CS-067 · API Security

JSON Web Tokens (JWT)

A compact, signed way to carry identity and claims between services, and how to use it safely.

Executive Summary

A JSON Web Token, or JWT, is a compact, digitally signed container that carries identity and claims between parties in a way each side can verify. It lets a server confirm who a caller is without storing session state, which makes it popular for APIs and single sign-on. The security of a JWT depends entirely on validating its signature, its claims, and its lifetime correctly on every use.

What It Is

A JWT is a string made of three parts joined by dots: a header describing the token type and signing algorithm, a payload of claims such as the subject, issuer, audience, and expiration time, and a signature. The header and payload are encoded, not encrypted, so anyone can read them; the signature is what makes the token trustworthy, because it proves the token was issued by a party holding the signing key and has not been altered. JWTs are commonly used as access tokens and identity tokens in modern authentication flows. Because the token itself carries the claims, a server can verify a caller by checking the signature and the claims rather than looking up a session in a database, which is why JWTs are often called stateless.

Why It Matters

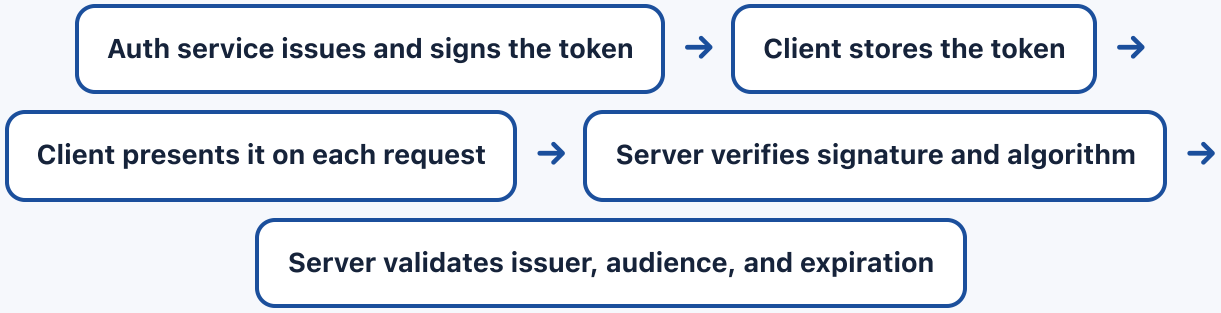
JWTs sit at the center of authentication for a large share of modern web and mobile applications, so a mistake in how they are validated can undermine an entire system. If a server accepts a token without properly checking the signature, the algorithm, the issuer, the audience, and the expiration, an attacker may be able to present a forged or stale token and impersonate any user. Because the payload is readable by anyone, developers who mistake encoding for

encryption may place sensitive data in the token by accident. For engineers, understanding JWT validation is one of the highest-leverage security skills, because it is both widely used and frequently implemented incorrectly.

How It Works

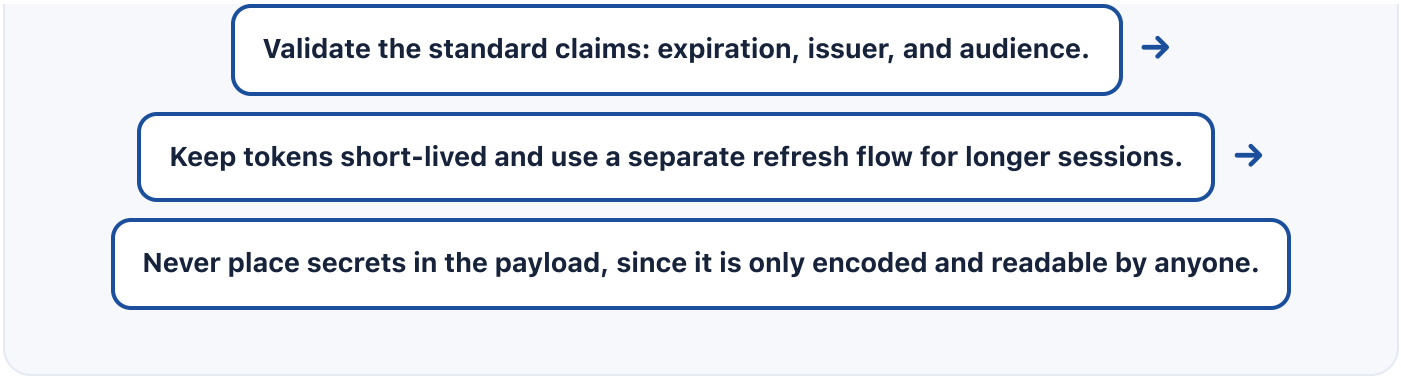
An identity provider or authentication service creates a JWT, signs it with a secret key or a private key, and returns it to the client. The client presents the token on later requests, typically in an authorization header. The server verifies the token by checking the signature against the expected key, confirming the algorithm matches what it expects, and validating the standard claims: that the token has not expired, was issued by the trusted issuer, and is intended for this audience. Only after all of these pass does the server trust the claims inside. Signatures can use a shared secret or a public and private key pair; with a key pair, services can verify tokens using a public key without ever holding the signing key. Tokens should be short-lived, and longer sessions should use a separate refresh mechanism rather than one long-lived token.

Architecture Diagram



A JWT is issued and signed once, then verified on every request by checking its signature and claims.

Visual Workflow



Common Attacks

- Accepting a token whose algorithm was switched to none, bypassing signature checks
- Confusing a public key for a shared secret so an attacker can sign their own tokens
- Replaying a stolen token that has a long or missing expiration
- Weak signing keys that can be brute forced to forge valid signatures
- Trusting claims without verifying the issuer or audience, allowing token reuse across services

Common Mistakes

- Not verifying the signature at all, or verifying it inconsistently
- Allowing the token to specify any algorithm instead of pinning an expected one
- Storing sensitive data in the payload, mistaking encoding for encryption
- Issuing long-lived tokens with no expiration or no way to revoke them
- Storing tokens where scripts can read them, exposing them to theft
- Failing to check the audience, so a token for one service is accepted by another

Best Practices

- Pin the expected signing algorithm and reject the none algorithm outright
- Verify the signature and validate expiration, issuer, and audience on every request
- Use short expiration times and a separate, revocable refresh mechanism
- Use strong keys and rotate them, favoring key pairs for multi-service verification
- Keep only non-sensitive claims in the payload
- Store tokens securely on the client and transmit them only over TLS

Quick Checklist

- [x] Signing algorithm pinned, none algorithm rejected
- [x] Signature verified on every request against the expected key
- [x] Expiration, issuer, and audience claims validated
- [x] Token lifetimes short, with a revocable refresh path
- [x] No sensitive data placed in the payload
- [x] Tokens transmitted only over TLS and stored securely on the client

Recommended Tools

JWT verification library
Validates signature, algorithm, and standard claims in code

Identity provider
Issues and signs tokens and publishes verification keys

Token inspection tool
Decodes a token for debugging without exposing it in production

Key management service
Stores and rotates signing keys securely

Industry Standards

JSON Web Token (RFC 7519)
Defines the JWT structure and standard claims

JSON Web Signature (JWS)
Defines how tokens are signed and verified

OWASP JSON Web Token Cheat Sheet guidance
Practical validation and handling recommendations

Career Relevance

JWT handling is a daily concern for backend engineers, application security engineers, and API security engineers, who issue, validate, and troubleshoot tokens. Security engineers and penetration testers routinely test JWT implementations for the well-known validation flaws, and identity and AI governance professionals need to understand tokens because they underpin access to modern APIs and AI services.

Interview Questions

- What are the three parts of a JWT, and which one makes it trustworthy?
- Is a JWT encrypted? What does that mean for the data you put in it?
- How does the none algorithm attack work, and how do you prevent it?
- Why should JWTs be short-lived, and how do refresh tokens fit in?
- What standard claims should you validate on every request, and why?

Related Certifications

OWASP-aligned application security training (GWAPT)

GIAC Web Application Penetration Tester

CompTIA Security+ (foundational)

Further Reading

- RFC 7519: JSON Web Token
- OWASP API Security Project
- OWASP Cheat Sheet Series

Key Takeaways

- A JWT is signed, not encrypted, so its payload is readable by anyone.
- The signature is what makes a token trustworthy, and it must be verified every time.
- Pin the expected algorithm and reject the none algorithm to prevent forgery.
- Validate expiration, issuer, and audience, and keep tokens short-lived.
- Never store secrets in a JWT, and always transmit tokens over TLS.

FAQ

Related Careers

RELATED ROLES

Application Security Engineer

Api Security Engineer

Backend Engineer

Security Engineer

RELATED CERTIFICATIONS

OWASP-aligned application security training

GIAC Web Application Penetration Tester (GWAPT)

CompTIA Security+ (foundational)

CURRENT OPENINGS

Live roles are on the job board.

Browse all jobs

GUIDES & SALARY

Career guides

Role roadmaps

Salary data

Career tools

SUGGESTED LEARNING PATH

1. Ground the basics with CS-001 Cybersecurity
2. Study this sheet: **JSON Web Tokens (JWT)**
3. Go deeper: REST API Security
4. Go deeper: OAuth 2.0
5. Validate it: work toward **OWASP-aligned application security training**
6. Find the role: browse current openings

RELATED SHEETS

CS-066 REST API Security | CS-068 OAuth 2.0 | CS-069 OpenID Connect | CS-062 Authentication | CS-063 Authorization