Assignment 3
Programming and Security

Segurança em Tecnologias da Informação
University of Coimbra

Authors:

António Lima, nº 2011166926

Paulo Pereira, nº 2011164879

June 3, 2015

Contents

1 Introduction

Security is, and should always be, the backbone of modern communications. Due to it secrets can be shared, privacy can be maintained and we can exchange information without worrying about prying eyes. That is, if certain guidelines and measures are followed.

This assignment is focused on developing a chat service that goes beyond the traditional ones by improving communication security through authentication of its clients, integrity and authenticity checking, non-repudiation and confidentiality among other features. The goal is to provide as much security as possible without hindering the system's performance or drastically increasing its overhead.

2 Network Communication Scenario

The following picture demonstrates how our simple chat service works. Any message sent from a user to the server will be replicated and sent to the others users.

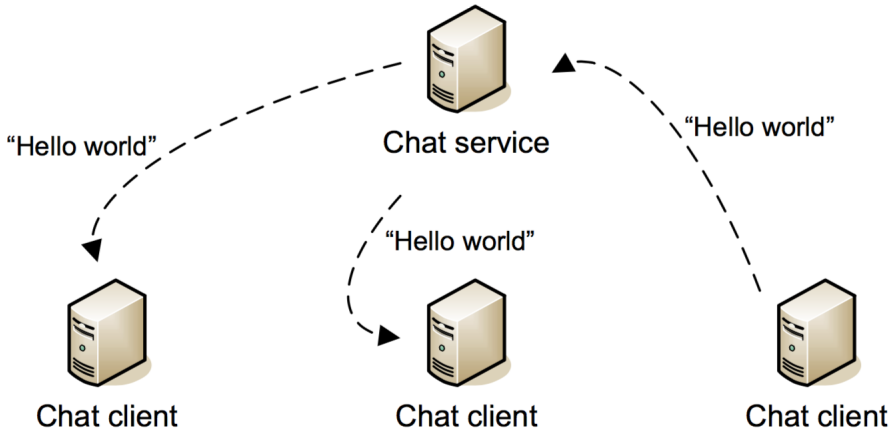

Figure 1: Example of message exchange within the chat service

3 Security requirements implemented and how

The authors decided to use the source code provided with this assignment for a simple chat system. That simple chat is very similar to the previous picture, wherein the user can write any message to be sent to all the participants in the conversation and the ".quit" message to exit said chat.

Alongside the message sharing will be added the necessary mechanisms to **encrypt, authenticate, check for integrity and non-repudiation**. These stages will be described in the following sub-sections.

3.1 Confidentiality

Its purpose is to shield all messages between the server and its users in a way that only the intended recipients and sender can read them. For this purpose, **symmetric ciphers** were the obvious choice as they provide **less overhead** than their asymmetric counterpart. A symmetric key is shared between each client and the server, with the latter being responsible for **generating, dismissing and sharing** said keys with its users.

One problem that arises is how to **share securely** the same symmetric key over the connection. To solve this problem we added a RSA key in the server and when a client connects to the server, the server sends its public key to the client. After that the client creates a symmetric key and sends that key encrypted with the server's public key. After that all the messages sent from and to that client are encrypted with that symmetric key.

Additionally we provided another simple security method by **not** sending the messages in **plain text** but rather **bytes of an an object class** containing the it.

3.2 Authenticity

Authenticity is responsible for verifying that each entity is reliable. To accomplish this task it's necessary in the beginning of a connection to ensure that **each entity is correct** and, after that, that each message in that connection is sent by that entity.

The first part is guaranteed by a very straightforward scheme of **user/password** or by any other similar authentication scheme. In the implementation of the chat the authors consider that there was a CA **(Certificate Authority)**, in which the server and each client have a secure

connection with the CA. It was considered that the CA was in a **secure and trustworthy machine** in which the certificates are sent safely for each entity. In this way, each time a client connects to the server, both the server and the client share their certificates. Afterwards they connect to the CA to check the authenticity of the **exchanged certificates**.

The last part, **ensuring authenticity between messages**, is strongly linked with the non-repudiation and because of that will be explained later on.

3.3 Integrity

Even if with encryption nothing ensures that a hacker cannot intercept and the modify the messages shared. Hence the need to ensures that the content of the message received it's the same that the message sent.

That can be assured with the addition of a hash function. However only with the use of this method it's not possible to ensure that the message was sent by that entity, opening doors to a **man in the middle attack**. So with the use of a pair of keys for each entity we resolved the problem that opens the door to man in the middle attacks. Additionally, this method is used to ensure that the entity sent the message and because of that that entity cannot claim that the message was not sent from him.

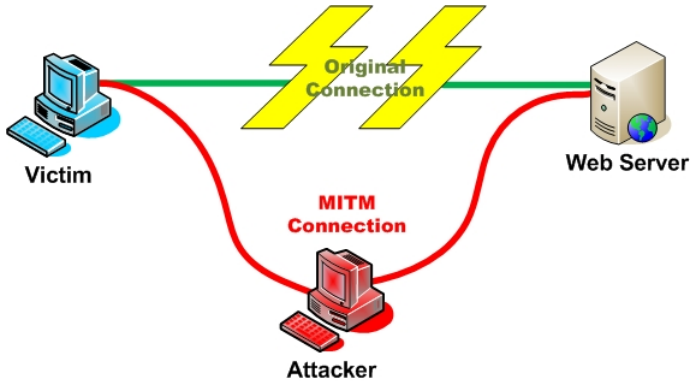

Figure 2: Standard man-in-the-middle attack concept

3.4 Non-repudiation

As mentioned in the previous sub-section the problem that arise with the use of hash functions is that it is not possible prove that the sender did indeed send the received message. One way to resolve that problem is **signing the hash with a private key** that only the sender knows. It is also necessary to **send a public key to the receiver** so that it can verify if the sender was the one who sent the messages.

As we've mentioned, each message that is sent through the Chat Service is **signed by its sender**, therefor, every communication can be **unequivocally related to a particular user**. This way no entity within the system can deny their contributions to a conversation.

3.5 Key management

Unfortunately, although symmetric keys provide security at a lesser overhead than their counterparts, **they can be broken faster**. As such, we implemented a **Key Chain** mechanism within the server which keeps hold of all **valid keys** (one for each client) and that, after having been used a certain number of times are **replaced**, thus diminishing the odds of it being broken while it is still valid.

This validity time frame, of life span if you will, has to be adequate to the type of communications performed. It shouldn't be so small as to represent almost constant renewal, but it shouldn't be so big as to allow for brute-force attacks to be attempted.

Since this is an academic demonstration of concepts we chose a life span of **10 messages** in order to demonstrate the renewal and exchange mechanism taking place, but it shouldn't be taken as the *de facto* standard.

It is worth mentioning that changing the keys requires a synchronizing mechanism in order to avoid conflicts when decrypting other messages.

3.6 Secure management of confidential information

What good is a secure prison if the keys can be attained with little effort? Following this train of thought, the server needs to ensure that all shared keys are securely protected and stored.

In the project built all the shared key and the pair of private/public keys used are generated during the execution and when the program end all these keys are lost.

The weakness of this program it's the use of certificates. In the final version the certificates are save in the machine of the sender. The authors consider the existence of a CA which solves the problem of save certificates in the same machine of the server and of the client.

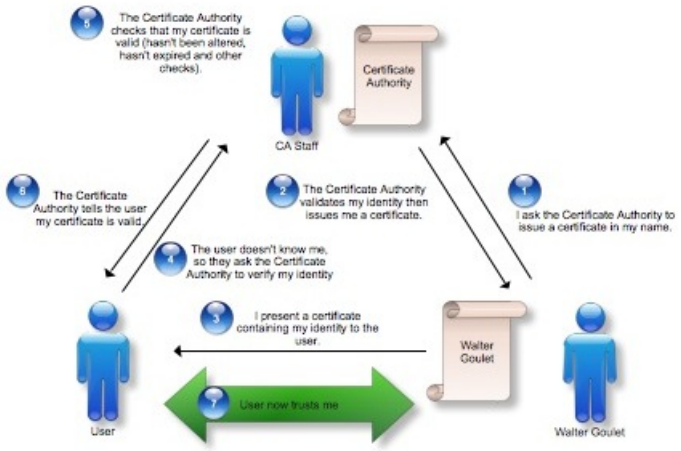

Figure 3: Authentication using a Certificate

That CA was not implemented only emulated and the authors also considered that the CA was in a secure and trustworthy machine in which the certificates are sent safely to each entity.

3.7 Additional Precautions

As we mentioned before, one way to slightly increase the protection of the content in the messages is storing it in an object, thus sending the bytes of said object instead of sending the plain text. In that way if a hacker cracks the encryption he has to discover the protocol of the data. However, it is very likely that if a hacker cracks the connections he is also capable of finding the protocol used by the data.

Another mechanism we also added was the inclusion of timestamps to the messages. This way a hacker cannot replicate previous messages aiming to find a pattern in the messages received.

4 Conclusion

In this project the authors studied and implement several mechanisms to protect from eavesdroppers the shared messages in a service chat.

These mechanisms aim to implement confidentiality, authentication, integrity and non-repudiation. We were also requested to use symmetric keys to encrypt the messages and this requires an additional effort to change the key periodically.

Additionally the authors decided to implement a timestamp mechanism to prevent replication of messages.

5 References

Granjal, Jorge (2013). Gestão de Sistemas e Redes em Linux (3.ª Edição Atualizada). FCA.

http://www.java2s.com/Tutorial/Java/0490__Security/CBCusingDESwithanIVbasedonanonceIn
htm

http://www.java2s.com/Tutorial/Java/0490__Security/RSASignatureGeneration.
htm

http://www.java2s.com/Tutorial/Java/0490__Security/RSAexamplewithrandomkeygeneration.
htm

http://www.java2s.com/Tutorial/Java/0490__Security/GenerateX509certificate.
htm

http://www.java2s.com/Tutorial/Java/0490__Security/Validatecertificate.htm

http://ldapwiki.willeke.com/attach/Certificate%20Validation/pki-simple-diag.
jpg

https://www.owasp.org/images/2/21/Main_the_middle.JPG