**S-38.3159 Protocol Design – Assignment 2 – Robust SyncCFT**

The assignment extends the capabilities of the Assignment I in terms of security and robustness. The protocol now has to implement new mechanism that ensures end-to-end security for the users and protects against most common attacks such as man-in-the-middle or spoofing of packets.

1. **Based on the specifications and design of the Assignment 1, we have identified the following potential vulnerabilities in our system.**

**Spoofing of legitimate packets**

The first version of our protocol did not account for any security mechanism such as authentication or encryption. As a consequence, once the *connection* has been negotiated between the end peers, it would be become virtually possible to hijack the connection by using the Man-In-The-Middle attack.

Our protocol implements ID fields for the sender and the local and remote sessions. In addition the sequence flag guarantees the ordering of packets processed by the application. This is necessary because the implementation has been carried out on top of the User Datagram Protocol (UDP). Remember that UDP does not provide reliable data transfer or ordering of packets, so these functions had to be delegated to the application. Assuming that the sequence number has to be incremented by one each time the application sends a message, it becomes rather easy to intercept legitimate packets and inject spoofed packets in the network thus affecting the standard operation as described in the protocol specifications.

**Protocol attack**

The protocol could be vulnerable for example to faulty transmission reports requesting the retransmission of previously sent data and therefore entering a transient state where the same information is forwarded all over again resulting in an endless loop. The retransmission of previously captured messages from an attacker could also exploit the recipient peer by allocating non-necessary resources or retransmission of previously acknowledged data.

**DoS attack**

The protocol could be vulnerable to a storm of “Update” messages from an *authorized* or *authenticated* host. In conjunction with the spoofing of packets, a storm of update messages with different   
“version-ID” used to trigger the update mechanism could develop into a DoS attack. Under these circumstances a rogue client could establish a legitimate connection with another peer and start a DoS attack that will result in the latter unavailability of the attacked peer.

1. **Provide a fix to the protocol based on the categories identified.**

**Spoofing of legitimate packets**

In order to address the vulnerabilities detected in the previous phase we decided to implement a mechanism for enabling security. To that end, we support the authentication of remote peers and cryptographic encryption of the messages exchanged. The operation reuses the connection establishment phase defined in the first version of the protocol but with a different set of options. The mechanism introduced prevents from the first and more difficult problem to address, spoofing of packets.

Since there are 3 messages in the HELLO establishment we have established 3 phases for the adjacency establishment. Upon starting the application, a password is requested from the user to start the synchronization session, so that together with a dynamically generated Public and Private RSA Key the authentication of users is granted.

The generation of RSA Keys guarantees that the information ciphered with the public key can only be deciphered with the private key. The adoption of RSA Keys also allows us to securely transfer dynamically generated session keys each time the application requests it. For the cryptographic encryption of the messages, we create a shared AES key for the session of the two peers. The key can be securely transferred from the originating peer by using the remote public key exchanged during the HELLO phase.

The following lines describe the process of the connection establishment specifically designed for the Assignment 2. Please consider the case where a certain Peer-A initiates connections to a certain Peer-B.

**Phase 1**

Peer-A sends HELLO Req with the security flag activated. A security TLV is generated containing the Public Key of Peer-A and the MD5Sum of the Public Key and the local password configured in Peer-A. When the Peer-B receives the messages, extracts the Public Key of Peer-A and verifies the MD5Sum with its local password. A matching result indicates that both of the passwords are identical and the communication can continue, otherwise the host cannot be authenticated and the packet is dropped.

Once Peer-A has been authenticated in Peer-B, the Peer-B stores the public key of A to cipher subsequent messages.

The contents of the TLV are as follow:

**“PubKA\_plaintext ? Hash(PubKA\_plaintext + Local\_pwd\_A)”**

**Phase 2**

After the Phase-1, the Peer-B will send HELLO Res back to Peer-A with security flag activated as well. The purpose of this message is to confirm the authentication to Peer-A and continues with the procedure of 2-way authentication so that Peer-A can trust Peer-B. The Peer-A then stores the public key of Peer-B to cipher subsequent messages.

The contents of the TLV are as follow:

**“PubKB\_plaintext ? Hash(PubKB\_plaintext + Local\_pwd\_B)”**

**Phase 3**

Once the two peers have been successfully authenticated, the connection is finally established following a similar 3-Way handshake procedure like in TCP. Then the originator, Peer-A, creates a shared secret AES key to be used for the encryption of subsequent messages between the peers. The key is securely encrypted with the public key of the remote user that was retrieved during Phase-2. A final HELLO Res message is sent acknowledging the connection establishment to Peer-B

**Protocol attack**

By implementing the security mechanisms described earlier, we are solving to a great extent the problems related to protocol attack. A packet that fails to be decrypted and verified the checksum is automatically dropped regardless of the originator or session IDs. In that sense, it becomes virtually impossible for an attacker terminate a session between two peers during the course of a data transmission. The shared secrets exchanged during the HELLO phase provide a trustworthy framework that protects against these attacks. The attacker is required to know the SEQ and ACK number together with the shared key so that the packet is not dropped by the recipient side.

An attacker could also engage in the retransmission of old packets such as HELLO requests in order to create state in the recipient side. The application can protect from this attack by storing the already known hashes for the session. If a new connection uses the same hash it can be classified as “retransmission attack” and therefore be discarded.

**DoS attack**

These attacks are intrinsically related to rogue clients that have been previously authenticated and send legitimate information. For those attacks attempting some sort of “SYN” attack, the protocol should only keep state information once the user has been validated, otherwise the message is discarded.

In order to protect against DoS attacks that would trigger constant requests of the manifest file on the attacked peer, it is possible to limit the rate of updates for a particular host. The state machine should store then the timestamps for the last update messages received and enforce a policy that prevents from requesting the updated manifest file.

The same procedure can be applied to Data transmission where a Data transmission constantly requests the retransmission of a packet thus limiting and constraining the synchronization.

As a summary, the timeouts and the number of retransmissions allowed for a certain type of packet should be also taken into account to tear down the *connection* if necessary.

1. **Implement a fix**

The implemented solution of authentication and cryptographic encryption provides a suitable framework for the “Spoofing of legitimate packets” and to a great extent of the “Protocol Attack”. Further robustness is dependent on the processing of the application rather than the definition and operations of the protocol.

1. **Terminate Session SyncCFT**

The designed solution for terminating the signaling connection is based on the expiration of timeouts. If a packet is not received during certain consecutives amount of timeouts, the session is marked as disconnected.

With regard to the termination of the data connections, our protocol attends to explicit signaling over the DATA channel that is used for transferring the chunks of information. This particular packet is of the type BYE Req.

On a regular operation, once all the chunks of information have been retrieved and the resulting file has been accordingly hashed and validated, a Bye Req message is forwarded.

In Assignment 1 this operation introduced a security hole easy to exploit, that would allow an attacker to terminate data sessions of other peers by sending spoofed BYE Req messages. The processing of these messages only takes into accounts the Local\_Session\_ID and the Remote\_Session\_ID.

For the Assignment 2, the protocol is no longer affected by such messages due to the encryption.