

Authentication & Authorization

↙
Who are you?

↓
What can you do?

Most famous authentication techniques:

- OAuth 2.0
- JWT
- Zero trust architecture
- Password less auth

Three components that are important:

Sessions: A way for a server to remember who a user is across multiple requests.

HTTP is by default stateless. So, server forgets everything after responding. Session adds state on top of HTTP.

Workflow

- 1. User logs in with credentials
- 2. Server creates a session record
- 3. Server generates a session ID
- 4. ID is sent to the browser (cookie)
- 5. Browser sends the ID with every req
- 6. Server use the ID to find the session

JWT: Lets the server trust a signed token sent by the client instead of keeping session state.

It is a string token with three parts:

- Header → Token type & signing algorithm
- Payloads → User data (claims)
- Signature → Created using a secret or private key to prevent tempering

Workflow

- 1. User logs in
- 2. Server creates a JWT & signs it
- 3. Token is sent to the client
- 4. Client stores the token
- 5. Client sends token with each req
- 6. Server verify & No session lookup needed

JWT is stateless. So the server don't need to store the session which is memory inefficient, time consuming & not scalable.

Problem with JWT is that if anyone can access my token, he/she can impersonate as me & the server side don't have any mechanism to invalid that token untill it expire manually or the server change its secret.

But if the server change the secret to invalid one token, all the users of that server have to login again to get new JWT. Also JWT can't be easily revoked because there is no way to check the status of the token.

One way around is maintaining a blacklist of JWT in the memory of the server.

Cookies : Are a small piece of data that browser stores & sends back to a server with every request.

Mainly the server stores some data in browser & other server can't access that data.

Types of Authentication

- ① Stateful
- ② Stateless
- ③ API key based
- ④ OAuth 2.0 OIDC

OAuth solved a major problem in resource sharing.

If we want to give access to the resources of one platform to another we had to give the credentials of the platform could access all the resources.

OAuth solves this using tokens limiting access & allowing revocation at any time.

Although it solved the authorization problem, the authentication problem remains & therefore came Open ID Connect (OIDC).

Authorization → Providing specific permission to specific user.

One of the famous authorization technique is Role Based Access Control (RBAC).

Defending Attacks

While authenticating we send user friendly error messages:

Username not found

Invalid Password

But these could help attackers. By seeing the messages they can confirm the the username is correct. So, use like the following:

Invalid credentials

There is another type of attack "Timing Attack".

Time take to get response if authentication failed at username & password is different.

So, we need to equalize the response time.

const. time operation method

or

Simulate response delay