JWT (JSON Web Token) Schema Deep Dive

Introduction

In our authentication system, we utilize JSON Web Tokens (JWTs) to securely transmit information between parties. JWTs are compact, self-contained tokens that consist of a header, payload, and signature. They are widely used for authentication and authorization purposes.

Custom JWT Schema

Our custom JWT schema includes a variety of claims and roles to provide granular access control and authorization. The JWT payload contains the following key components:

Standard Claims

- *iss* (Issuer): Identifies the principal that issued the JWT.
- *sub* (Subject): Identifies the principal that is the subject of the JWT.
- *aud* (Audience): Identifies the recipients that the JWT is intended for.
- *exp* (Expiration Time): Identifies the expiration time on or after which the JWT must not be accepted for processing.
- *iat* (Issued At): Identifies the time at which the JWT was issued.
- *jti* (JWT ID): Provides a unique identifier for the JWT.

Custom Claims

- *userId*: Represents the unique identifier of the user associated with the token.
- *username*: Indicates the username of the authenticated user.
- *email*: Contains the email address of the user.
- *roles*: Specifies an array of roles assigned to the user. Examples of roles include "admin", "manager", "editor", "viewer", etc.
- permissions: Includes an array of specific permissions granted to the user. Permissions can be granular, such as "read:articles", "write:comments", "delete:users", etc.
- *accessLevel*: Defines the overall access level of the user. It can have values like "basic", "premium", "enterprise", etc.
- *departmentId*: Indicates the department or team to which the user belongs.
- *subscriptionTier*: Specifies the subscription tier associated with the user, such as "free", "pro", "enterprise", etc.
- *profileComplete*: A boolean value indicating whether the user's profile is complete or requires additional information.
- *lastLoginTimestamp*: Represents the timestamp of the user's last successful login.

Access Control and Authorization

The custom claims and roles included in the JWT enable fine-grained access control and authorization throughout the system. Here's how it works:

1. When a user authenticates successfully, the authentication server generates a JWT containing the relevant claims and roles based on the user's identity and privileges.

2. The generated JWT is then sent back to the client as part of the authentication response.

3. For subsequent requests, the client includes the JWT in the Authorization header of the request, typically in the format Bearer <JWT>.

4. The server receiving the request verifies the JWT's signature to ensure its integrity and authenticity. If the signature is valid, the server proceeds to extract the claims and roles from the JWT payload.

5. Based on the extracted claims and roles, the server can make authorization decisions. For example:
   - If a user has the "admin" role, they may have unrestricted access to all resources and functionalities.
   - If a user has the "editor" role and the "write:articles" permission, they can create and modify articles but not delete them.
   - If a user has the "viewer" role, they may only have read-only access to certain resources.

6. The server can also utilize other custom claims to enforce additional access control rules. For instance:
   - The *accessLevel* claim can determine the level of access a user has to specific features or content.
   - The *subscriptionTier* claim can be used to grant or restrict access based on the user's subscription plan.
   - The *departmentId* claim can ensure that users can only access resources relevant to their department.

JWT Generation and Expiration

When a user successfully authenticates, the authentication server generates a JWT using a secret key known only to the server. The JWT is digitally signed using this secret key, ensuring its integrity and preventing tampering.

The JWT includes an expiration time (exp claim) that determines the validity period of the token. Once the token expires, it is no longer considered valid, and the user must re-authenticate to obtain a new token.

The expiration time is typically set to a reasonable duration based on the application's security requirements. Common expiration times range from a few minutes to a few hours or days, depending on the sensitivity of the application and the desired user experience.

Revoking Access

In certain situations, it may be necessary to revoke access for a specific user or invalidate a particular JWT before its expiration time. Here are a few approaches to achieve this:

1. Token Blacklisting: Maintain a blacklist of revoked or invalid JWTs. When a token needs to be revoked, add its identifier (jti claim) to the blacklist. On each request, check if the incoming JWT's identifier exists in the blacklist. If found, consider the token invalid and deny access.

2. Token Versioning: Include a version claim in the JWT payload. When a user's permissions change or their access needs to be revoked, increment the version number associated with their user ID. On each request, compare the version claim in the JWT with the current version stored for the user. If the versions don't match, consider the token invalid and require re-authentication.

3. Short-lived Tokens: Issue JWTs with relatively short expiration times and require users to refresh their tokens frequently. This approach reduces the window of opportunity for using revoked or compromised tokens. When a user's access needs to be revoked, simply stop issuing new tokens for that user.

Conclusion

The custom JWT schema, with its varied claims and roles, provides a powerful and flexible mechanism for access control and authorization. By including relevant information in the JWT payload, the system can make informed decisions about what actions a user is allowed to perform and what resources they can access.

Proper JWT generation, expiration handling, and revocation mechanisms are crucial to maintain the security and integrity of the system. Regularly reviewing and updating the JWT schema based on evolving application requirements helps ensure a robust and secure authentication and authorization process.