Ecole Supérieure d'Informatique
HE2B/ESI

Report

Luyckx Marco 496283
Taes Julien 474196

Professor: Absil Romain

Course : Secure software design and web security INFO-Y115

Delivered in January 2023

Academic year 2022-2023

Table of contents

1 Introduction

For our project of INFO-Y115, we have to implement a small system allowing users to **securely** handle an agenda made of events, under a client / server architecture. The main (and only) aspect that is going to be evaluated is the **security**. With this in mind, every decisions that we took was made with consideration for the **threat model**.

In this report, we will delve into the various security measures we have implemented.

We used **Express 4.18.x** to do this project (instead of another framework like Django or Laravel for example) because, after looking for the best, minimal and flexible framework that provided a robust set of features (regarding security and scalability), we found that Express would fulfill our needs. (Also, we both wanted to realize the project in JavaScript.)

We choose that particular version of Express because it is the latest security patched version at the moment.

The git ssh url of the project is : `git@gitlab.com:maluyckx/ssd_online_agenda.git`

For testing purposes, we assumed that you are using a Linux distribution with Docker.

2 Setup

In this section, you will find all **important elements** that we used for the project.

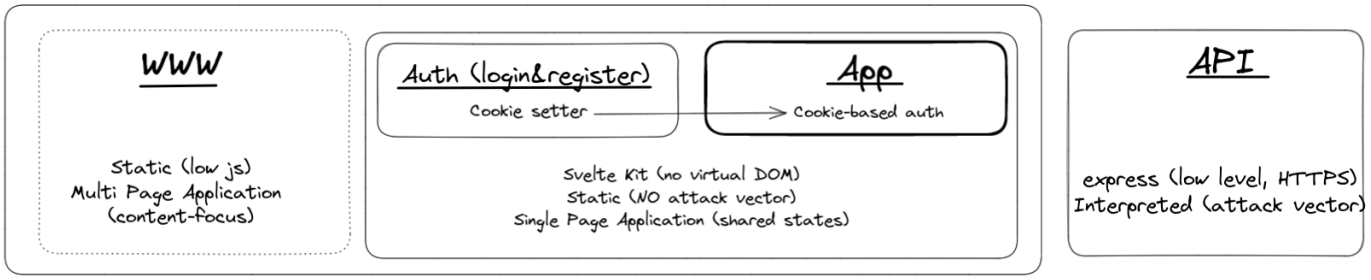

Figure 1: Architecture of the project

Depending on which state the user is, he has access to different things :

Public : A random user can see the contents of the "www" and "auth" directories, but cannot access anything else.

Logged in : Upon logging in, you will have full access to the "app" section.

Admin : As an administrator, you will have access to all areas of the system, including the database. However, all data in the database will be encrypted to ensure the confidentiality of users, even in the event that the administrator is malicious.

Of course, all the permissions are cumulative (for example, an admin can see everything that a public and logged-in user can see).

Below, you will find justifications for each of our decisions :

We used **NodeJS** because it is the only one that is made to do full-stack development and because it is really easy to deploy a server in NodeJS with TLS.

To implement our database, we choose **Better-sqlite3** because it is really lightweight, easy to integrate, self-contained and has a small footprint.

To encrypt and sign our data, we choose **OpenPGP.js** because, it is a widely-used encryption standard (which means that it has undergone thorough review and testing by the security community), open-source (which means that people review the code and ensure that it is secure and free of vulnerabilities) and because it has a really simple API which makes it easy to integrate into your app.

Regarding the parameters that can be tuned in OpenGPG, we choose the **curve P-256** for the performance (fast, making it suitable for the use in our application that needs to be efficient). Additionally, it is natively supported in most browsers as it is compliant with standards (NIST).

All usernames are hashed using **SHA-256**. To make it less susceptible to generic rainbow attacks, we added a string "SuperSec0reUsern4meIs" to the front of the username before hashing it. The resulting hash is then unique and should not be present in rainbow tables. For passwords, we used **pbkdf2** with **SHA-256** and employed the same technique, adding the string "SuperSec0reP4ssw0rdIs" to the front of the password before hashing it.

To protect against excessive or abusive usage (DDoS or brute-force attack) and ensure the availability and responsiveness of our system to legitimate users, we have implemented a rate limiter using Express called **rate-limiter-flexible**. This rate limiter operates using tokens instead of using fixed or sliding window.

To prevent user from inputting 'simple' (and by 'simple', we mean obvious, stupidly short and predictable), we used **Zod**, which is a TypeScript validator.

For example, when a new user wants to register, his password needs to :

- Be at least 3 characters long;
- Be at most 32 characters long;
- Contain at least 1 uppercase;
- Contain at least 1 digit;
- Contain at least 1 special character;

In order to streamline the deployment process for both ourselves and the end user, we have implemented the use of **Docker** to automate the build and launch of our project. We made it using Debian. We initially considered using Alpine, but ultimately decided that Debian was more secure. Additionally, the entire operation system is in read-only mode, providing an extra layer of security.

To enhance the user experience, we implemented the use of **cookies**. However, it is important to note that if someone were to obtain another person's cookies, they would be able to access

the same information that an authenticated user can see, but not sensitive data belonging to the individual from whom the cookies were stolen.

To insure that the cookies are safe, we added security features :

- **expires** : in 15 minutes;
- **secure** : true;
- **httpOnly** : true (to prevent access to cookie values via JavaScript);
- **path** : "/" (indicate a URL path that must exist in the requested URL in order to send the Cookie header);
- **sameSite** : "strict" (ensures that the authentication cookie isn't sent with cross-site requests, it provides some protection against cross-site request forgery attacks).

Concerning the expiring date, we store **absolute dates** in the database but display the current time of the user. This is to prevent a malicious user from altering the time zone and prolonging the life of an outdated cookie.

There is a potential attack vector that we are aware of. If a malicious user steals the **session cookie**, which contains the password stored in plain-text, before the cookie is flushed, the malicious user would be able to log in with the stolen credentials.

In making our decisions, we prioritized options with **minimal dependencies** and **low-level** implementation in order to reduce the attack surface.

3 Features

We divided the various characteristics of our system into smaller components in order to provide a more thorough explanation of how they work. The different categories are Account, Contact, Agenda and Miscellaneous.

To improve the clarity of our explanations regarding our application, we have created some diagrams using Excalidraw.

Important note : we are using Javascript in **"strict mode"** to enhance security.

If you attempt to access our web app using only HTTP, you will be redirected to HTTPS.

All data is transmitted over **TLS** to ensure that private and sensitive information cannot be viewed by eavesdroppers or hackers.

3.1 Account

We have hashed the username to prevent it from being revealed in the event that the server is compromised. This is to protect against dictionary attacks (assuming that a database containing the username has already been leaked).

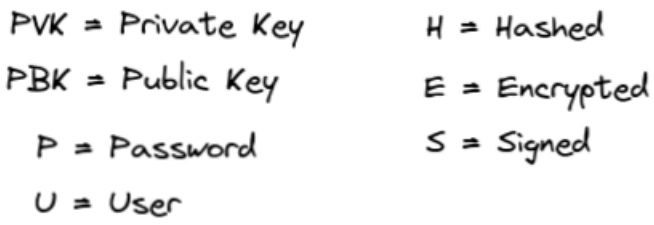

Figure 2: Nomenclature used for the drawings

3.1.1 Registration

To register, a user must provide a username and password. They may also choose to provide their real name and email address, although these are not required. For the purposes of this report, we will refer to those as "credentials".

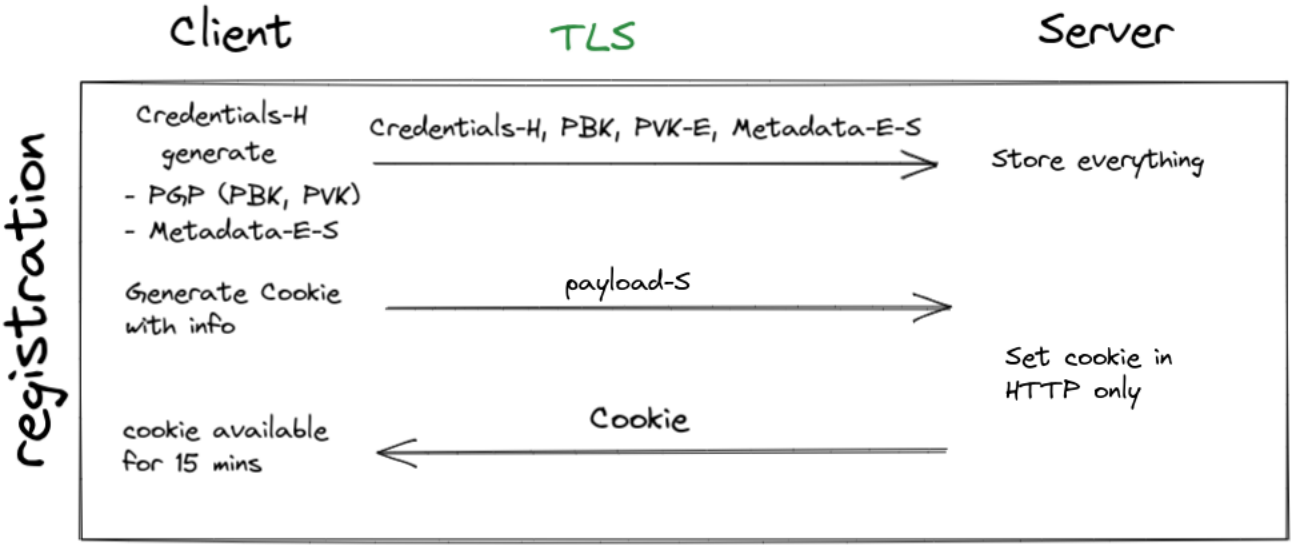

Figure 3: Registration

3.1.2 Authentication / Log In

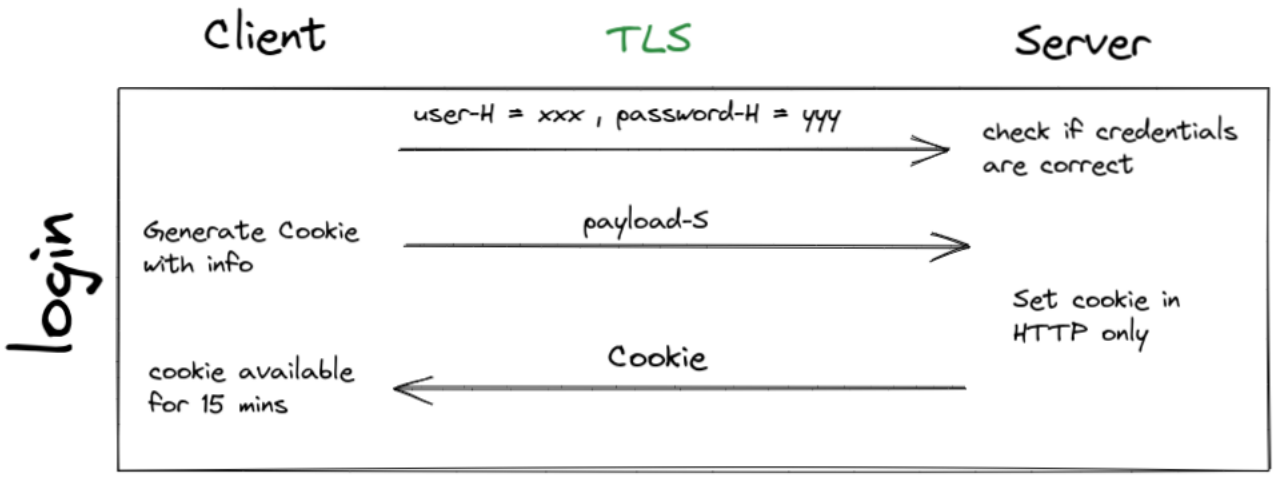

Figure 4: Log in

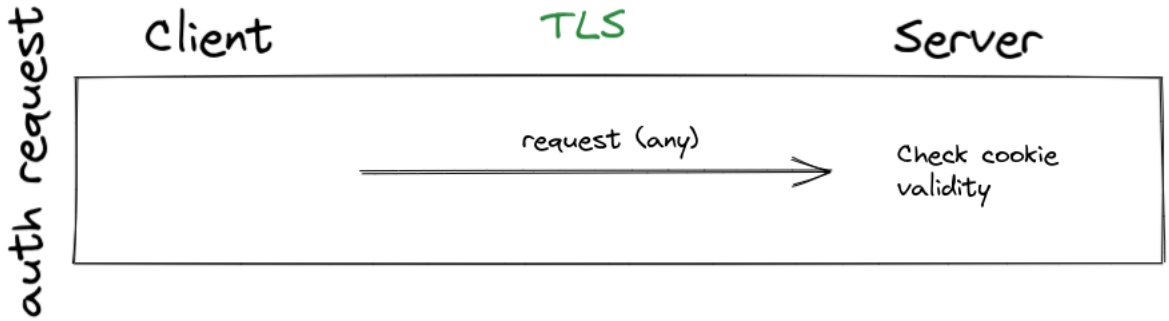

Figure 5: Authentication

3.1.3 Revocation

The revocation process involves deleting all records associated with the username from the database.

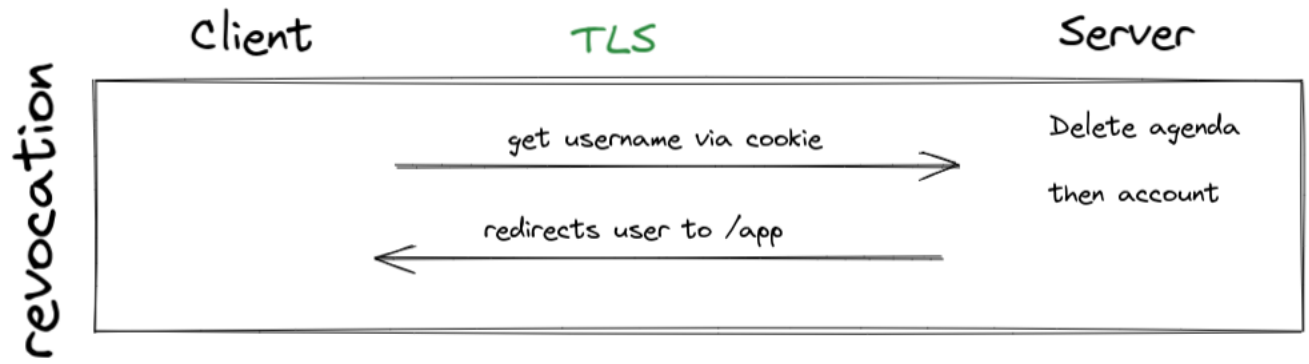

Figure 6: Revocation

3.2 Contact

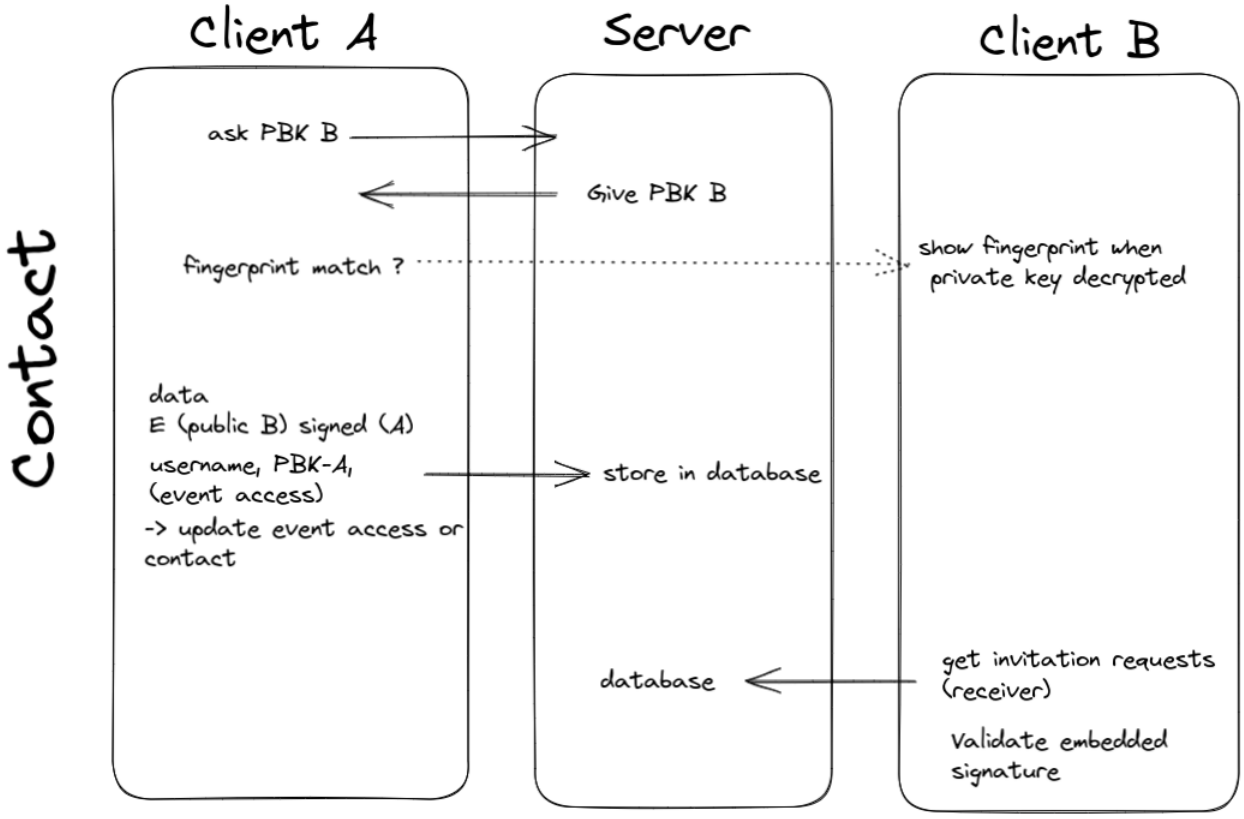

Figure 7: Invitation

3.3 Agenda

An agenda is a collection of events. We have added a feature that allows a user to have multiple agendas, each with its own set of events.

3.3.1 Creating events

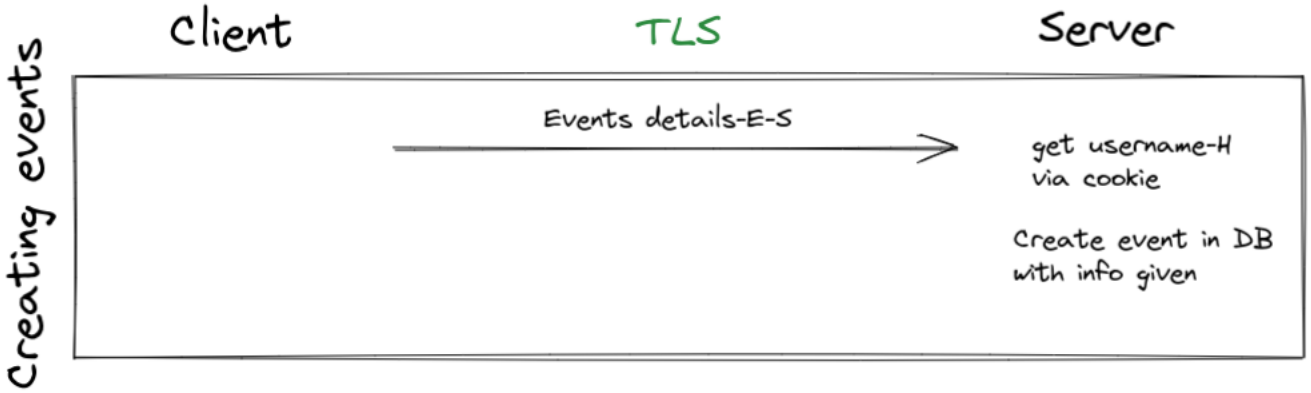

Figure 8: Checking events

3.3.2 Editing events

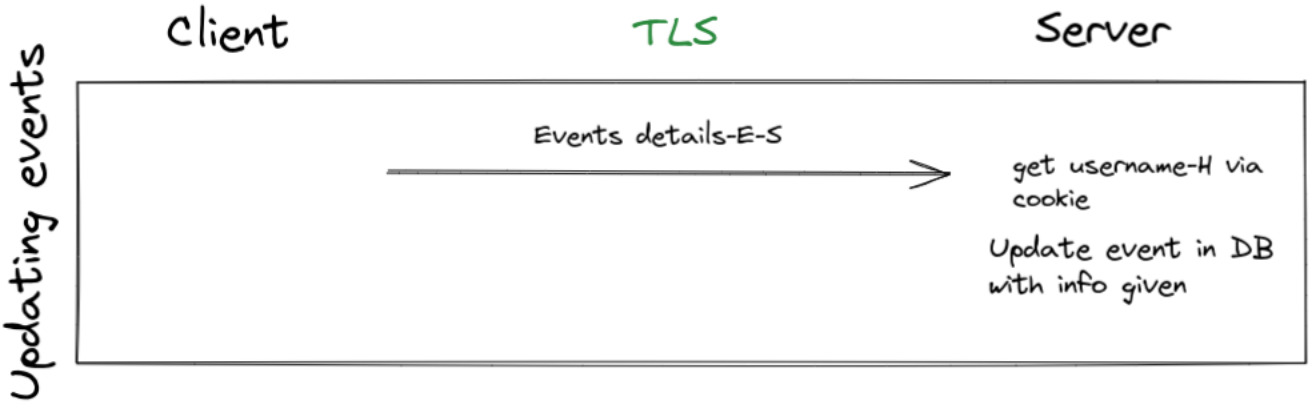

Figure 9: Editing events

3.3.3 Deleting events

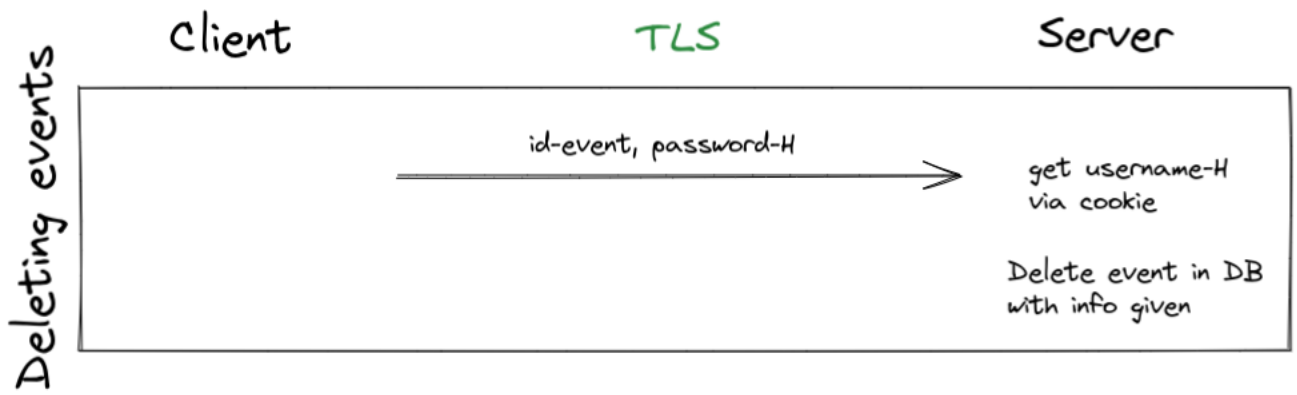

Figure 10: Deleting events

3.4 Miscellaneous

3.4.1 Database

As previously mentioned, we chose to use a sqlite3 database as it was the most suitable for our needs. One aspect that we had to consider was the lack of native permissions management in sqlite3. However, we were able to address this by 'playing' with micro-services.

3.4.2 Logging

We considered various options for implementing a logging system and in conclusion, we determined that **Docker** was the most suitable for our needs due to its low footprint. Another significant benefit of using Docker is that logs are **always** written and cannot be intercepted.

4 Conclusion

In conclusion, we believe that we have successfully addressed almost all (we hope) identified threats in our security agenda. It is worth noting that there is one specific vulnerability that we are aware of: the potential session storage leak with a plaintext password for a very brief period of time.