SecFS – 6.858 Final Project

Shantanu Jain
Michael T Kelessoglou
{jains, mkel}@mit.edu

Fall 2015

Problem Statement

Implement SecFS, a subset of the serialized SUNDR (Secure Untrusted Data Repository) file system, given a single-client, insecure, functional skeleton codebase. See README.md for a detailed description of SecFS's required functionality.

Symmetric Key Encryption

All non-world readable files and directories were encrypted in order to protect their contents from non-authorized users (Exercise 3). We used symmetric key encryption here achieve better performance than asymmetric encryption.

To protect the symmetric keys, they were stored encrypted with the user's private key, using RSA asymmetric encryption. One symmetric key was also created per group. For each user in the group, a copy of the group symmetric key encrypted by the user's public key was also stored. These encrypted symmetric keys were stored on the server in a special data structure called the Symmetric Key Store (SKS). This did not expose any critical information, as every entry in the SKS is encrypted, and can only be decrypted by using the user's private key. A user can only get his own symmetric key, and the symmetric key of groups he is a member of, via his private key. Thus a malicious server can at worst perform a DoS attack by overwriting the SKS with invalid data or refusing to return it.

Group Implementation

When a file is created and assigned group permissions, the most recent user to modify the file stores its blocks on the server, updates its inode to point to these blocks, adds/modifies an entry in his itable to point to the updated inode, and updates the group itable with an I that points to the entry in his itable. This means that a group file's entry may jump between different users in a group as these different users update it. Group folders are treated in the same manner.

A user attempting to access a group folder or file has his membership in the group checked from the .groups file. If he is not listed as a member there, he is denied access.

Read Protection of Files

By looking at the umask, determine if world-read permissions have been granted. If these permissions have not been granted, encrypt the file's datablocks using symmetric key encryption. Secure the symmetric key using the aforementioned SKS data structure. If a user attempts to access an encrypted file, then his read permissions on that file are equal to his write permissions on that file – the same as access.can_write().

Garbage Collection

Data Block

Whenever a data block is updated, a new entry (hash-data pair) is created on the server. We implement a special interface on the server to remove certain entries by hash, called free(). When the client updates an inode to point to a new data block, the client requests that the hash be freed, deleting that entry on the server. Because the hash space is large, a malicious authorized client could not easily remove entries from the server.

VSL

We implement server-side garbage collection of the VSL. A new VSL is created every time any file or directory in the file system changes. This leads to a lot of small used blocks on the server. We implement a special interface on the server to handle VSL retrieval and writing, getVSL() and storeVSL() in secfs-server. These methods always read and write the VSL from the same hash block on the server, overwriting the previous VSL.