Get Started with JSON Web Tokens

All you wanted to know about JSON Web Tokens but were afraid to ask.

WHAT IS JSON WEB TOKEN?

JSON Web Token (JWT) is an open standard (RFC 7519) 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 **HMAC** algorithm) or a public/private key pair using **RSA**.

Let's explain some concepts of this definition further.

- **Compact**: Because of its size, it can be sent through an URL, POST parameter, or inside an HTTP header. Additionally, due to its size its transmission is fast.
- **Self-contained**: The payload contains all the required information about the user, to avoid querying the database more than once.

For full details on JSON Web Tokens, check the JWT Handbook.

WHEN SHOULD YOU USE JSON WEB TOKENS?

These are some scenarios where JSON Web Tokens are useful:

- **Authentication**: This is the typical scenario for using JWT, once the user is logged in, each subsequent request will include the JWT, allowing the user to access routes, services, and resources that are permitted with that token. Single Sign On is a feature that widely uses JWT nowadays, because of its small overhead and its ability to be easily used among systems of different domains.
- **Information Exchange**: JWTs are a good way of securely transmitting information between parties, because as they can be signed, for example using a public/private key pair, you can be sure that the sender is who they say they are. Additionally, as

the signature is calculated using the header and the payload, you can also verify that the content hasn't changed.

WHICH IS THE JSON WEB TOKEN STRUCTURE?

JWTs consist of three parts separated by dots (`.`), which are:

- Header
- Payload
- Signature

Therefore, a JWT typically looks like the following.

```
xxxxx.yyyyy.zzzzz
```

Let's break down the different parts.

Header

The header ***typically*** consists of two parts: the type of the token, which is JWT, and the hashing algorithm such as HMAC SHA256 or RSA.

For example:

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

Then, this JSON is **Base64Url** encoded to form the first part of the JWT.

Payload

The second part of the token is the payload, which contains the claims. Claims are statements about an entity (typically, the user) and additional metadata. There are three types of claims: ***reserved***, ***public***, and ***private*** claims.

- **Reserved claims**: These are a set of predefined claims, which are not mandatory but recommended, thought to provide a set of useful, interoperable claims. Some of them are: **iss** (issuer), **exp** (expiration time), **sub** (subject), **aud** (audience), among others.

Who, when, about what , and who can see and so on.

> *Notice that the claim names are only three characters long as JWT is meant to be compact.*

- **Public claims**: These can be defined at will by those using JWTs. But to avoid collisions they should be defined in the IANA JSON Web Token Registry or be defined as a URI that contains a collision resistant namespace.
- **Private claims**: These are the custom claims created to share information between parties that agree on using them.

An example of payload could be:

```
{
  "sub": "1234567890",
  "name": "John Doe",
```

```
  "admin": true
}
```

The payload is then **Base64Url** encoded to form the second part of the JWT.

Signature

Signature depends on the first two parts, header and payload.

To create the signature part you have to take the encoded header, the encoded payload, a secret, the algorithm specified in the header, and sign that.

For example if you want to use the HMAC SHA256 algorithm, the signature will be created in the following way.

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

The signature is used to verify that the sender of the JWT is who it says it is and to ensure that the message was't changed in the way.

Putting all together

The output is three Base64 strings separated by dots that can be easily passed in HTML and HTTP environments, while being more compact compared to XML-based standards such as SAML.

The following shows a JWT that has the previous header and payload encoded and it is signed with a secret.

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4
gRG9lIiwiaXNTb2NpYWwiOnRydWV9.
4pcPyMD09olPSyXnrXCjTwXyr4BsezdI1AVTmud2fU4
```

Header

Payload

signature

You can browse to jwt.io where you can play with a JWT and put these concepts in practice. jwt.io allows you to decode, verify and generate JWT.

HOW JSON WEB TOKENS WORK?

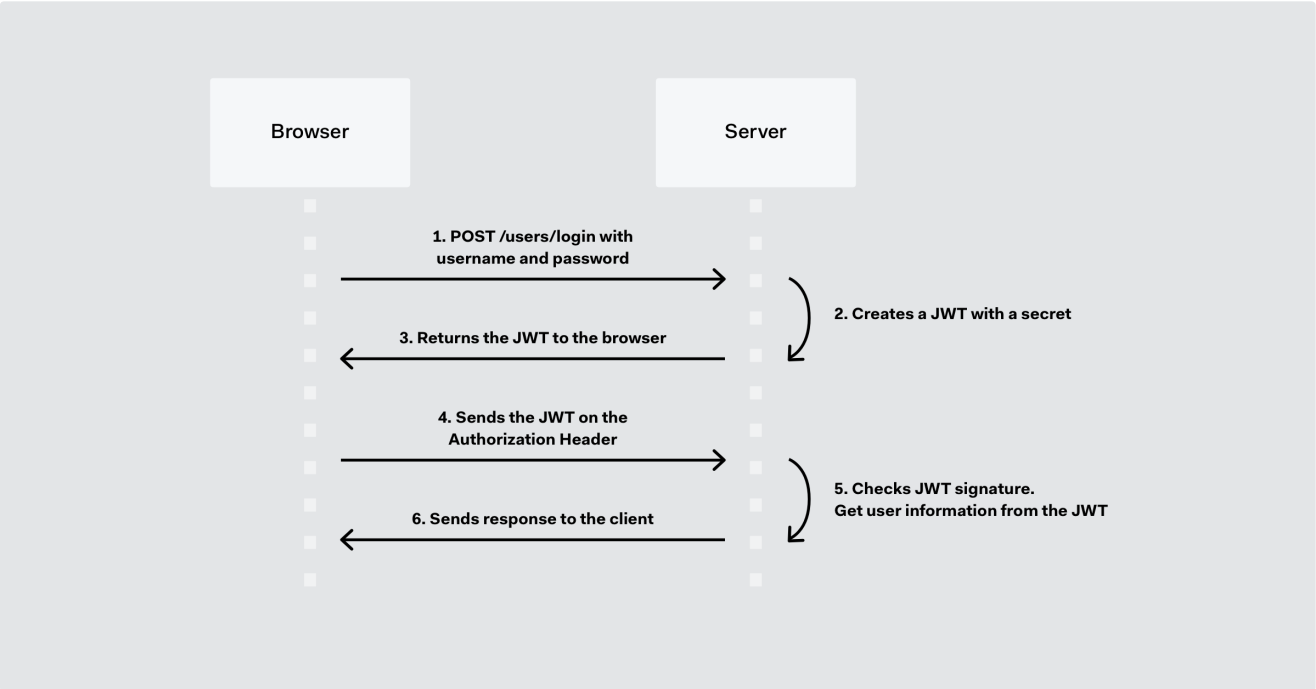
In authentication, when the user successfully logs in using his credentials, a JSON Web Token will be returned and must be saved locally (typically in local storage, but cookies can be also used), instead of the traditional approach of creating a session in the server and returning a cookie.

Whenever the user wants to access a protected route, it should send the JWT, typically in the **Authorization** header using the **Bearer** schema. Therefore the content of the header should look like the following.

```
Authorization: Bearer <token>
```

This is a stateless authentication mechanism as the user state is never saved in the server memory. The server's protected routes will check for a valid JWT in the Authorization header, and if there is, the user will be allowed. As JWTs are self-contained, all the necessary information is there, reducing the need of going back and forward to the database.

This allows to fully rely on data APIs that are stateless and even make requests to downstream services. It doesn't matter which domains are serving your APIs, as Cross-Origin Resource Sharing (CORS) won't be an issue as it doesn't use cookies.



WHY SHOULD YOU USE JSON WEB TOKENS?

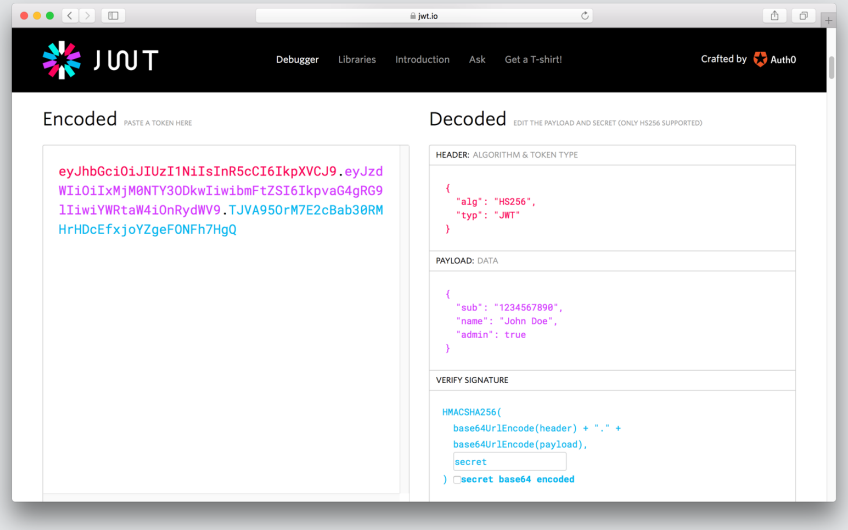
Let's talk about the benefits of **JSON Web Tokens (JWT)** comparing it to **Simple Web Tokens (SWT)** and **Security Assertion Markup Language Tokens (SAML)**.

As JSON is less verbose than XML, when it is encoded its size is also smaller; making JWT more compact than SAML. This makes JWT a good choice to be passed in HTML and HTTP environments.

Security-wise, SWT can only be symmetric signed by a shared secret using the HMAC algorithm. While JWT and SAML tokens can also use a public/private key pair in the form of a X.509 certificate to sign them. However, signing XML with XML Digital Signature without introducing obscure security holes is very difficult compared to the simplicity of signing JSON.

JSON parsers are common in most programming languages, because they map directly to objects, conversely XML doesn't have a natural document-to-object mapping. This makes it easier to work with JWT than SAML assertions.

Regarding usage, JWT is used at an Internet scale. This highlights the ease of client side processing of JWTs on multiple platforms, especially, mobile.



HOW WE USE JSON WEB TOKENS IN AUTH0?

In Auth0, we issue JWTs as a result of the authentication process. When the user logs in using Auth0, a JWT is created, signed, and sent to the user. Auth0 supports signing JWT with both HMAC and RSA algorithms. This token will be then used to authenticate and authorize with APIs which will grant access to their protected routes and resources.

We also use JWTs to perform authentication and authorization in Auth0's API v2, replacing the traditional usage of regular opaque API keys. Regarding authorization, JSON Web Tokens allow granular security, that is the ability to specify a particular set of permissions in the token, which improves debuggability.

Contact Us

Name *

Your Name

Email *

Your Email

Company *

Company

Role *

Software Developer

Message *

Message

SEND

PRODUCT

Pricing

Why Auth0

How It Works

COMPANY

About Us

Blog

Jobs

Press

SECURITY

Availability & Trust

Security

White Hat

LEARN

Help & Support

Documentation

Open Source

EXTEND