# JWT Attacks

**JSON Web Token (JWT)** is a **compact, URL-safe** way of representing claims between two parties. JWTs are commonly used in:

* API authentication
* Session tokens
* Identity assertions (OAuth2, OpenID Connect)

A JWT has three parts: <Header>.<Payload>.<Signature>

# What Makes JWT Vulnerable?

JWTs are **only as secure as:**

* The secret key (HMAC) or private key (RSA)
* The signing algorithm
* The implementation validating them

# Common JWT Attack Vectors

## 1. None Algorithm Attack (alg: none)

**Most Famous**

| Description | JWTs specify their signing algorithm in the header. Some servers (misconfigured) accept alg: none, skipping signature verification. |  
Payload

{

"alg": "none",

"typ": "JWT"

}

Attacker removes the signature and modifies claims.

**Fix**: Disallow none algorithm server-side.

## 2. Algorithm Confusion Attack (HS256 to RS256)

**High Impact**

| Description | If the server accepts asymmetric (RS256) and symmetric (HS256) algorithms, attacker can confuse it. |

* Server expects RS256 (uses private key).
* Attacker switches to HS256, and uses **public key as HMAC secret**.

{

"alg": "HS256"

}

Attacker forges signature using public key.

**Fix**:

* Do not allow algorithm switching.
* Enforce fixed algorithms.

## 3. Brute Force JWT Secret (HS256)

| Description | If server uses **weak secret key**, attacker can brute-force it and forge valid tokens. |  
| Tools | jwt-cracker, jwt\_tool.py, JohnTheRipper  
| Example | Try to guess secret used in HMAC SHA256.

**Fix**: Use long, random secrets (≥256-bit).

## 4. JWT Signature Stripping

| Description | Web proxies or middlewares strip Authorization headers and replace JWTs, leading to bypass.  
| Impact | Attacker may downgrade from secure JWT to anonymous access.

**Fix**:

* Validate token presence server-side.
* Secure all reverse proxies.

## 5. Expired Token Replay

Description | Reuse of expired tokens where **expiration is not enforced**, or only enforced on client-side.  
| Payload Manipulation |

"exp": 999999999999

Attacker alters expiry in payload if server doesn’t re-sign.

**Fix**: Enforce server-side expiration; don’t trust payload.

## 6. Kid Header Injection (Key ID Confusion)

| Description | kid (Key ID) header used to select the key for signature verification.  
| Attack | Supply kid as a file path or command injection:

{

"kid": "../../../../etc/passwd"

}

**Fix**: Sanitize kid input, don’t use dynamic file loads.

## 7. Client-Side JWT Storage Vulnerabilities

| Description | If JWT stored in **localStorage**, vulnerable to XSS.  
| Risk | Attacker can steal JWT and impersonate user.

**Fix**:

* Store JWTs in **HttpOnly cookies**
* Prevent **XSS** in entire app

# Tools

| Tool | Description |
| --- | --- |
| jwt.io | Decode and analyze JWTs |
| JWT Tool (ticarpi) | Forge, brute-force, test JWTs |
| jwt\_tool.py | Test for alg:none, HS256 brute-force, RS/HS confusion |
| Postman / Burp Suite | Modify tokens manually |
| Crackstation / Hashcat / John | Crack weak secrets |

# Mitigation

## 1. Strong Key Management

* Use **long (≥256-bit)** secrets for HMAC (HS256)
* Use **secure key storage** (e.g., AWS KMS, HashiCorp Vault)

## 2. Enforce Signing Algorithms

* Hardcode to **HS256** or **RS256**
* Reject alg: none or mixed algs

## 3. Validate All JWT Claims

* Check:
  + exp (Expiration)
  + iat (Issued at)
  + nbf (Not before)
  + aud, iss, sub

## 4. Secure Token Storage

* Use **HttpOnly**, **Secure** cookies
* Avoid localStorage or sessionStorage for sensitive tokens

## 5. Prevent XSS

* JWT theft via XSS is common
* Use **CSP**, sanitize DOM inputs, encode outputs

## 6. Log and Monitor Token Usage

* Detect token reuse, expired tokens, sudden IP changes

# Points

“JWTs are not encrypted — they’re base64 encoded. People mistake that for security.”

“JWT vulnerabilities often stem from misconfiguration, not flaws in JWT itself.”

“Always test for alg: none, key confusion, weak secrets, and expired token reuse during JWT assessments.”

# Real-World Cases

| Org | Issue |
| --- | --- |
| Auth0 (old version) | Accepted alg: none in older lib |
| Buffer (2018) | Poor token validation logic led to privilege escalation |
| Several APIs | Accepted forged tokens due to missing signature verification |