Technical University of Denmark

Authentication Lab

02239 - Data Security

Group 7

Authors

Student Number:	Name:
s202789	Adamantios Alexandros Kounis
s202518	Christos Grigoriou
s213685	Nikolaos Karavasilis
s202074	Joachim Morken

October 29, 2021

Introduction

In the context of this lab, a short description of authentication in client/server applications is presented in order to help the reader understand the problem. In a client/server system, the goal is for the former to communicate with the latter. In this report we are going to address the problem of how these entities can establish and maintain a secure communication throughout the session. This is attained when the client can make authenticated requests to the server.

An issue that emerges here, is whether the communication protocol used among entities is secure enough to deter attacks. For example, using the unsafe *http* protocol in a client/server application gives chances for an intruder to interfere and intercept the messages being exchanged. In our client/print server system, the attacker could intercept messages (passwords, files etc.) sent from client to the server. Thus, integrity and confidentiality compromise entails in a non-secure application.

In order to eliminate such "man-in-the-middle" attacks, a secure *https* protocol can be used. However, this would allow any benign client (even a malicious one) to use the services provided from the server. In other words, the server needs to accommodate a mechanism which allows only registered users to access and use its services.

Our solution focuses on problems such as transmitting the passwords from the client in a cryptographic form, in order to avoid sending passwords in plain text. In our proposed client/print server system, the authentication of already authorized users takes place on the server's side. Clients that are enrolled in advance have their information stored in the database accessible by the server. These users are the only ones allowed to send files to be printed. After a client requests to log in, the server authenticates the user and upon successful completion it replies back with an access token. Essentially, this token allows the client to use the services provided by the server. Subjects of password storage, transport and verification are discussed in the following sections.

Authentication

The authentication mechanism in a client/server system is undoubtedly a matter of great interest. For decades it has been the only mechanism in order to accomplish authentication. However, the past couple of years, passwordless ideas have risen and already been adopted. The main reason of this phenomenon is the the fact that password based authentication has some disadvantages. According to Microsoft[1] [2] for a password to be effective it needs to be at least 8 characters long, use both upper and lower case characters, contain a combination

[1] `https://docs.microsoft.com/en-us/windows/security/threat-protection/security-policy-settings/password-must-meet-complexity-requirements`

[2] `https://docs.microsoft.com/en-us/microsoft-365/admin/misc/password-policy-recommendations?view=o365-worldwide`

of alphanumeric characters and symbols and not be used across different websites. These requirements outline some of the drawbacks of password based authentication:

1. **Weak passwords:** People usually find it frustrating to change or reset their passwords. Similarly, having to remember multiple of them is also not an option. As a result, re-using passwords or picking naive ones is very common, and thus simple and weak passwords are picked that are easy to remember.

2. **Leaked passwords:** Although the client might have been conformed with the guidance of creating secure passwords, there are cases where the server is not able to safely keep them. This leads to leaked passwords from the server side.

3. **Cracked easily:** When a client uses weak passwords, then it is not difficult for the attacker to crack them. This can be accomplished either by brute force or password guessing.

4. **Unable to recall passwords:** A problem with password based authentication is the common phenomenon of forgetting the password accompanied with the difficulty to reset them. In many websites, regaining access after forgetting the password is not even supported or requires other information in order to complete the identification.

The fact that our implementation has a password-based authentication for the principals, raises a discussion for password storage, secure transport between the client and the server, and verification. This section describes the alternatives of password storage options and concludes with the choice that our group decided to include in the solution.

Password Storage

The system's server must first authenticate all clients' requests before proceeding in printing tasks. All users that are allowed to use the print server of the small company are enrolled in advance. Thus, server first authenticates users against their credentials upon requests. The options for storing user's credentials are:

- **System File**: Using system files to store passwords means that the files are kept locally in the server's system. The operating system of the server stores the passwords in a hashed form and is unable to read them in plain text. When new passwords are created they are hashed and stored in the system file. The authentication is accomplished by comparing the hashed value of the submitted password with the stored value in the system file. Since the file is kept in private inside the system, accessing it requires certain privileges that often only the administrator possesses.

  Modern operating systems offer password storage mechanisms in system files in a way similar to a database. Examining the Windows OS (XP, Vista, 7, 8.1 and 10), the Security Accounts Manager (SAM) database [3] is responsible for storing user passwords

[3] `https://en.wikipedia.org/wiki/Security_Account_Manager`

in several formats (plain text, hashed and salted, encrypted). Passwords are stored using the LM or NTLM[4] hashing algorithms.

The disadvantage of this strategy is that the system is vulnerable to potential malicious threats that can cause harm in the system. In our system, this is interpreted in password retrieval from an attacker.

- **Public File:** Using public files to store passwords means that a lot of users has access to them. A way to implement this would be to choose secure hash functions. Furthermore, choosing a sufficient salt and hashing the joined value of the salt and the credentials, prevents password guessing attacks. Therefore, each hashed value would be different among pairs of username and password. Since basic users do not have write privileges to the public file, the server should have administrator rights to the file.

- **DBMS**: Maintaining a database where user's credentials are kept seems to be the best solution for several reasons. Firstly, the database is stored in a server which is a third-party entity, independent from the client/server system. Every time a user attempts to log-in the system, the printer server first queries the database to verify that the user is registered with the system and subsequently provided that the given password is correct, the user is allowed to use the printing services.

  With the use of DBMS we avoid data redundancy since there is a single database (one table in our case) with all users information needed. On the other hand, with the system / public file approach, in some cases multiple files are stored to keep the different user information and most of the times in different locations in the system. Although designing a database is more complex and time consuming compared to a file-based system, it consequently offers easier data entry, storage and retrieval.

  Since DBMS operates as a server it can be extremely help-full for the discretion of data that it follows and enforces a set of rules (constraints, triggers etc.). Moreover, user updates like password changes are easy to make and directly reflected in the whole application.

  Finally, DBMS gives the ability to use timestamps. This way the system is aware of both user registration and every database access. Subsequently, a user cannot convincingly deny that a print job has been requested as the system can trace related actions. Thus, non-repudiation and auditability security aspects are covered as well.

- **Choice of our solution**: Our groups decision has been made after discussing and evaluating how potential harmful actions could affect the application. Evaluating the three basic security properties we proceeded with a DBMS password storage option. Hence, in case the print server is compromised, integrity and confidentiality are still maintained since passwords remain consistent as they are stored remotely in a third-party entity. Considering the availability aspect, our system relies on the DBMS provider who is responsible to have the database secure and available to the database client

[4] `https://en.wikipedia.org/wiki/LAN_Manager#Password_hashing_algorithm`

(our system's print server). Threats that can cause harm through natural disasters, could of course compromise availability. However this is part of our risk management policy, it cannot be covered by any control and constitutes residual risk of our system.

Password Transport

A prerequisite of client/server interaction is that entities communicate in secure channels. We adopt the assumption that TLS protocol is used, and secure connection is established via a three-way handshake. Hence, we eliminate *replay* and *"man-in-the-middle"* attacks, which are the dominant problems of password transport. This way confidentiality, integrity and authenticity guarantees are met. There are two alternative options for exchanging authenticated messages between client and server, individual request authentication or authenticated sessions.

- **Individual request authentication:** This implementation is stateless, since the server does not keep a history of successful connections with the clients. On the other hand, each client keeps a state of his connection with the server. As a result, client needs to authenticate himself in each request. A token-based authentication approach is the most common practice that satisfies individual client authenticated requests. As a first step, the client attempts to log in to the system providing username and password. Subsequently, the server verifies that the client is registered before he responds with a signed generated token. Token is stored in the client's side in order to be included in every future interaction with the server. If the token is valid the server handles the request. Lastly, when the user logs out, the token is deleted on the client's side.

- **Authenticated sessions:** This implementation is stateful, meaning that the state of the authenticated session needs to be stored both in the client and the server. Specifically, it constitutes a prerequisite for the server to keep records of the active sessions, as well for the client to store the session identifier. As a first step, the client provides his login credentials. Then, the server verifies that the client is registered and subsequently creates a session which is stored in a database or in memory. Afterwards, the client stores the session identifier as well. In every future interaction with the server the client sends the session id alongside with the request. Then the server validates the session using the database or the memory. Lastly, when the user logs out, both the client and the server destroy the session registration.

Password Verification

The implementation of our system uses a database password storage. The database accommodates two columns consisting of usernames and passwords. Having the username field as the primary key of our database we ensure that there can only ever be at most one row that can equal a specific pair of username and password. The server verifies each user against the database.

Design and Implementation

The implementation of the authentication lab follows a 3-tier like architecture. The project consists of the server and client side with the communication of the two being done with Remote Method Invocation. The application also consists of a database tier (JDBC connection) which is used for storing the authorized users and for authenticating them upon the login feature. The implementation approach we followed regarding the password transport, is a stateful one, using authenticated sessions.

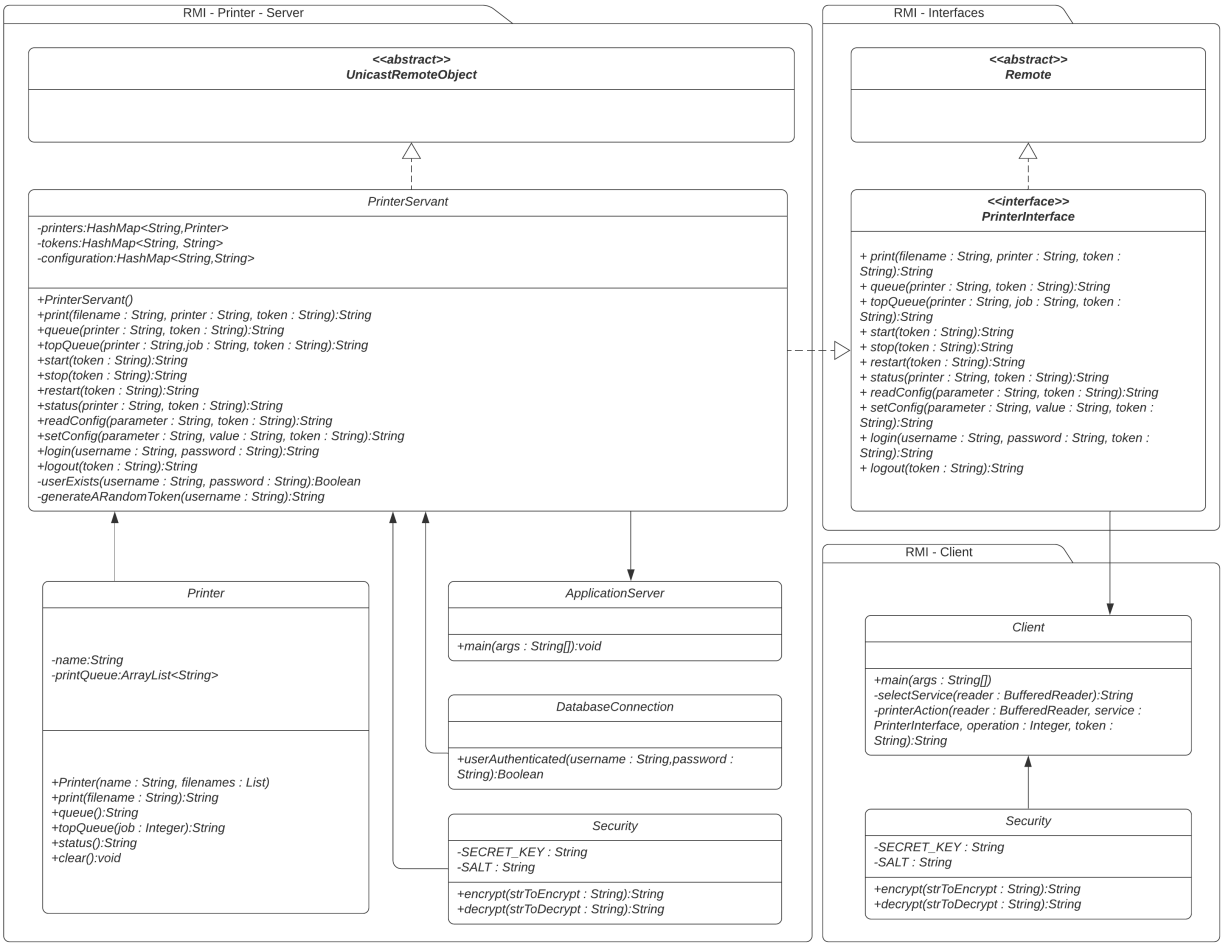

Figure 1: UML Diagram

As shown in the UML diagram in figure 1, the project is broken down to 3 sections. Those include the RMI - *Printer - Server*, RMI - *Interfaces* and the RMI - *Client*. The RMI - *Interfaces* section consists of the PrinterInterface interface and the Remote class that it extends to. It consists of all the methods (functionalities) of the printer server as well as the login and logout methods. In the RMI - *Printer - Server* section, the PrinterServant class implements all the methods from the PrinterInterface class. The implementation is achieved by also using the Printer object and the methods it contains, the Security class which holds

the encryption and decryption methods used in all of the messages, the DatabaseConnection class which makes the verification of users possible and lastly the ApplicationServer class which is used to instantiate the server side of the system. Lastly, there is the RMI - *Client* section.

This section contains the main method in which there is an infinite loop that presents the interface to the user, as well as certain helper methods for invoking the methods remotely and presenting the functionalities of the printer server. The section also includes the same Security class as the server which is also used for the equivalent encryption and decryption of all communication between the server and client. The implementation of the Security class is implemented by encrypting all parameters of all the methods and then decrypting the result that is being returned. The encryption is done using AES, which is an advanced method for ciphering. The AES encryption method was selected, because it constitutes an appropriate method for sensitive data. The padding scheme we used is the 256-bit variant of AES.

On the technical aspect, our DBMS was implemented using the Azure Services, which offers a TLS protocol that exists within the service which results in encrypting the data that is being transmitted between our printer server and the database server. The encryption that comes along with it offers a self-signed certificate that uses a 128-bit encryption which is not easy to break. Moreover, Azure offers firewall settings for the database server. Therefore, the connection to the database that needs to happen first, cannot be successful if it is being initiated by an unknown printer server instance with a foreign to the database server IP-Address. For the purposes of this lab this setting is turned off, for the convenience of any user that wishes to test the functionality of the system.

Evaluation

This section demonstrates how the system's requirements are satisfied by presenting screenshots showing the functionality of our implementation. The user initially provides username and password to log in the system[5]. Provided that the user's credentials are correct they are being granted a session token with which they can trigger all the functionalities of the printer server. The generated token is printed on the console for demonstration purposes as this can be observed in the first screenshot. This token is being passed as a parameter in all the methods that the client invokes onto the server this way authenticating him with each one of them.

[5]In order to test our system's functionality one can provide:
username: username
password: password

Screenshots from console output:

```
Welcome to the Printer software
Input your Username to login :
christos
Input your password:
s202518
TuJevy6AQtS7lVdcK13TFtDsSFdene
Successfully Logged In
Select a service you would like to use : (Type in Index)
(1) - Print
(2) - Queue
(3) - Set file to top of the Queue
(4) - Start Server
(5) - Stop Server
(6) - Restart Server
(7) - Get Status
(8) - Read Configuration
(9) - Set Configuration
(10) - Log Out
```

Welcome screen that prompts the user to choose among indexes after a successful login

```
1
Write in the file to print
file1
Write in the printer to print from
printer1
Printing file1 on printer1
```

User attempts to print file1 using printer1

```
2
Write in the printer to select the queue from
printer1
Listing the print queue for printer1
Job number: 0, file name: file2
Job number: 1, file name: file3
Job number: 2, file name: file4
```

User requests the print queue for printer1

```
7
Write in the printer in which to move the file to the top of the queue
printer1
Printer is online +
Printer: printer1 has 3 files in queue
```

User requests the status of printer1

```
2
Write in the printer to select the queue from
printer1
Listing the print queue for printer1
Job number: 0, file name: file4
Job number: 1, file name: file2
Job number: 2, file name: file3
```

User moved job 2 on the top of the queue

```
6
Shutting down server...
Clearing print queue...
Print queue cleared
Starting server...
```

User restarts the print server

```
2
Write in the printer to select the queue from
printer1
Listing the print queue for printer1
```

User confirms that queue is empty after restarting the print server

Conclusion

With the RMI implementation and the secure authentication between principals, initially with TLS and subsequently with tokens, we can infer that the project's requirements are satisfied. When a user provides the correct credentials, the print server replies with a token. The server also saves this token for as long as the session is active, and thus an authenticated session is in action. In all of the following requests the client also attaches his token. Users that are enrolled in the system, after authentication they can request print jobs, and use all other methods (queue, topQueue, stop, start etc.) of the server successfully. However, in case the user is not registered, the print server will not create a token and thus reply with login attempt failure (unauthorized status).

Our implementation supports passwords of different size and each password can contain alphanumerical and special symbols. Another key component of our implementation, is encrypting the credentials before sending them to the server. Although, we have assumed that a secure communication channel is established using TLS, avoiding the transmission of the credentials in plain text, adds an extra layer of security. Furthermore, the server is capable of having multiple printers connected and every job can be submitted to a specific one. The database is designed to provide username uniqueness, which means there can not be two users with the same username. Since the server keeps a state of the active session tokes with the clients, upon restart or crash all of the tokens will be destroyed and the users will be instantly logged out.

Future work: During this project we have put emphasis on providing a secure way for authenticating the users. In the future, we could focus on the passwords themselves. For instance, the server could only accept passwords with some characteristics, such as making it necessary that each password is at least of a specific length and contains both alphanumerical and special characters. Also, is could be obligatory that the user avoids names or words to be included in his password. Although, these would cause inconvenience for the users, it would contribute significantly towards preventing password guessing attacks and brute force methods.

Finally, one of the reasons we used DBMS for password storage is that it can easily facilitate the development of new features added on our system. For example, we can enforce our security policy by establishing access control where users' privileges differentiate according to their identity in the system's hierarchy.