478
End to End Secure Chat Application

EternalBlue

Luke Clements: 014736434
Marc Marcelino: 014625700

The Need

Users want an end-to-end encrypted chat.

Assets:

- Server hardware (AWS), server/client code, user specific information

Stakeholders:

- Users

Adversaries

- The server
- Somebody in their basement
  - Other network traffic analysis
  - Replay attack

Server Setup

- Ubuntu AWS Server
- Nginx
- Node.js
- mongoDB
- Key Pair
  - Allows connection to server via SSH
- Accessed via Git Bash

Design - Client Side

- User public keys will be exchanged offline in person
- User ids will be exchanged offline in person
- User friends will be stored in a json file client side
- Messages to a friend will be encrypted with that friend's public key on the clientside
- The encrypted message will be sent to the server with the toId set to the friend's userId
- Messages are retrieved by sending the user's userId in a GET request and are decrypted clientside

Client Side

- Using Python
- Menu:
  - Menu:
    - Register
    - Login
  - Menu:
    - GetMessages
    - PostMessage
    - DeleteMessages
  - Add Friend

Design - Server Side (Registration and Login)

- Users will register with a username and password
  - This password will be salted and hashed before storage with SHA256
- User login requires the username and password and returns a JWT
- All other routes are secured via the token received in the login process

User Format:

```
{
        id: String,
        Username: String,
        Password: String,        //this is hashed
        createDate: Date,
}
```

Design - Server Side (Messages)

- Messages will be posted with a toId and fromId
- Message data (the actual message) is encrypted before it hits the server
- Messages will be retrieved by finding all messages with a specific toId (the user's id)
- Messages will be deleted by deleting all messages with a specific toID (the user's id)

Message Format:

```
{
        id: String,
        toId: String,
        fromId: String,
        data: String,           //encrypted message
        createDate: Date,
}
```

Attack Surfaces

- Hardware is protected by AWS
- Software
  - Theft of ssh login to login to server
- User info
  - Passwords are hashed
  - Database is not accessible except by require GET, POST, DELETE routes
- MITM
  - RSA keys are never known of the server
    - No manipulation or impersonation of messages is possible
    - Manipulated messages would never decrypt and it would be clear that the message had been changed
    - Encrypted messages can be seen as well as the origin user and to user and the date the message was sent
- Replay attack

Analysis

Benefits:

- Message encryption/decryption takes place on the clientside
- Friend data for specific users is not stored in the server
- User RSA keys are never know by the server
    - The server cannot decrypt any message that is stored on it

Downsides:

- Requires non-server storage for each user (costs the user)
- RSA keys and friendIds must be exchanged in person

Future Work

- More work to mitigate replay attacks
- RSA key exchange not in person
- Deleting messages on server after a set time
- Add Diffie-Hellman for full end-to-end encryption