**Web Security**

**Oauth 2**

Oauth is used for authorisation, not authentication. It gives permissions as to what you can do, authentication is done elsewhere.

How it works e.g.

* You give a website access to post FB posts on your behalf, the website sends the user to Facebook to authenticate
* Facebook confirms the users credentials and generates a token which is passed to the website to use when posting on the users behalf.
* Now, when the website wants to post on the users behalf, it sends requests with that token. The type of requests a website can make is limited by the permissions given (by FB in this example)
* The user never shares their password!

Oauth lingo

|  |  |
| --- | --- |
| Flow | Exchange between website and oauth provider (e.g Facebook) |
| Scopes | The permissions given to the website |
| Tokens | Used to access provider |
| Provider | Used to authenticate, e.g. Facebook, GitHub, Google |

Approaches for API authentication & authorisation

1. API keys
   1. Appended to URL or passed as header
   2. Most common
   3. Hard to manage as these keys can be passed all over
   4. All or nothing, i.e. hard to manage permissions – user can do all CRUD actions
2. Account identifier and secret
   1. Similar to API key
3. Oauth
   1. Easy to delegate selective access
   2. Easy to revoke access
   3. Common, well supported in most tech stacks
   4. Not easy to implement

Oauth is for authorisation, it does not specify payloads, though it often uses JWT. Oauth does not specify *how* someone authenticates, it just happens and we don’t know *who* just authenticated.

Oauth end points

* /authorize (required)
  + What the end user interacts with to establish permissions for the resource. Could return an authorisation or an access token.
* /token (required)
  + Where the client application can exchange the authorisation from the first end point for an access or refresh token.
* /revoke
  + The end point which applications use to deactivate a token
* /introspect
  + Check whether active, i.e. got an expiration time
* /register
  + The end point which applications use to create new oauth clients (client id and client secret), for provisioning new applications or users

Token management

Access tokens are credentials used to access protected resources. An access token is a string representing an authorisation issued to the client.

Tokens represent specific scopes/permissions and durations of access, granted by the resource owner, and enforced by the resource server and authorisation server.

Access tokens can have different formats and structures. They might include data in them or reference data on the server/DB.

Authorisation in microservices

Designed specifically for server-to-server interactions where there is no user.

|  |  |  |
| --- | --- | --- |
| Client | Auth server | Resource server |

1

2

3

4

1. Client credentials given to the auth server in order to authenticate the user
2. Access token returned in order to use in request header/url to the resource server
3. Access to protected resource using token
4. Protected resource response returned from the resource server

Private clients only. Secrets must be in backend code. We therefore cannot use this in mobile applications because the secret would need to be on the client.

The client id and client secret are tied to the application and not the user.

Authorisation for web/mobile applications

Designed specifically for *untrusted* or public clients, where a malicious user could get access to their source code. Therefore, we don’t use the client secret.

This is only for users who explicitly grant authorisation via an authentication and user consent flow.

|  |  |  |
| --- | --- | --- |
| Browser | Trusted 3rd party oauth | Web app |

1

2

3

4

5

1. Navigate to application
2. Respond with client front-end
3. User requests sign-in, which sends client id and reply url to request an access token
4. Redirect to reply url with access token in url
5. Client sends access token in auth header to web app or API