Security Project Improvement Report

João Maia, nº 76364

February 4, 2018

Abstract

This document focus on the changes and comparisons between the current version and the previous version of the security project proposed by the teachers of the Security discipline, from the course Telematics and Computer Engineer from University of Aveiro, which consists in a secure messaging repository system.

In this document the reader will find the changes performed on the project and their reasons.

Chapter 1

Introduction

The project described in this report is an improvement from the previous version of the project delivered in 31 of December, 2017.

As the previous project, the main goal of this project is to implement a secure messaging repository system which would be used to exchange messages between two users without the message's contents being eavesdropped or intercepted.

The changes and improvements were made taking in consideration the feedback and the opinions from the teachers of the Security discipline, from the course Telematics and Computer Engineer from University of Aveiro.

Chapter 2

Implemented Changes

2.1 Establishing a Session Key

The first change implemented was the removal of the exchange of public keys messages, since the client will no longer need the server's public key.

In this version, it is assumed that the server has a trustworthy self-signed certificate. Due to the ephemeral Rivest–Shamir–Adleman (RSA) asymmetric keys generated by the server, which are used to create the certificate, a new certificate must be generated every time the server starts running.

The method to establish the session key is still the same, it is used the Elliptic-Curve Diffie Hellman Ephemeral (ECDHE) method, however, in the new version the server sends the public component of the ECDHE, a signature, generated with the ECDHE public component and the server's certificate private key, and the server's certificate.

When the client receives the message, it verifies the validity of the certificate by comparing with a copy stored in a directory with all the trustworthy certificates. If the certificate is invalid, it will terminate the program, otherwise the signature is verified, terminating the program in case it is invalid. If both the certificate and the signature are valid, then the client responds with its ECDHE public component, a signature, created with the ECDHE public component and the user's Citizen's Card (CC)'s private key, the user's CC's authentication certificate and all the certificates from the certification hierarchy. After sending the message, the session key is calculated and the client awaits a response from the server confirming the message was received and valid.

Meanwhile, the server verifies the certification hierarchy and the certificate. If the certificate or the hierarchy are not valid, the server terminates the connection with the client, otherwise it verifies the signature, also terminating the connection in case this one is invalid. If the signature and the certificate are valid, then the client creates the session key and sends a message confirming that the session key was created.

In the previous version, the server and the client would exchange public keys after establishing a connection and before negotiating a session key, signing them with the correspondent private keys. The messages in establishing a session key also were signed with the private keys. This would not guarantee the authentication and integrity of the message from the server, making it susceptible to Man-in-the-Middle (MitM). With the introduction of certificates, the authentication and the integrity of the messages are guaranteed, since certificates are public documents that bind a public key to an entity and tempering with the certificates will invalidate both the certificate and, consequently, the message, easing the identification of the Man-in-the-Middle attacks.

2.2 Sending and Receiving Messages

Another change was in sending and receiving messages. Instead of signing with the client's private key, the message is now signed with one of the two user's CC's private key, in this case the authentication private key. To verify the signature, the authentication certificate is also sent along with the message, accompanied with the certificates from the certification hierarchy, similar to the establishing the session key messages.

When the receiver receives the message, it verifies the certificate and its hierarchy, informing the receiver in case any of them is invalid, stopping the process of verifying the message. If

the certificate and the hierarchy are valid, then the signature is verified, executing the previous procedure in case it is invalid and deciphering the message in the other case.

The use of certificates authenticates the sender and guarantees the integrity of the message, since, as described in the previous section, the certificates bind a public key to an entity and tempering with them would invalidate the certificate and the message, facilitating the detection of MitM attacks.

2.3 Receipts

Another change performed was in the receipts. In the previous version, the receipts were signed with the client's private key, invalidating it after the receipt's sender restarted a session, because of the ephemeral RSA symmetric keys. So in this new version, the receipts are signed with the CC, authenticating the sender and making it valid after the user restarted the session, making difficult to fake receipts.

As in the messages previously described, the receipt message is accompanied with a certificate and the certificates from the hierarchy, being the reasons for this choice the same as described previously.

2.4 User Interface

A change was made in the client's interface, simplifying the interface, making it more intuitive for the user. The option of exiting the program was also added to the list of possible commands.

2.5 Server Databases

The public keys database and the session keys database were also changed.

The public keys database is still stored in a file, but the structure was remodeled from an array into a dictionary, where the keys are the ids given by the server to the client. This eases the process of updating and searching the public keys of each user.

The session keys database structure and values were not changed, although the keys are now the socket in which the client is connected. This change also eases the process of updating and searching the session keys of a client.

2.6 Remove of Login

The previous version had an option at the start of the client program to either create or login into an account. The new version does not support this option, and the login is automatic.

When the server receives the create message, which is now automatically sent after connecting with the server, it checks if the user already exists and reassigns the same id as in the previous session, in case the account already existed.

Chapter 3

Important Notes

3.1 Verifying the Certificates

The certificates are verified by checking if the certificate has expired and if the hierarchy is valid. Both the server and the client possess a directory which contains copies of the trustworthy certificates, comparing certificates received with the ones present in the directory. For example, a client receives a message from another client with a certificate and the certificates from its hierarchy. It will check the hierarchy and see if the anchor is trustworthy, i.e., if it is present in the directory. If the anchor is valid, the certificates in the hierarchy are verified and in case they are valid, they are added to the directory, speeding the validation of the certificate in the next verification. Otherwise, the process of the verification stops and the client informs the user of an error in the verification of the certificates.

3.2 Libraries Used

The libraries used are still the same: the pykcs11 1.4.4, the pyOpenSSL 17.5.0, the cryptography version 2.1.4, among others. Only the library getpass was removed, since there is no longer the need to use passwords in the creation of a user.

3.3 Problems

Even with the implemented changes, some problems still remain from the previous version:

1. **The problem with the ephemeral RSA asymmetric keys** still remains, i.e., if the user A send a message to the user B and the user A restarts the session, the user B won't be able to decipher the message, since the public key it will receive will be the wrong one.
2. **There is no use of a jail or *chroot* mechanism**, so if a server is compromised, the attacker can access important files in the server, besides the ones stored in the directory tree where the program is stored.
3. **The messages saved in the client side are in plaintext** and an attacker can read the messages saved by the user during the process of receiving them.
4. There is no countermeasure against **side-channel attacks**, if a session lasts too long.
5. **There are no permissions to access files**. The user can only access its message box during the use of the application, however, the directories can be accessed using a normal search and browsing, and the files can also be tempered by an user in the side of the server.

Acronyms

CC Citizen's Card

ECDHE Elliptic-Curve Diffie Hellman Ephemeral

MitM Man-in-the-Middle

RSA Rivest–Shamir–Adleman

Bibliography

[1] PyOpenSSL documentation (https://pyopenssl.readthedocs.io/em/latest/api/)

[2] PyKSC11 documentation (https://pkcs11wrap.sourceforge.io/api/)

[3] Cryptography Library documentation (https://cryptography.io/en/latest/)

[4] Python 3 documentation (https://docs.python.org/3/)

[5] StackOverflow (https://stackoverflow.com/)