JSON Web Tokens

JSON Web Tokens (JWT), pronounced “jot”, are a standard since the information they carry is transmitted via JSON. We can read more about the draft73 here, but that explanation isn’t the most pretty to look at. JSON Web Tokens work across different programming languages: JWTs work in .NET, Python, Node.js, Java, PHP, Ruby, Go, JavaScript, and Haskell. So you can see that these can be used in many different scenarios.

JWTs are self-contained: They will carry all the information necessary within itself. This means that a JWT will be able to transmit basic information about itself, a payload (usually user information) ko and a signature.yuyu

JWTs can be passed around easily: Since JWTs are self-contained, they are perfectly used inside an HTTP header when authenticating an API. You can also pass it through the URL.

What does a JWT look like?

A JWT is easy to identify. It is three strings separated by .

For example:

Let’s break down the 3 parts and see what each contains.

Breaking Down a JSON Web Token

Since there are 3 parts separated by a ., each section is created differently. We have the 3parts which are:

• header

• payload

• signature

JSON Web Token Overview

Header

The header carries 2 parts:

• declaring the type, which is JWT

• the hashing algorithm to use (HMAC SHA256 in this case)

Here’s an example:

```
1 { 2 "typ": "JWT", 3 "alg": "HS256" 4 }
```

Now once this is base64encode, we have the first part of our JSON web token!

```
1 eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
```

Payload

The payload will carry the bulk of our JWT, also called the JWT Claims74. This is where we will put the information that we want to transmit as well as other information about our token.

There are multiple claims that we can provide. This includes registered claim names, public claim names, and private claim names.

Registered Claims

Claims that are not mandatory whose names are reserved for us. These include:

• iss: The issuer of the token

• sub: The subject of the token

• aud: The audience of the token

• exp: This will probably be the registered claim most often used. This will define the expiration in NumericDate value. The expiration MUST be before the current date/time.

• nbf: Defines the time before which the JWT MUST NOT be accepted for processing

• iat: The time the JWT was issued. Can be used to determine the age of the JWT

• jti: Unique identifier for the JWT. Can be used to prevent the JWT from being replayed. This is helpful for a one time use token.

Public Claims

These are the claims that we create ourselves like user name, information, and other important information.

Private Claims

A producer and consumer may agree to use claim names that are private. These are subject to collision, so use them with caution.

Example Payload

Our example payload has two registered claims (iss, and exp) and two public claims (name, admin).

```
1 { 2 "iss": "scotch.io", 3 "exp": 1300819380, 4 "name": "Chris Sevilleja", 5 "admin": true 6 }
```

This will encode to:

```
1 eyJpc3MiOiJzY290Y2guaW8iLCJleHAiOjEzMDA4MTkzODAsIm5hbWUiOiJDaHJpcyBTZXZpbGxlamEi\ 2 LCJhZG1pbiI6dHJ1ZX0
```

That will be the second part of our JSON Web Token.

Signature

The third and final part of our JSON Web Token is going to be the signature. This signature is made up of a hash of the following components:

• the header

• the payload

• secret

• The token is sent on every request so there are no CSRF attacks. There is no session based information to manipulate since, well, we don't have a session!

This is how we get the third part of the JWT:

```
1 var encodedString = base64UrlEncode(header) + "." + base64UrlEncode(payload);
2 3 HMACSHA256(encodedString, 'secret');
```

The secret is the signature held by the server. This is the way that our server will be able to verify existing tokens and sign new ones. This is the only thing that our server holds in order to verify the user.

This gives us the final part of our JWT.

```
1 03f329983b86f7d9a9f5fef85305880101d5e302afafa20154d094b229f75773
```

Now we have our full JSON Web Token:

```
1
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzY290Y2guaW8iLCJleHAiOjEzMDA4MT
k\                                                                          2
zODAsIm5hbWUiOiJDaHJpcyBTZXZpbGxlamEiLCJhZG1pbiI6dHJ1ZX0.03f329983b86f7d9a9f5fe
f\ 3 85305880101d5e302afafa20154d094b229f75773
```

Auth075 has created a great site76 to go through and test out how JWTs are made. You can see as you change the content on the fly, you are able to see the JWT get updated immediately. Auth0 provides great tools and they also maintain the jsonwebtoken77 Node package to handle creating and verifying JWTs in Node.

With an understanding of both token based authentication and the mechanism to handle authentication, JSON Web Tokens, let's see how we can implement both of these into our Node API that we just built.