Published By:
Divyanshu Mishra

Secure Authentication using JSON Web Token (JWT) in a Web-based Application

Abstract:

Modern web applications have integrated data sharing that occurs between cross origins over the network. The application users are required to be authenticated beforehand so they can access and modify the specific contents of a particular web application. This results in the issue of implementing a secured authentication approach that can transmit user information based upon the user's authentication state based on which he can access all the different routes and resources of the application. JWT basically handles the "authorization" when verifying whether the user is cleared to access the route. Generally, sessions have been used to authorize the user which is a back and forth procedure involving the client and server interaction to pass on session IDs and verify the users. JWT on the other hand is a JSON Web Token that is created for every user and signature verification can assess the credibility of the user.

What is JWT?

It is a technique to securely transmit information between different parties using **Session Tokens.** Moreover, JWT is a stateless authentication mechanism that is based on client-side authentication and not the server-side.

Here, Session Tokens are the encrypted strings that can fetch the session instances to verify the authenticity of a particular user.

The typical JWT Structure is as follows:

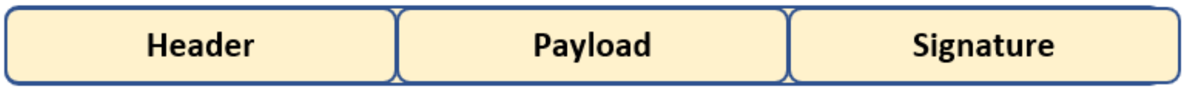

Fig 1: JWT Structure

Here, the Header for example in JSON format is as follows:

```
{
   "alg": "HS256",
   "typ": "JWT"
}
```

alg: The type of signing algorithm being used can vary as **HMAC SHA256 [**Hash-based Message Authentication Code**]** or **RSA.**
typ: Type of algorithm which is **JWT.**

The payload contains the claims which provide details about the user and other data.

Further, Payload in JSON format is as follows:

```
{
  "sub": "user1001",
  "name": "Divyanshu mishra",
  "admin": true
}
```

sub: The subject which is a registered claim
name and admin are the other user derived data

Finally, the Signature part consists of the **Header, Payload,** and a **Secret** all signed together after being encoded in **Base64Url.**

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret)
```

Ultimately, we get an encoded and signed URL string which for example can look as follows:

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfaWQiOiI2MjZhMmNmODQ2ZGM1YjUzNmNhZTU0MTAiLCJpYXQiOjE2NTIxOTM0MzAsImV4cCI6MTY1MjI3OTgzMH0.7w_1PKRTVgxztoJrj11umIuo4OLmJfrxRWjGBVHE-is

Working of JSON Web Token:

When the user logs in to a web application that uses JWT as its authentication implementation, a JWT is returned which is sensitive and has an expiry date that ensures its validity for only a short period.

Now, suppose the user wants to access protected routes of the application. In that case, the user agent can send the JWT as a header to verify the credibility of the user and provide access.

The user agent uses the **Bearer** to send the JWT in the **Authorization** header as follows:

```
Authorization: Bearer <token>
```

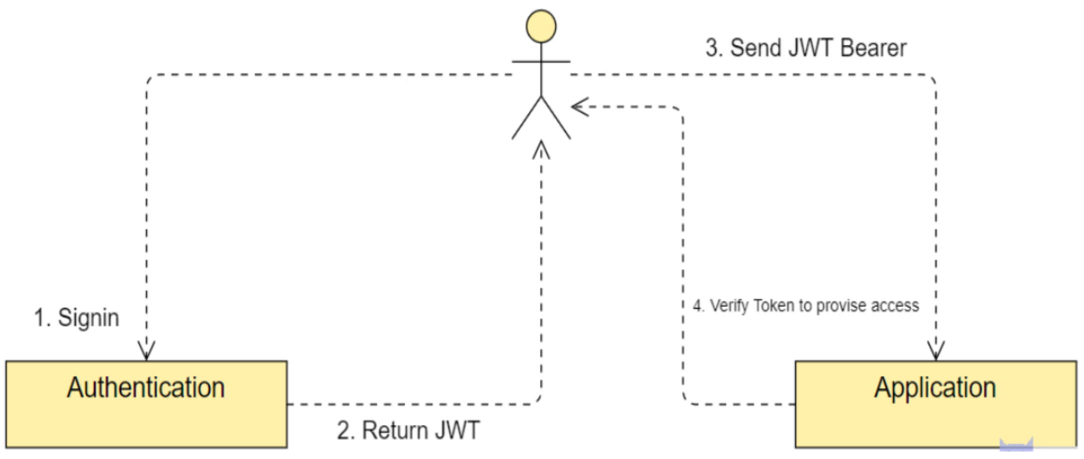

Fig 2: JWT Process Flow

When to use JSON Web Token?

JWT can be used when working with the RESTful APIs and also when the server cannot use any sessions as with JWT, the token itself can contain the data.
Moreover, when the issuer and the receiver belong to different independent parties, JWT can be a good option to be implemented.

Conclusion:

JSON Web Tokens are used to authenticate users securely and share information across origins.
When APIs are being developed/implemented JWTs are a standard to use as they can very well offer a compact and safe method of information exchange without a centralized token database.
Therefore, due to JWT being lightweight, versatile, and self-contained, it is deemed to become more popular in web development.