RESEARCH DOCUMENT

How to use JWT in a secure way for web applications?

Andreea Șindrilaru
Fontys University of Applied Sciences
S3-CB-01

Table of Contents

1. *Introduction*

The purpose of this document is to give a better understanding of JWT and how we can use it in a secure way for the backend and frontend.

Nowadays, it is crucial for every application to have well-done security, as the IT industry is developing quite fast, compared to how it was, for example, in the '90s. Over the last few decades, developers had to come up with solutions to strengthen up the security of applications to avoid hacker attacks and to protect each user's personal data, since most of our data is stored in databases.

2. *Research methodology*

This document will use DOT Framework as the research methodology.

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

- Library
- Field
- Lab
- Showroom
- Workshop

In this document, the methods 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 a strategy.

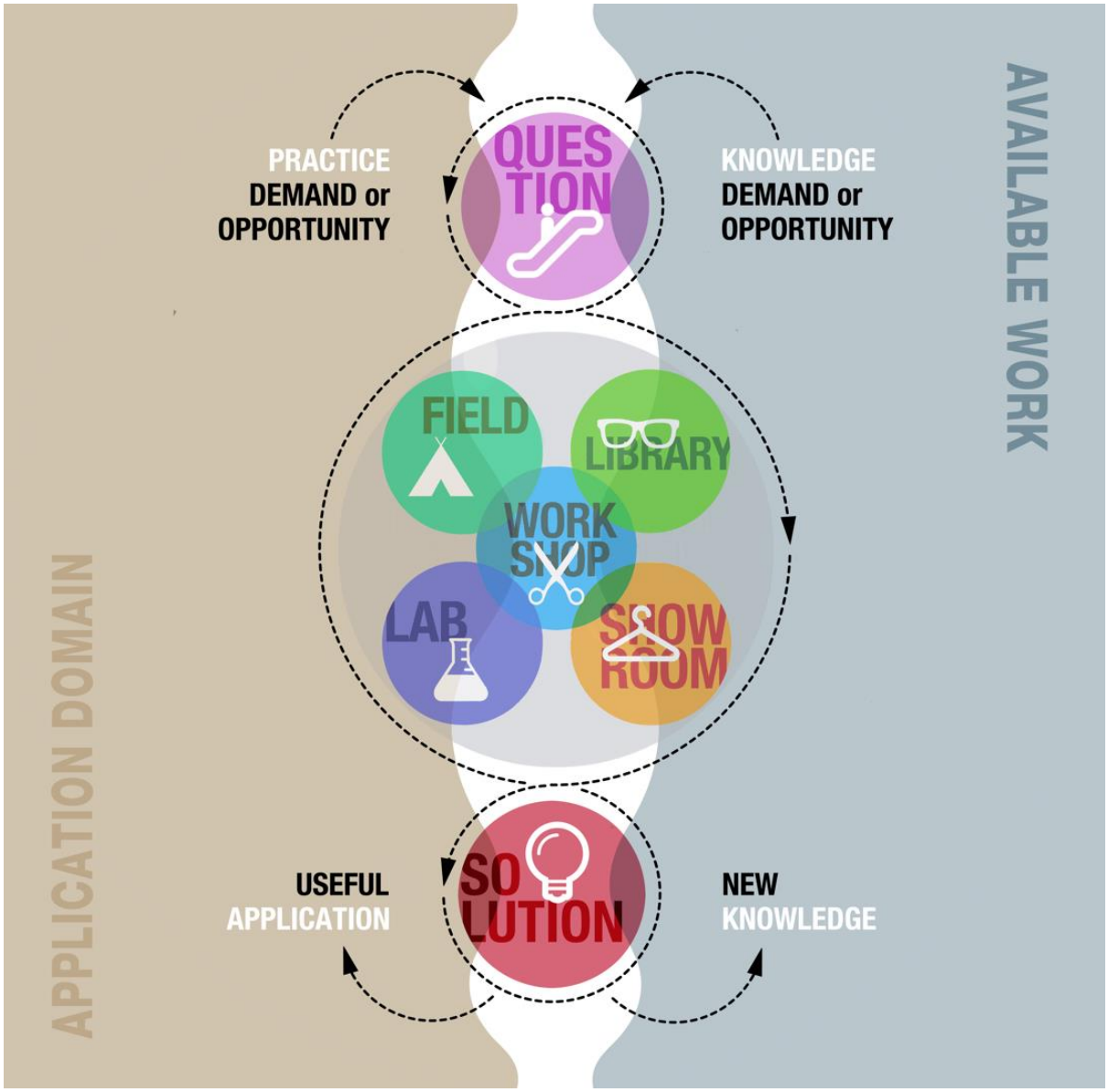

Figure 1, DOT Framework, source: https://maken.wikiwijs.nl/129804/DOT_frameworkEN #!page-4594577

3. *What is a JSON object?*

JSON (JavaScript Object Notation) is a data-interchange text format that is both human and machine-readable, used to send data between devices (e.g.: from server to client), based on the JavaScript object syntax. Thanks to its common data format, it can be used in electronic data interchange, including web applications. It is built based on two universal data structures:

- A list of names and values.
- An ordered collection of values.

Some people make some mistakes calling a JSON object an actual object which is wrong because the data is in a string format. As we can see in *Figure 1,* JSON is only showing the proprieties of the object, so you can't see any methods, constructors, and fields. If we have an object

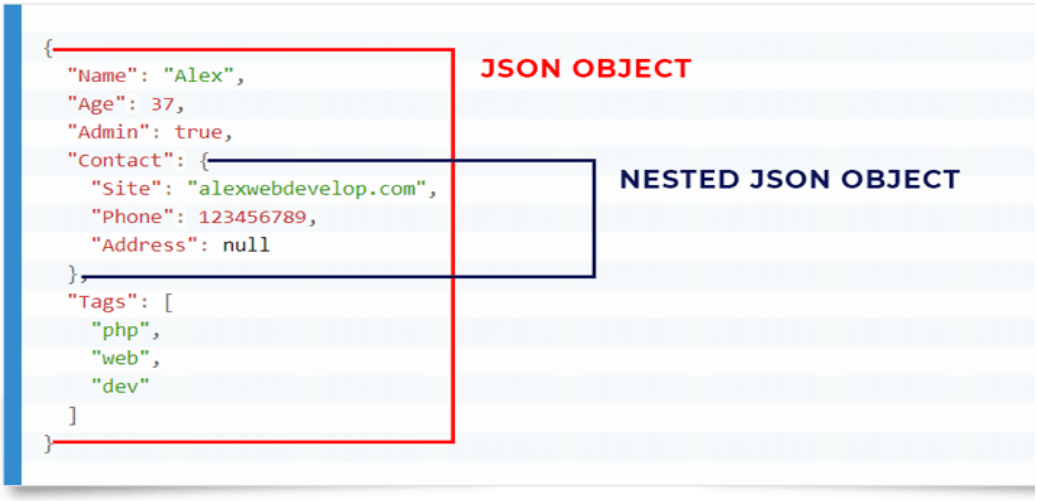

Figure 1, the actual representation of a JSON object, source: https://alexwebdevelop.com/php-json-backend/

within an object, JSON will again, display only the proprieties of it.

Due to the fact that JSON is showing all the proprieties of the object, developers need to create classes that will only deliver the information they want the client to see, in order to avoid compromising the functionality of the application. These classes are called DTOs (Data Transfer Objects).

An actual example of using DTOs can be when we get a list of products on a webshop. The actual Product objects are in JSON format, in the back end, which is transmuted in the front-end in such a way that the client will only see the important information. (e.g.: Product name, price, and available quantity)

4. *What is a JSON Web Token (JWT) and how does it work?*

Security plays a major role in our modern days, by not implementing the needed security, hackers can either steal all the details of the users by hacking their accounts with different techniques

According to (Introduction, n.d.) *JSON Web Token (JWT) is an open standard that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed. JWTs can be signed using a secret (with the **HMAC** algorithm) or a public/private key pair using **RSA** or **ECDSA**.*

In other words, JWT is a set of encoded statements about an entity (e.g.: user) and additional information (later used as claims), in JSON which can be transferred between devices with an expiration value. Claims are checked and signed digitally, by the server which issues the token so that the generated token can be used in the verification of the ownership.

JWTs are broken down into 3 parts:

- Header
- Payload
- Signature

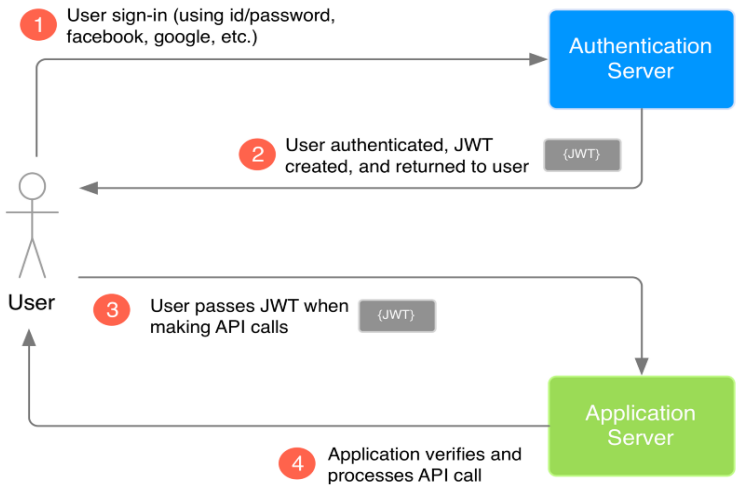

Figure 1, visual representation of how JWT is created, source: https://morioh.com/p/79f6f8b073f8

4.1 Header

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

Figure 2, an example of JWT Header, source: https://jwt.io/introduction

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

4.2 Payload

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

Figure 3, an example of JWT Payload, source:

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 log in, but he is trying to access the admin endpoint. Because his role is not admin, the server will send a response saying "Forbidden".

4.3 Signature

Before creating the signature, developers need to make sure they have encoded the header and the payload so that sensitive info is not available to anyone. We must specify which encoding algorithm we are going to use. What we can see in *Figure 4* is the signing process for a token. Each signature is unique thanks to the "secret key" which is known only by the server. Therefore, users cannot falsify tokens as they would have to know the secret key.

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret)
```

Figure 4, an example of JWT Signature, source:

5. *Why does JWT need to have an expiry value?*

Let's suppose that an attacker has found a way to steal your JWT which is very bad. The attacker will be able to use the JWT as long as possible because the API which accepts the token verifies the token independently and does not check the source from where it was sent. In short, the API has no knowledge if your token was stolen or not.

In order to avoid such events, every token must have an expiry value that should be kept short (e.g.: 15 minutes) so that any leaked token will become invalid quickly.

6. 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

6.1. local/session Storage

Most people store their JWTs in the browser's storage, leaving their applications open to attacks. (e.g.:XSS). In this kind of attack, the attacker uses JavaScript code that 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 store 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.

6.2. Cookies

Another way of storing is via cookies with HttpOnly (created by the 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 the interest of avoiding these attacks, using synchronized token patterns could do the trick. Angular JS provides a 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 cross-domain 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 that can support this functionality, otherwise, the HttpOnly cookie will be treated as a regular cookie.

7. Testing JWT

As mentioned at the beginning of the document, a JWT consists of a header, payload, and signature. One of the most crucial vulnerabilities which might be encountered with JWT tokens would be when the application fails 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 be the algorithm used for the creating signature, in the header. What would happen if there were no algorithms for signing? That would result in a token with no signature, opened for modifications. Most implementations 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 that use no algorithm (aka. none algorithm).

A further exploit to be taken into account and tested would be 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 that 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

```
// Verify a JWT signed using RS256
jwt.verify(token, publicKey);

// Verify a JWT signed using HS256
jwt.verify(token, secretKey);
```

Figure 1, how is JWT signed using different algorithms in the header, source: https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/06-Session_Management_Testing/10-Testing_JSON_Web_Tokens

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 the HS256 algorithm.

As JWT is used pretty much everywhere for authentication, it is important to test it as there are many ways to compromise an application.

Nowadays, there are lots of tools that can test JWTs by covering the most exceptional cases. (John the Ripper, jwt2john, jwt-cracker, JSON Web Tokens Burp Extension, etc.)

8. Conclusions and recommendations

In conclusion, JWT is a quite complex and compact way for protecting your application and its users from different kinds 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, ensure that a strong and unique key is used for signing the token while no sensitive data is being shown in the payload

Bibliography

auth0.com. (n.d.). *JWT.IO - JSON Web Tokens Introduction*. JSON Web Tokens - Jwt.Io. https://jwt.io/introduction

Cardenes, E. A. (2020, September 26). *Managing JWT token expiration - eduardo aparicio cardenes*. Medium. https://medium.com/@byeduardoac/managing-jwt-token-expiration-bfb2bd6ea584

Copes, F. (2021, July 2). *JWT authentication: Best practices and when to use it*. LogRocket Blog. https://blog.logrocket.com/jwt-authentication-best-practices/

Koniaris, G. (2020, July 11). *How to securely store JWT tokens.* DEV Community. https://dev.to/gkoniaris/how-to-securely-store-jwt-tokens-51cf

Methods - ICT research methods. (n.d.). DOTFramework. https://ictresearchmethods.nl/Methods

The Ultimate Guide to handling JWTs on frontend clients (GraphQL). (n.d.). Hasura GraphQL Engine Blog. https://hasura.io/blog/best-practices-of-using-jwt-with-graphql/#login

Wikipedia contributors. (2021, November 2). *JSON Web Token*. Wikipedia. https://en.wikipedia.org/wiki/JSON_Web_Token

WSTG - Latest | OWASP. (n.d.). OWASP. https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/06-Session_Management_Testing/10-Testing_JSON_Web_Tokens

Freeman, J. (2019, October 25). *What is JSON? A better format for data exchange*. InfoWorld. https://www.infoworld.com/article/3222851/what-is-json-a-better-format-for-data-exchange.html