universidade de aveiro

Security

Professors: André Zúquete, andre.zuquete@ua.pt
João Paulo Barraca, jpbarraca@ua.pt

Grupo p1g8
Álvaro Martins 72447
Alvaro.martins@ua.pt

universidade de aveiro

Abstract

In this project we will try to eliminate some of the problems in today's instant messaging systems, security and privacy. To achieve this we provide end-to-end encryption between clients and between client-server, we use Diffie Hellman, RSA, AES128 and AES256, SHA256 and SHA512 to make sure the encryption is not broken.

Although the problem is not yet solved it was managed to solve some of the core problems

Introduction

Instant messaging applications are very used in today's communications, through which private information flows everyday. By carrying private information it becomes a target to attackers who use the information to get money using stolen credentials or to steal data from a company and sell it to the rivals. Therefore security is very important in this systems.

The main objective of this project is to develop a system that prevents the theft of information so that users can safely communicate. The system is composed by a central server and several clients that exchange information between them.

The system supports as of now this features:

- Message confidentiality and integrity
- Multiple Cipher Support
- Forward secrecy
- Backward secrecy

Message confidentiality and integrity is obtained encrypting conversations between client-server and client-client and the use of HMAC.

Forward secrecy is obtained through the use of Diffie Hellman and Backward secrecy is obtained by switching symmetric keys every message and Diffie Hellman values and RSA keys every session or when the limit of messages is reached

Secure Instant Messaging System

Architecture

The system is composed by two main components: client(N clients), through which users communicate, and a server, which serves as central for all clients to connect:

The server will create a tcp/ip stream socket connection through which clients can exchange json messages with. All messages will always go through the server, which either handles or routes messages to the appropriate client. Internally the server stores some information from the clients like the id, ciphers used between the server-client communication, state, level and some more useful information but never private data from the client.

The client will serve as the access point from where the user will interact with the system. When it starts it automatically tries to connect to the server but not list all the client, that has to be the user to request the client to do it.All communications between clients are end-to-end encrypted, every client has a debug mode where all communications will be in plain text(json).

After the client is connected to the server it counts all secure messages that it receives from the server so when the count number hits 30 the client sends a refresh message to the server to refresh the keys from the session, this is used so the keys are ephemeral and provide backwards secrecy. The same happens to client-client connections however since the client-client communication is not so vital to the system functioning when the clients hit the limit of messages they simply disconnect and connect again refreshing the keys albeit taking more time.

There are two types of json messages exchanged between client-server or client-client:

- Connect
- Secure

Connect messages are used to establish a connection between client-server and consists of key exchanges to provide the basis for encryption between client-server. Both the server and the client must share a supported cipher or the connection will fail. A connection takes exactly 6 phases:

1. The client sends a connect to the server with all the ciphers that he supports
2. The server iterates over the supported client ciphers and picks the first that he supports and sends it to the client
3. The client sends the Diffie Hellman data(primitive root, modulus prime) and his Diffie Hellman public number
4. The server saves the client data and calculates the shared number. Then sends his public number

5. The client saves the server public number and calculates the shared number. Then calculates the HMAC of his RSA public key and sends it bundled with his RSA public key to the server
6. The server saves the client RSA public key and validates it with the HMAC and then calculates the HMAC of his RSA public key and sends it bundled with his RSA public key. At this point they finish the connection

Secure messages are only possible if the client is connected to the server and can be encrypted up to two times, when the message is client-client it is encrypted at client with the destination client data and then the result is encrypted with the server data and when the message is client-server it is only encrypted one time with the server data. Secure messages can carry one of 6 other messages inside:

- client-connect
- client-disconnect
- list
- ack
- refresh
- client-com

Client-connect messages are very much alike connect messages except that they are encrypted from client-server and are used to start a session between client-client. If a client receives a Client-connect but doesn't know the sender he sends a list request to the server. Like the server both clients must share a supported cipher or there will be no connection.

Client-disconnect messages are used to end a session between two clients and when they disconnect they automatically generate new keys for a new session between the same clients.

List messages are exchanged between client-server and are used to obtain the list of clients connected to the server.

Ack messages are used to acknowledge other received messages(for now).

Refresh messages are used to renegotiate all the keys between client-server to keep the system safer. It is solved in 3 phases:

1. the client sends a refresh message with new diffie hellman values, and the asymmetric public key and saves the values in a temporary variable.
2. the server saves the new values and sends the respective new values encrypted with the old keys.
3. the client replaces the old values with the new ones and the process is complete.

While the refresh process is not complete the client will store the messages that were to be sent while refreshing and waits until the refresh is finished and then flushes them.

Client-Com messages are the user text messages sent between clients and contain data that users wishes to send to the other client.

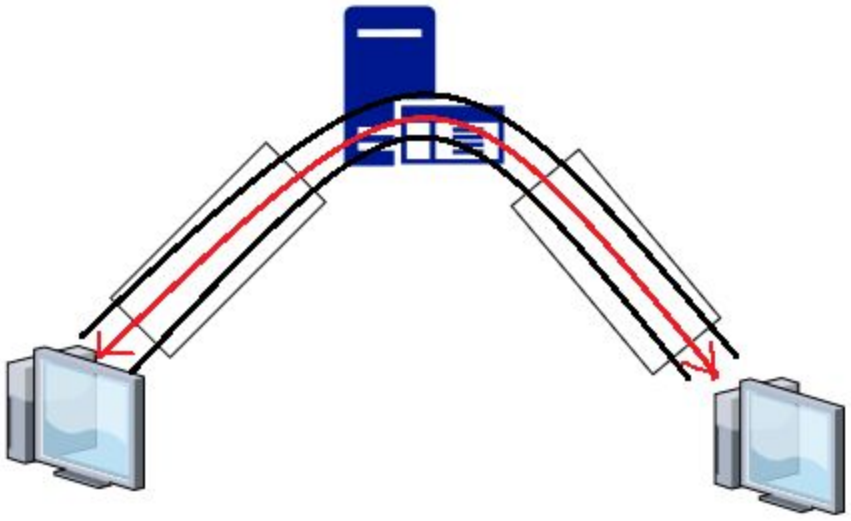

Fig 1. Architecture of the system with two tunnels. the outer tunnel indicates the encryption between client-server and the inner tunnel indicates the client-client end to end encryption

User interaction

The system shows the user a menu containing all the actions that the user can do(Fig 2):

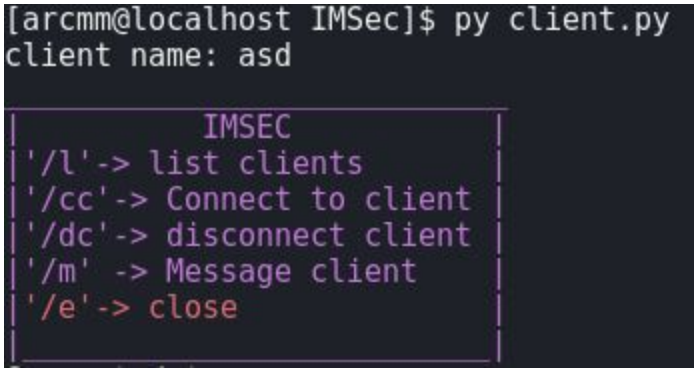

Fig 2. Client menu

- List all the available clients
- Connect to other clients
- Disconnect from another client
- Send message to another connected client
- Exit the system

When a user receives messages from other clients or there is a system status update it is shown in the terminal with hand picked colours to try and make the UI less messy although it is still messy.

Cipher

The system supports 4 ciphers:

- DHE_RSA_AES128_SHA256
- DHE_RSA_AES128_SHA512
- DHE_RSA_AES256_SHA256
- DHE_RSA_AES256_SHA512

The ciphering process is as follows:

1. Generate a key using scrypt(more on this later) and a random number
2. Encrypt the message using a symmetric cipher(AES128 or AES256) with the previous generated key
3. Encrypt the key using the asymmetric cipher(RSA) with the destination public asymmetric key
4. Calculate the HMAC of the result of 2. and the HMAC of the result of 3. Using a key generated by scrypt and the DHE shared key
5. Bundle it all

Cipher choice

In this section will be discussed why the ciphers were picked:

- There was no specific reason DHE was picked over ECDHE. DHE provides forward secrecy and is good at it.
- For asymmetric cipher ElGamal was considered but after some research it was found that both RSA and ElGamal provide similar levels of security but RSA provides a much smaller result. As such RSA was picked with the key size of 4098
- For symmetric cipher AES was considered because of both symmetric ciphers we used at practical class(AES,DES) it was the safest and it is a standard. Both AES128 and AES256 are secure so they are both supported
- For hashing it was picked SHA256 and SHA512 because there are not many hashing methods and SHA is widely used and proven effective at its job.
- Although not in the cipher directly scrypt was used as the KDF to generate the key for the symmetric cipher(derived from a random number) and the key for the HMAC(derived from the diffie hellman shared number). It was picked over PBKDF2 because after some research it was found that scrypt was more secure than PBKDF2

Results

As of now the system is able to successfully provide:

- Message confidentiality, integrity
- Multiple Cipher Support
- Forward secrecy
- Backward secrecy

These problems still remain:

- Destination validation
- Identity preservation
- Information flow control
- Participant consistency
- Conversation consistency

It does not solve all the problems that arise from the original problems however it is expected that those remaining problems will be solved by the next delivery.

Although the system solves some problems it still could be improved in some aspects such as:

- provide a faster way to refresh client-client(maybe replicating the solution for the client-server refresh)
- Use the Ack messages to check if the message reached the destination
- Support for El-Gamal and ECDHE
- Improve the UI of the client for better usability

During the making of this project the concepts of security were explored by the group and it was able to learn about the importance of security and how hard it is to create a perfectly secure system.

References

[1]http://stackoverflow.com/questions/956867/how-to-get-string-objects-instead-of-unicode-ones-from-json-in-python 27/10/2016

[2]Segurança em Redes Informáticas, A. Zúquete ISBN: 978-972-722-767-9