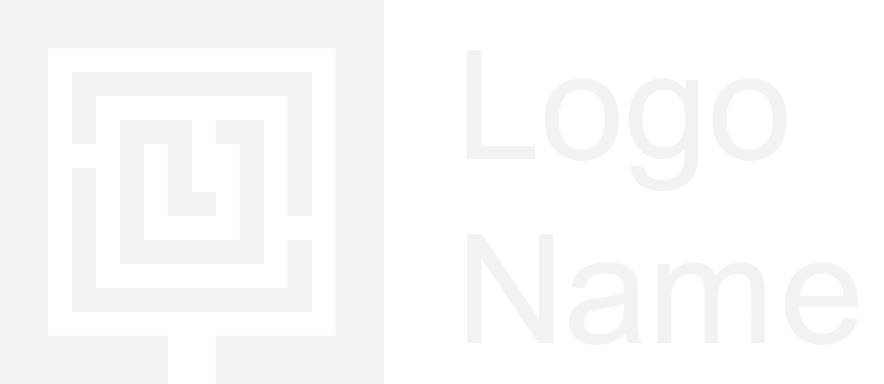
|  |
| --- |
| Research Document  What are the security implications and best practise for using JSON Web Tokens(JWTs) for web applications? |
| Fontys University of Applied Sciences Alaa Tarakji  S3-CB2 |



## Introduction:

The purpose of this document is to give a better understanding of JWTs and how can we use it in a secure way for web applications.

Nowadays, it is crucial for every application to have well-done security, as the IT industry is developing quit fast, comparing to how it was, for example in the 90’s. Over the last few decades, developers had to come up with solutions to strength up the security to avoid hackers attack and to protect each user’s personal data, since most of the data is stored in databases.

Therefore, a secure way of using JWTs needs to be found in order to improve the security of the application, this being the main question of this research document. To answer this, 3 sub questions have been added to understand the basic knowledge about JWTs, following ways on how to store it and test it, and my recommendations on what would be the most secure way of using JWTs in the fastest way.

## Research Question:

What are the security implications and best practice for using JSON Web Tokens (JWTs) for web applications?

## Research plan:

This document will use DOT Framework as research methodology.

DOT (Development Oriented Triangulation) framework is a research methodology which can help ICT developer to structure the research document by coming with a problem for which we need a solution. It consist of five main method:

1. Library
2. Field
3. Lab
4. Showroom
5. Workshop

In this document, the method used will be:

* Library: exploration on what is already known is done, by using literature study, community research and best good and bad practices as strategies.
* Field: exploration in the JWT context is done, by using problem analysis and task analysis as strategies.
* Lab research: testing for JWT is done, by using security tests as strategy.

Diagram

Description automatically generated

400px-DOT-Framework. Source: <https://ictresearchmethods.nl/Category:Field>

## Subquestions:

### What are JWTs and how do they work?

A JSON Web Token (JWT) is a standard for representing claims securely between two parties. It is a compact, self-contained mechanism for transmitting information between parties as a JSON object. JWTs are often used for authentication and authorization purposes in web applications and APIs.

The JWT consists of three parts: header, payload, and signature. The header contains information about how the JWT is encoded, such as the algorithm used for signing. The payload contains the claims, which are statements about an entity (typically, the user) and additional metadata. The signature is used to verify that the sender of the JWT is who it says it is and to ensure that the message wasn't changed along the way.

JWT is broken into 3 main parts:

* Header
* Payload
* Signature

Diagram

Description automatically generated

*Figure 1, visual representation of how JWT is created, source:* [*https://morioh.com/p/79f6f8b073f8*](https://morioh.com/p/79f6f8b073f8)

When a user logs in, the server creates a JWT containing the user's information as claims in the payload. This JWT is then sent to the client, which can store it in a cookie or local storage. The client sends the JWT with every request that requires authentication, typically in the Authorization header. The server then verifies the JWT's signature and decodes the payload to obtain the user's information. This allows the server to authorize the user to perform the requested action.

1. Header

Text

Description automatically generated

*Figure 2, example of JWT Header, source:* [*https://jwt.io/introduction*](https://jwt.io/introduction)

According to Figure 2, the header is responsible for identifying which algorithm (Mentioned in JWT definition from jwt.io) will be used for generating the signature and the token type which in our case is JWT.

1. Payload

Text

Description automatically generated

*Figure 3, example of JWT Payload, source:* [*https://jwt.io/introduction*](https://jwt.io/introduction)

This part contains the claims, which are split into 3 types: reserved, public, and private.

The payload is displayed as an ordinary JSON string, having only a few fields in order to keep everything close-packed. It is used to check what type of user is sending the token and if he has any permission to do the requested action. For example, a client is trying to login, but he is trying to access the admin endpoint. Because his role is not admin, the server will send a respond saying “Forbidden”.

1. Signature

Text

Description automatically generated

*Figure 4, example of JWT Signature, source: https://jwt.io/introduction*

When a JWT is issued, the issuer signs the token by combining the token header and payload into a single string and then encrypting it with a secret key. This produces a signature that can be included in the token. When a recipient receives a JWT, they can validate the signature by re-encrypting the header and payload using the same secret key, and comparing the resulting signature to the signature included in the token. If the signatures match, the recipient can be sure that the token has not been tampered with and that the data it contains is trustworthy.

This mechanism provides a way to verify that the token has not been modified, and that the data it contains is authentic. It ensures that the recipient can trust the information in the token, and that the token has not been tampered with by unauthorized parties.

JWTs have several advantages over traditional session-based authentication methods. They are stateless, meaning that the server doesn't need to keep track of the user's session, which can improve scalability. They are also self-contained, meaning that the payload can contain all the necessary information for authorization, reducing the need for additional database queries.

### What are the most common security tasks associated with JWTs?

1. Authentication: JWTs are commonly used for user authentication. The token is generated upon successful authentication and then sent with each subsequent request to authenticate the user.
2. Authorization: JWTs can also be used for authorization. The token can contain information about the user's roles and permissions, which can be used to grant or deny access to specific resources.
3. Data integrity: JWTs can be signed using a secret key, which provides a way to verify that the data in the token has not been tampered with.
4. Confidentiality: While JWTs are not typically used for confidential data, it is possible to encrypt the token to protect sensitive information.
5. Token expiration: JWTs can have an expiration time, after which they are no longer valid. This is useful for limiting the amount of time an authenticated user can access a resource.

### What are the best practices to store JWT?

When it comes to storing the token, we have to take into account all possible attack scenarios. Therefore, the tokens are stored in two common places:

* Local/Session storage
* Cookies

### Local/Session storage

Most people store their JWTs in the browser’s storage, leaving their application open to attacks. (e.g.:XSS). In this kind of attack, the attacker uses JavaScript code which is running on the domain the web application is hosted. Even though you manage to get rid of all untrusted data, there is still a high probability of malicious code still existing in some third parties’ modules.

Another reason why this kind of storing is not secure is the fact that Web Storage does not provide secure standards for transfer. JWT needs to be sent over via HTTPS and not HTTP.

1. *Cookies*

Another way of storing is via cookies with HttpOnly (created by server). Cookies are considered to be the most secure in this case, thanks to the fact that they are sent automatically in every request.

However, they are still vulnerable to CSRF attacks by sending all cookies on cross-domain requests. In interest of avoiding these attacks, using synchronized token patterns could do the trick. Angular JS provides good implementation for synchronized tokens.

Luckily, modern browsers have been implemented with a cookie policy called “SameSite” which consists of 3 values:

• None – allowing each cookie to be sent in every request

• Lax – allowing cookies to be sent in cross-domain requests via GET methods.

All other requests will not have cookies with this policy.

• Strict – not allowing cookies to be passed through requests in any way. Considered to be the safest value.

In order to make use of this policy, we need to make sure the user is using a modern browser which can support this functionality, otherwise the HttpOnly cookie will be treated as a regular cookie.

## Testing JWT:

As mentioned in the beginning of the document, a JWT consists of header, payload, and signature. One of the most crucial vulnerabilities which might be encountered with JWT tokens would be when the application fails in validating the signature is correct. According to (Testing JSON Web Tokens, n.d.) this can happen when a developer uses a method to decode the body instead of verifying it. The verification procedure checks the signature before decoding the token. This can be tested by editing only the body of the token, without touching the header or the signature, and adding to a request to see if the application will accept it or not.

Another part of the token to which is crucial testing would the algorithm used for the creating signature, in the header. What would happen if there were no algorithm for signing? That would result into a token with no signature, opened for modifications. Most implementation now have a basic check to prevent this, by checking if a key has been provided. If there is no key, then the verification will fail for tokens which use no algorithm (aka none algorithm).

A further exploit to be taken into account and tested would the HMAC vs Public Key Confusion. What happens behind the scenes, in case the application uses JWTs with public key-based signatures (Testing JSON Web Tokens, n.d.), is that the application itself cannot check if the algorithm found in the token is correct. There are some conditions which should be followed so that this attack would be successful:

• Back-end expects the token to be signed with a public key based algorithm

• Back-end doesn’t check which algorithm the token is using for signing

• The public key used in the verification process should be known to the attacker

If we want to combat this vulnerability, we should modify the contents of the token and use the previously obtained key to sign it with HS256 algorithm.

Text, letter

Description automatically generated

*Figure 1, how is JWT signed using different algorithms in the header, source: https://owasp.org/wwwproject-web-security-testing-guide/latest/4-Web\_Application\_Security\_ Testing/06- Session\_Management\_Testing/10-Testing\_JSON\_Web\_Tokens*

As JWT is used pretty much everywhere for authentication, it is important to test it as there many ways to compromise an application. Nowadays, there lots of tools which can test JWTs by covering most exceptional cases.(John the Ripper, jwt2john, jwt-cracker, JSON Web Tokens Burp Extension etc.)

## Conclusion and recommendation

In conclusion, JWT is a quite complex and compact way for protecting your application and its users from different kind of attacks. It is important, when setting it up, to think about all possible scenarios which may be encountered, starting from which is the best place to store the secret key used for decoding, up to testing each part of the token. My recommendations are to use a secure and up to date library for handling the tokens, make sure they are stored properly by using HttpOnly cookies and not local/session Storage to be securely transmitted and ensure that a strong and unique key is used for signing the token while no sensitive data is being shown in the payload.