JSON Web Tokens (JWTs)

Authentication is a crucial part of any web application, as it verifies the identity of the user and controls access to protected resources. One popular authentication method is JSON Web Token (JWT), which allows for secure and scalable identity verification via stateless authentication.

JSON Web Token (JWT) authentication is a stateless method of securely transmitting information between parties as a JavaScript Object Notation (JSON) object. It is often used to authenticate and authorize users in web applications and APIs.

Research on JSON Web Tokens (JWTs) spans across diverse areas, reflecting the multifaceted nature of their applications in modern digital ecosystems. Here's a breakdown of key research domains:

- Security Analysis: Focuses on assessing JWTs' security properties, including vulnerability assessments and cryptographic flaws.
- Authentication and Authorization: Investigates JWTs' use for authentication and authorization, comparing them with other methods in terms of security and scalability.
- Real-World Applications: Investigates JWTs' applications in web authentication, single sign-on, mobile app development, and IoT ecosystems.

In the authentication world, “stateless” means a mechanism in which the server does not maintain any session state between requests. In a stateless authentication system, each request is self-contained and includes all the necessary information to authenticate and authorize the user or entity. In the case of JWT authentication, this comes in the form of a token.

Structure of a JWT:

A JSON token consists of **three parts**

- **Header** containing information about the type of token and algorithms used to **generate the signature**.

- **Payload** containing the “claims” or the JSON object (ID and authentication verifications) **made by the user** that can include a User ID, the user’s name, an email address, and metainformation about the operation of the token.
- **Signature** A string that is generated via a cryptographic algorithm that can be used to verify **the integrity of the JSON payload**.

Together, **the header, payload, and signature** make up the JSON Web Token, typically passed between the **client and the server** in the HTTP Authorization header or in the body of an HTTP request or response. The server can then verify the signature to ensure that the token is valid and has not been modified and use the information in the payload to authenticate the user.

Outlined Basic steps of JWT authentication quite well

How JWT authentication works. a structured breakdown of the process:

1. **User Login**: The user provides their credentials (such as a username and password) to the web application or system for verification, which is transmitted to the authentication server.
2. **Token Generation**: Upon successful authentication, the server generates a JSON token containing critical information about the user and the authentication session. The server sends the token to the client for verification.
3. **Token Storage**: The client stores the token, usually in a cookie or purpose-marked local storage, and includes it in subsequent requests to the server.
4. **User Verification**: When the client sends a request to the application server, it verifies the signature in the token and checks the claims in the payload to ensure that the user can access the requested resource.
5. **Server Response**: If the JWT is valid and the user can access the requested resource.
6. **Token Expiration**: When the JWT expires, the client must obtain a new JWT by logging in again.

JWT authentication provides several advantages over traditional session-based authentication, including improved scalability and reduced server-side storage requirements. However, it is important to properly secure and manage JWTs to prevent unauthorized access to sensitive information.

What Are the Best Practices for Using JWT Authentication?

Generally, there are best practices for **when** and **when not to use** JWT Authentication:

- **When to Use**: JWT authentication can be useful in scenarios where the server **needs to handle many requests and sessions or in stateless APIs**. JWTs can simplify authentication by reducing the number of database calls required for session management and can be passed between microservices to maintain stateless communication.
- **When Not to Use**: JWT authentication may not be suitable for applications where the payload contains sensitive information, such as **payment details**, that must be protected against unauthorized access. JWTs can also pose a security risk if not properly secured, as anyone with access to a valid token can access protected resources. In these scenarios, a session-based authentication mechanism may be more appropriate.

Additionally, there are always i**mplementation best practices** when you've decided to use the technology, including:

Use Strong Encryption: Choose a strong cryptographic signing algorithm, such as, RS256, to sign JWTs. Avoid using insecure algorithms or plaintext.

Keep **Sensitive Data** on the Server: **Do not include** sensitive information in the token **payload**, such as passwords or credit card numbers. Instead, store this information server-side and retrieve it as needed.

Use Short Expiration: Set a short expiration time (around 15-30 minutes) for tokens to reduce the risk of a stolen token being used maliciously..

Use **Secure HyperText Transfer Protocol** (HTTPS): Use HTTPS to encrypt data in transit and prevent man-in-the-middle attacks.

Implement Token Revocation: Consider implementing token revocation to invalidate tokens that have been compromised or are no longer needed.

What Are Some Challenges to Avoid When Implementing JWT Authentication?

While there are some significant benefits in implementing JWT authentication, there are always assumptions and pitfalls to avoid. These include:

Storing Private Data in the Token: Even with encryption, passing sensitive data across authentication requests is not good practice and can result in compromised accounts.

Encryption Failures: Weak algorithms can open the JWT to attacks, such as signature forgery or token tampering. It's best to use strong cryptographic signing algorithms, such as RS256.

Not Validating Tokens: Failing to validate token signatures or expiration times can allow attackers to use stolen or expired tokens to access protected resources.

Using Long or No Expiration: While it may be tempting to set long expiration times to reduce the frequency of logins and "improve" user experience, doing so can increase the risk of a stolen token being used maliciously.

The Advantages and Disadvantages of Using JSON Web Tokens (JWTs) in Web Security

Advantages:

1. **Statelessness**: JWTs eliminate the need for server-side session storage, making authentication stateless and scalable.

2. **Cross-Domain Compatibility**: JWTs can be used across different platforms, enabling consistent authentication and authorization in heterogeneous environments.

3. **Security**: JWTs can be signed and encrypted, providing integrity and authenticity verification, mitigating security threats.

4. **Decentralized Authorization**: JWTs support fine-grained access control policies, reducing reliance on centralized servers for authorization.

5.**Performance**: JWTs improve application performance by reducing database round-trips and network latency for token validation.

In Brief, while JWTs offer advantages like statelessness and cross-domain compatibility, addressing security risks and scalability challenges is crucial for effective implementation. Understanding the trade-offs can guide developers in leveraging JWTs for secure authentication and authorization mechanisms.

Disadvantages:

1. **Security Risks**: Vulnerabilities like insecure token storage and insufficient signature validation pose security risks.

2. **Token Size**: Base64 encoding increases token size, leading to increased bandwidth usage and latency.

3. **Limited Revocation**: JWTs cannot be revoked before expiration without complex mechanisms like token blacklisting.

4. **Payload Size**: Including excessive information in token payloads can risk information leakage and privacy violations.

5. **Cryptographic Overhead**: Verifying JWT signatures introduces computational overhead, impacting performance, especially under high loads.

6. **Non-revocable:** Due to their self-contained nature and stateless verification process, it can be difficult to revoke a JWT before it expires naturally. Therefore, actions like banning a user immediately cannot be implemented easily. That being said, there is a way to maintain JWT deny / black list, and through that, we can revoke them immediately.

7. **Dependent on one secret key**: The creation of a JWT depends on one secret key. If that key is compromised, the attacker can fabricate their own JWT which the API layer will accept. This in turn implies that if the secret key is compromised, the attacker can spoof any user's identity. We can reduce this risk by changing the secret key from time to time.

Is JWT necessary over HTTPS communication?

JSON Web Tokens (JWTs) are **not strictly necessary** over HTTPS communication, but they are commonly used together for enhanced security. Reasons:

1. **Data Integrity**: HTTPS ensures that data transmitted between the client and server is encrypted and cannot be tampered with by malicious actors. While JWTs can provide integrity through signature verification, HTTPS adds an additional layer of protection at the transport layer.
2. **Confidentiality**: HTTPS encrypts the data in transit, preventing unauthorized parties from eavesdropping on the communication. This is particularly important when JWTs contain sensitive information.
3. **Authentication**: HTTPS verifies the identity of the server to the client, mitigating the risk of man-in-the-middle attacks. While JWTs authenticate users to the server, HTTPS authentication ensures the server's authenticity to the client.
4. **Compliance**: Many security standards and regulations, such as PCI DSS (Payment Card Industry Data Security Standard)and GDPR(General Data Protection Regulation), require the use of HTTPS to protect sensitive data. Compliance with these standards often necessitates the use of HTTPS alongside JWTs.

In brief, while JWTs can provide security features such as authentication and data integrity, using them in conjunction with HTTPS communication enhances overall

security by encrypting data in transit and verifying the server's identity. Therefore, while not strictly necessary, employing JWTs over HTTPS communication is a best practice for securing web applications and APIs.

In conclusion, JSON Web Tokens (JWTs) offer a streamlined and versatile approach to web security, facilitating efficient authentication and authorization mechanisms. While JWTs provide numerous benefits, including statelessness and cross-domain compatibility, it's essential to address associated challenges such as security vulnerabilities and token management. By implementing JWTs alongside HTTPS communication and adhering to best practices, developers can enhance the security of web applications and APIs, ensuring the integrity and confidentiality of user data in today's digital landscape.

The references:

1. **"What is JWT?" Retrieved from https://supertokens.com/blog/what-is-jwt#**
2. **Stack Overflow. "Is JWT necessary over HTTPS communication?" Retrieved from https://stackoverflow.com/questions/45978013/is-jwt-necessary-over-https-communication**
3. **Reddit. "Is there any benefit to using a JWT token in an HTTPs only web application?" Retrieved from https://www.reddit.com/r/django/comments/19dxhuj/is_there_any_benefit_to_using_a_jwt_token_in_an/**

By: Ghaliah Almutairi