CJ McAllister
Andrew Powell
Andrew Tribone
April 10, 2011
CS 1653

Phase 4 Write-Up

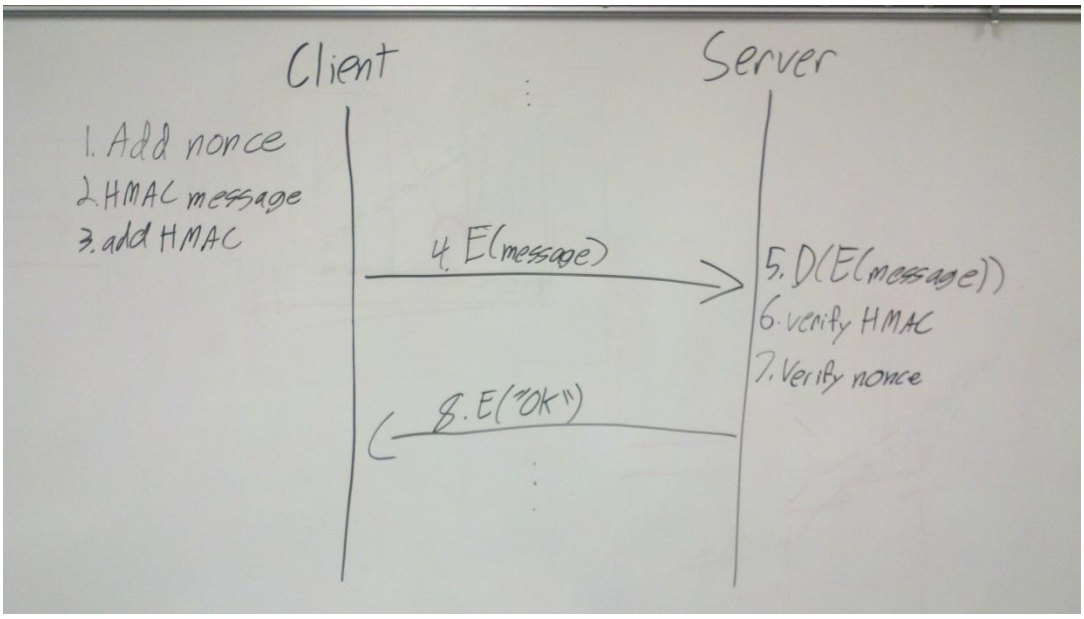
In this phase of the project, our file sharing system needs to be prepared to defend against an active attacker. An active attacker is a far more formidable adversary than the passive attacker that can only listen in on a connection. This new attacker has the ability to replay messages, resend old messages, or modify message at whim. Additionally, the attacker may impersonate a file server to leak files and/or user tokens. To deal with this threat, enhanced security measures are required. For example, to deal with our first threat model, the reordering, replay, and modification of messages, it becomes necessary to implement certain features that ensure the integrity of the message exchanges. In particular, we will make use of nonces and HMACs in our messages. This will help to ensure the identity and authenticity of the sender of a given message. In regards to our second threat, file leakage, we will have a symmetric key system for encrypting files. This system will only allow those with the proper key to access files. These keys will be group-specific, and will be changed as necessary. For the final threat model, we will again be issuing new tokens just before a client connects to a file server. These tokens will contain the name of the file server, and this name will also be part of the signature; effectively binding the token to that specific server.

T1: Message Reorder, Replay, or Modification

The first threat that must be dealt with is the reordering, replay, and modification of messages. This threat is very common of the active attacker, as it is quite simple to make a log of messages sent and received to be studied offline in order to craft an attack. This attack is particularly dangerous because, if executed properly, an attacker could bypass security protocols by sending a specific message that may be one step beyond the security measures in a given exchange.

As a first line of defense against this threat model, each message passed between a server and client will contain an HMAC and a nonce, each of which will be verified when received. The nonces will serve to protect against any replay or reordering attacks. It accomplishes this by incrementing the value on both the client and server side for each message passed. If an attacker attempted to send a message that was not in-line with the nonce, the client or server will drop the connection. The HMAC ensures the integrity of a message, because the client and server had shared a secret HMAC key, and both will use this to digest the messages that they receive. If at any point, the HMAC does not match, the connection is dropped. To accomplish this, the HMAC and nonce will be attached as an additional object in each Envelope that is sent to the client or server. Once the Envelope is received on the other end, the first step will be to verify each. If both check out, the function moves on; if the either is invalid, the connection is dropped at that moment.

Figure: Generic message passing w/ nonce & HMAC



These implementations will successfully protect the system from this threat model because each component of the threat is dealt with on each message exchange. Because of this, we can be assured that even if the attacker sends a modified, reordered, replayed, or any combination therein, our system will reject him on at least one front. Additionally, any legitimate client or server will pass these tests, so the legitimate users will not be hampered by the security measures.

T2: File Leakage

This threat model is particularly important, because it involves the fundamental purpose of this system – securely storing files. If file leakage is allowed to occur, our system's defenses will have been fully compromised. With free reign over reading and modifying files, an attacker could download a given file, and replace it with a malicious file that could execute arbitrary code on whatever system it ended up on. This obviously would wreak havoc on clients and servers alike.

Our method for preventing this extremely dangerous scenario is a symmetric key encryption scheme. Each group would have an ArrayList of AES keys that is used to encrypt and decrypt files that belong to that particular group. The group server would keep track of these keys for each group, storing this information in a HashMap of ArrayLists that contain an object containing an AES key and its corresponding number. With this security measure, even if a file were wrongfully downloaded by an attacker, they could not decrypt the contents to gain access to the information, and they also would be unable to insert their own data, because they lack the AES key required for encryption or decryption. This is sufficient to protect the files if the groups remain static; however, this is not the case.

In order to maintain security with dynamic groups, a method for key replacement must be implemented. We accomplish this by generating a new key each time a member of the group is removed. This ensures that old group members cannot access new or newly modified files once they have left the group. We have made the assumption that a former group member is able to download all the files available to that group the moment before they are removed, so we are not concerned with changing the older keys. Because of this, we only need to add new keys to the ArrayList, rather than changing all of them when a member is removed. Additionally, we will match the keys to their files via a simple numbering system, where a number is assigned to the key and the file it encrypted upon completion of encryption. This allows for efficient lookup when the file needs to be decrypted.

This implementation adequately secures the files on the file servers, because it only allows clients who are members of a given group at the time of connection to access their group's files. The AES key prevents an attacker from reading or modifying any existing files, maintaining integrity and confidentiality of the data, all while allowing the user uninhibited access to their rightfully-owned files due to the efficient nature of symmetric key cryptography. Additionally, we ensure that only authorized users can access the files at all by changing the AES key whenever a member is removed from the group.

T3: Token Theft

If an attacker posing as a file server were able to hijack a legitimate user's token and pass it on to a malicious user, that malicious user would have full access to any file that the legitimate user previously did. This poses obvious issues, as the attacker could then alter files at will and insert arbitrary, executable code into the hijacked user's groups' files.

To deal with this threat, we will be changing the structure of our tokens. A file server name field will be added to the token data, and it will be included in the signature of the token. Upon initial connection to the group server, this field will simply be an empty string, but when the user is about to connect to a file server, the field will contain the name of the server. This identifying information ties the token to a specific server. This way, if an attacker posed as a server to capture the token, even if it were passed on to another malicious user, that user could not use the token to connect to any other file servers except for the initial thief server. The token is created by the group server just before connecting to the file server and the file server then verifies the signature, and also checks the server name and from its contents against its own name. A successful verification allows the process to continue, an unsuccessful verification will drop the connection.

Conclusion

We feel that this set of security measures will effectively route any of the threats that we are concerned with at this point in the project. Each new security component will be effective because we took care in considering what the implications of each component would be. For instance, we make use of symmetric key cryptography whenever possible in order to maintain efficiency, and therefore

usability of the system. Additionally, we made sure that no one security component was too heavily relied upon, such is the case with T1, where we address each part of this threat model individually, rather than one large, clunky solution. In designing our solutions, we took a step-by-step approach, tackling each security issue in turn and making sure it was locked down before moving on. This helped us from looking over details of the individual security issues. Throughout this process, we had to discard several approaches to certain issues. One such discarded approach was using public key cryptography for guarding against file leakage, which we tossed due to performance issues as well as shaky security. We also considered having the file server keep track of the AES keys for T2, but we realized that this was unsafe due to the untrustworthy nature of our file servers in this phase.