Security Testing

Security testing is appropriate for our project, because at this point in the semester we have added features that have made the system far more secure than the bare bones system we started with. In a few different phases, our group from our applied cryptography class added functionality to the project which helped it combat specific threats and reduced vulnerabilities. Not only did we have to worry about active attackers (men in the middle), but we also had to protect against nosy administrators listening over the network. In order to do this, we used fundamental tools like symmetric keys, hashing, and digital signatures in order to uphold the CIA (Confidentiality, Integrity, and Availability) properties.

The system we are testing uses symmetric key cryptography in order to get us the confidentiality that we desire. In the case of this system, messages sent between clients and servers, as well as files uploaded to file servers are encrypted so that attackers can gain no information from tapping the wires or breaking into the server itself.

In order to provide confidentiality of messages sent from the clients to the servers, both public/private key cryptography and symmetric key cryptography is used. First, public/private key cryptography is used to establish a shared key for the symmetric cryptography that will be used by the user and the server for that session. All traffic must be confidential, so the user first uses the group server’s public key to send over log in information. This log in information would look like garbage to anyone listening over the line, and only the group server has the private key that decrypts it. After the user logs in, the dialogue continues with the set up of a symmetric key that is used to encrypt and decrypt all messages until the user logs out.

Another feature of our system gives users the ability to upload files that other authorized users can download. These files need to remain confidential in the event of file leakage, so they are encrypted before they are uploaded and decrypted once downloaded.

With all that being said, the confidentiality of sensitive data is fully protected by our system. Every message from the log-in procedure to when the user selects “quit” from the menu is encrypted using either asymmetric, or symmetric cryptography with a 128 bit key, from which the best known attack takes exponential complexity to crack. In addition, a user only uses a symmetric key for one single session which isn’t very long anyway. For all intents and purposes of a file sharing system like the one we are testing, confidentiality of data is not an issue.

Integrity is also a fundamental part of systems security. Users want to know if the data they receive has been tampered with or botched in transit, and servers want to be able to properly authenticate users who can prove they are who they say they are. For our system, there is a group server which is in charge of managing user permissions, and there are also separate file servers who grant users access based on their group server permissions. File servers can not directly communicate with group servers, so it is up to the users to prove what type of access rights they have. In order to prove what rights a user has, the group server gives the user a “token” object that lists permissions, that the user then gives to the file server. Based on the token, the file server allows or denies access to files.

One obvious attack on this setup would be if a user could manipulate his or her token to gain additional permissions than what they got from the group server. In this case, the integrity of the token object would be violated, and a file server should be able to tell if this has happened. In order to combat this vulnerability of the system, a digital signature of the token object is passed along with it so that the file server can verify whether the token it is receiving is the same one that the group server issued in the first place.

When a user logs into the group server, he or she is issued a token object which outlines which groups they can share files with. Not only are they given this unencrypted object, but they are also given a “digital signature” on that object which is used as a mechanism for providing integrity protection. This signature is computed by the group server, and is just a hashed and encrypted version of the token it just issued. When the token and signature are given to the file server, the file server does a decryption and a hash of the token to check to see if they match. If a user did not tamper with his or her token, then the matching indicates that integrity is upheld. If a user tampered with even 1 bit of the token, the signature will not be able to be verified and the token will be rejected.

Further, our system actually does integrity protection with every single message being sent back and forth between users and servers. Before a message is sent, a hash of the message is computed and attached. Upon receiving a message, the first thing that a party does is check whether or not a hash of the plaintext message matches the signature that was sent. As described in the confidentiality section, all of this traffic is already encrypted with symmetric key cryptography. To an outside attacker, this just appears as garbage text going from point A to B, but they could still mess around with the contents as it is being sent; confidentiality is not the same thing as integrity. Ultimately, these signatures give the system a way to verify the authenticity of all messages and user permissions in a computationally inexpensive way.