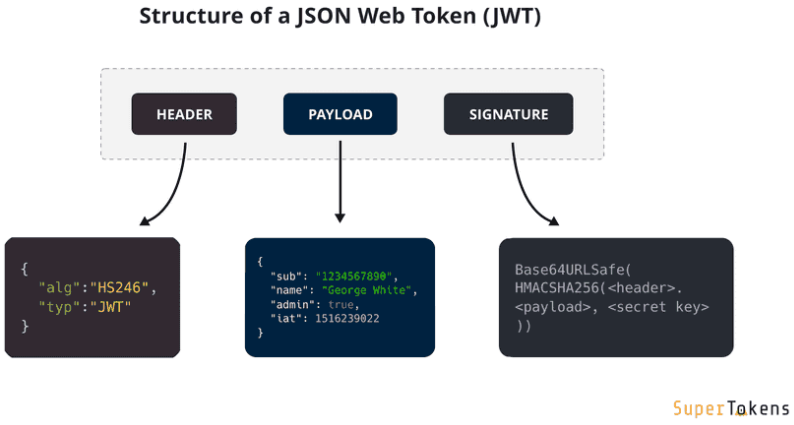
# JSON Web Tokens

JWT stands for JSON Web Token. It's a compact, URL-safe means of representing claims to be transferred between two parties. The claims in a JWT are encoded as a JSON object that is used as the payload of a JSON Web Signature (JWS) structure or as the plaintext of a JSON Web Encryption (JWE) structure, enabling the claims to be digitally signed or integrity protected with a Message Authentication Code (MAC) and/or encrypted.

JWTs can be used for authentication and authorization in web applications and APIs. They are often used as tokens in authentication and authorization protocols such as OAuth 2.0.

**Structure of JWT:**




1. **Header**: The header contains metadata about the type of token and the cryptographic algorithms used for its protection. This section typically specifies the token's type (JWT), along with the signing algorithm employed (e.g., HMAC SHA256 or RSA).

2. **Payload**: The payload encapsulates the claims being conveyed. Claims are statements about an entity (e.g., user identity, permissions, and additional data), categorized into three types: registered, public, and private claims. This section allows for the inclusion of custom data relevant to the application's context.
3. **Signature**: The signature provides a means of verifying the token's authenticity and integrity. By signing the header and payload with a secret key or a private key in the case of asymmetric cryptography, JWTs can be validated to ensure they have not been tampered with during transit. This step prevents unauthorized modifications and guards against token forgery.

**Applications in Authentication and Authorization**

JWTs serve as tokens in authentication and authorization processes within web applications and APIs. Their versatility and efficiency make them ideal for facilitating secure communication between clients and servers. Here's how JWTs are typically used in this context:

**Authentication:** When a user logs in or authenticates, the server issues a JWT containing relevant user information (e.g., user ID, roles) as claims in the payload. The client receives this token and includes it in subsequent requests as a means of identifying itself. The server verifies the JWT's signature to authenticate the user's identity and grants access accordingly.

**Authorization:** JWTs can also carry authorization data, specifying what actions or resources the client is permitted to access. This information is conveyed through the claims in the payload. By examining these claims, servers can enforce fine-grained access control policies, ensuring that clients only interact with resources for which they have the requisite permissions.

**Advantages of Json Web Tokens:**

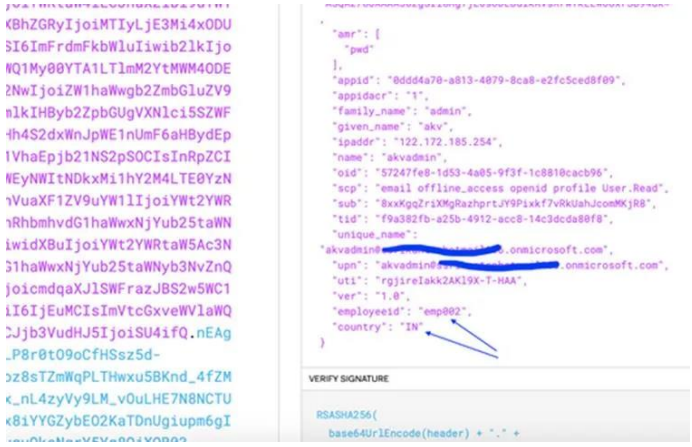
The widespread adoption of JWTs underscores their importance in modern web development for several reasons:

- **Statelessness**: JWTs are self-contained and do not require server-side storage of session state. This statelessness simplifies scaling and enhances performance, as servers do not need to maintain session data for individual clients.
- **Interoperability**: JWTs are based on open standards and enjoy broad support across various programming languages and platforms. This interoperability facilitates integration with existing systems and promotes compatibility in diverse technological ecosystems.
- **Security**: Through cryptographic signing, JWTs ensure data integrity and guard against tampering or unauthorized access. By verifying the signature, servers can trust the claims contained within the token, mitigating security risks associated with transmitting sensitive information over untrusted networks.

## Types of JWT attacks:

1. **Information Leakage:**

- JSON Web Tokens (JWTs) are commonly employed for access control, often carrying user-related information within their payload. However, when these tokens lack encryption, their contents are vulnerable to exploitation. By simply base64 decoding the token, any party can access and read the token's payload. Consequently, if sensitive information is stored within the token, it becomes susceptible to leakage.
- While a well-implemented signature section in the JWT ensures data integrity, it does not guarantee confidentiality. Therefore, even though the token's integrity remains intact, the information it carries can still be exposed if adequate encryption measures are not in place.




2. **None Algorithm Attack:**

- JWT supports a “none” algorithm. If the alg field is set to “none”, any token would be considered valid if their signature section is set to empty.

```
{
 "alg" : "none",
 "typ" : "JWT"
}


{
 "user" : "admin"
}
```

**3. JWT Token Tampering:**

- Attackers may modify the token's contents (header, payload, or signature) to impersonate other users, escalate privileges, or bypass access controls. Tampering with the token can lead to unauthorized access to sensitive resources.

**4. Brute Force Attacks on JWT Signing Key:**

Attackers attempt to brute force the signing key used to generate JWT signatures. By guessing the key through exhaustive trial and error, attackers can forge valid JWTs, leading to unauthorized access and data manipulation.

**5. Side Channel Attacks:**

Attackers exploit side channels, such as timing attacks or error messages, to glean information about JWTs or cryptographic operations. By analyzing subtle variations in response times or error messages, attackers may infer sensitive details and mount targeted attacks.

**4. KID Parameter Vulnerability:**

The KID (Key ID) parameter is a critical component within JWT tokens. However, if this parameter lacks proper validation, it opens the door to various attacks, including Command Injection, Local File Inclusion (LFI), SQL Injection (SQLi), and more. Typically, the KID is utilized to fetch a key file from the file system. If this parameter isn't adequately sanitized before utilization, it can pave the way for a directory traversal attack, enabling malicious actors to access unauthorized directories.

## Impact of JWT Token Hacking:

1. **Sensitive Information Disclosure:** Unauthorized access to JWT tokens can lead to the disclosure of sensitive information contained within the token's payload, potentially exposing user credentials, personal details, or other confidential data.
2. **Compromised Client Authenticity:** Hacking JWT tokens can compromise the authenticity of clients, allowing malicious actors to impersonate legitimate users and gain unauthorized access to resources or perform malicious actions on behalf of the compromised client.

3. **Account Takeover:** By exploiting vulnerabilities in JWT tokens, attackers can orchestrate account takeover attacks, gaining full control over user accounts and their associated privileges. This can result in unauthorized transactions, data manipulation, or other harmful activities.
4. **Access to Server Files:** Successful JWT token hacking may grant attackers access to server files, potentially compromising sensitive data or exposing proprietary information. This unauthorized access can lead to data breaches, system compromises, or disruption of services.
5. **Exfiltration of Database Data:** In some cases, attackers exploiting JWT token vulnerabilities may leverage their access to read data from the underlying SQL database. This can result in the exfiltration of sensitive data, including user records, financial information, or other critical assets, posing significant risks to the organization and its stakeholders.

**Mitigation of JWT attacks:**

1. **Sensitive Information Disclosure:**
    - **Use Encryption:** Encrypt sensitive information within the JWT payload to prevent unauthorized access. Employ strong encryption algorithms to safeguard data confidentiality.
    - **Limit Information:** Avoid storing excessive sensitive information within JWT tokens. Only include necessary data required for authentication and authorization purposes.
2. **Compromised Client Authenticity:**
    - **Implement Secure Authentication:** Utilize multi-factor authentication (MFA) and strong authentication mechanisms (e.g., OAuth 2.0, OpenID Connect) to enhance client authentication security.
    - **Token Revocation:** Implement token revocation mechanisms to invalidate compromised tokens promptly. Utilize token blacklists or token expiration strategies to mitigate the impact of stolen tokens.

3. **Account Takeover:**
    - **Security Awareness:** Educate users about the importance of strong passwords, account security practices, and the risks associated with phishing attacks.
    - **Monitoring and Detection:** Implement monitoring systems to detect unusual account activity, such as login attempts from unfamiliar locations or devices. Employ anomaly detection techniques to identify potential account takeover attempts.
4. **Access to Server Files:**
    - **Strict Access Controls:** Apply strict access controls to server files and directories, limiting access only to authorized personnel. Employ the principle of least privilege to restrict access based on user roles and responsibilities.
    - **File Integrity Monitoring:** Implement file integrity monitoring systems to detect unauthorized changes to server files. Regularly audit file permissions and configurations to ensure compliance with security policies.
5. **Exfiltration of Database Data:**
    - **Parameterized Queries:** Utilize parameterized queries or prepared statements to prevent SQL injection attacks. Sanitize user input to mitigate the risk of malicious SQL queries.
    - **Database Encryption:** Encrypt sensitive data stored in the database to protect against unauthorized access. Implement robust encryption algorithms and key management practices to safeguard data confidentiality.
6. **KID Parameter Vulnerability:**
    - **Input Validation:** Validate and sanitize user input, including the KID parameter, to prevent injection attacks and directory traversal vulnerabilities. Implement strict input validation routines to reject malicious input.

- **Whitelisting:** Implement whitelisting of acceptable values for the KID parameter to restrict access to authorized directories or resources.

**Reference:**


https://supertokens.com/blog/what-is-jwt

https://medium.com/@musab_alharany/10-ways-to-exploit-json-web-token-jwt-ac5f4efbc41b

https://book.hacktricks.xyz/pentesting-web/hacking-jwt-json-web-tokens

https://medium.com/@rajeevranjancom/jwt-json-web-token-attacks-6b82185ffed

https://portswigger.net/web-security/jwt