Lab report #2

SLH - Authentication

Alexis Martins

January 8, 2024

Contents

1 Questions

1.1 What is the purpose of a JWT access token? And what is the purpose of a JWT refresh token? Why do we have both?

The access token is used as a “credential” to access protected resuorces. It grants access to a resource. For instance, it’s possible to protect some API endpoints and to request for a valid access token to use the resource. The token will contain the credentials/identifier or any kind of informaton that will allow the server to determine wether or not the user can access the resource.

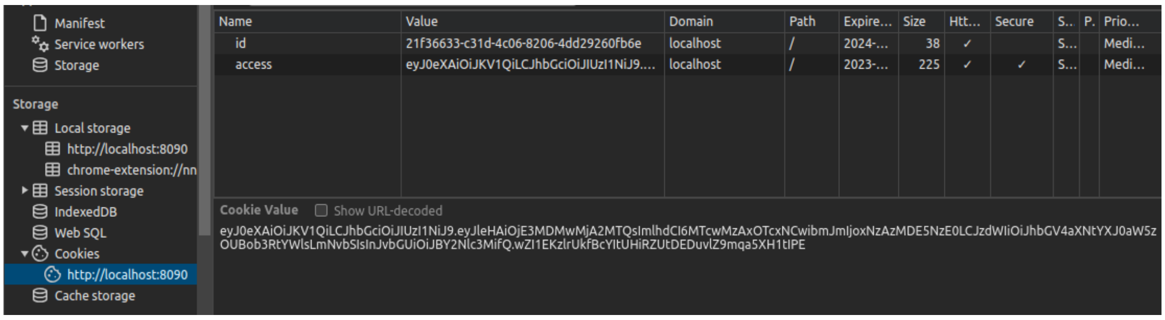
The refresh token is used to retrieve access tokens. When the access token is expired, we can contact the server with the refresh token to ask for a new access token.

It’s not mandatory to have both tokens in our architecture, but it’s highly recommended. The access token has a short lifetime (recommended few minutes and maxium few hours) and the refresh token has a longer lifetime (at least few days and maximum a pair of months).

Having both JWT access and refresh tokens offers a balance between security and usability. Access tokens are short-lived and grant temporary access to resources, minimizing the risk if they are compromised. However, frequently re-authenticating can be inconvenient. Refresh tokens, which are longer-lived, allow users to obtain new access tokens without re-entering credentials. This is a good balance between UX and security (and we know it’s always the challenge).

1.2 Where are the access tokens stored? Where are the refresh tokens stored?

The access tokens are stored in the cookies. Note that I enabled the HttpOnly and Secure flag.



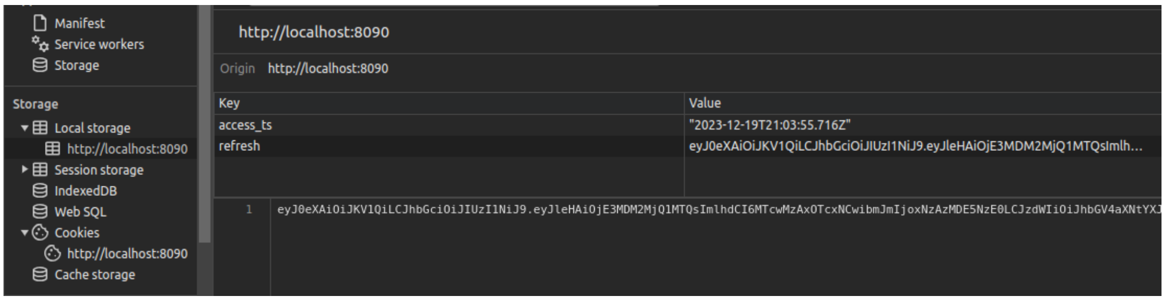
Name	Value	Domain	Path	Expire...	Size	Htt...	Secure	S..	P.	Prio...
id	21f36633-c31d-4c06-8206-4dd29260fb6e	localhost	/	2024-...	38	✓		S...		Medi...
access	eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9....	localhost	/	2023-...	225	✓	✓	S...		Medi...

Cookie Value ☐ Show URL-decoded

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJleHAiOjE3MDMwMjA2MTQsImlhdCI6MTcwMzAxOTcxNCwibmJmIjoxNzAzMDE5NzE0LCJzdWIiOiJhbGV4aXNtYXJ0aW5z
OUBob3RtYWlsLmNvbSIsInJvbGUiOiJBY2Nlc3MifQ.wZI1EKzlrUkfBcYItUHiRZUtDEDuvlZ9mqa5XH1tIPE
```

Figure 1: Access token in the cookies storage

The refresh tokens are stored in the local storage.



http://localhost:8090

Origin http://localhost:8090

Key	Value
access_ts	"2023-12-19T21:03:55.716Z"
refresh	eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJleHAiOjE3MDM2MjQ1MTQsImlh...

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJleHAiOjE3MDM2MjQ1MTQsImlhdCI6MTcwMzAxOTcxNCwibmJmIjoxNzAzMDE5NzE0LCJzdWIiOiJhbGV4aXNtYXJ
```

Figure 2: Refresh token in the local storage

1.3 Is it a good idea to store them there? Is there a better solution?

Cookies can be a good solution if they are well-parametered. If we had the HttpOnly flag, secure flag (if HTTPS) and configure the same-site, cookies will be pretty much fine. This is a good idea for a JWT access token.

On the other hand, local storage doesn't have any security mechanism to prevent a basic XSS to steal the values stored inside. Knowing that, local storage just looks less secure than the good old school cookies with the right flags (named above).

But it isn't the optimal way of using a refresh token. According to this document section 8 from IETF that references the RFC 8252, the best practice is to implement a refresh token rotation. The flow is still the same, we have an short-time access token that grants access to services and a refresh token that lasts much longer and which allows us to have an access token. The subtelty comes from the managment of the refresh token, this one is one time use. To explain the workflow, I will partially take the example from the document named above :

- The user connects to the application, if the connection is authorized he will receive an access token (1h duration) and a refresh token (24h duration).
- He will store both these tokens in a secure place (like the cookies we have seen before)
- Meanwhile the access token is available, the user can access any service. Once it's expired, he will use the refresh token to get a new valid access token (and a refresh, see below).
- The server on its side will add the old refresh token to a "blacklist" database and generate a new access and refresh token to send it to the user.
- When the refresh token expires, the user has to reconnect.

What does this method offers in terms of security ?

If the token is one time use, we have another question "What happens if it re-use ?". There are several solutions, if the servers detects token reuse :

- Blacklist all the existing tokens (access and refresh), so neither the user or the attacker will have access to the service. This will require the user to log in again.
- The solution, I read was to lock the user's account and send him a message by a side-channel. I find this solution "extreme" compared to the previous, but it works.

To summarize, while there is no attacks the flow will work as it was without the server. If there is an attack and if the attack tries to use the new refresh token after or before me, the server will detect a suspicious activity and deny the access to the application.

Figure 3: Thanks to GPT