Date: 24-06-2022

Group: S3-CB01

Student: Oleksandr Gurianov

Version: 1.0

Applied Research

Table of content

Introduction

In this document, you will get acquainted with the applied research that was done during the project development. Web Security was chosen as the topic of the research since its implementation was one of the objectives of the Individual Project Assignment. Up until now we were familiar only with the basics of web security principles and researching such a topic can greatly contribute to the improvement of project security.

How to use JWT securely in Frontend & Backend?

What data should be stored in the JWT?

Research method: *Literature Study, Problem Analysis*

JWT is simply a signed JSON intended to be shared between two parties. The signature is used to verify the authenticity of the token to make sure that none of the JSON data were tampered with. The data of the token themselves are not encrypted.

The method of authenticating users does not change with JWT. [...] The difference is only in how you manage the user authorization (how you let your service know that the user has permission to do something). (Novak, 2021)

Before we decide what data should be stored in the JWT, we need to make sure that it is clear what kind of data can be stored. There are several layers of the application, all of which require some kind of Authentication & Authorization. To answer this question, let's first consider what kind of data can be stored.

The JWT RFC (Internet Engineering Task Force (IETF), 2015) establishes three classes of claims:

- ***Registered claims*** like sub, iss, exp or nbf
- ***Public claims*** with public names or names registered by IANA contain values that should be unique like email, address, or phone_number
- ***Private claims*** to use in context and values can collision

The following Claim Names are registered in the IANA "JSON Web Token Claims" registry established by Section 10.1 (Bradley, Campbell, B. Jones, & Mortimore, 2015; DA, 2021):

- ***iss (issuer):*** identifies the principal that issued the JWT.
- ***sub (subject):*** identifies the principal that is the subject of the JWT. Must be unique
- ***aud (audience):*** identifies the recipients that the JWT is intended for (array of strings/uri)
- ***exp (expiration time):*** identifies the expiration time (UTC Unix) after which you must no longer accept this token. It should be after the issued-at time.
- ***nbf (not before):*** identifies the UTC Unix time before which the JWT must not be accepted
- ***iat (issued at):*** identifies the UTC Unix time at which the JWT was issued
- ***jti (JWT ID):*** provides a unique identifier for the JWT

Now that we are aware of the kinds of data the JWT can store we can decide what data is necessary. Depending on your application, the token may also contain capability claims, such as statements that the subject has access to a particular system service. The user's email address and other data can also be stored in the cookie, of course. However, the user identifier (subject claim) is usually enough.

Where the JWT should be stored in the browser?

Research method: *Literature Study, Problem Analysis*

There are three options (DA, 2021) available for storing the data on the client-side and each of those has its advantages and disadvantages:

- ***Cookie***
- ***localStorage***
- ***Session Storage***

If you set the JWT on the ***cookie***, the browser will automatically send the token along with the URL for the Same Site Request. However, it is vulnerable to the CSRF. We can protect the site against CSRF by setting a cookie with SameSite=strict.

The ***localStorage*** doesn't send the data automatically along with the URL. So you need to implement the system for the auth token for every URL. But the best part is that this method is not vulnerable to CSRF.

Session Storage is pretty much the same as Local Storage, except the token will accessible only one tab, once the tab is closed the session got destroyed. So it is not useful for feature use like "remember me".

In conclusion, neither localStorage nor session storage is by default XSS protected. However, the Cookie offers several security choices, like HttpOnly, SameSite, etc. Therefore, using Cookie is a good decision.

How to make use of Refresh Tokens?

Research method: *Literature Study*

Before we decide whether Refresh Tokens should be used in the application let's first clarify its meaning. Access tokens may only be valid for a brief period, as we are well aware. Client apps can "refresh" the access token by using a refresh token after it has expired. In other words, a refresh token is a credential artifact that enables a client program to obtain new access tokens without requiring the user to re-login.

As long as the refresh token is active and hasn't expired, the client application may request a new access token. Therefore, a refresh token with an extremely long lifespan might conceivably provide the token holder limitless ability to obtain a new access token and access restricted resources whenever they choose. Either a trustworthy user or a malevolent user could be the refresh token's bearer. As a solution, security firms design procedures to make sure that the intended parties are primarily holding and using this strong token continuously (Arias & Bellen, 2021).

A refresh token is revoked and replaced by a new refresh token each time it is used to create a new JWT. Refresh token rotation is a method that improves security by shortening refresh token lifetimes and decreasing the likelihood that a hacked token would still be valid (or valid for long) (Watmore, 2022).

Conclusion

Emails, passwords, roles, and other sensitive information that users often used have to be stored in a secure place while maintaining the ease of getting this information. That’s where the use of JWT Access Tokens and Refresh Tokens comes into place. By using those two types of authentication and implementing them into the application we can easily provide a user with access when they need to perform an authorized action while keeping this information stored in a safe place.

References

- Internet Engineering Task Force (IETF). (2015, May). *JSON Web Token (JWT)*. Retrieved from RFC: https://www.rfc-editor.org/rfc/rfc7519#page-8
- DA, N. (2021, July 6). *Where Should You Store JSON Web Tokens (JWT)?* Retrieved from JavaScript in Plain English: https://javascript.plainenglish.io/where-to-store-the-json-web-token-jwt-4f76abcd4577
- Bradley, J., Campbell, B., B. Jones, M., & Mortimore, C. (2015, January 23). *JSON Web Token (JWT)*. Retrieved from IANA: https://www.iana.org/assignments/jwt/jwt.xhtml
- Novak, M. (2021, November 24). *JWT: Ultimate How-To Guide With Best Practices In JavaScript*. Retrieved from Better Programming: https://betterprogramming.pub/jwt-ultimate-how-to-guide-with-best-practices-in-javascript-f7ba4c48dfbd