Distributed Systems Project Report

Oisin Cleary
13325102

Introduction

The goal of this project was to create a distributed file sharing system. I modeled my system off a github repository. As such all files are visible to all users. The exception to this is the user level system which I created. Users cannot view files which are of a higher priority than they have permission to view.(Note 1 is the highest level of priority. Increased level means decreased priority.)

Usage:

- login $username $password:
    - User cannot any other commands without logging in. It provide the client with a token used for authentication. This token also contains the user's level so the file server knows which files the users can access.
        - Note that 'admin' 'password' are provided for testing
- addUser $username $password $level:
    - Used to add other users.
        - This is not seen as a core feature so it is not secure. Users can add as many other users as the want at any level.
- listUsers:
    - Lists all users in the system.
- add $filepath:
    - Uploads file to server. If file already exists it is overwritten. If existing file has a higher level than the current user the user is told they cannot overwrite.
        - All file paths taken from the client/files directory for testing.
- list:
    - List files on the server and their paths
        - Only shows files the user has permission to see.
- get $filepath:
    - Download a file from the server
        - User can only download what they have permission to.
- lock $filepath
    - Used to lock a file. Locked files can only be written to by the user with the lock. Unlock is used to free the lock.
- unlock $filepath:
    - Used to unlock a locked file. Files can only be unlock ed by the lock holder or a user with a higher level of priority (Not the same level or lower)
        - Note in this system 1 is the highest level and increasing level numbers decrease priority

Section 1 Distributed Transparent File Access

Files on my server can be accessed by the user through my client application. Much of the detail of what is happening is hidden from the user. The user never sees the redirects or cache accesses performed by the system. The user interacts only with files. The server exists in 3 parts; the authentication server, the directory server and the file server. The authentication server manages users and their metadata. The directory server manages file metadata, caching and locking. The file server simply stores the files.

Section 2 Security Service

The user logs in by providing a username and password to the authentication server. The authentication server checks these against its database. If the details are valid the user is logged in. When the user logs in a key is generated for the user. This key is returned to the client along with an encrypted token. When the client sends information such as files or filenames to the directory or file servers it encrypts the data with the key it is given. It also includes the token as a parameter. The token is encrypted with a key known to the servers. It contains the key given to the client and the client's name and level. The key can be used to decrypt the client's messages and the level to restrict access appropriately.

I found python's crypto libraries to be clunky so I wrote my own trivial library. It uses caesarean encryption. It also only encrypts the alpha characters as I had trouble with invalid transmission characters otherwise. This is of course completely unsuitable for actual encryption. It is merely used to demonstrate the protocol.

Another hole in the security of this system is that the directory server only checks that a user exists within the system before distributing the file. It does not check level. As such a malicious user could get access to any file on a server if they were granted access to any one file on that server by the directory server. This could be resolved by having the directory server generate a token which would be passed to the file server.

Section 3 Directory Service

The Directory server serves as a middleman between the client and the file server. It holds a database of metadata associated with the files on the file server. When the client requests a file the request is sent to the directory server. This server tells the client if the copy of the file the client has in its cache is up to date. If it is the exchange ends. If the file is not up to date the directory server passes on the address of a file server with the file on it. The client then sends a request to this server and the file is sent on to the client. When writing files the client asks the directory server if it has permission to write to the file. The directory server checks the user's level and whether the file is locked. If the user can write the file a file server url is passed to the client. If the user does not have permission the no url is passed.

Section 4 Replication

Section 5 Caching

Caching is managed through version numbers in my solution. Each file has a version number associated with it. Every time the file is overwritten this number increases. The client maintains a dictionary of filenames and version numbers. When the client uploads a file the server returns the version number. The client adds this to its cache. When the client requests a file it gives the server the versionnumber from its cache if it has one. If this version is the most recent version the file is not sent and the client is told their copy is up to date. If it is not the most recent version, the file is sent and the cache updates its version number to match this version. I this system the file is only ever sent if it is out of date.

Section 6 Transactions

My solution does not implement transactions.

Section 7 Lock Service

Locking is also tied into my levels of access system. A user who has permission to view a file can lock the file. When a file is locked no user can write to it. The file can still be read by those with permission. The file can be unlocked only be the user with the lock or any person with a higher level of clearance. This allows administrators to unlock files held by malicious or unavailable users. Locks are stored in the file database. The name and level of the user holding the lock are stored. These are retrieved from the token provided by the authentication server. When unlocking the name of the user requesting the unlock is compared to the name on the lock. If they match the file is unlocked. If they do not the level of the user requesting the unlock is checked. If the level is of a higher priority then the file is unlocked anyway.