Computer Security EIT060
Project 2: Medical Records and Secure Connections

Jakob Krantz
Christine Boghammar
David Andersson
Klas Sonesson

February 2015

1 Introduction

The point of this project is to build a program with a secure setup and implementation. To learn how to avoid different weaknesses that can make an application breachable and to be able to make it strong against software attacks. The problem that this project is based on is to create a software for a hospital to ensure the functionality to be strictly available to authorized personnel and users being doctors, nurses, patients and outside agencies (governments). The point is to provide access to the hospital database which contains journals and records to whoever is authorized to access it.

2 Project description

The implementation has been developed in a Java environment with a solution using certificates to provide authentication for both client and server. To reach a secure communication between them, this solution is based on encrypted information exchange that goes both ways. The implementation is structured as shown below:

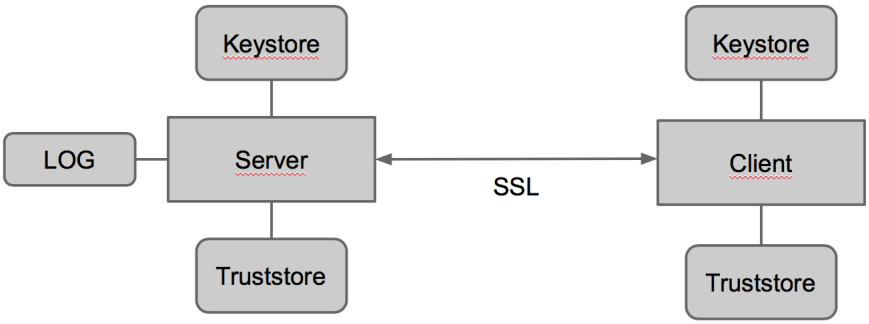

Figure 1: Communication scheme

To get a smoother understanding, some terms that will be repeatedly mentioned in the report will be briefly explained:

- Certificate: A certificate is a file that works as an identification to a specific user. It contains information such as a public key, name, id and other significant data for the user.
- Certificate Authority: An entity that issues certificates. It certifies the ownership of a public key by the named subject of the certificate. When the client and the server tries to make a connection, common CA will ensure a trusted and authenticated connection between the two parts.
- Keystore: The keys used for authentication and data integrity is stored as keystore. There is a keystore for both client and server where the certificates are saved.
- Truststore: A database where the CA:s are stored. It contains information about who to trust and authentication of certificates.

- Log: A file where information regarding the actions towards the patients are kept.
- SSL: Secure Socket Layer protocol communicate through a shared session key for secure transmission of data.

2.1 Secure Socket Layer

Secure Sockets Layer (SSL) is the most widely used protocol for implementing cryptography. SSL uses a combination of cryptographic processes to provide secure communication over an untrusted network. By using SSL you can be sure that the entity with whom you are communicating is really who you think it is. Encrypting the communication provides privacy and therefore addresses the risk of data being intercepted by an unauthorized third party, sometimes known as an attacker. Encryption also prevents attackers from modifying the data that's being transmitted.

2.2 SSL Handshake Protocol

The implementation of the project is based on the communication mentioned above through an information exchange protocol called SSL handshake. There are two main functions with how SSL handles the communication with sending messages through handshakes which are to establish information security by agreeing on encryption mechanisms and authenticate identity. How SSL establishes secure communication is shown in the table below [1].

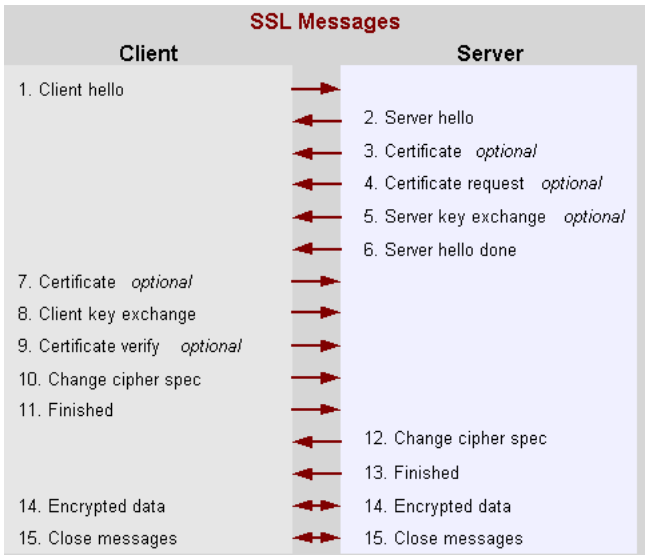

Figure 2: Scheme of SSL handshake protocol [1]

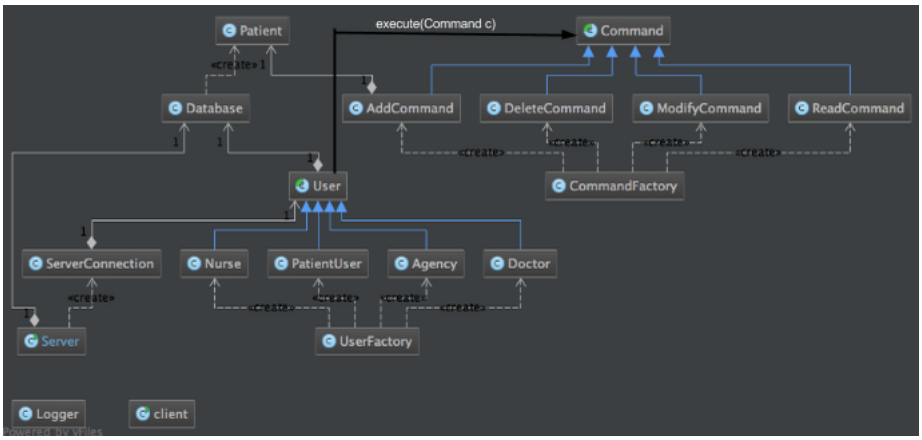

Figure 3: UML of the implementation structure

3 Architectural overview

3.1 Client

The users that are included and handled are doctors, nurses, patients and agencies. They are all given different authentication which will grant access to specific areas of information depending on what kind of user that is connecting to the database.

3.2 Server

The main purpose of the server is to establish a connection to the client being the user. When a client tries to connect to the server, the server is first obligated to check in the truststore if the client is to be trusted. If the certificate is signed by a trusted certificate authority, the server establishes an SSL connection to receive actions from the client.

3.3 ServerConnection

After a connection is set up by SSL, the Server class will start up an instance of ServerConnection and pass on the X509 certificate. This class is a new Thread, allowing the server to have multiple clients connected. When the client sends requests ServerConnection will handle those and send back information whether the client gets access or not and information about what actions the client is allowed to perform.

3.4 User

Is an abstract class which is a representation of the client connected to the server. Subclasses are Doctor, Nurse, Agency, PatientUser. Depending on which subclass is calling a Command, the Command may or may not be executed.

3.5 UserFactory

Used by the ServerConnection class to build a representation of the client as a User object based on a X509Certificate.

3.6 Command

Abstract class which represents a request from the connected client. A command is used and executed by a subclass of User. Subclasses are ModifyCommand, AddCommand, ReadCommand and DeleteCommand, each of those will modify the database according to the type of the subclass.

3.7 CommandFactory

Used to generate subclasses of the Command class by parsing a String received from the client.

3.8 Logger

Used in the server to log client connections, requests and so on.

3.9 Database

Stores all user data in a remote location. Can access and tamper with the data on the remote SQL database.

4 Security evaluation

To evaluate the security of the implementation of the program, this section will go through possible risks and vulnerabilities. As it seems realistic that journalists, relatives and insurance companies would in some extent be interested to get a scope in the confidential records that is held within the hospital database it is smart to try to work against a possible breach by unauthorized users. By listing different kinds of attacks and discussing how they work it can easier be determined whether the program is weakly or strongly protected against being breached.

4.1 Java SSL overview

The main goal of this assignment is the implementation of a two way communication containing sensitive information.To set up and maintain this, a secure connection between the user and server is created using Java secure socket extension (JSSE). The services that JSSE provides are mainly server and client authentication and data encryption, with the help of keystores and truststores and a agreed upon encryption method. The version of Java's Secure socket layer we use is TLS, emphasising that this is a transport layer security, and thus uses TCP/IP. To initiate a connecting using JSSE, a number of messages are transmitted, showing in Figure 2. Where after a brief discovery phase we move on to security settings. [1]

4.2 Encryption and authentication

Encrypting data and authentication is an important step to prevent a range of attacks, but also to cause non-repudiation. To do this SSL uses mechanisms which is divided into a secret and public phases.

The secret phase is based on symmetric cryptography and utilises the fact that the same key can be used for both encryption and decryption. The idea is that both the server and client has a range of enabled protocols which they compare and then agrees upon the strongest one, such as Rivest cipher or DES. Following is the key exchange which is done by for example RSA, which is then used until the session is over or someone initiates an update. These selections are called the cipher suite and is initiated by SSL handshake, which is a part of the socket setup process, but can also be called later to reconfigure the security settings.

The public phase uses a certificate, which is a document containing information about the author as well as a digital signature which is connected to the authors public key. These are, as mentioned, stored in the key and truststores. The digital signature is basically the original data, submitted to a hash function and encrypted with a key. The idea is that all users have their own certificate, which in itself does not provide protection. Instead the certificate has to be verified by an authority. It could be a chain of authorities that signs each others certificates, but eventually some authority has to sign both its own and someone else, this authority is called CA root. So if you trust a CA, you trust the user which certificate is signed by the CA. To verify a certificate, SSL checks a range of things including: Does the user and CA's public keys validate the digital signature? Has the certificate expired? Is the CA who signed among our trusted?

These phases together provide a secure connection, and when combined with a log they compose a versatile protection against attacks and human error. [6]

4.3 Possible attacks

4.3.1 Spoofing attack

The attack called spoofing are when packages is created with a false source address. A malicious party forges and impersonates another device or user on a network in order to launch attacks against network hosts. By impersonating another device the attacker can steal data, spread malware or bypass access controls by displaying a false login screen[2]. The way our program is implemented, the protection of spoofing is almost non existing. However, there are some ways that this kind of attack could be avoided for an example to only allow a few restricted subset of computers to have access to our program. Otherwise packet filtering can be done to prevent spoofing. A filter inspects the packets as they are transmitted across the network as the filter can block packets with conflicting source addresses. Another aspect of preventing spoofing is logging, where it can be seen if a specific user has accessed the system when it actually was a user who was pretending to be someone else.

4.3.2 Man in the middle

The man in the middle attack is sort of a spoofing attack. It is when a third party (man-in-the-middle) is sniffing after packages that are sent on a link between two parties. After retrieving some packages new packets are created using somebody else's IP address. This could be done if the third party would have created a certificate that would have the same CA that signed it and therefor made it a trusted certificate to our connection. [3]

4.3.3 Brute force attack

With brute force, the attack will render through all possible passwords up to a certain length. For this attack to be successful the attack will have to have access to the specified keystore file. Another restriction is that all the certificates have a life length and since this kind of attack is very exhausting the risk of a certificate being expired before it has been breached is very likely [4].

4.3.4 Dictionary attack

A dictionary attack is similar to brute force but instead of trying every possible password this attack uses every password in a dictionary. The protection is almost equal to a brute force attack. [9]

4.3.5 Replay attack

A replay attack is the method of reusing data that has been eavesdropped previously. For example if the attacker manages to intercept a challenge that is identical to a previous one, the eavesdropped response could be sent back and thereby authenticate the attacker. SSL prevents this by using a MAC which is computed for all outgoing data. If the MAC isn't correct, the receiver will know that the data has been tampered with[5].

4.3.6 Buffer overflow

Per definition means that a buffer has used the allocated memory and tries to write more causing the buffer to overflow. This kind of problem is already prevented by using Java's environment which will not allow this to happen. [8]

4.3.7 The human factor / Social Engineering

In the case of a social engineering attack where a person successfully poses as an authenticated hospital-worker or governmental official, the software could be very vulnerable. If this person manages to attain a copy of the truststore and a specific keystore they could potentially do a lot of damage. By brute-forcing the passwords and then for example adding a malicious certificate to the truststore, a man in the middle attack could be feasible.

4.3.8 Time-memory tradeoff

A time-memory tradeoff attack is the process of reversing hash functions by precomputing tables with the different possible solutions. This attack can be rendered infeasible by using salt while hashing the key. Our SSL-connection uses a cipher suite that implements AES-256 which renders rainbow tables useless. [7]

References

[1] http://docs.oracle.com/javase/7/docs/technotes/guides/security/jsse/JSSERefGuide.html

[2] http://www.veracode.com/security/spoofing-attack

[3] https://www.owasp.org/index.php/Man-in-the-middle_attack

[4] http://www.eit.lth.se/fileadmin/eit/courses/eit060/lect/Lect4.pdf

[5] http://tools.ietf.org/html/rfc4346#appendix-F.2

[6] http://docs.oracle.com/cd/E19528-01/819-0997/6n3cs0brm/index.html#aakhc

[7] http://www.cs.miami.edu/ burt/learning/Csc609.102/doc/36.pdf

[8] https://www.owasp.org/index.php/Buffer_Overflow

[9] http://en.wikipedia.org/wiki/Dictionary_attack