200014517

CS4203 Practical C02 Report

Introduction

In this practical I was asked to make a 'secure reddit' which I have decided to implement as a simple client server command line java program, I believe I have done a decent job at this. A lot of this practical was up to interpretation and so I will discuss the design choices and security in this report.

System Design & Security

The base idea was to use java (as it is the language I am most familiar with, and it is perfectly capable of doing networking) and create a simple command line interface to send messages to a server which gets relayed to other clients. The actual security of the system comes in the smaller details and the systems simplicity.

KeyUtils.java

The files of the system are broken into a client, server and also a class for a custom keyUtils implementation which encodes and decodes the public key which is the encryption model the system uses.

For the encryption, several options were considered: private key is secure but since there's a potential more than 2 users then it gets messy and complicated to implement, session keys would be useful but hard to make useable in this abstract system, there was also of course the Diffie-Hellman key exchange which would be incredibly secure but since the users aren't known I couldn't easily implement it. Thus, public key encryption didn't have anything inherently problematic to implement.

P2Server.java

The server keeps a port, a list of writers, public and private keys and also a map of user logins and credentials. If given the time the credentials would've been stored in a database with encrypted communications between it and the server.

The main method sets up some usernames and passwords for testing and then tries to open a new server socket object on the given port and initialises the public and private key and runs the server. Every time a connection is made a new client handler is made. The client handler uses an SSL socket and a reader and writer. It simply needs a connection to initialise and when run checks reads the first 2 lines of the connection and checks it's the username and password and then whenever it receives a message it broadcasts it to other users. Since the user sends an encrypted message then it doesn't need to be decrypted or re-encrypted.

The other methods are simply for decrypting passwords and authenticating the users which would be implemented more if there was database integration.

P2Client.java

The client is a lot simpler design wise; it simply connects to the server and sends out encrypted messages and receives them from the server. It takes the username and password from the command line. It then tries to set up a socket connection to the server. If done so correctly the system initialises a reader, writer input stream and byte array for the public key and converts it into the public key.

It then sends out the username and encrypted password as obviously the password needs to be protected to avoid man in the middle attacks. The program then reads any messages from the server and prints it out after decrypting it. The program then allows for the user to write messages and will encrypt and write them to the server.

The threats of this kind of scope of system are man in the middle attacks and intrusion. To avoid that the system encrypts all messages it sends except the usernames, as well as it has a basic authentication system so only registered users can utilise the system. As discussed above public key encryption was chosen as the model of encryption. Diffie-Hellman was a strong contender but wouldn't work for the purposes of this practical. Certification was considered but due to time constraints was not implemented. It would have had to use a stub certification service rather than an actual trusted one but if it were implemented and the service were properly used then a trusted certification service would have been contacted and used.

The main source of security for the system lies in its simplicity, since all messages are broadcasts then a user must be aware of that, the messages are protected and there is a base level of authentication implemented.

Testing

Due to time constraints proper rigorous testing could not be performed.

Conclusion

I was asked to create a secure messaging service and I believe I've made a reasonable attempt despite not having the time to test it.