JWT authentication: When and how to use it

Learn when JWT is best used, when it's best to use something else, and how to prevent the most basic security issues.

October 11, 2018 · 3 min read

JWT (JSON Web Token) is a very popular technology not without its share of controversy.

Some people say you should never use it, while others say it's amazing.

What's the truth? Should you use it or not? That's why we're here.

Brief introduction to JWT

A JWT technically is a mechanism to verify the owner of some JSON data. It's an encoded string, which is URL safe, that can contain an unlimited amount of data (unlike a cookie), and it's cryptographically signed.

When a server receives a JWT, it can guarantee the data it contains can be trusted because it's signed by the source. No middleman can modify a JWT once it's sent.

It's important to note that a JWT guarantees data ownership but not encryption; the JSON data you store into a JWT can be seen by anyone that intercepts the token, as it's just serialized, not encrypted.

For this reason, it's highly recommended to use HTTPS with JWTs (and HTTPS in general, by the way).

We're not going to cover how JWTs are generated in detail. There are various guides about this, and I suggest this one: "What the heck is JWT anyway?"

TL;DR: What are they good for?

JWT is a great technology for API authentication and server-to-server authorization.

It's not a good choice for sessions.

Read on to understand the nitty gritty details about those affirmations.

Using JWT for API authentication

A very common use of a JWT token, and the one you should probably *only* use JWT for, is as an **API authentication mechanism**.

Just to give you an idea, it's so popular and widely used that Google uses it to let you authenticate to their APIs.

The idea is simple: you get a secret token from the service when you set up the API.

On the client side, you create the token (there are many libraries for this), using the secret token to sign it.

You pass it as part of the API request, and the server will know it's that specific client because the request is signed with its unique identifier:

How to expire a single token

How can you invalidate a single token? A no-effort solution is to change the server secret key, which invalidates all tokens. Not really nice for users that should not have their token expired for no reason.

One way to do it is to add a property to your user object in the server database, to reference the datetime the token was created at.

A token automatically stores this value in the `iat` property.

Every time you check the token, you can compare its `iat` value with the server-side user property.

To invalidate the token, just update the server-side value, and if `iat` is older than this, you can reject the token.

Another way to achieve this is by having a blacklist in your database cached in memory (or, even better, a whitelist).

Store JWTs securely

A JWT needs to be stored in a safe place inside the user's browser.

If you store it inside localStorage, it's accessible by any script inside your page (which is as bad as it sounds, as an XSS attack can let an external attacker get access to the token).

Don't store it in local storage (or session storage). If any of the third-party scripts you include in your page gets compromised, it can access all your users' tokens.

The JWT needs to be stored inside an **httpOnly cookie**, a special kind of cookie that's only sent in HTTP requests to the server, and it's never accessible (both for reading or writing) from JavaScript running in the browser.

Using JWT to securely exchange information between two servers

Since JWT are signed, the receiver can be sure the client is really who it thinks it is.

Using JWT for SPA authentication

JWTs can be used as an authentication mechanism that does not require a database. The server can avoid using a database because the data store in the JWT sent to the client is *safe*.

Using JWT to authorize operations across servers

Say you have one server where you are logged in, SERVER1, which redirects you to another server SERVER2 to perform some kind of operation.

SERVER1 can issue you a JWT that authorizes you to SERVER2. Those two servers don't need to share a session or anything to authenticate you. The token is perfect for this use case.

When not to use JWTs: Session tokens for regular web apps

You might think that using JWTs for session tokens might appear like a good idea at first:

- You can store any kind of user details on the client
- The server can trust the client because the JWT is signed, and there is no need to call the database to retrieve the information you already stored in the JWT
- You don't need to coordinate sessions in a centralized database when you get to the eventual problem of horizontal scaling

Ultimately, if you already have a database for your application, just use a sessions table and use regular sessions as provided by the server-side framework of choice.

Why? There is a cost involved in using JWTs: they are sent for every request to the server, and it's always a high cost compared to server-side sessions.

Also, while the security risks are minimized sending JWTs using HTTPS, there is always the possibility that it's intercepted and the data deciphered, exposing your user's data.

I recommend you read these two articles on the subject if you want to get into more details about JWTs and sessions:

- "Why JWTs Suck as Session Tokens"
- "Stop using JWT for sessions"
- "Stop using JWT for sessions, part 2: Why your solution doesn't work"

How to choose the perfect JWT library

Head to jwt.io and look at the Libraries list. The site contains a list of the most popular libraries that implement JWT.

Select your language of choice and pick the library that you prefer, which ideally has the highest number of green checks.

Conclusion

JWT is a very popular standard you can use to trust requests by using signatures, and exchange information between parties. Make sure you know when it's best used, when it's best to use something else, and how to prevent the most basic security issues.

LogRocket: Full visibility into your web apps

LogRocket is a frontend application monitoring solution that lets you replay problems as if they happened in your own browser. Instead of guessing why errors happen, or asking users for screenshots and log dumps, LogRocket lets you replay the session to quickly understand what went wrong. It works perfectly with any app, regardless of framework, and has plugins to log additional context from Redux, Vuex, and @ngrx/store.

In addition to logging Redux actions and state, LogRocket records console logs, JavaScript errors, stacktraces, network requests/responses with headers + bodies, browser metadata, and custom logs. It also instruments the DOM to record the HTML and CSS on the page, recreating pixel-perfect videos of even the most complex single-page apps.

Try it for free.

Share this:

Twitter Reddit LinkedIn Facebook

Flavio Copes Follow

I write tutorials for developers at https://flaviocopes.com

8 Replies to “JWT authentication: When and how to use it”

Steve M Says:
December 6, 2019 at 5:55 pm
Reply

I don’t understand some of the claims here.

“Don’t store it in local storage (or session storage). If any of the third-party scripts you include in your page gets compromised, it can access all your users’ tokens.”

Why does this matter, when you protect against CSRF with CSRF tokens?

Rye Says:
March 22, 2020 at 3:45 am
Reply

I tried storing it in cookie httpOnly but my problem is I cannot pass as request authorization header when making a request to the backend. How will this be solved?

Arun Teltia Says:
March 26, 2020 at 11:14 am
Reply

did you solve the problem coz i also had the same problem

Rye Says:
March 26, 2020 at 8:22 pm
Reply

After some research, yes. It’s automatically passed into the request cookies. Before I use req.headers.authorization in my middleware, now I have to use `req.cookies[‘name’]`. The idea of setting cookie as httpOnly is that you can never call it using JS to alter like localstorage.

Oren Magen Says:
April 11, 2020 at 10:56 am
Reply

″ there is always the possibility that it’s intercepted and the data deciphered″ – deciphered is not the right word here since JWT are serialised, not encrypted

To Bao Thien Quan Says:
May 6, 2020 at 4:49 am
Reply

“the possibility that it’s intercepted and the data deciphered, exposing your user’s data.”
We only store enough information to identify the user in the jwt token. It can be the user’s id, email, or even another access token (in case you want to implement remote logout or similar features). We don’t store sensitive data (e.g. password,...) in the token, so this should not be an issue.

Karthi Says:
May 25, 2020 at 7:59 pm

Reply

"Using JWT to authorize operations across servers" do you have any examples for this?

Gowtham Says:
October 18, 2020 at 11:00 am

Reply

I am new to JWT.
If not through JWT how should we send sensitive data (like a password) to a server while logging in.

Leave a Reply

Enter your comment here...