BEST

PRACTICES

Today, JSON Web Tokens are widely used in applications to share security information. Still, they are not entirely foolproof and could open doors for attackers.

What is JWT?

A JWT is a mechanism to verify the owner of some JSON data. It's an encoded, URL-safe string that can contain an unlimited amount of data (unlike a cookie) and is 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 because it's just serialized, not encrypted.

JWT

Swipe-up

When to use JWT authentication

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

Do not use JWTs for sessions

There is an increasing number of frontend developers who claim JWTs have some benefits for use as session retention mechanism, instead of session cookies and centralized sessions.

This should not be considered as good practice.

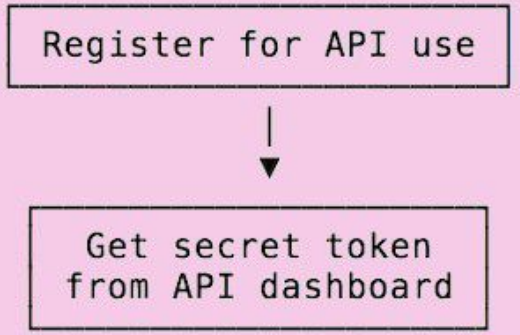
JWTs were never considered for use with sessions, and using them in such a way may actually lower the security of your applications.

Using JWT for API authentication

A very common use for JWT — and perhaps the only good one — is as an API authentication mechanism.

JWT technology is so popular and widely used that Google uses it to let you authenticate to its APIs.

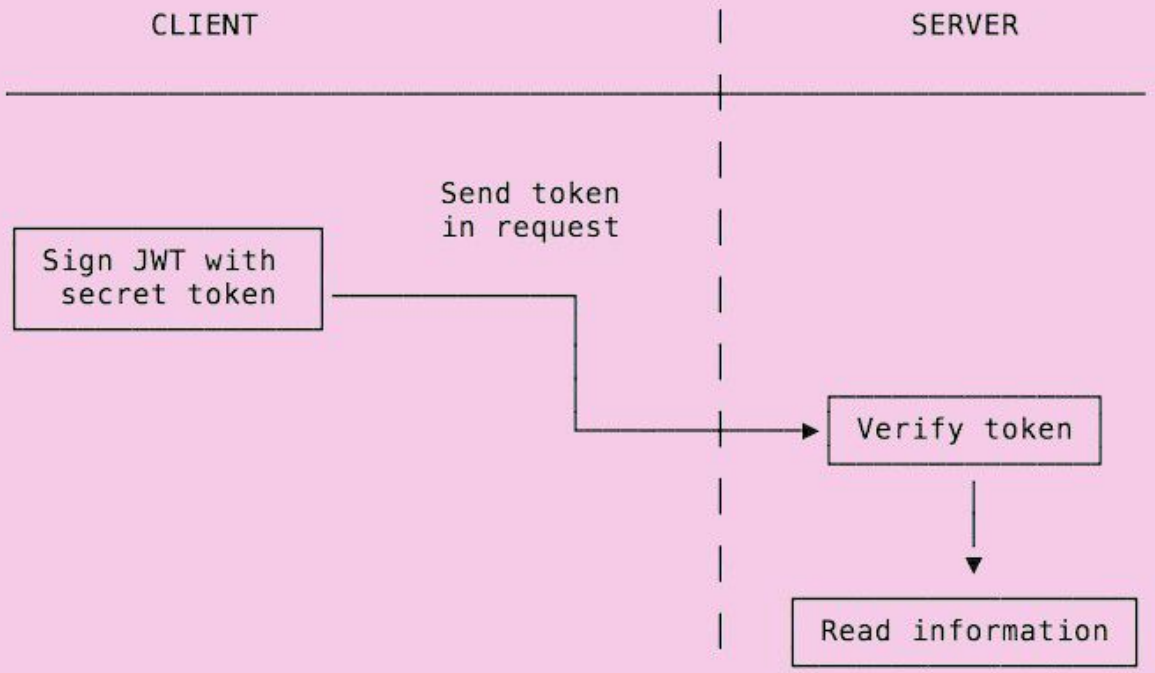
The idea is simple: you get a secret token from the service when you set up the API:



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

Using JWT for API authentication

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



Dealing with expiration, issued time and clock skew

JWTs are self-contained, by-value tokens and it is very hard to revoke them, once issued and delivered to the recipient.

Because of that, you should use as short expiration time for your tokens as possible - minutes or hours at maximum. You should avoid giving your tokens expiration times in days or months.

Remember that the exp claim, containing the expiration time, is not the only time-based claim that can be used for verification.
The nbf claim contains a "not-before" time.

The token should be rejected if the current time is before the time in the nbf claim. Another time-based claim is iat - issued at.
You can use this claim to reject tokens which you deem too old to be used with your resource server.

How to securely store JWTs in a cookie

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. This is as bad as it sounds; an XSS attack could give an external attacker access to the token.

To reiterate, whatever you do, don't store a JWT in local storage (or session storage). If any of the third-party scripts you include in your page is compromised, it can access all your users' tokens.

To keep them secure, you should always store JWTs inside an httpOnly cookie. This is a special kind of cookie that's only sent in HTTP requests to the server. It's never accessible (both for reading or writing) from JavaScript running in the browser.

Always Sign the Token

JWT signature is the fundamental security feature that ensures data (payload) within the token has not been altered.

To create a JWT signature, you need the encoded header, the encoded payload, a secret, and the algorithm specified in the header. For example, signature with HMACSHA256 algorithm would look like this:

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

So, if there are any changes in the header or payload, you will have to create a new token signature.

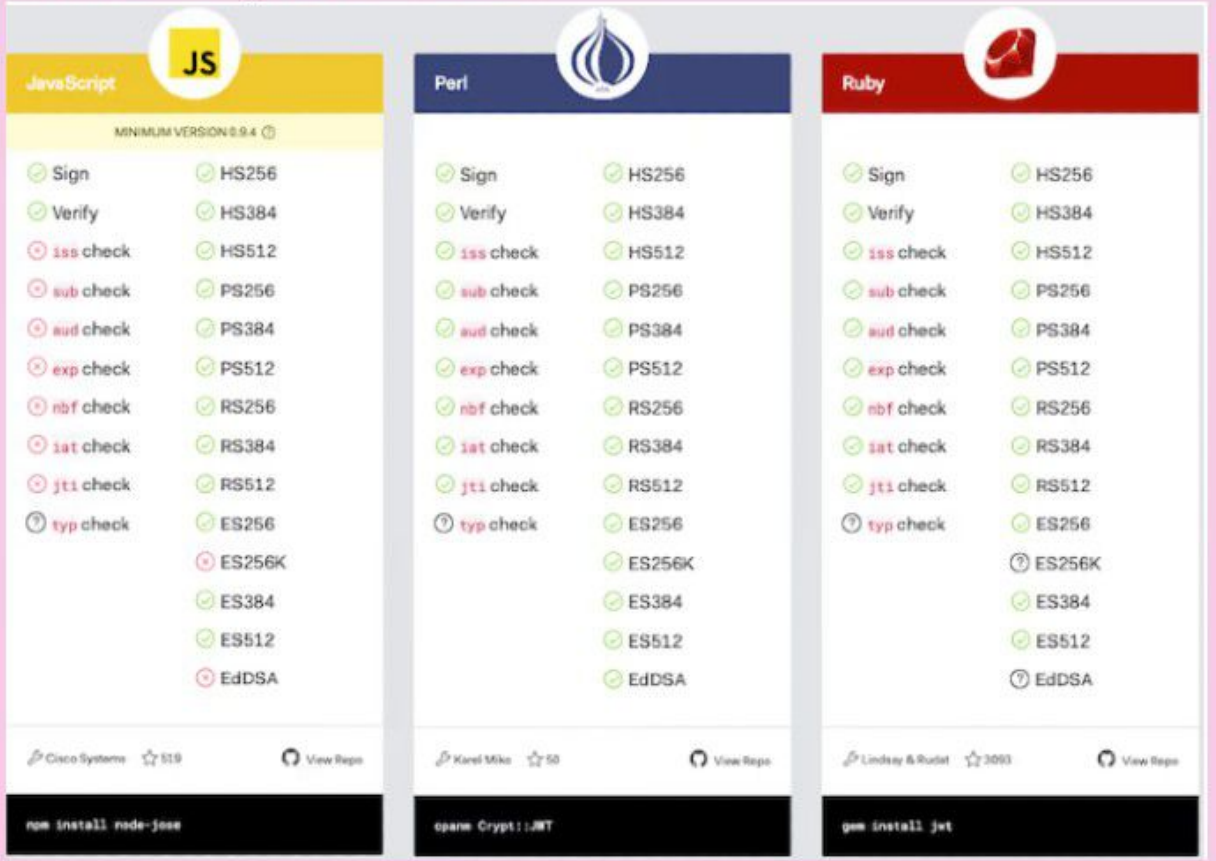
Also, you can include a random token ID in the jti claim to prevent two tokens from having the very same signature at the same time.

Swipe-up

How to choose the best JWT library

A list of the most popular libraries that implement JWT, including libraries for Node.js, Python, Rust, Go, JavaScript, and many more.

Select your language of choice and pick the library that you prefer — ideally, the one with the highest number of green checks.



JWT

Swipe-up

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.

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.

It's important to remember that JWT safety depends greatly on how the tokens are implemented and used. Just because a JWT contains a cryptographic signature it doesn't automatically mean that it's safe, or that you should blindly trust the token.

Unless good practices are observed you can quite easily become the victim of an attack.

The good practices outlined in this article are true at the time of writing.

You should remember that security standards and the security levels of our cryptography can change quite rapidly and it's good to keep an eye on what is happening in the industry.