Draft Report: JWT Security: Vulnerabilities and Proof of Concept

1. Introduction

JSON Web Tokens (JWTs) are a common way to handle login sessions in modern web apps. They are small pieces of data that contain information like who the user is and what role they have. A JWT has three parts:

Header – tells which algorithm is used.

Payload – the actual data (like username, role).

Signature – makes sure the token wasn't changed.

Because JWTs are self-contained and don't require saving session data on the server, they are very popular. But if JWTs are not used properly, they can open up serious security problems.

2. Common JWT Vulnerabilities

Here are some of the main issues that happen with JWTs:

<u>alg=none issue</u>

Some JWT libraries accept tokens where the algorithm is set to none.

If this happens, attackers can edit the token however they want, and the server might still trust it.

<u>Weak secrets</u>

If the secret key is too simple (like secret123 or password), an attacker can guess it.

Once they find the key, they can create their own tokens and pretend to be anyone (even an admin).

<u>Algorithm confusion</u>

Some apps mix up the difference between HS256 (symmetric key) and RS256 (public/private keys).

If not checked properly, attackers can trick the server into accepting fake tokens.

Replay attacks

If someone steals a valid token, they can keep using it until it expires.

This is especially dangerous if tokens have a long lifetime.

3. Proof of Concept Demonstration

For this project, I made a small demo that shows how weak secrets can be cracked.

First, I created a JWT with the secret "secret123".

Then, using Python, I wrote a script that tries different words from a wordlist.

The script was able to guess the correct secret and then create a fake token with "role": "admin".

This shows how easy it is to take over accounts if the secret key is weak.

4. Mitigations

To make JWTs secure, developers should:

- Use long and random secret keys (not simple words).
- Use RS256 or ES256 algorithms instead of HS256 when possible.
- Always add an expiry time (exp) so stolen tokens don’t last forever.
- Use secure storage for tokens (for example, HttpOnly cookies instead of localStorage).
- Never allow alg=none. Always check which algorithm is being used.
- Keep a way to block/revoke tokens if they are leaked.

5. Conclusion

JWTs are very useful, but if used carelessly, they can lead to big security risks. My PoC showed how a weak secret can be brute-forced and used to make a fake admin token. In real-world apps, this could mean complete account takeover.

By following good practices like using strong keys, proper algorithms, and short expiry times, JWTs can remain a safe way to manage user sessions.

6. References

1. OWASP: JSON Web Token Cheat Sheet
2. PortSwigger: JWT Attacks
3. Auth0: Critical JWT Vulnerabilities