Phase 3: Security Threats

Carlos Rios, Alice Turchaninova, Yuntian Zhang

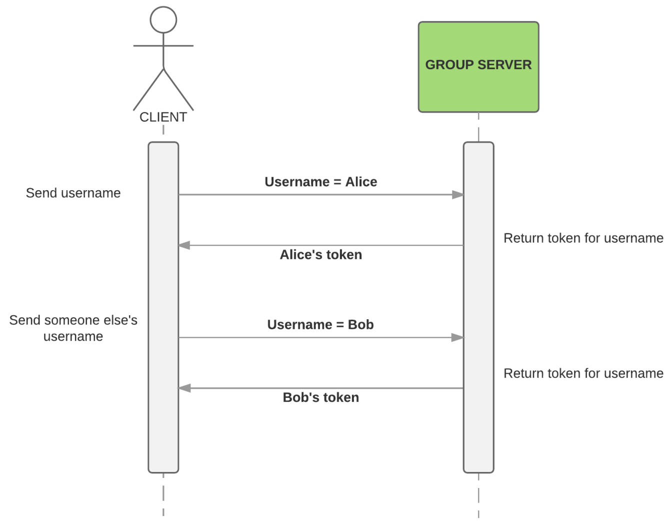
April 27, 2016

1 T1: Unauthorized Token Issuance

1.1 Insecure Implementation

In our Phase 2 implementation, we require only a username to receive a token. If Alice knows Bob's username, she can provide it and receive his token from the group server. She can then take advantage of any privileges Bob has, such as memberships in and/or ownerships of groups, to view, add, and remove group members, and view, upload, and download files in those groups. If Bob is an admin, Alice gets the privilege of creating users.

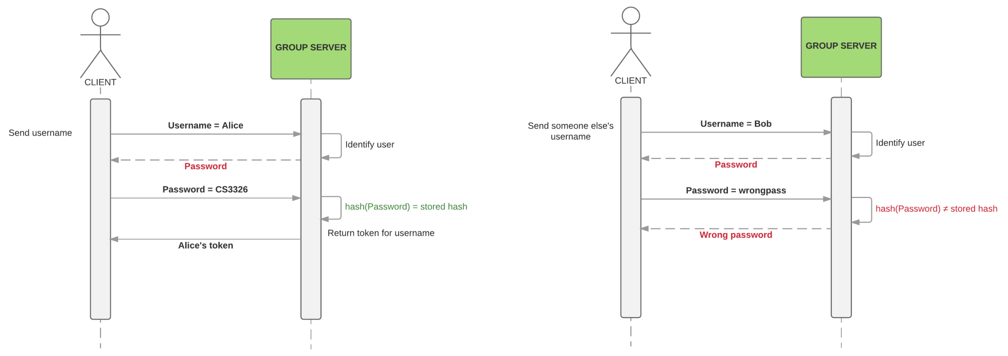
Figure 1: Insecure implementation: allows unauthorized token issuance.



1.2 Security Mechanism

In Phase 3, we have chosen to implement a password protocol to protect against this threat. When a client connects to the group server, a username and password must both be provided. The hash (using SHA-1 with RSA) of the entered password is checked against the hashed password stored in the user list for the user in question. If the hashes match, the user is authorized to receive a token from the group server.

Figure 2: Secure implementation: password mechanism prevents unauthorized token issuance.



The initial set-up is as follows: upon creation (by an admin) of a user account, a password is set by the admin and stored with that user's information. The admin may communicate the password to the user. After logging in with the provided password, the user can change the password to one of their choosing.

2 T2: Token Modification/Forgery

2.1 Insecure Implementation

In our Phase 2 implementation, a user receives a token from the group server at login and this token is used to authorize and execute group server and file server functions. Since the token contains information on which groups the user in question owns and/or is a member of, it determines whether a user is allowed to add or remove users in a group, list members of a group, and upload, delete, or view files in a group, among other actions. Therefore, by forging or modifying a token, a user can gain significant privileges. The token issued by the group server at login is not verified in any way before being used to authorize user actions.

2.2 Security Mechanism

In Phase 3, we have chosen to use RSA public-private key cryptography to provide a signature to each token, verifying it to have been issued by a legitimate group server. The RSA keys are generated at startup. A user is issued their token at login, created from the user information stored in the user list. Throughout the rest of the session, the token is authenticated prior to execution of any operations by verifying the RSA signature of its contents. The information in the user list and the signature in the token are kept up to date with the latest legitimate (via group server) operation. We also store the timestamp of the latest legitimate operation to prevent token reuse (a user attempting to use a once-legitimate, but outdated token). To allow the file server to verify the legitimacy of the token it receives from the client, we store the identifier of the trusted group server with the file server and the timestamp of the latest legitimate operation in the user list. Similarly to the token signature, the timestamp is updated in the token and user list after group server operations affecting the user's permissions. Upon receiving a token, the file server verifies the issuer against the trusted group server name and the timestamp against the timestamp stored in the user list. Thus, both the group server and file server are able to verify that a token has not been modified or forged after issuance.

3 T3: Information Leakage via Passive Monitoring

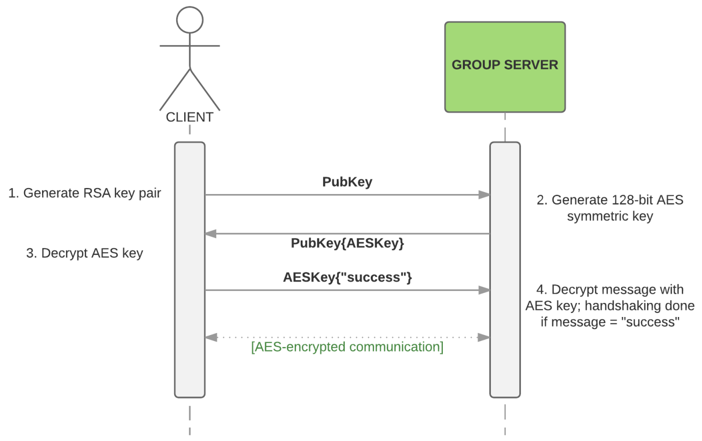
3.1 Insecure Implementation

In our Phase 2 implementation, messages passed between clients and servers are sent plainly, without any protection. These messages include usernames, passwords, user tokens, group and group member information, and files. Therefore, anyone able to observe these communications could glean a large amount of valuable user, group, and file information that should be kept private.

3.2 Security Mechanism

We have chosen to use symmetric key cryptography to encrypt messages passed between clients and servers, with RSA public-private keys used to share the symmetric key. Using RSA gives us a simple and secure way to share a session key between the client and server. The symmetric encryption will use AES with a 128-bit key in the CBC (cipher block chaining) mode of operation. We chose AES because it is the standard for symmetric encryption. We judged that a 128-bit key length and the CBC mode of operation are sufficient for our security needs; our system is not being used to store highly confidential information, and these choices meet with general security standards.

Figure 3: Secure communication using RSA public-private keys and AES symmetric shared key.


CLIENT
GROUP SERVER
1. Generate RSA key pair
PubKey
2. Generate 128-bit AES symmetric key
PubKey{AESKey}
3. Decrypt AES key
AESKey{"success"}
4. Decrypt message with AES key; handshaking done if message = "success"
[AES-encrypted communication]