NetLogFS: a diff_log based encrypted network file system

Leo Liu, Joseph Driscoll, Asa Oines, Clark Della Silva

---

1. Introduction

In today's world, many users request an external secure server to hold their files. However, many providers of such services often lack the ability to keep their file servers secure. Additionally, even when providers of file systems claim they are able to keep their file secure, users are hesitant to trust them. To remedy this situation, our team has set out to create a secure file system that can be housed on an untrusted server. To achieve this goal, many traditional file system methods need to be adjusted. In order to secure files on an untrusted server, the root user of the server must be unable to read files or modify files without detection. Furthermore, if a malicious user or server were to corrupt the data encrypted in a file, there must be some way to recover said file. In this paper, we describe NetLogFS, our implementation of a secure file system that accomplishes these goals.

1.1. Threat Model

We are assuming that root on the server is compromised but the server is not actively malicious. This means that we do not trust the server to perform checks or authentication correctly, but we do trust it to perform requested reads and writes accurately. We are also assuming that root will have access to all data on the server, so anything that is stored must be encrypted. We are also assuming that a trusted client is being used on a trusted machine, so that keys and unencrypted file names can be stored temporarily on a per session basis, which are deleted when a user logs out. There are also untrusted clients that may be attempting to gain access to information on the server using our code.

2. Design

2.1. Overview

In NetLogFS, each user has a home directory under "/" named after his username. All usernames are encrypted using a deterministic encryption. Users can only access files within their home directory, unless other users grant them access to their files by sending them permissions. Each file has two symmetric keys: a read key (RK) and a write key (WK). Every file is stored on the server using an encrypted filename obtained by encrypting the filename with its read key. The contents of the file are encrypted with the read key before they are stored on the server.

There are two types of permissions: read-only and write. To grant read-only access to a file, the owner shares the file's read key. The read key allows the recipient to determine the encrypted path of the file and to decrypt the contents of the file using symmetric encryption. The un-encrypted file will contain a digital signature and a digital signature public key that allows the user to verify the integrity of the file.

To grant write access to a file, the owner shares both the read and write key. The write key allows the recipient to gain access to a digital signature private key that can be used to generate the signature on every write. The write key also gives the recipient access to a secret number that the client must present to the server on every write attempt for authentication.

Revoking access to a file requires changing the read and write keys, re-encrypting the files, re-encrypting the filenames, and sharing the new keys only with users who should keep access to the file. Every user has a public and private key. To share a permission, the owner encrypts the appropriate read and write keys with the public key of the owner and sends the result to the server. Recipients' clients will download the permissions and decrypt them with their private key.

Finally, the write permission also gives access to a difflog. Each file has its own difflog. This difflog is encrypted with the file's write key and contains all edits to the file. The edits are signed by each user, which allows writers to ensure that their changes were included and to rebuild any revision of the file. Each logfile also contains the file secret and the digital signature private key. Directories are handled similar to files; they have read keys, write keys, and logs. However, instead of encrypting a directory with that directory's read key, we instead create a separate file call .meta_directory. This file contains all the meta data of the directory, including the digital signature of the directory, the digital signature public key, and the most recent edit number of the directory. The .meta_directory file also contains a record of all files that were added and removed to the directory, allowing us to record the creation deletion of files on the logfile of the directory.

2.2. Crypto

The client uses three kinds of crypto to handle information storage on the server; Deterministic, Symmetric, and Asymmetric. Deterministic encryption is used to obscure user names on the server. Symmetric encryption is used to encrypt all of our files on the server, and Asymmetric encryption is used to encrypt the symmetric keys on the server.

We used the module pyCrypto as the backend for our crypto functions. Our digital signature is generated using PKCS1_PSS with 2048bit keys. Symmetric Encyption uses AES in CBC mode with a block size of 16, and a 16 byte random IV. The key for AES-CBC is derived using PBKDF2. And, Asymmetric Encryption uses PKCS1_OAEP with 2048bit RSA keys.

2.3. Storage on Server

The server stores all the per-user directories under a single directory at a specified location. To hide filenames, each file and folder is stored on the server with an

encrypted filename. The per-user home directory is stored as DET(username). All other directories and files are stored with their filename encrypted by their corresponding read keys. A file with the path, '/user/dir1/dir2/file', would be stored on the server as 'baseDir + /DET('user')/enc(dir1_RK, 'dir1')/enc(dir2_RK, 'dir2')/enc(file_RK, 'file')'.

Each file has a difflog associated with it. The difflog's filename is obtained by adding '.log_' in front of the filename. The difflog for '/user/dir1/dir2/file' is '/user/dir1/dir2/.log_file'. On the server, the difflog is stored using the same encrypted filename of the corresponding file, except with a '.log_' in the last part of the filepath.

The server has four database tables: Secrets, Public_Keys, Permissions, and Passwords. Secrets contains the secret number for each encrypted filepath. Public_Keys is a table of the user public keys, which are used to encrypt permissions that a user grants to others. Permissions is a table that contains each permission, the owner's encrypted name, and the recipient's encrypted name. The Passwords table contains salted passwords.

All encryption and decryption is done on the client. The server never has access to plaintext usernames, filenames, and data.

2.4. Client-Server Interaction

The user logs in to the system with a password. After the client successfully connects to the server, it fetches all the permissions shared with a user. The client client uses the user's private key, which is stored on the client, to decrypt the permissions to get the file and folder read and write keys.

2.5. File Format

Files are stored on the server with both the metadata and contents included in the same file. The metadata includes: (1) a watermark, (2) a digital signature, (3) a digital signature public key, and (4) an edit number. The contents are the data written by the user. Both metadata and contents are concatenated and encrypted by the file's read key before they are sent to the server for storage. The file format is summarized as: sym_enc(file_read_key, watermark + signature + signature public key + contents).

2.6. Reading a File

To read a file the client needs to generate an encrypted path, retrieve the file from the server using the encrypted name, decrypt the results to obtain the plaintext metadata and contents, and then use the metadata to verify the integrity of the contents.

To generate the encrypted path, the client encrypts each component in the path separately and concatenates the result. Any non-home directories within the path are encrypted with the path read key. Thus, to read any file, a user must have read access to all the parent directories. The client then fetches the file from the server.

The client decrypts the downloaded data to obtain the metadata and the contents. It checks that the watermark is correct and verifies the digital signature using the digital signature public key. The digital signature is generated on every write and verified on every file open. To generate a signature, the client with write permission will concatenate the digital signature public key, the edit number, and the contents, hash the result, and sign it with the digital signature private key.

2.7. Difflog Format

The difflog file consists of a watermark and a difflog object. The Difflog is a class that extends list, where each element is a dif entry, specifying the change and the signature of the client how implemented that change. In addition to the list of diffs, the difflog object also contains the digital signature private key and the secret number for the file it logs. After creation, a diff entry is frozen so that the attributes can't be changed. A difflog is specified as .log_filename or .log_directory

2.8. Writing a File

To write to an existing file, the client first must open the file in write mode and download the file from the server. Once downloaded, the client decrypts the logfile using the write key of the file and from the logfile reads the secret number of the file. With this number, the client can prove that he has the write key, since he must have the write key in order to read the logfile. After editing the selected file, the client then sends his new file to the server, providing the secret number of the file.

If the file does not exist, then the client instead opens and downloads the metadatafile and logfile of the parent directory the client wants to place his new file in. The client then reads the secret number of the parent directory and uses this secret to prove that he has permissions to write to a directory. To create a file, the directory creates a read key, write key, digital signature public key, digital signature private key, and a new set of permissions for the client. The client then tells the server to create the file and gives the server a new permission to add to that file.

2.9. Directories

Directories are treated in a similar manner to files. However, instead of logging the changes to the inode map contained inside the directory, we create a separate metadatafile, named .meta_directory, that contains the additions and deletions to the directory. We then log the changes of that metafile in the logfile of the directory. The logfile of a directory and the metadata file exists in the parent directory of the directory. The name of a directory and the name of its metadatafile are encrypted with the directory's read key while the logfile of a directory is encrypted with the directory's write key.

2.10. Sharing Files

NetLogFS shares files by distributing permissions between different users. A permission is unique to the user who receives the permission. The permission consists of the encrypted path of the file on the server, the read key of the file, the write key of the file, and is encrypted using the users public key. These permissions are created by a client when they specify they wish to change permissions of a file. All permissions handed out are added to a database in the server. Users receive their permissions from the permission table by asking the server. They also store previous permissions handed to them in the client.

3. Implementation

We implemented the client and server in Python. The client and server communicate using a JSON object protocol. Crypto uses pyCyrpto as a backend, and pbkdf2 is borrowed from the lab code. Character based diffing is implemented with google's diff-match-patch library. Our client implements equivalents of the Unix system calls for files and directories. We also implemented a shell that allows the user to work with the file system using cd, ls, touch, rm, mv, mkdir, vim, and emacs.

3.1. Description of Source Code

client/api2.py → Defines the functionality for the api, essentially the Unix syscall equivalents.
client/client.py → Client shell.
client/crypt.py → The crypto api for the client, uses pycrypto as backend
client/difflog.py → Defines the difflog structure and class.
server/db.py → Functions for interacting with the databases on server.
server/server.py → The server daemon.

4. Guarantees of the System

User names, file names and folder names are treated as confidential. The server or a malicious client cannot obtain any information about the names of these files without breaking their encryption.

Assuming the server is not malicious, an unauthorized user cannot read or modify files he does not have the permissions to. Files are symmetrically encrypted, so unless the user has the read key, they cannot access the file. If a user has the read key for a file, they can still not modify the file without being detected, as the file must be signed with the private key stored in the log file, which is protected by a separate key. In addition, the original file can be reconstructed if corrupted because the log maintains enough information to rebuild the file to any saved state.

All modifications to a file can be tracked to a specific user, as each diff entry is required to be signed by a user's public key. If an unauthorized user writes to a file, the diff will have an entry with an invalid signature, so the file can be reconstructed to

a state before the unauthorized modification. In addition, entries in the diff log are frozen after creation so they cannot be modified, preventing future tampering with the diff log.

If a user has permissions revoked, they will no longer be able to access, modify or tell which file it is on the server, as when permissions are changed, the file and log are encrypted with new symmetric keys (this changes the encrypted name of the file), a new RSA key pair is generated for file signing, and a new secret number for the diff log is generated. In addition, all permissions for the file are removed and reissued with the new keys, so that only the user with revoked permissions is effected.

A server cannot supply one file in response to a request for a different one, as the users permission contains the encrypted file path and the symmetric keys. This means that the user would see that the file names would not match, and more importantly when the user attempts to decrypt the requested file, the decryption will fail letting the user know that the file is not the one requested.

Directories also maintain a log file that tracks creations and deletions of files/folders in it. Between the directory log and the file log, all changes to the system can be tracked. authorized changes are recorded in the logs, and signed by the user, and unauthorized changes would cause discrepancies between the filesystem state and log state, allowing for detection of these changes.