akd40-der62-erg56
Alex Dawson - David Rocker - Eric Ghildyal

Phase 3

Overall

All communication is being intercepted

Group Server

Fully trusted, properly authenticates users

File server

Untrusted unless properly authenticated

Clients

Untrusted

T1 Unauthorized Token Issuance

Description of threat:

As our system currently stands any user can request another's token, this problem leaves our system inherently insecure, as it breaks down the notion of groups and privileges in a file system. For example, with a minimal amount of effort, a user could log in as a professor and remove somebody from a group based on a class so they do not have access to files required to complete graded homework.

Mechanism:

1. User connects to group server, with a request for a token, their username and their password, encrypted with the group server's public key.
2. The group server decrypts this tuple with its private key, and authenticates the user's password with a database, by hashing the user's password so it is not stored in plaintext.
3. Once authenticated the server returns the user's token.

Arguments:

This process prevents a user from getting another user's token with the assumption that only the owner of the account has both the username and password. We also protect from the possibility of a malicious entity getting the password from the group server by hashing the password with SHA-256, using a user specific salt. We are using SHA-256 because it has yet to be broken and it it faster than SHA-384.

T2 Token Modification/Forgery

Description of threat:

Anything that a client interacts with has the possibility of being tampered with and tokens are no exception. A forged token can have the same effect as switching users by giving the forger access to groups they don’t own, allowing them to see files they should not be able to see, and performing actions under the guise of another user.

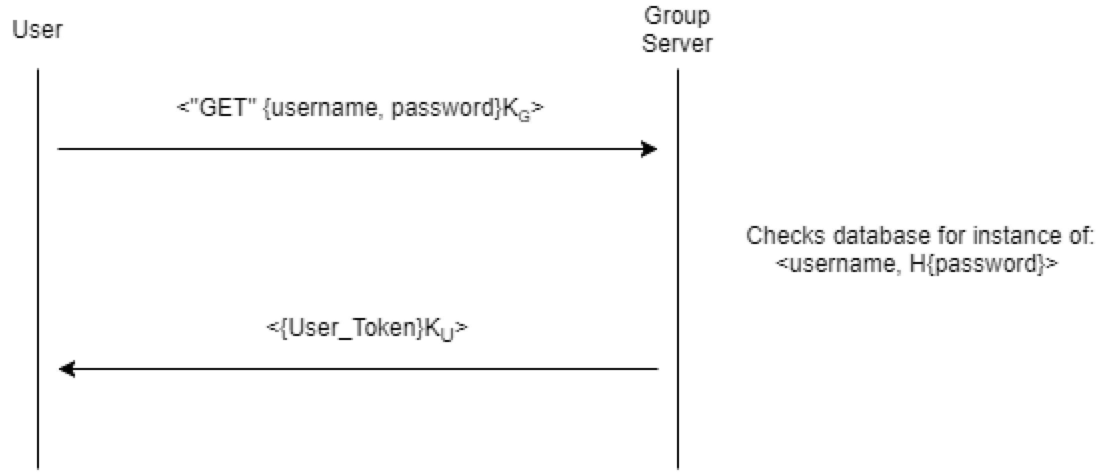
Mechanism:

1. This process is the same as T1, but the protection from this threat is provided by the use of the RSA signature at the end of this process..
2. User connects to group server, with a request for a token, their username and their password, encrypted with the group server's public key.
3. The group server decrypts this tuple with its private key, and authenticates the user's password with a database, by hashing the user's password so it is not stored in plaintext.
4. Once authenticated the server returns the user’s token signed with its public key .

Arguments:

The process protects against Token Modification by using an RSA signature from the Group Server on the token. This token can be authenticated by any fileserver, using the the Group Server’s public key. This protects against modification because if the user attempts to fake their token, the signature on the token will no longer be valid.

Fig. 1 Process for T1 and T2


User
Group Server
<"GET" {username, password}K_G>
Checks database for instance of:
<username, H{password}>
<{User_Token}K_U>

T3 Unauthorized File Servers

Description of threat:

Considering any user can run a file server, it should be assumed that there are a lot of unauthorized servers trying to break into the trust system. If a user connects to an unauthorized server, there is no guarantee the server will not steal their authentication and use it illegitimately. Due to this, knowing which servers one can and cannot connect to is a very important part of the overall system’s security.

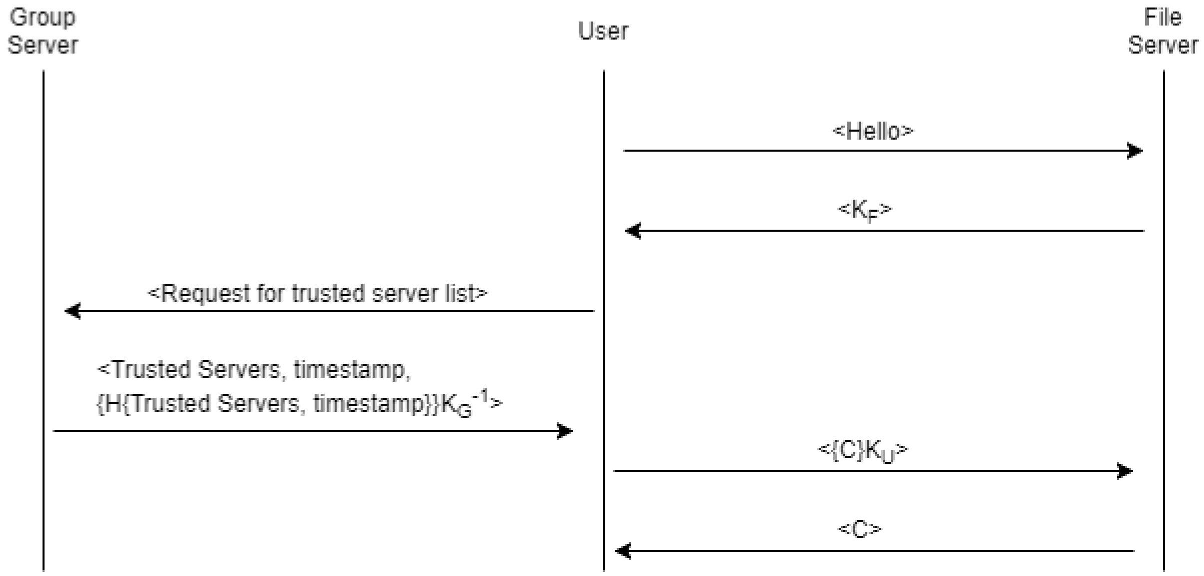
Mechanism:

1. We have set up three servers of our own and verified the authenticity
2. When a user attempts to connect, provide them the list of our verified public keys, and let them verify against the server they are connecting to.
3. If they accept it, connect to the server

Arguments:

Since we have a list of three verified file servers and are providing the key of the server they are attempting to connect to it is up to the user whether or not to trust it. At this point if the user chooses to continue on an unverified server it is at their own risk.

Fig. 2 Process for T3


Group Server
User
File Server
<Hello>
<K_F>
<Request for trusted server list>
<Trusted Servers, timestamp, {H{Trusted Servers, timestamp}}K_G^{-1}>
<{C}K_U>
<C>

T4 Information Leakage via Passive Monitoring

Description of threat:

Passive monitoring is extremely important in maintaining a secure system. No matter how many physical or policy steps are present, if data is being transmitted as plain text, anybody can read it and act upon the data. Even encrypted data is susceptible to attacks, due to increasing processing power, old algorithms can become outdated. Once an attacker has some bit of information, they can try a more narrowed set of attack vectors to break the rest of your system.

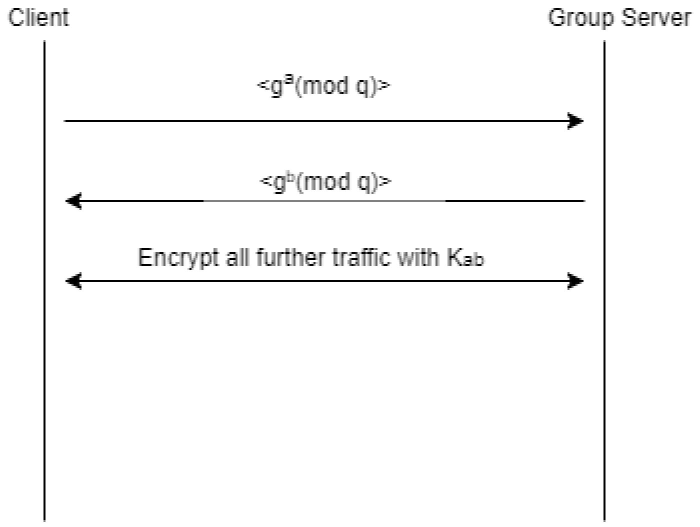
Mechanism:

1. Use Diffie Hellman to exchange keys between the client and server
2. Client encrypts all requests using the agreed upon key
3. Decrypt requests on the server using its key, perform the necessary actions, and encrypt the reply before sending it back to client
4. Client decrypts using its key and verify the appropriate action has been taken

Arguments:

Each request must be encrypted with the agreed upon key in order to ensure the inability of an outsider to decrypt and read any traffic between the server and the client. We are using Diffie-Hellman because it is a lightweight, cryptographically secure key sharing algorithm.

Fig 3 Process for T4



akd40-der62-erg56
Alex Dawson - David Rocker - Eric Ghildyal

From the mechanisms we designed the only two that go hand in hand are T1 and T2. They are both very similar in design and the implementation will be done using the same code. For each threat our group took the approach of each of us coming up with a proposed solution then discussing the pros and cons of each until deciding on one. This allowed us to have multiple options and potentially mix our solutions if one of us considered different risks than others. The discussion phase was probably the longest part of the design process as we didn't choose one until we all agreed with it.