Authentication using JWT

Agenda

In this session, we will discuss:

- JWT – Introduction
- Components of JWT
- JWT Structure Example
- JWT Operation
- Advantages of JWT
- JWT Authentication in Node JS

JWT – Introduction

- JWT (JSON Web Token) serves as an open standard, outlined in RFC 7519, offering a secure and self-contained method for exchanging information between entities in the form of a JSON object.
- This data can be relied upon and authenticated due to its digital signature.
- JWTs can be signed either through a secret using the HMAC algorithm or with a public/private key pair using RSA or ECDSA.

Components of JWT – Header

- The header typically comprises two components: the token type, identified as JWT, and the employed signing algorithm, which can be HMAC SHA256 or RSA, among others.
- Example of the Header:

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

Components of JWT – Payload

- The second part of the token is the payload, which contains the claims. Claims consist of statements about an entity, commonly the user, along with extra data.
- Example of the Payload:

```
{
      "sub": "1234567890",
     "name": "John Doe",
    "iat": 1516239022
}
```

- Common claims include iss (issuer), exp (expiration time), sub (subject), aud (audience), and more.

Components of JWT – Signature

- To generate the signature component, the user must utilize the encoded header, the encoded payload, a secret key, and the algorithm indicated in the header to perform the signing process.
- Example of the Signature:

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

JWT Structure – Example

Header :

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

Payload:

```
        {
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

Signature:

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

JWT Token:

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiAiMTIzNDU2Nzg5MCIsICJuYW1lIjogIkpvaG4gRG9lIiwgImlhdCI6IDE1MTYyMzkwMjJ9.TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ

JWT Operation

- User Authentication:
  - The user logs in, and the server verifies their credentials.
- JWT Creation:
  - Header: The server constructs a JSON object with information about the token, such as the type ("typ": "JWT") and the signing algorithm ("alg": "HS256").
  - Payload: The server creates another JSON object with claims about the user (e.g., user ID, username, expiration time).
  - Signature: The server takes the encoded header, the encoded payload, and a secret key. It then applies the specified signing algorithm to create the signature. The header, payload, and signature are concatenated to form the JWT.

JWT Operation

- Token Issuance:
  - The server sends the JWT to the client as part of the authentication response.
- Token Usage:
  - The client stores the JWT and includes it in the Authorization header of subsequent requests.
- Server Verification:
  - Upon receiving a request with a JWT, the server extracts the header and payload.
  - The server uses the stored secret key to recreate the signature by hashing the header and payload.
  - It compares the recreated signature with the one in the JWT. If they match, the token is considered valid.

JWT Operation

- Claims Verification:
  - The server checks the claims in the payload, including the expiration time.
  - If the token is expired or the claims are invalid, the server rejects the request.
- User Access:
  - If the JWT is valid, the server processes the request and grants access to the requested resources or actions based on the claims in the payload.

Advantages of JWT

- Stateless Authentication:
  - JWTs allow for stateless authentication, eliminating the need for servers to store session information and enhancing scalability.
- Efficient Cross-Domain Authentication:
  - JWTs are easily transmitted between different domains, making them suitable for microservices architectures and third-party service integrations.
- Compact and Versatile Information Exchange:
  - JWTs provide a lightweight and efficient way to transmit user information, roles, and permissions securely, making them ideal for mobile applications and efficient access control.
- Decentralized Identity Management:
  - In decentralized or distributed systems, JWTs facilitate conveying user identity and permissions without relying on a centralized authentication server, promoting flexibility and scalability.

JWT Authentication in Node JS

Live Demo

Summary

Here’s a brief recap:

- JWT (JSON Web Token) is a compact, self-contained token format for securely transmitting information, and it comprises a header, payload, and signature, making it ideal for authentication and authorization in web applications.
- JWTs also enable secure authentication without server-side sessions, making them efficient and scalable.
- The fundamental elements and architecture of JWT, along with practical examples, are understood. The inner workings of JWT, which is a critical concept in modern web authentication and authorization, are explored.
- JWTs are used in SSO systems to authenticate a user once and grant them access to multiple applications within an ecosystem without the need to re-enter credentials for each application.