

# JSON Web Token (JWT) Vulnerability Analysis Report

**Tester:** Blessing Isaiah  
**Report Date:** November 28, 2025

## Executive Summary

### Objective

The goal of this assessment is to examine the JSON Web Token and identify five critical weaknesses in its design or usage. Each weakness requires a clear explanation of why it poses a security risk and how an attacker could turn it into an exploit. The objective focuses on understanding both the technical flaw and the real world impact.

### Methodology

The evaluation focused entirely on the token itself, including its header, payload, signing method, and structural integrity. This allowed identification of flaws that an attacker could exploit without needing access to the underlying application.

### Major Tools Used

Kali Linux provided a controlled testing environment.  
JWT\_Tool and xJwt.io assisted in decoding, tampering, and inspecting the token.  
The JWT.io debugger allowed quick review of header, payload, and signature behavior during manipulation.  
These tools ensured consistent and reliable evaluation of the security properties of the token.

## Findings: Critical JWT Security Flaws

JWT Analyzed:

```
eyJhbGciOiJIub251IiwidHlwIjoiSldUIn0.eyJ1c2VyIjoiYWRTaW4iLCJleHAiOjE2MDAwMDAwMDAsInJvbGUiOiJhZG1pbiIsInZhbGkiOiJpmYWxzZX0[unsigned]
```

Header (decoded):

```
JSON CLAIMS TABLE COPY ↗
{
  "alg": "none",
  "typ": "JWT"
}
```

### Payload (decoded):

```
Token payload values:
[+] user = "admin"
[+] exp = 1600000000    ⇒ TIMESTAMP = 2020-09-13 08:26:40 (UTC)
[+] role = "admin"
[+] valid = False
[-] TOKEN IS EXPIRED!
```

## Flaw 1: Algorithm set to "none"

### Why it is a vulnerability:

The token carries no signature, so nothing prevents an attacker from altering its contents. A JWT with **alg: none** acts like an unsigned JSON object, so an attacker edits the payload directly and sends it back to the server.

### Recommendation:

Backend must reject tokens where the algorithm is set to none and enforce strict allowed-algorithm validation.

## Flaw 2: Role stored directly in the payload

### Why it is a vulnerability:

The backend risks trusting a client-controlled role claim that should never come from the user side. When roles like "admin" appear openly in the payload, an attacker modifies them because they are not protected by any signature.

### Recommendation:

Assign roles server-side only and avoid trusting any privilege-related claim from the client.

## Flaw 3: Expired timestamp with unclear enforcement

### Why it is a vulnerability:

The expiration field is already outdated, which hints that the server might not be validating token lifetimes. If the backend accepts an expired token, attackers keep using it long past its intended lifetime.

### Recommendation:

Enforce strict expiration checks and reject any token where **exp** is in the past.

## Flaw 4: Cleartext exposure of all claims

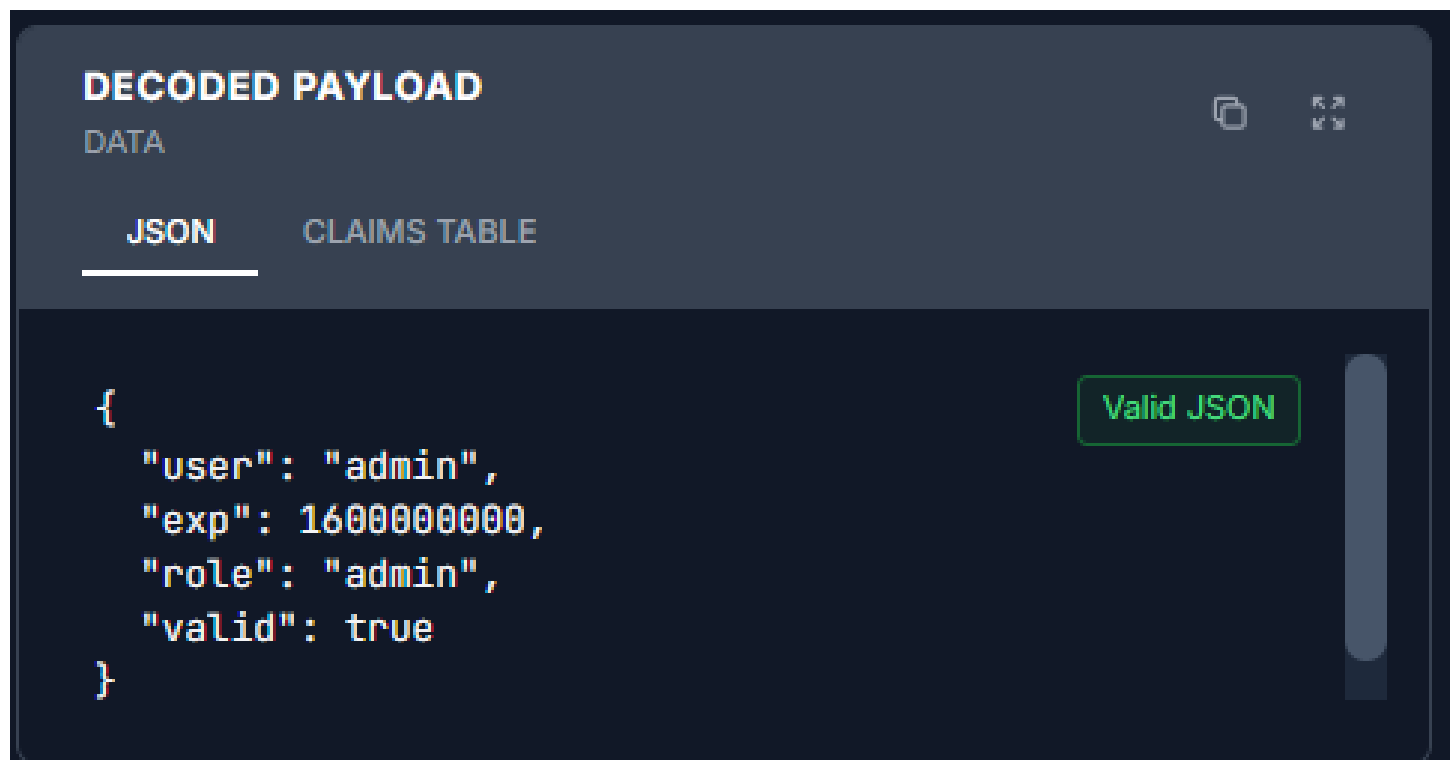
### Why it is a vulnerability:

Attackers decode the base64 payload to gather intelligence about users, roles, internal naming conventions, and system structure.

### Recommendation:

Avoid placing sensitive identifiers in the payload and encrypt or minimize claims when possible.

## Flaw 5: Client-controlled “valid” flag



New token

```
eyJhbGciOiJub25lIiwidHlwIjoiSldUIIn0.eyJ1c2VyIjoiYWRTaW4iLCJleHAiOjE2MDAwMDAwMDAsInJvbGUiOiJhZG1pbiIsInZhbGkIjp0cnVlfQ[unsigned]
```

**Why it is a vulnerability:**

The valid field is a Boolean value in the payload, which means the user can change it and regenerate a new token because there the restriction in place.

**Recommendation:**

Never rely on a user-provided field to determine account status or authorization.