COMPUTER PROJECT SSD&WS

Academic year 2022 - 2023

Mushingelete Aramson Felho 000574088 – Ayoub Touhami

HE2B ESI | M-SECUC

Table of Contents

Introduction *2*

Section 1: Security Implementation *3*

1. Creating a user account 3
2. Login in 4
3. Login out 4

Section 2: Data Model and Security *5*

1. User model 5
2. Event model 5
3. Invitation model 5

Section 3: Checklist *6*

1. **Do I properly ensure confidentiality?** 6
   - A. Are sensitive data transmitted and stored securely? 6
   - B. Are sensitive requests sent to the server transmitted securely? 6
   - C. Do I achieve end-to-end encryption ? 6
   - D. Does a system administrator have access to the sensible data of some arbitrary user? 6
2. **Do I properly ensure integrity of stored data?** 6
3. **Do I properly ensure non-repudiation?** 6
4. **Do I use a proper and strong authentication scheme?** 6
5. **Do my security features rely on secrecy, beyond credentials?** 7
6. **Am I vulnerable to injection?** 7
7. **Am I vulnerable to data remanence attacks?** 7
8. **Am I vulnerable to replay attacks?** 7
9. **Am I vulnerable to fraudulent request forgery?** 7
10. **Am I monitoring enough user activity so that I can immediately detect malicious intents, or analyse an attack a posteriori?** 7
11. **Am I using components with known vulnerabilities ?** 7
12. **Is my system updated ?** 7
13. **Is my access control broken (cf. OWASP 10)?** 8
14. **Are my general security features misconfigured (cf. OWASP 10)?** 8

Bibliography *9*

Introduction

As part of the SSD&WS course we had to carry out a project to develop an online agenda. The main constraint of the project was that the server was not a trusted entity. All information related to an event is considered sensitive. So even the system administrator cannot have access to this data.

To accomplish this task, we decided to develop a web application using the Django framework to create an API to manage the data. On the client side the React framework was used to manage the access to the data by the user via the web browser.

First, we will explain the mechanisms put in place to transfer the information in a secure way

Then we will explain the data model used and how we implemented the project.

Finally, we will evaluate our application by comparing it to a checklist to measure its security

Section 1: Security Implementation

To realize this project, we were inspired by Bitwarden's (Bitwarden, Inc, 2020) security paper, which gives guidelines on how to implement a project like ours.

1. Creating a user account

When creating an account, the user provides an email address and a password. The Password-Based Key Derivation Function 2 (PBKDF2) is used on the user's password, with more than 100,000 iterations and the user's email address as a salt. This gives us the Master Key value of 256 bits.

This Master Key is extended to 512 bits in length using the HMAC-based Extract-and-Expand Key Derivation Function (HKDF). These Master Key and the Derived Master key are not stored on the server.

In addition, a 256-bit Symmetric Key and a 128-bit initialisation vector are also generated using a Cryptographically Secure Pseudorandom Number Generator (CSPRNG).

The previously generated symmetric key is encrypted using AES-256-bit encryption with the Derived Master Key and the Initialisation vector. This encryption will result in the Protected Symmetric Key. This key is associated with the user and is the first element sent to the server when an account is created.

Then the 256-bit Master Key previously generated by the email and password is encrypted using the PBKDF-SHA256 algorithm but this time with the password as the salt and 100,000 iterations. The result is the Master Password Hash. This password is the second element sent to the server when an account is created. However, this Master Password Hash is encrypted by the Agron2 algorithm.

When this is completed, the user receives an email containing a link to activate its account

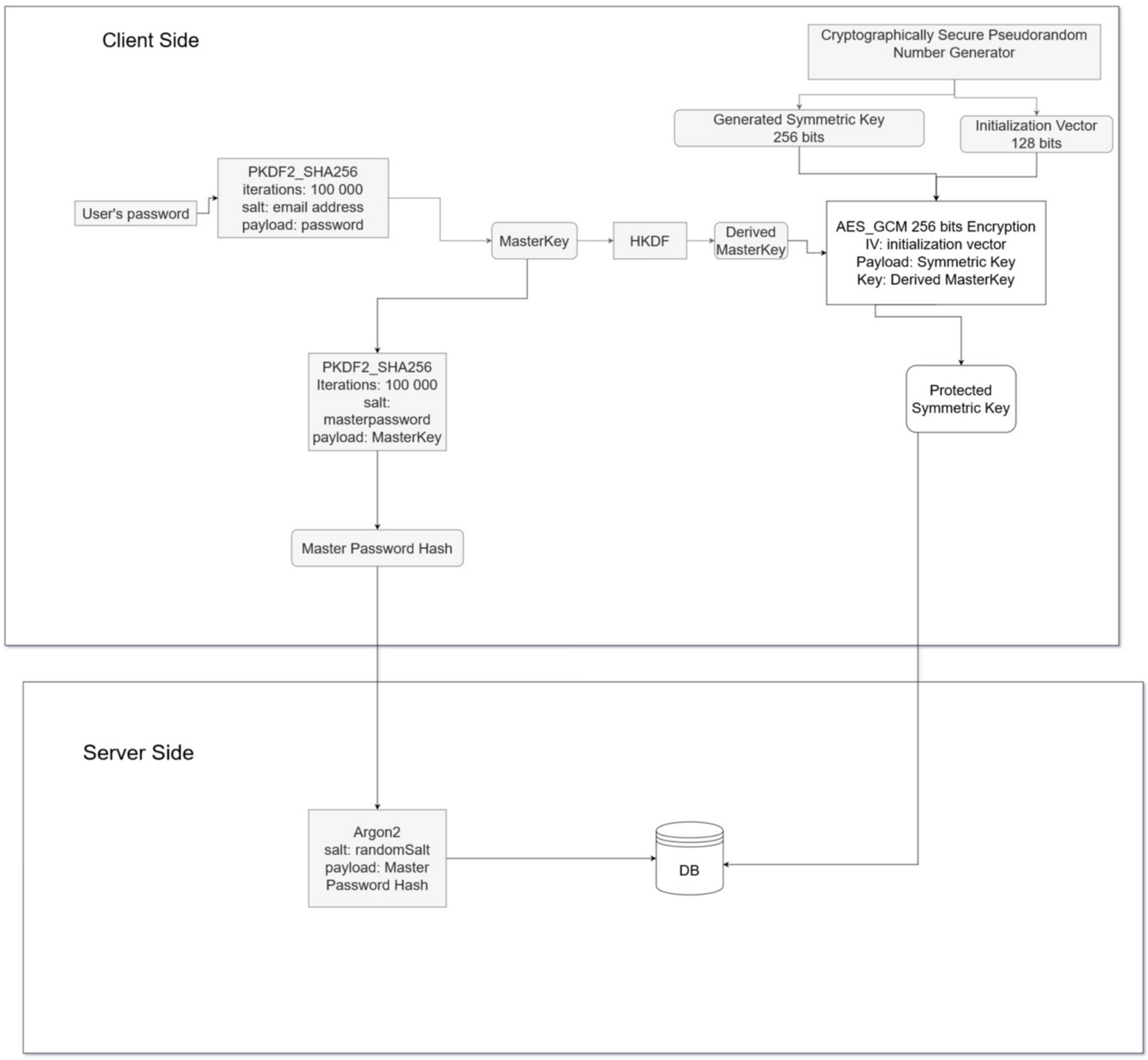
Client Side
Cryptographically Secure Pseudorandom Number Generator
Generated Symmetric Key 256 bits
Initialization Vector 128 bits
User's password
PKDF2_SHA256 iterations: 100 000 salt: email address payload: password
MasterKey
HKDF
Derived MasterKey
AES_GCM 256 bits Encryption IV: initialization vector Payload: Symmetric Key Key: Derived MasterKey
Protected Symmetric Key
PKDF2_SHA256 Iterations: 100 000 salt: masterpassword payload: MasterKey
Master Password Hash
Server Side
Argon2 salt: randomSalt payload: Master Password Hash
DB

Figure 1 Keys generation at account creation

2. Login in

At the time of connection, the user enters his email address and password. The encryption process takes place on the client side to obtain the Master Password Hash. Once this is sent and compared to the DB hash. A pair of JSON Web Token is generated. A refresh token and an access token.

3. Login out

Upon disconnection, the server retrieves the user's refresh token and blacklists it. Thus, a malicious person having succeeded in sniffing his token cannot use it after his disconnection. Concerning the token access, it is configured with a limited lifetime.

Section 2: Data Model and Security

1. User model

On the server we gave a user several keys on his model in addition to other information.

In addition to the Protected Symmetric Key generated by the account creation process. A key pair is generated for each user and stored on the DB. The public key is stored in clear text while the private key is encrypted on the client side with the symmetric key which gives the protected private key. Only the user can decrypt this key since it requires the symmetric key and therefore the Derived Master Key. This way we can securely store the key pair on the server.

2. Event model

When an event is created, a symmetric key is generated on the client side and all event information is encrypted with this key. Then we use the symmetric key of the creator to encrypt this key of the event. This becomes the protected event key and is also stored in the event table along with other information such as dates or locations.

3. Invitation model

If a user wants to invite another user to an event, the users must first be part of each other's contact list, then when the validation of the contact has taken place, the server retrieves the public key of this contact and sends it to the client side. The creator of the event can decrypt the protected event key with his symmetric or Derived Master Key. When he gets the symmetric key of the event, he will encrypt it with the public key of the user. Thus, we obtain a protected event key, but this time stored in the invitation table.

Only the invited user can decrypt this protected event key with his own private key. Moreover, in this model the event id is also encrypted to allow this user to retrieve the appropriate information.

Since one of the constraints of our project was that all data stored in the Event table be encrypted, the server does not have access to the list of guests to an event. This prevented us from filtering access to the event data in an optimal way.
To limit the risk of exposure we decided to expose the data of an event created by a user to his contacts only. Thus, an invited user will have access to all the events created by his contacts.

These being encrypted, without the key generated with the public key of the user, the access to the list of events does not entail great risk.

Section 3: Checklist

1. Do I properly ensure confidentiality?

A. Are sensitive data transmitted and stored securely?

Yeah, when transferring data to the server the data is already encrypted. Moreover, only the people who have the keys, the creator and the guest users, can access the data.

B. Are sensitive requests sent to the server transmitted securely?

The sensitives request are not transmitted securely but they are transferred in a ciphered state.

C. Do I achieve end-to-end encryption ?

Yes, thanks to our protected key system. Only the people for whom the data is intended can access it.

D. Does a system administrator have access to the sensible data of some arbitrary user?

The only data that the system administrator can see is the number of events created by a user and the number of events to which the user has been invited. The rest of the data can be seen by the system administrator.

2. Do I properly ensure integrity of stored data?

Since the event data is encrypted on the client side with an event-specific key, the modification of an event-related information will result in a total change of the basic information once decrypted.

3. Do I properly ensure non-repudiation?

Our server uses an authentication system with JSON Web Token. Apart from the creation of the account and the generation of these tokens, all requests sent to the server must be accompanied by an access token. Since these tokens are signed by the user, the legitimacy of the actions undertaken by this user is guaranteed.

4. Do I use a proper and strong authentication scheme?

Unfortunately, we did not implement the double authentication factor due to time constraints. However, in our implementation, the Master Key is encrypted to become the Master Password Hash and sent to the server. This means that if an attacker managed to recover our password on the server, he would not have access to our data since the Master Password Hash does not allow to obtain the protected symmetric key. Therefore, he would not be able to read our events as well our invitations since the private key of the user is needed to read them and therefore the symmetric key.

5. Do my security features rely on secrecy, beyond credentials?

No, the server cannot be trusted so obviously security by obscurity shouldn't be set up. We base our security on the decryption of the keys necessary for the operation and security of our application on the client side

6. Am I vulnerable to injection?

The data sent to the server is first secured, analysed and ciphered on the client side before reaching the server in a customed format. Moreover the URL only allow

7. Am I vulnerable to data remanence attacks?

No since all data is encrypted and must be decrypted on the client side. Without this, this data is useless.

8. Am I vulnerable to replay attacks?

Since we use JSON Web Tokens, we are indeed vulnerable to replay attacks. To remedy this, we have decided to limit the lifetime of these tokens to reduce the time of use in case of an attack.
In addition, we wanted to implement a method that allows us to revoke refresh tokens upon disconnection. This limits the risk of replay attacks. As this method cannot be applied to access tokens, we decided to reduce the lifetime of these.

9. Am I vulnerable to fraudulent request forgery?

10. Am I monitoring enough user activity so that I can immediately detect malicious intents, or analyse an attack a posteriori?

We have implemented a logging system to analyse the actions entered by a user. Each log generated has a sequence number, making it difficult to falsify. In addition, logs deemed critical will generate an email to the administrator to investigate suspicious behaviour. Finally, if data is sent that does not respect the encryption format we have set up, the user is blocked in addition to the email sent to the administrator. This suspicious behaviour proves that the user has managed to send data directly to the server.

11. Am I using components with known vulnerabilities ?

No, we don't.

12. Is my system updated ?

We have updated our systems and made sure used the up-to-date or recent versions of the various languages and frameworks used.

13. Is my access control broken (cf. OWASP 10)?

Our API only allows users who are not logged in to register and to generate a pair of tokens. All other actions require the user to be logged in. In addition, we have set up a rate limit to the number of requests sent to the API to prevent damage caused by possible automation tools.

Then, we use a JWT system combining at and rt. The use of such a system can lead to a vulnerability as an attacker retrieving the refresh taken could use it to impersonate a legitimate user. For this reason, we have implemented a system that allows a refresh token to be revoked when a user is logged in. Since it is not possible to revoke an at with our implementation, we have reduced the lifetime of the at so that it is not used despite the fact that an rt has been revoked for a user.

We have also implemented a system to limit access to resources for a user. Indeed, a user can only access data that belongs to him and to a lesser extent to that of his contacts. A user can see the public key of a contact to share his events with him but also the list of events of his contacts to decrypt an event that has been shared with him.

14. Are my general security features misconfigured (cf. OWASP 10)?

When we put our application into production, we took the necessary steps to deactivate the options we were not interested in. On the Django side, the admin user was disabled, thus removing a risk of attack on it.

Bibliography

Bitwarden, Inc. (2020, October). *Bitwarden Security Whitepaper*. Retrieved from https://bitwarden.com/images/resources/security-white-paper-download.pdf