Try my new website - Ambient-Geek.github.io Please click on Ads to support the site

Geekly Articles each Day

» Five easy steps for understanding JSON Web Tokens (JWT)

Continue listening

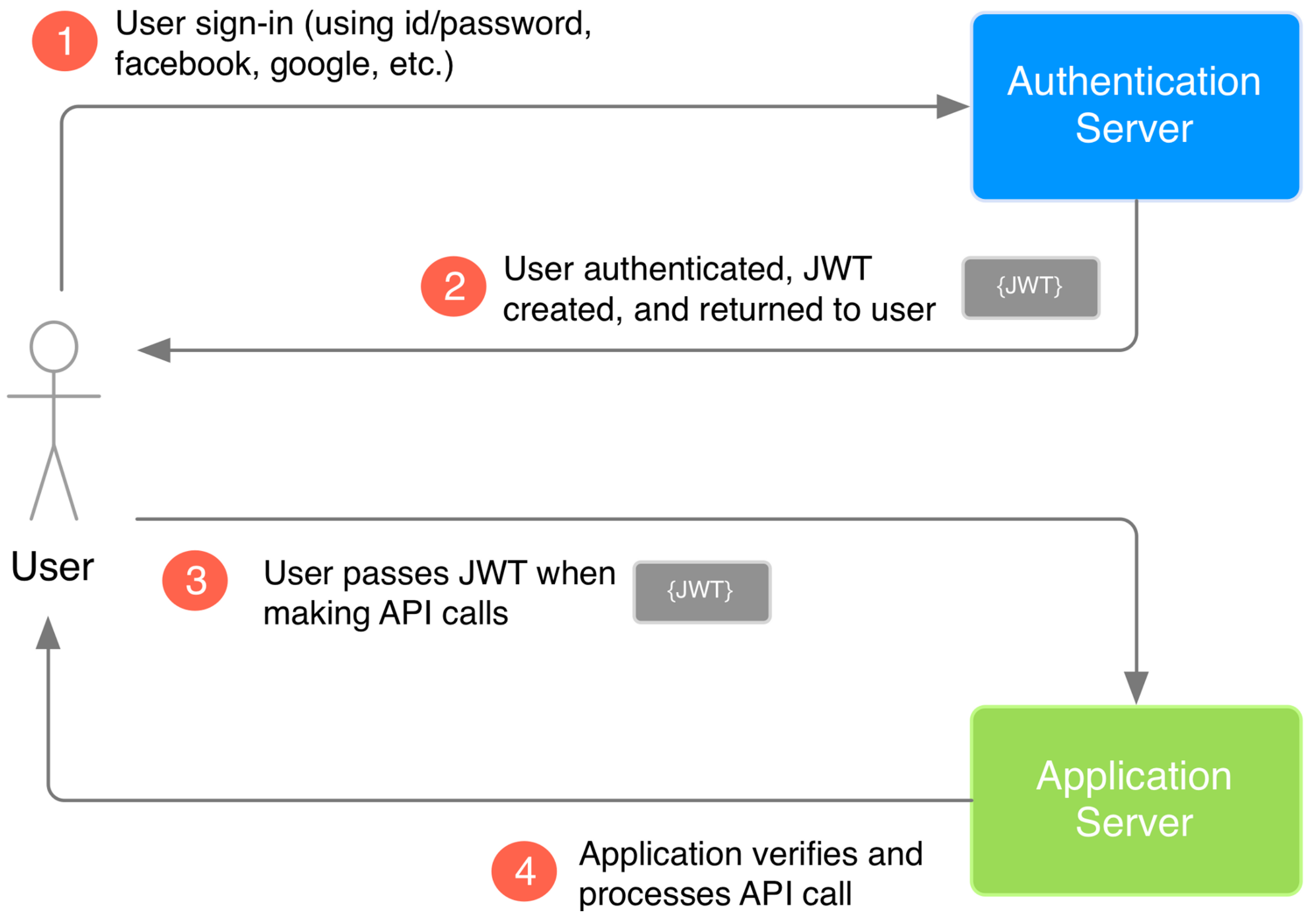
I present to you my rather loose translation of Article 5 Easy Steps to Understanding JSON Web Tokens (JWT) . This article will discuss what *JSON Web Tokens (JWT) are* and what they eat. That is, what role do they play in user authentication and ensuring the security of application data.

First, consider the formal definition.

> `JSON Web Token (JWT)` is a *JSON* object that is defined in the open standard RFC 7519 . It is considered one of the safe ways to transfer information between two participants. To create it, it is necessary to define a header with general information on the token, payload data, such as user id, its role, etc. and signatures (signature).
> By the way, *JWT* is correctly pronounced as `/dʒɒt/`

Suppose we want to register on the site. In our case, there are three parties — user `user` , `application server` and `authentication server` . The authentication server will provide the user with a token with which he can later interact with the application.

»


1
User sign-in (using id/password, facebook, google, etc.)
Authentication Server
2
User authenticated, JWT created, and returned to user
{JWT}
User
3
User passes JWT when making API calls
{JWT}
Application Server
4
Application verifies and processes API call

Continue listening

The application uses *JWT* to verify user authentication as follows:

1. First, the user enters the authentication server using an authentication key (it can be a *login / password* pair, or a *Facebook* key, or a *Google* key, or a key from another account).
2. The authentication server then creates the *JWT* and sends it to the user.

4. When a user makes an API request, the application can check on the *JWT* submitted with the request whether the user is who he claims to be. In this scheme, the application server is configured to check whether the incoming *JWT is* exactly what was created by the authentication server (the verification process will be explained later in more detail).

JWT structure

»

JWT consists of three parts: header `header` , `payload` and signature `signature` . Let's go through each of them.

Step 1. Create a HEADER

The *JWT header* contains information on how the *JWT* signature should be calculated. A header is also a *JSON* object that looks like this:

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

The `typ` field does not tell us anything new, just that it is *JSON Web Token* . More interesting here is the `alg` field, which defines the hashing algorithm. It will be used when creating the signature. `HS256` is nothing more than a `HMAC-SHA256` ; only one secret key is needed to calculate it (for more details, see step 3). Another `RS256` algorithm can also be used - unlike the previous one, it is asymmetric and creates two keys: public and private. Using a private key, a signature is created, and using a public key, only the authenticity of the signature is verified, so we do not need to worry about its security.

Step 2. Create PAYLOAD

Payload is **payload** that is stored inside *JWT* . This data is also called *JWT-claims* . In the example that we are considering, the authentication server creates a *JWT* with information about the user *id* - **userId** .

```
payload = { "userId": "b08f86af-35da-48f2-8fab-cef3904660bd" }
```

We put only one claim in the *payload* . You can put as many *applications* as you like. There is a list of standard *applications* for *JWT* payload - here are some of them:

- *iss* (issuer) - identifies the application from which the token is sent.
- *sub* (subject) - defines the topic of the token.
- *exp* (expiration time) - token lifetime.

These fields can be useful when creating *JWT* , but they are not required. If you want to know the entire list of available fields for *JWT* , you can have a look at the Wiki . But it is worth remembering that the more information is transmitted, the more *JWT* itself will end up as a result. Usually there are no problems with this, but still it can negatively affect performance and cause delays in the interaction with the server.

Step 3. Create SIGNATURE

The signature is computed using the following pseudo-code:

```
const SECRET_KEY = 'cAtwa1kkEy' const unsignedToken = base64urlEncode(header) + '.' + base64u
```

The **base64url** algorithm encodes the header and payload created in steps *1* and *2* . The algorithm connects the encoded strings through a dot. Then, the resulting string is hashed by the algorithm specified in the header based on our secret key.

```
// header eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9 // payloadey J1c2VySWQiOiJiMDhmODZhZi0zNWRhLTQ
```

Step 4. Now combine all three JWT components together.

Now that we have all three components, we can create our *JWT* . It's quite simple, we connect all the received elements into a string through a dot.

```
const token = encodeBase64Url(header) + '.' + encodeBase64Url(payload) + '.' + encodeBase64Ur
```

You can try creating your own *JWT* at [jwt.io.](jwt.io.)
Let's return to our example. Now the authentication server can send to the user *JWT* .

How does JWT protect our data?

It is very important to understand that using *JWT* **does NOT** hide or mask data automatically. The reason why *JWT is* used is to verify that the data sent was actually sent to an authorized source. As demonstrated above, the data inside *JWT is* encoded and signed, note that this is not the same thing that is encrypted. The purpose of data coding is structure transformation. Signed data allows the recipient to verify the data source authentication. Thus, encoding and signing data does not protect them. On the other hand, the main purpose of encryption is to protect data from unauthorized access. For a more detailed explanation of the difference between encryption and encryption, as well as how hashing works, see this article . Since *JWT is* only encoded and signed, and since *JWT is* not encrypted, *JWT* does not guarantee any security for sensitive data.

Step 5. Verify JWT

In our simple example of 3 participants, we use *JWT* , which is signed using the `HS256` algorithm, and only the authentication server and the application server know the secret key. The application server receives the secret key from the authentication server during the installation of authentication processes. Since the application knows the secret key when the user makes an API request with a token attached to it, the application can perform the same signing algorithm for *JWT* as in step *3* . The application can then verify this signature by comparing it with its own calculated hashing. If the signatures match, then *JWT is* valid, i.e.

atures do not mat perhaps this is a sign of a potential attack. Thus, while testing JWT , the application adds a *layer of trust* between itself and the user.

Finally

We walked through what *JWT is* , how it is created and how it is validated, how it can be used to establish trust between the user and the application. But this is only a piece of the puzzle of a large topic of authorization and ensuring the protection of your application. We considered only the basics, but without them it is impossible to move on.

useful links

1. 5 Easy Steps to Understanding JSON Web Tokens (JWT)
2. Securing React Redux Apps With JWT Tokens
3. Why do I need Refresh Token if there is Access Token?

Source: https://habr.com/ru/post/340146/

More articles:

- Planning practices. Assessment of tasks
- Weekend Reading: 22 independent development, security, testing and game development blogs
- The digest of interesting materials for the mobile developer # 225 (October 9-October 15)
- WiFi roaming - 802.11i / r / k / v / OKC, what we really need and how to recognize it
- Waf eyes hackers
- Digital events for the week
- The digest of fresh materials from the world of the frontend for the last week №284 (October 9 - 15, 2017)
- Change sex and race on a selfie using neural networks

Try my new website - [Ambient-Geek.github.io]() [ction, optimization]() Please click on Ads to support the site

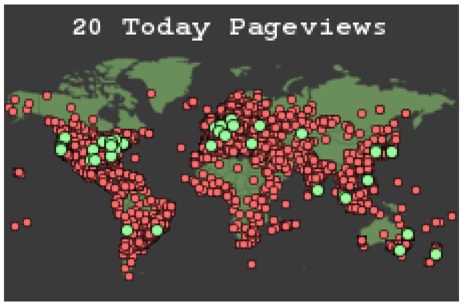
- [(not) get depressed with Angular]()
- [VoIP server for a small company (FreePBX 14, Asterisk 15, Ubuntu 16.04) part 2]()

[All Articles]()

This site may contain Advertising and Affiliate Program links. As an Amazon Associate I earn from qualifying purchases.

»

Weekly-Geekly | 2019



Continue listening