Web Security

Using JWT Tokens for secure authentication

Ivan Ovcharov

Software Engineering, Fontys UAS

Student number: 4090993

This page is left blank on purpose

Abstract

This research paper uses the DOT framework under the LAB testing category. After a brief explanation of the structure and context of the topic, the paper will introduce different testing methods of JWT's uses and implementation.

Nowadays, a lot of companies rely on using browser cookies to temporarily store information about the application user. Despite it being very convenient, it could be a potential weak point for malicious software to exploit and gather sensitive data. To help prevent this, JWT (Java Web Tokens) come into place. They serve as a way to store payload of a user locally safely by creating a representational string using an algorithm.

Web Security: Using JWT tokens securely in back/front-end

1. Introduction

A JWT token, usually found in space constrained environments such as HTTP authorization headers and query UI parameters, is a string of claims that can be encrypted using a “secret” stamp. The tokens are stored locally on a user’s machine and are used to ensure authorization by decryption of the string. All java web tokens are always presented by JWE/JWS Compact serialization.

2. Terminology

Claim string/claim

Information about a certain object. It is represented as a pair of name and value and consists of a Claim Name and Claim Value.

Claim name

A string that represents the name section of a claim

Claim value

Any JSON value that represents the value section of a claim.

3. JSON Web Token overview & examples

A JSON Web Token usually consists of three dot separated parts that include:

1. Header
2. Payload
3. Signature

A usual visual representation of a JWT is usually found in the form:

aaaaaa.bbbbbbb.cccccc

3.1 Header breakdown

A JSON Web Token header can be broken down into two parts: a signing algorithm (HMAC, SHA256 or RSA) and the type of the token itself.

An example of a header would *typically* look as follows:

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

3.2 Payload breakdown

A payload contains all of the claims about a specific user. They represent different data and statements and have three types of forms: *public*, *private* and *registered* claims.

- **Public claims:** Claims of such sort can be defined upon necessity by those using JWTs. To avoid redundancy and duplication issues, they *should* be defined in the IANA JSON Web Token Registry or as a URI, containing a collision resistant namespace.

- **Private claims:** Such claims are instantiated by parties that agree upon using them and do not represent neither *public* nor *registered* claims

- **Registered claims:** This is usually considered the standard set of predefined claims that are *recommended*, as they provide useful & interoperable claims. A list of them is:
  - **exp** (expiration time)
  - **sub** (subject)
  - **aud** (audience)
  - **iss** (issuer)

  & others.

An example of a proper payload *could* look as follows:

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

(figure. 1)

The payload is then converted to a string containing a set of unique characters. Such a process goes by the term **Base64Url encoding**. Such a string is readable by anyone, so avoiding the inclusion of secret information is necessary to avoid security issues.

3.3 Signature breakdown

A signature is done after an encoded header, encoded payload, signing algorithm & a secret are instantiated. It is then used to "sign" the set of claims and encrypt it.

An example of a signature using the HMAC SHA256 algorithm is represented as follows:

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

(figure. 2)

A signature is a key component to ensure security as it makes sure the message is not modified along the way and, in case of signed tokens using a private key, is used to verify the identity of a JWT sender and approves that it is the correct user.

After combining all three elements, we get a complete & structured JWT that has a secret signature, a body containing the encrypted data and a header that is used as interpretation.

4.Testing JSON Tokens and their security

- *LAB Dot framework method*

4.1 What is the DOT Framework lab methodology?

”Measuring is knowing”

Lab research is done to test your ideas with the users of your product. It is used to learn if things work out the way you intended them. Creating different unit tests allows us to easily see the application of a given subject.

4.2 Introducing tests to JSON Web Tokens

JWT Tokens consist of three components and are used to authorize a given user and his persona. Naturally, a JWT token can have weak points in their provided security. This is where testing comes into place.

Testing the payload of a JWT

A payload contains the data that is being encrypted by a given token. In order to verify its validity and to ensure it is bound to the correct token, a unit test can be introduced.
A way to approach this would be to manually integrate a person's information and then compare it to the given payload. Even if there is a single difference, this could possibly signify a leak in security and may even lead to broken data

.

In order to follow the LAB methodology, the development team must issue a testing version of the application where different users fill in dummy credentials to be then encrypted into a payload. If the data is recovered correctly and the users are successfully authorized, we can then come to the conclusion that the payload works correctly.

Testing the signature of a JSON Web Token

The signature (or “secret”) of a token is used to “sign” the payload component of a JWT. It is usually the most exploitable part, as it could be easily done in an incorrect manner. It is usually built out of a string of characters that represent a “secret” sequence.

In order to test, a simple checking algorithm may be deployed that compares common words used in such signatures. Furthermore, a software engineering team may represent themselves as users and try different instances of such a signature.

Testing the header of a JWT

A header breaks down in two parts - a signing algorithm and the type of token itself. The header is also the most important part of a JWT token as it determines how a token is constructed and the technology behind generating the other two components. To test if a header is secure enough, different signing algorithms have to be deployed in the respective situations.

A lot of websites encode certain REST API’s with such headers, instead of showing an id/naming convention of the page. This could be potentially dangerous and exploited as an experienced cyber software engineer could determine what signing algorithm is used by decrypting the header.

Putting a JWT header to the test can be proven quite difficult as it is a randomly generated set of characters and is claimed to be ninety-nine percent sure to avoid redundancy (repeating headers).

Having said that, a software engineering team could issue a testing version of the software where headers are present visually and see if it could be exploited by the website's users. There are some famous cases where users who are not professional could change some characters of such headers and end up in restricted parts of a web application.

5. Deploying JWT unit tests

Another key strategy of verifying the integrity and security of a JSON web token is by deploying various unit tests. One of the most proven and popular ways of doing so is by using the following extensions:

Chai & Mocha

Chai & Mocha are Java libraries that introduce functionality such as *.should(), could()* & *would()*. With these technologies, unit tests become more flexible as they allow easy verification of certain parts of a JSON web token.

Chai is a Behavior Driven Development / Test Driven Development assertion library for Node and the browser and can be used with any testing framework (like Mocha). These are (assertions) simple statements that are always expected to evaluate to true, and if not, they throw an error

Mocha is a framework that does not have a built-in assertion library like Chai, but its developed to run test cases and reports any errors encountered during the process.

6. Conclusion

JWT tokens are a modern and secure way of encoding pages, user information and user data in a fast and easy to deploy manner. They provide a variety of different algorithms that can be used in the respective situations.

Furthermore, it is of key importance that such functionality should be tested by deploying a dummy version to the public or creating unit tests with the necessary extensions & libraries.

JSON Web Tokens also provide functionality of refreshing a certain token, forcing a token verification if a user is suspicious and managing its expiry date, allowing for a brand new token to be generated thus providing further security.

References

Jones, M. - Bradley, J - Sakimura, N. & Microsoft (2015). *Official IETF Standards documentation.* (ISSN: 2070-1721). https://datatracker.ietf.org/doc/html/rfc7519#section-1

CMD methods pack. CMD Methods Pack - find a combination of research methods that suit your needs. (n.d.). Retrieved November 3, 2021, from https://www.cmdmethods.nl/more-info.

Auth0 (2021). *Official JWT.io documentation introduction.* https://jwt.io/introduction

Oluyege, P. (2020, November 3). *Unit testing JWT secured node and express restful API with Chai and mocha*. Buddy. Retrieved November 3, 2021, from https://buddy.works/tutorials/unit-testing-jwt-secured-node-and-express-restful-api-with-chai-and-mocha.