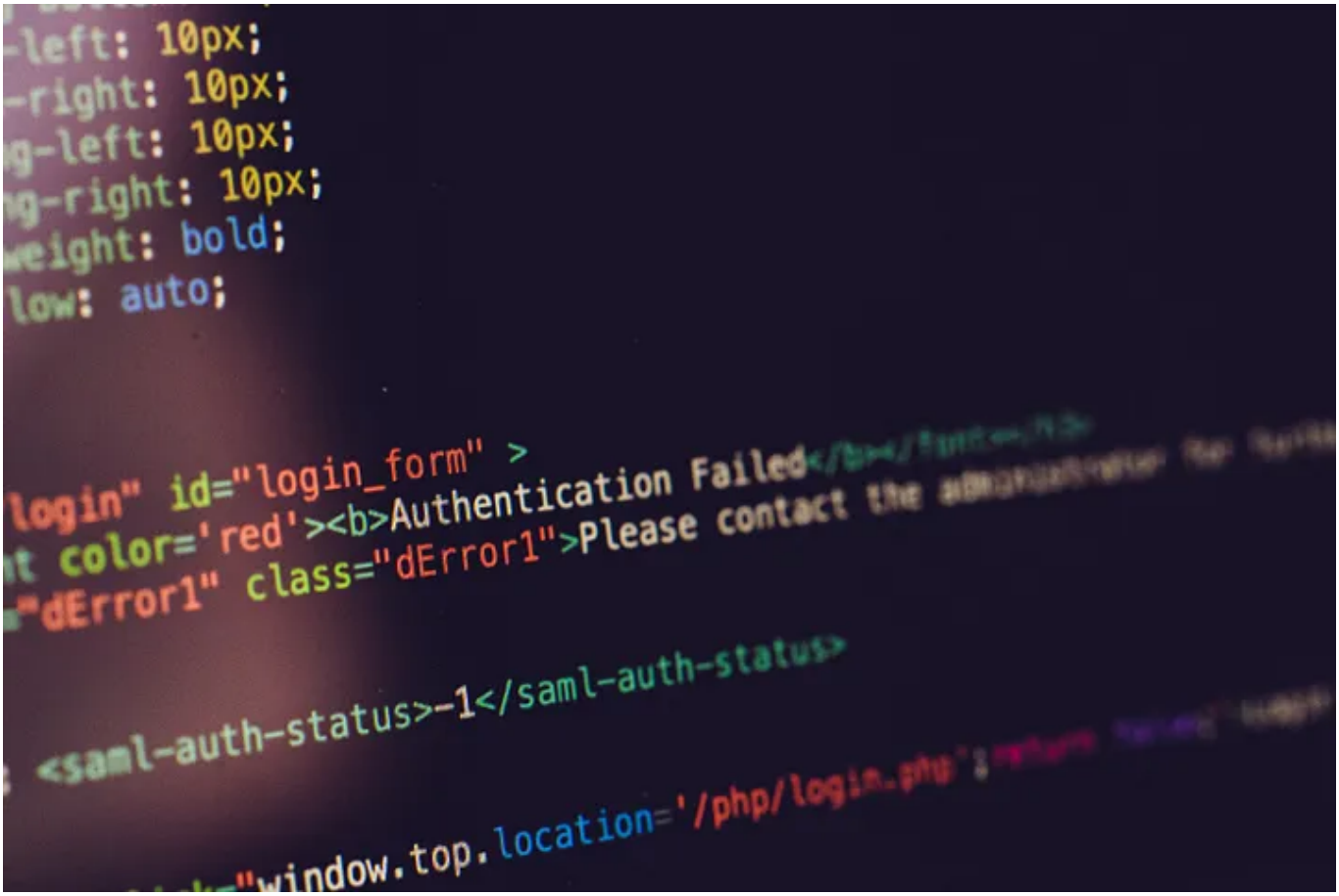
JSON Web Token(JWT) vs Opaque Token

Piyumi M Dasanayaka · Follow

5 min read · Oct 17, 2020



Authentication is the method of deciding if, in fact, someone or something is who or what it claims itself to be. Authentication technology offers device access control by testing to see whether a user's credentials match the credentials in an authenticated user database or in a server for data authentication. Token-based authentication works by ensuring that a signed token is followed by each request to a server, which the server verifies for validity and responds to the request only then. A token is a piece of data that has little meaning or use on its own, but becomes a crucial player in securing the application in accordance with the correct tokenization method.

Why use tokens instead of traditional methods

- ***Tokens are stateless*** -The token is self-contained and includes all the authentication information that it needs. For scalability, this is perfect as it frees your server from having to store the session state.
- ***Tokens can be generated from anywhere*** -The generation of tokens is decoupled from token authentication, allowing you to manage token signing on a separate server or even via another company such as Auth0.
- ***Fine-grained access control***- You can easily define user roles and permissions, as well as services that the user can access, within the token payload.

There are a number of techniques, but two of the most common are:

1. JWT (JSON Web Tokens)
2. Opaque Tokens

Let's see what are these tokens and why we select JWT instead of Opaque and why we select Opaque instead of JWT.

1. JWT (JSON Web Tokens)

Actually, JWT Tokens are a complete JSON object encoded with base64 and then signed with either a symmetric shared key or a public / private key pair. The difference is that if you have a customer who needs to verify that the token is signed, but that customer should not be permitted to generate tokens, you can give the customer the public key that can not generate but still verify tokens.

Three sections consist of a JSON Web Token: **Header**, **Payload,** and **Signature**. The header and payload are encoded by Base64, then concatenated by time, and the result is finally signed algorithmically, creating a token in the form of header.claims.signature. The header consists of the metadata used to sign the token, including the token form and the hashing algorithm. The payload includes information about the statements encoded by the token.

Look at the JWT :

header.claims.signature

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJtZXNzYWdlIjoiSldUIFJ1bGVzISIs
ImlhdCI6MTQ1OTQ0ODExOSwiZXhwIjoxNDU5NDU0NTE5fQ.-
yIVBD5b73C75osbmwwshQNRC7frWUYrqaTjTpza2y4
```

One of the advantages of JWTs is that they can be used without a back-up shop. Inside the token itself is all the information needed to authenticate the user. It makes it easy not to rely upon centralized authentication servers and

databases in a distributed microservice world. To handle verifying the token, the individual microservice only requires some middleware (JWT libs are publicly accessible for anything from Express to JVM MVC Frameworks) and also the secret key needed for verification.

Drawbacks of JWT

One of the downsides to JWTs is that if you need the action to be instant, blocking users or adding/removing roles is a little more difficult. Know, there is a predefined expiry date for the JWT that can be set for a week in the future. Since the token is stored on the client-side, even if you mark the user as disabled in your database, there is no way of directly invalidating the token. Instead, you must wait for it to expire. In particular, this can impact the architecture when developing a public API that could be starved by a single power user or an e-commerce app where it is important to ban fraudulent users. For example, if all you care about is banning compromised tokens or users, you can have a blacklist of tokens or user-ids, but this can reintroduce a database back into your authentic structure. A suggested way to blacklist is to guarantee that each token has a JWT Id that can be stored in the Db.

On the other hand, if you have an enterprise app with several positions, such as admin, project owner, service account manager, and you want to have an immediate impact, then it can be tricky to create. In particular, consider the case where an admin is changing the authorized roles of someone else, such as his / her immediate reports. Thus, without refreshing the JWT, the updated user doesn't even realize his / her roles have changed.

The token will expand as more fields are added as a second drawback. The token is sent for almost every request in stateless apps, so there can be an

effect on data traffic size. For example, the enterprise app we alluded to earlier may have several functions, which may add bloat and complexities for what to store in a token. Think about designing the APIs that back up the Web Portal of AWS or Azure. On each resource for each user, you have targeted permissions.

2. Opaque Tokens

A ***JWT*** has readable content, as I explained in earlier. Everyone can decode the token and read the information in it. An ***opaque token*** on the other hand has a format that is not intended to be read by you. Only the issuer knows the format.

Literally, opaque tokens are what they sound like. The opaque token is simply a primary key that references a database entry that has the data, instead of storing user identity and claims in the token. Fast key-value stores such as Redis are perfect for leveraging O(1) search of the payload in-memory hash tables. Since the roles are read directly from a database, roles can be modified and as soon as the modifications propagate to the backend, the user can see the new roles.

> *In Auth0's case, opaque tokens can be used with the* `/userinfo` *endpoint to return a user's profile. If you receive an opaque Access Token, you don't need to validate it. You can use it with the* `/userinfo` *endpoint, and Auth0 takes care of the rest.*
>
> *A "opaque JWT refresh token" is a contradiction . What actually is meant here is, that in some JWT frameworks only the authentication token is a JWT, but as refresh token they use opaque tokens.*

Thank you.

Reference

JWT.IO

JSON Web Token (JWT) is a compact URL-safe means of representing claims to be transferred between two parties. The...

jwt.io

Steps to building authentication and authorization for RESTful APIs

Authentication & Authorization of RESTful APIs and single page apps. An overview from JWTs vs opaque tokens and cookies...

www.moesif.com

Jwt | Jwt Token | Opaque | Authentication

Written by Piyumi M Dasanayaka

31 Followers

Software engineer

Follow