JSON Web Token (JWT)

Prashant Walke

Overview

What is JSON Web Token?

JSON Web Tokens Uses

- Authorization
- Information Exchange

How do JSON Web Tokens work

What is JSON Web Token?

- JWT is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object.
- This information can be verified and trusted because it is digitally signed.
- JWTs can be signed using a secret (with the HMAC algorithm) or a public/private key pair using RSA or ECDSA.

JSON Web Tokens Uses

Authorization

- Once the user is logged in, each subsequent request will include the JWT, allowing the user to access routes, services, and resources that are permitted with that token.

Information Exchange

- JSON Web Tokens are a good way of securely transmitting information between parties

Why should we use JSON Web Tokens?

- **Security -** Securely transmitting information between parties using public/private key pairs
- **Ease -** Ease of client-side processing of the JSON Web token on multiple platforms, especially mobile.
- **Compact -** Because of its size, it can be sent through an URL, POST parameter, or inside an HTTP header. Additionally, due to its size its transmission is fast.
- **Self-Contained -** The payload contains all the required information about the user, to avoid querying the database more than once.

How do JSON Web Tokens work?

JWT format

header.payload.signature

- **Header -** consists of two parts: the type of the token, which is JWT, and the signing algorithm being used, such as HMAC SHA256 or RSA.

  For example: {

  "alg": "HS256",

  "typ": "JWT"

  }

JWT format

header.payload.signature

- **Payload-** Contains the claims. Claims are statements about an entity (typically, the user) and additional data. There are three types of claims: registered, public, and private claims.

  For example: {

  "user_id": "4"

  }

JWT format

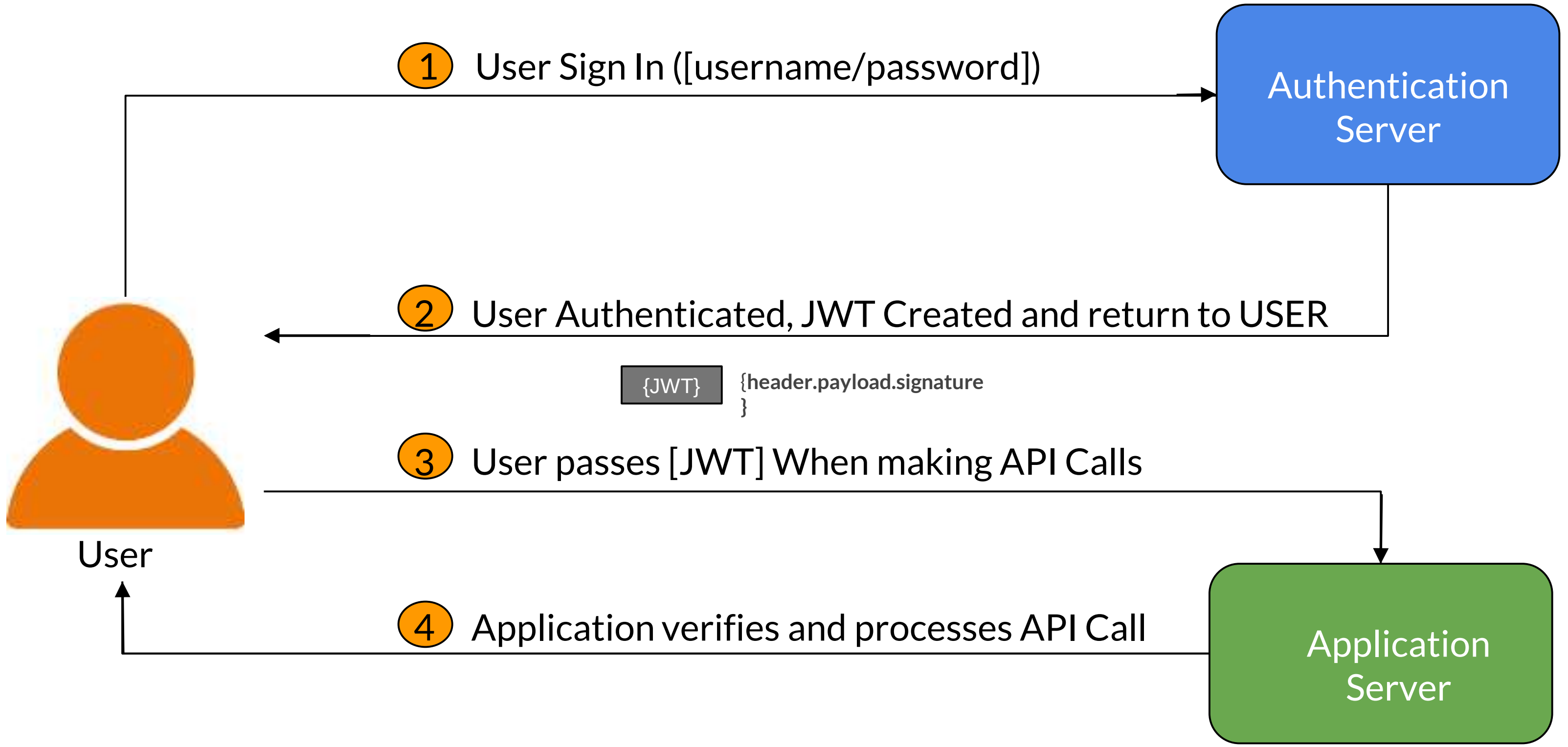
header.payload.signature

- **Signature -** To create the signature part you have to take the encoded header, the encoded payload, a secret, the algorithm specified in the header, and sign that.

  For example (HMAC SHA256 algorithm):

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

JWT format


1 User Sign In ([username/password])
Authentication Server
2 User Authenticated, JWT Created and return to USER
{JWT}
{header.payload.signature }
3 User passes [JWT] When making API Calls
User
4 Application verifies and processes API Call
Application Server

JWT to verify the authenticity of a user

- User first signs into the authentication server using the authentication server’s login system (e.g. username and password, Facebook login, Google login, Twitter etc).
- The authentication server then creates the JWT and sends it to the user.
- When the user makes API calls to the application, the user passes the JWT along with the API call.
- In this setup, the application server would be configured to verify that the incoming JWT are created by the authentication server
- When the user makes API calls with the attached JWT, the application can use the JWT to verify that the API call is coming from an authenticated user.

Conclusion

Definitely having reliable way to authenticate user is the first thing on the list and using JWT Authentication as an best authentication method.