Team Project Phase Three:
Cameron Buck, Nathan Ong, Dave Stein

Introduction

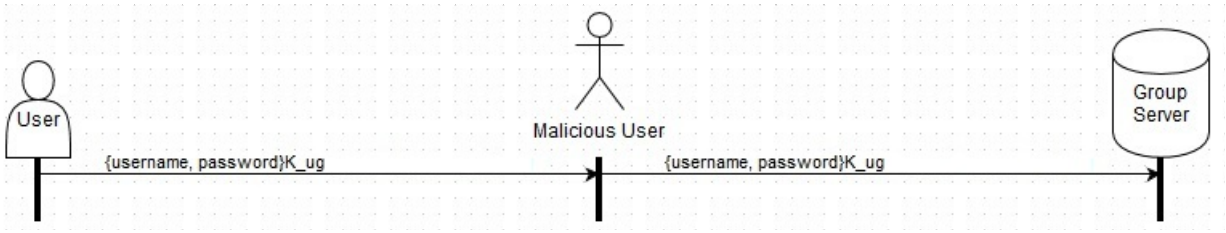
In our attempt to protect against threats, we have come to the conclusion that threat number four, Information Leakage via Passive Monitoring, is the primary threat we wish to counteract. Threat four is a major factor in enabling malicious users to attack the system through threats one, two, and three. Examples of sensitive information that may be compromised include: authorization tokens, files, and passwords. As a result, our primary objective is to protect against threat four, which will be used as a basis for determining mechanisms for threats one, two, and three.

In order to protect against threat four we plan to implement Secure Socket Layer (SSL) to create a tunnel to prevent information leakage. While observers may notice that a client is connected to the server, a malicious user will be unable to decipher the data that is communicated between the client and server. To protect against threat one, Unauthorized Token Issuance, we will implement a simple password authentication protocol to verify a user's identity. In regards to threat two, Token Modification/Forgery, we plan to utilize a cryptographic hash function to ensure the integrity of a token. Finally, to mitigate the effects of threat three, we will establish a certificate authority to handle public key distribution of properly authenticated group servers and file servers. Should a client wish to connect to a file server, he or she must retrieve the public key from the certificate authority before connecting to the file server; this is the same with any group server. We believe that these mechanisms will ensure confidentiality of data between the client and server.

Threat 1: Unauthorized Token Issuance

In the original implementation of the group-based file sharing application, users can simply request for a user's authentication token by providing a valid username. This leads to the possibility that untrusted clients can impersonate another user and gain access to files or privileges that he or she should not have. In addition, users who should not have access to the system at all would be able to gain access by simply providing a valid username, which may be as simple as trying permutations of common names.

We wish to prevent unauthorized users from impersonating other users to ensure confidentiality and integrity of data and keep the system accessible to users who should have access. In order to mitigate this threat, we will assume that T4 has been mitigated first (see Section 5), meaning that there is no possibility that users can monitor the system passively by setting up a symmetric key. We furthermore assume that components of T3 have been set up (i.e. a certificate authority) to ensure that the public key of the group server can be accessed such that the symmetric key can be sent securely. Then, in order to prove that a user is not impersonating another user, in addition to sending a username, the user must also send a password through the SSL tunnel created when the user connects with the group server after being encrypted with the group server's RSA public key.



Whenever a user is created, a password will be asked for whenever an admin creates him or her, with a minimum length of eight characters. The password will be stored on the group server after being concatenated with a random salt of 128 bits generated by Java's SecureRandom and hashed through 160-bit SHA-1 after ten cycles. This ensures that the user accounts are not compromised, should the group server be compromised. The group server will check passwords through hashing before issuing any token.

This solves the problem because only the user should know his or her own password, which verifies his or her identity. Encrypting with the predetermined symmetric key ensures that the username and password can be read by the intended receiver. Storing it on the group server as a hash after a salt ensures that should the group server be compromised, the group server has not provided the attacker with a direct username-password pair.

The act of issuing a token is addressed in Section 3. We require that running any command for a file server requires that the user re-authenticate with the group server.

Threat 2: Token Modification/Forgery

Hashing can be used to prevent token modification. The goal is to prevent malicious users from modifying their privileges. Every time a user logs into the group server, they are given a token based on the following three categories:

- Issuer: the Group Server
- Subject: the user's username
- Groups: any groups the user is in

This token is a concatenation of the issuer, subject, and groups in order. The groups A, B, C, and D will always be concatenated in the same order to avoid confusion: alphabetically. They would be concatenated into the group category as "A~B~C~D". Tildes ("~") are used as delimiters to separate the group names and avoid any ambiguity. Notice that the tilde character will be disallowed from usernames.

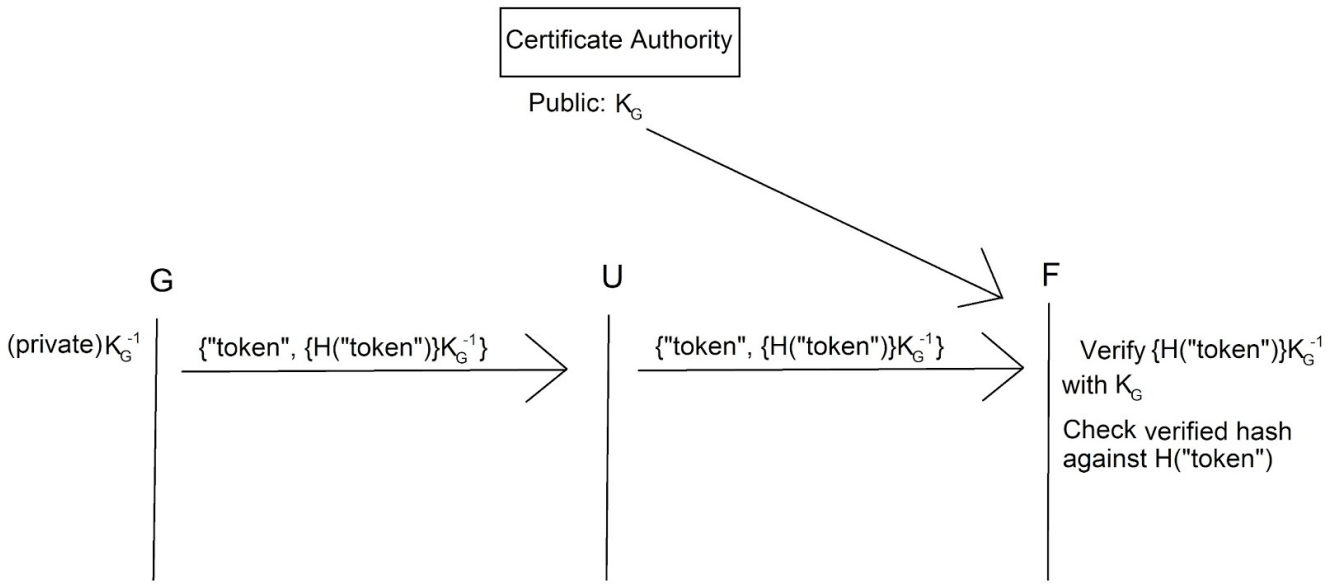
When creating a new username, the server should check for a tilde. No username should contain "~" because this would be a potential way to gain increased privileges. If the username "bob~A~B~C~D" were entered for bob's username, the server would concatenate the groups A, B, C, and D to bob's token and possibly give him privileges he didn't have. The concatenated clear-text token would look like this (issuer, subject, group):

"groupserver~**bob~A~B~C~D**~A~B".

The underlined groups A and B are bob's actual groups. The **bolded** text is bob's username. The server would interpret Bob as having access to all four groups with there being two instances of groups A and B. This is bad. Different delimiters besides tilde could be used to delimit the Issuer, Subject, and Groups, but the easiest thing to check is the user's input for a username. Disallowing tilde is easy, and there is no reason for a username to have the tilde character in it. Applying the principle of economy of mechanism, we choose the easiest way because it is the best way for our needs. Coding more to support tilde in usernames would take more time and possibly introduce new bugs.

These categories are concatenated into a string, and that string is hashed with 160-bit SHA-1. The token will consist of two parts: the clear-text concatenated string and the 160-bit compressed SHA-1 string. This is not secure because we assume the malicious user knows how we hash. The user can easily make up his or her own privileges by hashing a new concatenated clear-text string with modified privileges. If the user gives the current insecure token to the file server, there is no way to confirm or deny the group server's authorship of the key. The current token can be forged, and there is no way to trace its origin to the group server.

To prevent the token from being forged, the group server will sign the hash of the concatenated string with its 1024-bit RSA private key, K_G^{-1}. All parties will have the group server's public key, K_G, from the certificate authority and will be able to decrypt the hash. Once the hash is decrypted with K_G, the clear text can be hashed to check it against the decrypted hash. Both hashes should be identical. Two identical hashes give the file server high confidence that the group server signed the hash with K_G^{-1} due to to RSA being hard to factor. Bouncy Castle's support of SHA-1 and RSA will provide this cryptographic randomness.

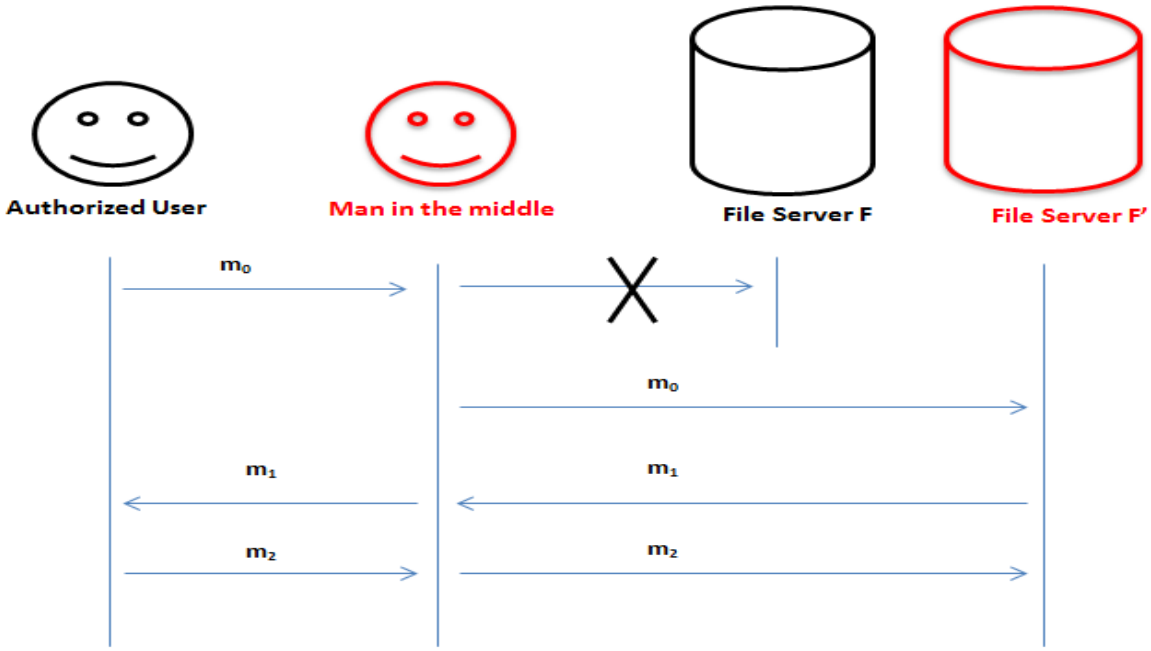


We leave out the security details of the last message (file server sends file to user) because it requires integration of threat four. Thus, the details can be found in the conclusion.

Threat 3: Unauthorized FIle Servers

The unauthorized file server threat can appear in the event that a user is attempting to communicate with file server F. However, it may be the case that they are in fact communicating with a server F' impersonating file server F. This threat could be problematic for multiple reasons. Most importantly, the user may not be able to complete the intended task they set forth to accomplish. In other words, a disruption in service is created. Other examples of why this threat is problematic is because the user may unknowingly provide sensitive information about the overall system (e.g. group server, any other file server), download malicious files, etc. In terms of the three security properties confidentiality, integrity, and availability, Threat 3 is mainly focused on integrity and availability. However, communication confidentiality is equally important. The goal is to maintain the communication integrity, ensuring the file server with which we are communicating is trusted. In regards to availability, the user needs to be able to accomplish the tasks they intended to do without disruption.

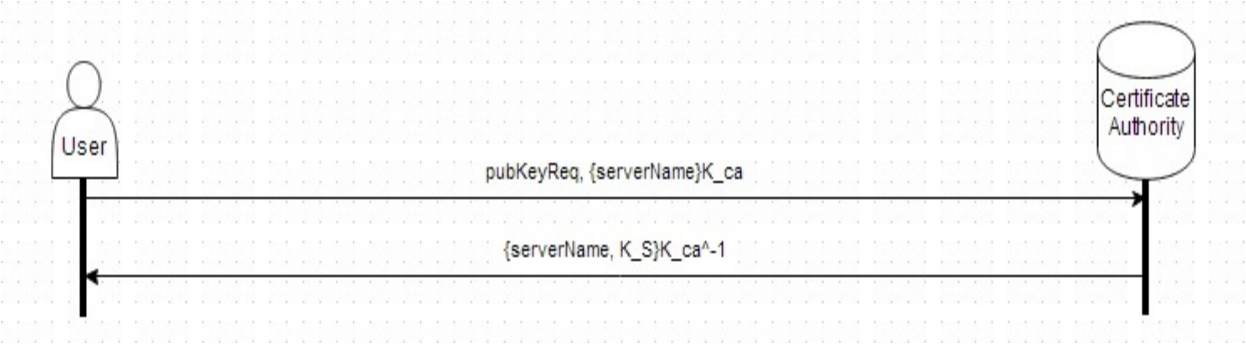
The following diagram depicts a scenario in which a man in the middle intercepts communication between an authorized user and redirects communication to a malicious file server F'.



The mechanism we plan to utilize to address this threat is the use of a Public Key Infrastructure (PKI). The goal is to utilize a certificate authority to issue digital certificates. Digital certificates will be stored for the group server and all file servers, which contain each server's public key. It will be assumed that

the certificate authority is trusted for this scenario; otherwise this can be an area of vulnerability due to impersonation. To implement this mechanism we plan to create a single server dedicated to distributing digital certificates. We assume that this certificate authority is always online, and every piece of software released to the client has the certificate authority's private key K_ca (RSA 1024-bit).

The first step is for the user to retrieve the RSA public key (1024 bit) from the certificate Authority:



The user will provide the Certificate authority with the corresponding server name. The Certificate Authority will return a message containing the server's name and the public key K_S of the server digitally signed by the Certificate Authority with K_ca^-1 (also RSA 1024 bit). The user can then verify that it came directly from the Certificate Authority and the public key is for the correct server.

This public key will be used to ensure that the symmetric key that is sent to establish a valid connection (See Threat 4). The reason why a trusted certificate authority will properly address this threat is that it will provide an unforgeable digital certificate which contains the server's public key with which to communicate. For example, the user will be able to determine that he or she is communicating to file server F because only server F knows how to decrypt messages encrypted with its public key.

We assume that all software downloaded has the public key of the certificate authority to prove that whenever the user retrieves the public key of the group server or file server, it really came from the trusted certificate authority. Any change in the key will require a new download of the software.

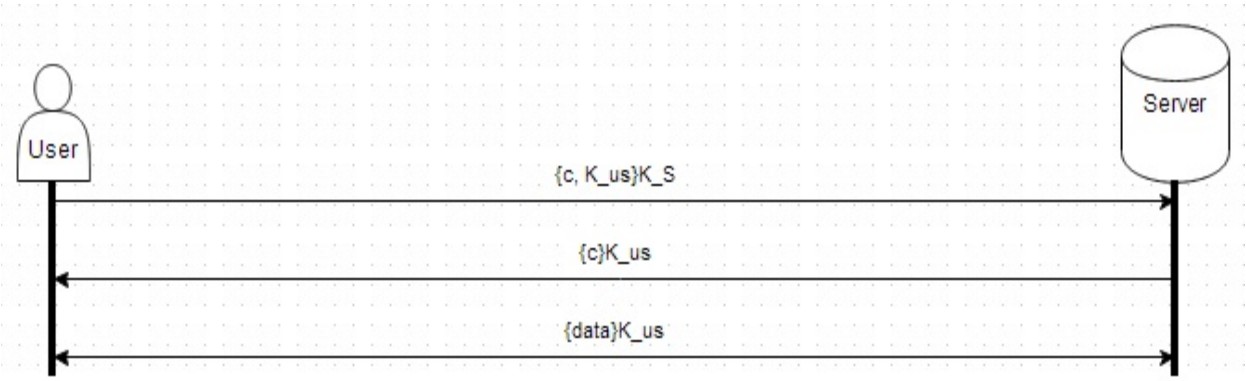
We also require that any new file server must send its public key to the certificate authority through some offline manner. We assume that the public-private key pairs are secure for at least a few years, which goes beyond the scope of the assignment. Thus, there is no mechanism to alter the key pairs.

Threat 4: Information Leakage via Passive Monitoring

Connections that are unsecure have the possibility of malicious users passively analyzing the data that is sent by both parties. This data may be sensitive, including passwords or files, which probably should not be shared with other entities. As a result, there is a need to protect against monitors. The mechanism to do so requires that a Certificate Authority is already established, its public key distributed with the software, and the server has given its public key to the Certificate Authority (See Threat 3).

We also assume that the user has already retrieved the public key of the corresponding server (K_S).

In order to set up the secure connection, the user will create a symmetric key (128 bit key randomly generated, K_us) for the server and the user to correspond on. It will encrypt the symmetric key and a challenge with the given public key, and send it to the server:



Once the server receives the message, it will decrypt the message and send back the same challenge (a 128 bit secure random number) to the user encrypted with the symmetric key. We ensure that the server actually can use the symmetric key and we are talking to the correct server. Now, both the server and the user can interact in a closed channel encrypted with the symmetric key. Once the channel is closed, the symmetric key is discarded.

To ensure security with the random number generator, we will use Java's SecureRandom class with its implementation provided by Bouncy Castle. We will use the SHA1 implementation of a pseudorandom number generator, which has seeds with 160 bits. Java ensures that a new seed is generated per instance, as long as the seed creation method is not bypassed with a given seed.

This mitigates the threat because all data that is sent by the user and the server are encrypted with the symmetric key, which any passive observer would need to read any of the messages. In addition, the keys are discarded after use, which means that the messages will be theoretically unreadable afterwards.

Conclusion

In the discussion of the threats, we mention that solutions to certain threats can be parts of solutions to other threats. We mention that T4 (Section 5) is the only threat that does not use other threat solutions. SSL in every connection ensures that passive monitors will be unable to gather any data in transit. T3 (Section 4) uses this solution to be able to set up communications with the trusted certificate authority, which the public keys then verify the identity of the file server and group server. T1 (Section 2) uses mechanisms from both T4 and T3 to ensure that the password is not seen being transferred, but the group server's public key is used to encrypt the username and password to ensure

that it arrives at its intended destination. T2 (Section 3) also uses the solutions provided for every threat to ensure that the tokens are unalterable by the group server.

In a full view of the solutions to the threats, we take an example of having the user upload a file to the file server from scratch. First, we ensure that the certificate authority is running. Without this, we cannot retrieve public keys for any server. We generate the public and private key pair of the certificate authority, then update the software to reflect that. Then, we create the group server and file server and its public-private key pairs, and send their public keys off to the certificate authority, digitally signed with their private keys. Then the user connects first to the certificate authority to retrieve the public key of the group server. The user then encrypts his username and password with the group server public key and sends it off to the group server. Once verified, then the user can now connect to the file server, to which the user sets up a session symmetric key by encrypting it with the file server's public key. Then, the user encrypts a request for his token. The group server this request after unencrypting it, and sends the user a token, which includes a hash of the token digitially signed by the group server, which is sent to the user. The user will then send the token, upload command, a hash of the file, and the file he or she wishes to upload, encrypted with the symmetric key generated before between the user and the file server. The file server receives it, unencrypts it, checks the token, and the command. If everything checks out, then the file server runs the command with the file, and sends a success message to the user.

The most difficult portion was to figure out how to prevent the user from altering tokens (T3). We had to figure out how to be able to pass the token from the group server to the file server to show that the user does have the privileges to do modifications within the file server, without allowing the user to have access to modifying the token. We finally figured out that we could force encryption of the token, which prevents the user from modifying it or forging it without access to the private key of the group server.