**CS 6349 - Network Security – Fall 2017 Programming Project Report**

**Anirudh Rathinam (net-id: axr167530)**

**Utsav Vijay Dholakia (net-id: uxd150130)**

**Project Overview**

This project attempts to implement a secure file transfer protocol to transfer data securely between the client and the server. The client authenticates the server allowing data to be uploaded to and downloaded from the server in a secure fashion. The project considers issues such as authentication, maintaining confidentiality and performing integrity checks to ensure secure data transfer between the client and the server.

**Functional capabilities model:**

The functional capabilities are as follows:

* The client authenticates the server using the certificate from the CA
* A session key is generated and exchanged using public key cryptography
* Based on the session key encryption key, integrity key and IV values are generated.
* This happens each time a connection is established
* In addition to the above, the project also ensures that the integrity of the data is maintained.

With the help of the above, the protocol allows for secure data exchange between the client and server. It allows the client to upload files to and download files from the server in a secure fashion and ensures that the integrity of data is maintained with the help of a keyed hash function.

Various well-known security attacks are considered such as the possibility of data fabrication, man in the middle attacks etc. and the protocol provides adequate preventive measures for these attacks.

The program has been implemented in Java.

**Authentication**

Authentication is achieved with the help of a Certification Authority (CA). The client initially has the CA’s certificate and public key. The Server has its private key, its public key and the certificate issued with the CA.

When the client initializes a conversation with the server, the server sends its certificate issued by the CA to the client. The client can verify the identity of the server by comparing the Server’s certificate against the public key of the CA which it already has. Once the verification is done, the client knows it is talking to the correct server.

Once the verification is done, the next step is to arrive at a shared secret through which all future communication can take place. To do this, the client first generates a ‘pre-master key’. To share this information with the server, the key is encrypted with the public key of the server. The client has the public key of the server from the certificate it initially sent for verification purposes. The encrypted secret key is then sent to the server where it is decrypted using the Server’s private key. Hence both the client and the server have the pre-master secret from which a session key can be derived for data encryption.

**Data Confidentiality**

The data to be exchanged once the encryption key has been established can be done as using a scheme described as follows:

* The message can be broken down into fixed length chunks (p1...pn) of the size of the message digest.
* We use intermediate blocks (b1...bn) and ciphertext blocks (c1...cn) computed as follows:

b1 = SHA(encryption key|IV) c1 = p1 XOR b1

b2 = SHA(encryption key|c1) c2 = p2 XOR b2

...

bn = SHA(encryption key|cn-1) cn = pn XOR bn

* Since only the ciphertext blocks are being transferred over the network, confidentiality is maintained.
* To obtain the original plaintext message, we simply need to XOR the ciphertext using the intermediate blocks (b1...bn). For example, pn = cn XOR bn.

**Integrity**

To ensure integrity, each 8-byte ciphertext block (c1...cn) is hashed using the SHA message digest. The one byte of the digest is appended to the 8 byte ciphertext block and is sent to the receiving end. The receiver hashes the first 8 incoming bytes and compares the first byte of the digest to the last byte on the receiving end. If the results are the same, this means that the data has not been tampered with.

**Threats, possible attacks and attack prevention measures**

The protocol deals with several threats which are listed below:

* Identity theft/spoofing: The authentication step handles this with the help of the CA using public key cryptography.
* Traffic monitoring: Since the ciphertext is not decipherable to outside observers without knowledge of the shared session key, traffic analysis is no longer a threat.
* Man in the Middle attacks: If a third party intercepts the data, it will not compromise the security since the data has been encrypted and can only be decrypted by the intended parties.
* Data fabrication: This is handled in the integrity check. If the data has been tampered with, the hash values will not match the value sent along with the ciphertext block.

**Challenges Faced**

The biggest challenge was in the authentication. Translating the certificates and making sure java (the language we used) recognized them was among the biggest challenges we faced. The issue was solved only after we managed to get a better understanding of how certificates work and how languages like Java parsed these certificates.

Another major issue was deciding how to transfer the data between the client and server. This issue was resolved by using the ObjectInputStream and the ObjectOutputStream classes.

**Contributions**

While we did most of the project tasks together and would often help each other while designing the modules, the contributions can be broadly summarized as follows:

- Authentication: Anirudh Rathinam

- Encryption: Utsav Vijay Dholakia

- File Transfer: Utsav Vijay Dholakia

- Integrity Check: Anirudh Rathinam