CS4203 P2 - Report

190022658

November 17, 2022

1 Overview

In this practical, we needed to create a secure messaging system which consists of:

1. the client with which the users interact;
2. a server or broker that puts the clients in touch with each other;
3. a crypto infrastructure that ties everything together.

I created all three components in JavaScript.

2 Design

The libraries used were tls, fs and crypto. I chose JavaScript because the documentation was easy to read and was filled with example. I also had experience in building client and server in JavaScript. There are two components for a messaging application, client and server.

2.1 Security

The components here are tls, certificates and encryption.

2.1.1 Crypto

I used the crypto module in nodejs for encryption. The type of encryption I chose was assymetric key encryption. The key pair were generated using the generateKeyPairSync from the crypto module. This was generated when the client started and the public key was sent to the server upon login. I used the privateDecrypt and publicEncrypt methods to perform the encryption. These implementation are in utils/crypto.js.

2.1.2 TLS

I used the tls module in nodejs to connect the server and clients instead of the net module. This adds a layer of security creating a secure communication channel between the client and the server. The certificate creation process is shown in the README.

2.2 Server

The server has a list of all users (as User objects) which contain the usernames, public keys and sockets. The program uses event listeners from tls server and sockets. It then utilizes those to create new Users (login), send messages, send public keys and sending errors.

2.3 Client

The client keeps all public keys to encrypt the data and uses its own private key to decrypt messages that it receives. It also uses event listeners from tls socket to send message to server, receive publickeys and login.

For ease, the messages sent are of certain types. They are the following form:

```
Message = {
    type: "login" / "message" / "keyinfo",
    // login
    username,
    publicKey

    // message
    receiver,
    text
}
```

3 Functionality and Testing

3.1 Login

```
> p2@1.0.0 client /cs/home/es299/Documents/CS4203/P2
> node client.js

Enter your username: adi
```

```
To message someone       - @<username> message
To refresh public keys   - /refresh
To exit                  - /exit

>> Server: Hello from server, adi!
```

3.2 Sending a message to a non-existant user

```
>> Server: Hello from server, adi!
@someone hi
>> User not online, try /refresh
```

3.3 Sending a message to a user without having its public key

```
-- client 1 --
Enter your username: eeee

To message someone       - @<username> message
To refresh public keys   - /refresh
To exit                  - /exit

>> Server: Hello from server, eeee!

-- client 2 --
>> Server: User connected: eeee
@eeee hi
>> User not online, try /refresh
```

3.4 Sending a valid message

```
-- client 1 --
Enter your username: eeee

To message someone       - @<username> message
To refresh public keys   - /refresh
To exit                  - /exit

>> Server: Hello from server, eeee!
```

```
-- client 2 --
/refresh
>> Keys refreshed!
>> Users online:
>>    adi
>>    eeee
@eeee hello!

-- client 1 --
...
>> Server: Hello from server, eeee!
>> adi: hello!
```

3.5 Shows when a new user has logged in

```
Enter your username: adi

To message someone       - @<username> message
To refresh public keys   - /refresh
To exit                  - /exit

>> Server: Hello from server, adi!
>> Server: User connected: eeee
```

3.6 Exit

```
>> eeee: hello!
>> adi: hi!
/exit
```

3.7 Show when user disconnects

```
-- client --
>> adi: hello!
>> Server: User disconnected: adi
```

3.8 Server closes

All the clients exit in this case.

4 Evaluation

Man in the middle attack has been prevented by the use of tls. Non-repudiability has not been implementated, to implement this the sign and verify methods can be used from the crypto module. Messages cannot be read on the server since the server only receives encryted data and does not have the private keys to decrypt them. It's important to ensure that the server database is secure, incase the public key is changed.

5 Conclusion

This practical was challenging to start out with, but got easier. If I had more time, I would have implemented non-repudiability and database with user passwords. I would also have added a queue to save the messages and user credentials. The last thing would have been to send all the existing public keys to the user when they register and send public key of the user when they register. This would completely remove the use of '/refresh' command and would make it easy to implement non-repudiability.