SICMAPP: Secure Instant Messaging Application

*Team : Aditya Rao (001903899), Yogesh Tewani (001991072)*

**Architecture**

**Server**

**User 1**

**User 2**

**Authenticate**

**Authenticate**

**Ticket/Communicate**

**Assumptions**

* Users will have already been registered with the Server
* Users only need to remember their username and password
* For each session, users will generate their own public/private key pair using RSA 2048 bits. These keys will be deleted at the end of the session
* Server will keep a list of all usernames encrypted with its public key and a list of users’ hashed passwords (hashed using SHA-256)
* All clients will have the Server’s public key built-in in order for all communication to the server to be only understood by the Server.
* One Central Server will store information in a database and will store public key of all clients (established upon login). Any client that wants to talk to another client has to go through the Server first.
* Public/Private key pair is generated by the client for each session.
* All R and nonce will be 128 bit random numbers.
* RSA will use 2048 bit keys and AES will use 128/256 bit keys in CBC mode.

Implemented Architecture

# Chat Initialization/ Mutual Authentication

W = Password

IPA = IP Address and port number combination of A.

PKA = Public Key of A.

A = Username

UA = Salt of the hash of the password to store

HW = Hash of the password with salt UA

DS = ga mod p, DA = ga mod p, KAS = gas mod p

TSS = Timestamp of Server at the time of login

Client A

Server

Login Request , {A}S

Hash[IPA, SA], UA, DS

, { DA }S

, Hash[IPA, SA]

KAS{A, HW, PKA, N1}S

KAS{ N1 - 1, TSS}

1. The Client will send a login request to the server along with its username encrypted with public key of server.
2. Server sends a Session Cookie which is a hash of IP address, port of client; and a randomly generated secret used as a cookie to prevent DOS attack; salt of the user and NS which is the DH contribution of the server that is used to generate the session key.
3. Client A send the session cookie; a DA, which is the DH contribution of the client, encrypted with server’s public key; and username, hash of the password, public key generated by the client, and nonce N1 all encrypted through AES using the newly generated session key KAS. The size of the can either be 128 bit or 256 bits.
4. Server then authenticates the user. If successful, the server sends decremented nonce N1 and the timestamp of the server at that time, TSS, encrypted using the session key KAS computed at the server using the DH contribution of the client and itself. This finishes the authentication of the user and the server stores the public key of the client for the current session.

# Get List of Online Users

Server

Client A

List, KAs{N2}

KAS{{List},N2-1}

1. Client A request list of online users using the List command and sends it along with nonce N1 encrypted with session key.
2. Server sends back the list (list of usernames and corresponding ID) and N1 – 1 encrypted with session key.

# Request to sever for chat initialization A – B

Ticket = KBS{PKA, IPA, A, TSAB, NAB}

TGTAB = Ticket, [HMAC{Ticket}]S

TGTAB, KAS{PKB, TSAB, N3 – 1}

Talk, KAS{B, NAB, N3}

Client A

Server

1. When the user wants to talk to a user in the list sent by the server, the client sends a “Talk” request along with the encrypted message containing the user id to talk to, a nonce generated to be used for talking to B, and a nonce N3 to maintain freshness of the message.
2. If the user is present in the list, the server then sends a Ticket granting ticket or TGT, which contains the public key of A, IP address and port number of A, the timestamp of the server TSAB at the time of the request and the nonce sent by A. This TGT is encrypted using the session key shared with B and server KBS. It also sends the HMAC of the TGT signed by the server and another message that contains the public key of B, TSAB and decremented nonce N3 encrypted using the session key KAS.

# User-User Connection Establishment

Client B

Client A

TGTAB, [NAB]A, {KAB}B

{TSAB, N1}A

KAB {message, N1-1, N2}

KAB {message, N2-1, N3}

1. To talk to B, it sends a “CHAT” request along with the TGTAB, NAB signed by A, and a AES key KAB, which will be the session key that will be used for future communication. This session key will be encrypted using the public key of B.
2. B then decrypts the TGTAB using the session key shared with the server, which verifies the authenticity of TGT, verifies the signature of A using the public key of A in the TGT and then obtains the session key KAB after decrypting using its private key. B sends back to A the timestamp TSAB encrypted using the new session key.
3. A then verifies the encrypted timestamp received and the connection is established. The future messages are sent encrypted with the session key.

# Logging Out

Client A can log out by sending a request to the Server with a nonce encrypted with the session key. This will signal to the Server to remove Client A’s username from its list of usernames and server will send back the nonce decremented by 1 and signed by session key and close the connection.

KAS{N5 - 1}

Logout, KAS{N5}

Client A

Server

Services Provided

# Perfect forward Secrecy

Perfect Forward Secrecy is maintained since Clients generate Public/Private key pair during each user session and during conversation with other clients, a session key is established using the keys generated. This ensures that past communication cannot be decrypted even if the clients are compromised or if server is malicious.

# Denial of Service Attacks

When the Server receives a login request, it will send a hash of the workstation’s IP Address, port and a randomly generated secret to that IP Address. This will make sure that no users are spoofing as other IP Addresses so the Server less likely to be overloaded.

# Addressing Online/Offline Attacks

Online attack is handled in the ideal architecture where if an intruder tries to guess the password by sending login requests, then the server will prevent login for 5 minutes after 3 failed attempts and reset user password after more than 10 login attempts and notify the user.

For addressing, the offline attack, the hashes of the user’s password are stored at the server side. To mitigate the effects when server is compromised, we implement a mechanism in the ideal architecture to change the passwords of all the users and send notification about the change.

Since the passwords are hashed along with salt, our implementation is resistant to offline attacks for most passwords.

# End Points Hiding

The Server keeps a list of usernames encrypted with the Server’s public key. When the user sends his username or request to speak to a user, all usernames are sent encrypted and so an intruder won’t be able to view usernames in plaintext.

# Secrecy from Server

Any conversation between clients will be encrypted with a session key established by the clients. There is no way for the Server to decrypt communication between Clients since the session key will not be known by the Server and will be deleted after each session.

# Application Trust

Since, the client sends the hash of the password, the server cannot derive the password in the event that the server is compromised. This ensures user’s passwords are not leaked.

We recommend users not to use the application if they do not trust the same.

ARCHITECTURE WITHOUT TIME CONSTRAINT

# Additional Services

1. Server can add the validity in the ticket so that ticket will remain valid for particular time period.
2. When W’ is weak and server is attacked, possibility is there to get clear text password by brute force. Server will use pseudo random function which encrypt and store passwords on the server to prevent brute force.
3. We would like to derive the session key established between the server and the client using function computed over the nonce and a derivative of the password.
4. If client is having session for more period of time, key rollover will happen and new session key will be established between two clients.
5. After fixed numbers of login attempt client will be blocked and password reset request will be sent to server.
6. Protection against Online/offline attack

Passwords will contain 8-16 characters and combination of alphabets, numbers and special characters. The Server will only keep a hash of all passwords so even if an attacker finds the hashes; he will not be able to perform an offline dictionary attack since the passwords won’t be that simple.

Since there are far too many variables in a password, intruders won’t be able to easily guess the password. The Server will also prevent login for 5 minutes after 3 failed attempts and reset user password after more than 10 login attempts and notify the user.

1. Server will maintain the status of each client.
2. For DOS attack following recommendations are proposed:
   * Using application firewall which will keep tab on number of connections and will monitor the Incoming connections.
   * Increase the number of concurrent connections:-

When under DoS attack, the number of concurrent connections will be abnormally high. Server should still have to handle them all because legitimate client will also need a connection to reach server. Increasing connection will facilitate server to keep its state alive.

* + Setting Rate Limit

Tools like Ntop which checks the network traffic will help to check number of connections server can handle at a time and set rate limit of the number of connections.

* Increased difficulty level of challenge sent by server.

Server will send hash of 24 bit number and clients has to brute force hash and send the quantity to server.

Flaws in our Project

1. Ticket is valid until user B logout. Validity of the ticket needs to be implemented to avoid this.
2. Active user check should be there from server which will check periodically which users are still online and not crashed.
3. DDoS attack is possible on server as well as client. An efficient puzzle can be implemented which will reduce the risk.
4. An attacker can perform online attack through password guessing using common passwords.
5. If an attacker compromises the server, then he can get the list of all hashes and salt for the users and can mount dictionary attack to get the password. If passwords are weak, then it possible that attacker can get the password. This is handled in the ideal architecture where the length of password is more than 8 characters and a combination of alphabets, numbers and special characters. In the ideal architecture application firewall will detect such break in and reset the passwords of the users and will notify the respective users about the event.
6. The hash of password is used to authenticate, if someone can get the hash by reading the database of server, then he can impersonate the users.
7. It is possible for an attacker in our current implementation to reuse the ticket generated in a previous communication and impersonate a user. This is partially avoided by storing the previous tickets, as no two tickets are the same.