is part of the shared list) will fail. Since everything is signed, we can be sure that the contents of the revocation list and keys have not been tampered with.