Secure Design Software

Report

CROCHE Loïc et COCHEZ Benjamin

1 Introduction

1.1 Description of project

We decided to develop project in Java. This choice isn't trivial because a good advantage is that we are more protected against buffer overflow in Java. Java is also a useful language with secure open source library as "java.security" for example.

It's a client/server project, therefore it's possible to several clients to communicate with the server. A thread is created for each connection to server. Multiplexing in TCP allows multiple connections on our server.

The goal of project is to allow students to access their notes, teacher to access or modify their student's notes and administrator to modify or see any student of any course.

1.2 Security constraints

We must implement several security concepts to protect our application and ensure secure interoperability between each application systems. Security implementations ensure confidentiality, integrity, authentication, availability (CIA) and non-repudiation.

According to course, here is a non-exhaustive list of security constraints which we are taken in consideration:

- Buffer overflow.
- Memory modification.
- Replay attack (resending packets to usurp an identity).
- Password hashed discover (using rainbow table).
- Brute force attack.
- Fuzzing attack (sending random data).
- Spoofing and information disclosing with a Man-In-The-Middle (MITM) attack.

2 Code implementation

2.1 Functional code implementation

2.1.1 Client side

First of all, we are designed project and we are implemented code without security. All request is transferring to server.

2.1.2 Server side

Server receive request from client and uses corresponding process. For example, if client logs in and ask all its notes (a student), server checks into its database to pull all notes correspondent about this student and send it to client.

2.2 Security implementation

2.2.1 Client side

Client generate and use a RSA key on 4096 bits to sign its messages.

We assume that most security are managed by server to avoid attack but also ensure a simple client system.

2.2.2 Server side

AES key is stock in a KeyStore on the server and we assume that server keep the KeyStore locally because server is normally in a trusted environment.

Server implements several security concepts. We explain them below:

- Consolidating user authentication:

  We use multi-authentication with a combination of password/email. Server check user credentials and send to its email a random Integer code which has an exponential time waiting if code entered is incorrect.

  This method permits to avoid **brute force attack**.

- Information security exchange:

  We use SHA-256 as cryptographic hash function to ensure information's integrity, an AES key on 256 bits to ensure information's confidentiality through symmetric cryptography. We use a self-signed X509 certificate to ensure authentication and no repudiation. Certificate is signed with a RSA key on 4096 bits (asymmetric cryptography). We assume that self-signed certificate isn't optimal, but we don't have time to ask a real certificate provided by a certificate authority. Asymmetric cryptography is also used to exchange AES key securely.

  All keys are protected, we choose our key specifications in accordance with (NIST) recommendations and we don't use only asymmetric cryptographic because it's too gourmand in term of performance to transfer data.

  This method allows to mitigate **information disclosure** and **spoofing**.

- Database protection:

  We store all password into a hash ciphered with an AES key. Each hash is salted to avoid disclosing if passwords are stealing and found with a **rainbow table**.

- Memory space protection:

  In java, problems about **memory exploits** are mitigated because buffer overflow isn't present. However, we avoid memory modification by setting to zero all non-used secret variables.

- Data checking:

  We check each message which arrive at server to see if structure message has been altered or not. It avoids **fuzzing attack**.

- **Replay attack** protection:

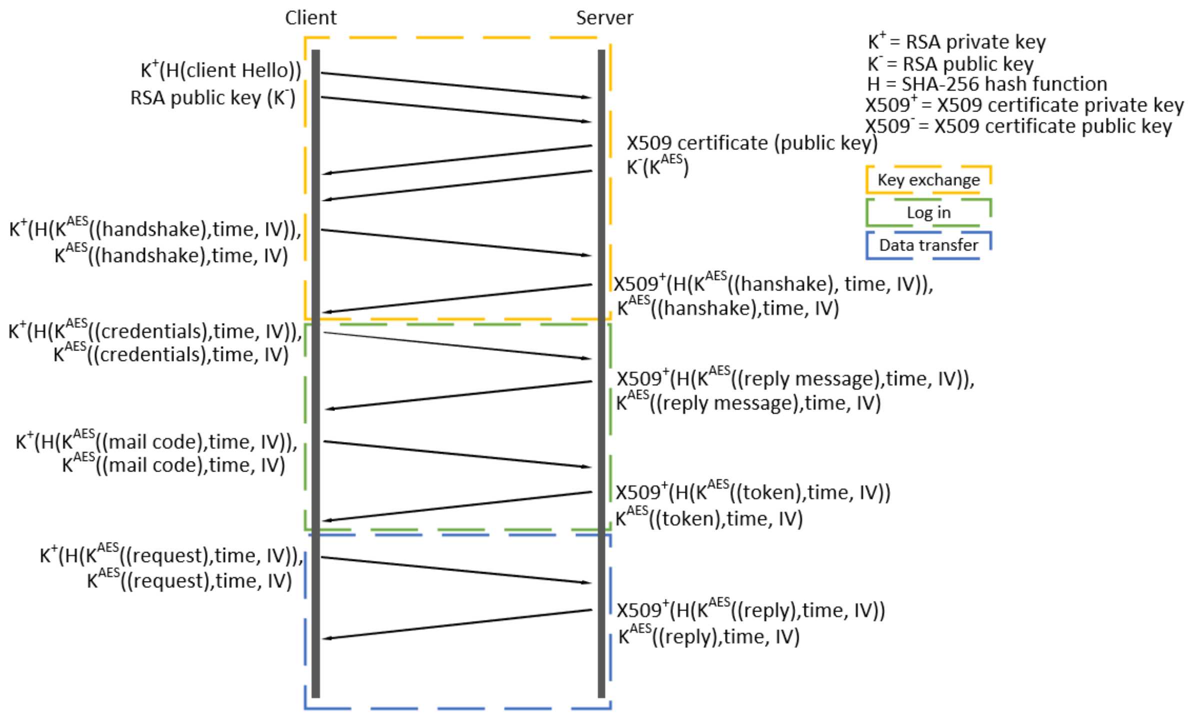
  We implement the time into messages to avoid that a malicious guy intercepts an authentication message and resend it one day later to log in with another user credentials. Each message has a validity of five minutes.

- Separation of privileges:

  We implement separation of privileges with different paths assigned by token and provided by server. It’s important to avoid that a student modify its own grades, for example.

3 How does it work?

3.1 Protocol schema



Client
Server
K^{+} = RSA private key
K^{-} = RSA public key
H = SHA-256 hash function
$X509^{+}$ = X509 certificate private key
$X509^{-}$ = X509 certificate public key
Key exchange
Log in
Data transfer
K^{+}(H(client Hello))
RSA public key (K^{-})
X509 certificate (public key)
K^{-}(K^{AES})
K^{+}(H(K^{AES}((handshake),time, IV)),
K^{AES}((handshake),time, IV)
$X509^{+}$(H(K^{AES}((hanshake), time, IV)),
K^{AES}((hanshake),time, IV)
K^{+}(H(K^{AES}((credentials),time, IV)),
K^{AES}((credentials),time, IV)
$X509^{+}$(H(K^{AES}((reply message),time, IV)),
K^{AES}((reply message),time, IV)
K^{+}(H(K^{AES}((mail code),time, IV)),
K^{AES}((mail code),time, IV)
$X509^{+}$(H(K^{AES}((token),time, IV))
K^{AES}((token),time, IV)
K^{+}(H(K^{AES}((request),time, IV)),
K^{AES}((request),time, IV)
$X509^{+}$(H(K^{AES}((reply),time, IV))
K^{AES}((reply),time, IV)

3.2 Explanation

Secure implement force a key exchange and an access control managed (multi-authentication) before data communication.

3.2.1 Key exchange phase

> A client initiates a TCP connection with server which creates a thread. To communicate securely, client and server must exchange their public keys. First, client generates a RSA pair keys and sends it with “Hello” and hello hashed with SHA-256 and signed by its private RSA key.

> The server receives data and checks if signature is correct. After it, server generate AES key and sends its X509 certificate and the private AES key to client. AES key is encrypted by client public key.

> Client receives X509 certificate and checks if structure of certificate is correct. Then, client deciphers AES key. After it, client sends an handshake ciphered, the time and an IV with the correspondent signature.

  Note: an IV it's an initialization vector. It allows to random AES ciphering. Without IV, a cryptanalyst can be easier find a ciphered message.

> Server receives handshake, deciphers with the AES key and verifies signature. After it, server sends also an handshake ciphered, the time and an IV with the correspondent signature.

> Client receives handshake and check server signature to see if it's correct information and if hash is the same. If it's okay, client and server are both AES key. They can communicate with each other securely.

3.2.2 Authentication phase

> Now client must authenticate it. Client sends its credentials ciphered with AES key, with time and the IV. Correspondent signature is also sent.

> Server receives it, deciphers and checks into its database if mail and password corresponding are correct. If it's the case, it replies to client with "Good credentials" and server generates a random Integer code sent into client mailbox to perform multi-authentication.

> Client receives reply message and must enter the random code sent into mailbox. Ciphered and signed code is sent to server.

> Server checks if code is correct. If it's the case, it pulls in its database type of client connection and sent to client a token corresponding to choose menu (ADMIN || TEACHER || STUDENT). If it's incorrect, a waiting thread is iterated, and client must wait a certain time to resend code. Waiting time is exponential.

> Client receives token and it is redirected into corresponding range (admin, teacher or student).

3.2.3 Data transfer phase

> Now, client and server can send request and reply each other securely because all messages are ciphered and signed.

4 Conclusion

As a conclusion, we are implemented different security concepts into our project. client is able to communicate with server securely mitigating risk of attacks.