Security Principles and Considerations

Research document

The research was done by
a student Vitali Bestolkau(4138872)
from IPS-DB03 group

Table of Contents

Version history

Version	Date	Change
1.0	13-01-2022	Created file and added some content
1.1	17-01-2022	Added all the rest of information that was need
1.2	18-01-2022	Added Sources

Introduction

I am creating a website that involves storing some private data about its users such as their credentials, for example, that should make sure that people are using their own accounts but not a stolen one and that my website should break due to some actions outside its own system. So, obviously, I realised that I would have to implement some security principles applicable to my situation. But I didn't know that there were quite a number of issues to be concerned about, before I began doing this research.

That’s why, in this document I will share with you tasks I’ve encountered while trying to make my application secure and how I solved them. My research was mainly based on the renowned security-related website OWASP. The guys working there, by the way, have recently published the updated OWASP Top 10 list, which shows main vulnerabilities of websites within a range of 3-4 years.

Broken Access Control

First and foremost issue for my application was “Broken Access Control”. My website is based on the fact that there are users that can go through the site and check anything they can, and admins, which are responsible for adding new data inside the application and they can manage any type of data any way they want to. So imagine what would happen if some person with intentions could get admin credentials and use the admin account consequently. It could lead to the whole data inside the application to be deleted in a snap of a finger. That’s why I took this issue in the most serious way.

To begin with, it would be impossible to use JWTs that were generated outside my service, as thanks to Firebase Admin SDK feature every time the token is changed the server checks if a token was provided by Firebase or not. And if it was not, then a user won’t be able to log in.

Moreover, even in case some person manages to get to the content that is supposed to be only visible for admins, the user will not be able to do much as the server side of my application grants access to certain actions only in case they were logged in as admins.

Identification and Authentication Failures

Authentication problem became one of the concerns as well. Luckily Firebase Authentication helps with this issue as well quite easily.

Firstly, in order to sign up in the application, as far as I am using an email address as a part of sign up credentials, I send a user a confirmation email to verify their registered email address. In case they don’t verify the provided email, the access won’t be granted, which reduces the potency of using numerous fake accounts. Also Firebase uses its own hashing for passwords which proved to be reliable.

Secondly, this Firebase service also provides a pre-built feature such as “sign-up quota” which will not allow a person to spam new accounts from the IP that they are using. Moreover, if you consider a default quota to big/small, then you can easily change that in the settings.

Thirdly, as well as in the “sign-up quota” situation there is one more significant feature that was already implemented in Firebase Authentication service. That is refreshing its authentication tokens, aka JWTs. With this feature included, users will not be able to use an account after the next token refresh, in case the account will be either deleted or disabled by the admin user. Additionally, they will have to log in again in case the password was changed by one of the users.

Fourthly, I should mention that I may include reCAPTCHA to my application as Firebase also may help with that. That way the potency of getting any problems with applications spamming to my website will be quite close to 0.

Injections

In order to avoid facing Injections issues, in the very start of the development of my application I decided to make use of a famous ORM system applicable to Java web developers: Spring Boot JPA.

However the main decision was made when I began reducing the code related to authentication on my server side. The major part of authentication logic is managed by Firebase service, except on my server I create custom tokens, verify them and create new users using Firebase Admin SDK. But the passwords and UIDs are stored (passwords are also hashed) on the Firebase server.

Vulnerable and Outdated Components

From the very beginning of the development of my application I wanted to make it a long lasting project. That is the reason I went for the most updated versions of the tools I was using. For example, I am using Vuejs as my front-end development tool. You may not know, but Vue has recently presented a new 3 version of this JavaScript framework, which I decided to use. Even though it is new for the public, as I mentioned, I wanted my website to be a long lasting one. Also from the security point of view, it is obvious that in several years Vue 2 will not be supported any more, which may lead to main security principles not being updated and becoming vulnerable to the attacks.

Vuejs was just one of the examples which showed the most that I am deeply concerned with keeping all my tools uptodate. This also counts for Vuejs dependencies as well as Spring Boot libraries.

Software and Data Integrity Failures

I don’t think I have anything to worry about in this section either. All my dependencies that I use in my project are coming from trustworthy providers and only from them. If we talk about CI/CD pipeline, there I have Azure and Firebase on behalf of Microsoft and Google being responsible for that. Also this pipeline was configured following all the rules and additionally I use Azure MySQL database as the main source of data, which was configured according to the rules as well.

Security Logging and Monitoring Failures

Logging is the main source of data when it comes to some problems the application encounters. That’s why it is quite obvious that this tool should be up and running, especially on the production version of the application.

Although, right now I have only one logging system that I use and that’s the one that is pre-built in the App Service by Azure, that I use to deploy my back-end part of the application, I will probably include another logging system inside my Spring Boot application.

Sources

[OWASP Top Ten Web Application Security Risks | OWASP]