JWT security, common attacks and defenses.

A JWT (JSON Web Token) is a self-contained authentication token: the application puts the user's identity and claims into a small JSON document, signs it cryptographically, and gives it to the browser to send back on every request. JWTs are the default token format for most modern APIs. The format is simple to use and the cryptography is well-understood; the security problems come from configuration mistakes that turn the token from an authentication primitive into a backdoor.

HOW IT WORKS

01 What are the most common JWT attacks?

Five patterns we see in production engagements:

- alg=none. The JWT header names the algorithm. Some libraries historically accepted `alg: none`, which means 'no signature required'. An attacker forges a token with arbitrary claims, sets `alg: none`, sends it. The server reads the claims and trusts them.
- Algorithm confusion (RS256 -> HS256). RS256 uses a public/private key pair; HS256 uses a single shared secret. If the application accepts both and the public key is, well, public, the attacker can sign a token using the public key as the HS256 secret. The server validates and accepts it.
- Weak HS256 secret. When HS256 is used with a guessable secret, anyone who can capture a real token can crack the secret offline and then forge arbitrary tokens.
- JKU / X5U / KID injection. Some libraries fetch the verification key from a URL or key ID in the token header. If the application does not strictly validate that source, the attacker points it at a key under their control and signs the token themselves.
- No signature verification at all. The application reads the claims and uses them without verifying the signature. Common when a developer adds 'just decode the JWT to read the user ID' without realizing decode and verify are different operations in most libraries.

SOURCES

[1] OWASP API2:2023 Broken Authentication

[2] OWASP JSON Web Token Cheat Sheet

[3] RFC 7519 JSON Web Token

[4] CVE-2015-9235 (JWT alg=none in node-jsonwebtoken)

02 What do attackers do with a compromised JWT setup?

Most JWT attacks lead directly to account takeover. The attacker forges a token with another user's ID, sends it, and now operates as that user for the token's lifetime.

The blast radius escalates when:

- The claims include roles or scopes. Forge `role: admin` and the attacker is now an admin.
- The token has no expiry or a very long expiry. The compromise persists.
- The same JWT setup is shared across multiple services. One forged token works against all of them.
- The application stores sensitive data in the JWT payload thinking it is private. The payload is base64-encoded, not encrypted. Anyone with the token can read it.

03 How do you configure JWTs safely?

- Pin the algorithm. Hardcode the expected algorithm in your verification code. Reject tokens with any other algorithm, including `none`.
- Use asymmetric signing where possible. RS256 or EdDSA. The signing key stays on the auth service; verifiers only ever see the public key.
- If using HS256, use a long random secret. At least 256 bits. Generated, never typed. Never committed to a repository.
- Set short expiries. Minutes for access tokens. Refresh-token flows handle the user experience.
- Validate every claim that affects authorization. Issuer, audience, not-before, expiry, and any custom role / scope claims.
- Do not put secrets in the payload. The payload is readable by anyone with the token. Put references (user ID, session ID) and look up the secret server-side.
- Use a well-maintained library. Roll-your-own JWT verification is where alg-confusion and missing-checks bugs come from.

04 How does SecureLayer7 test JWT implementations?

Every API engagement that uses JWTs runs through the standard attack matrix.

- Forge a token with `alg: none`. Does it work?
- Try algorithm confusion (RS256 -> HS256 with the public key). Does it work?
- Crack the HS256 secret offline. Can we recover it?
- Modify the claims (`role`, `user_id`, `scope`) without re-signing. Does the server accept?
- Plant a JKU / X5U URL pointing at our server. Does the verifier fetch from there?
- Decode the payload. Is anything sensitive in there that should not be?
- Replay an expired token. Is expiry actually checked?

Deliverable maps findings to OWASP API2:2023 (Broken Authentication) with the specific library and configuration changes required.

Test your JWT setup against the full attack matrix.

securelayer7.net/learn/application-security/jwt-attacks

Open online