Authentication and Authorization

Basic Concepts & Introduction to OAuth

Authenticate & Authorize

- **Authentication** - validate the identity of a "user", agent, or process
- **Authorization** - specifying rights to access a resource

Authentication is responsible for identifying who the user is.

Authorization is responsible for deciding what the user has permission to do.

Other Aspects of Security

- Access Control - controls access to resources
- Data Integrity - prevent data from being modified or corrupted, and *prove* that data hasn't been modified
- Confidentiality & Privacy - privacy is about people, confidentiality is about data
- Non-repudiation - prove that user has made a request
  - "repudiate" means to *deny* having done something
- Auditing - make a tamper-resistant record of security related events

Authentication Methods

Authentication methods for humans:

1. Username & password

2. Username & one-time password (TOTP, codes, SMS)

3. Biometrics - fingerprint, facial recognition, iris scan

4. Trusted 3rd Party - OAuth and OpenID
   "Login with Google" or "Login with Facebook"

5. Public-Private Keys

6. SQRL - a new method by Steve Gibson

Mantra of Authentication

Use at least 2 of these...

Something you _____

- *a username and password*

Something you _____

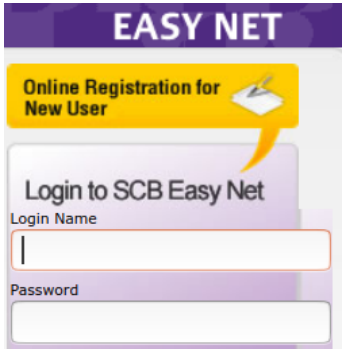
- *key card, registered mobile phone*

Something you _____

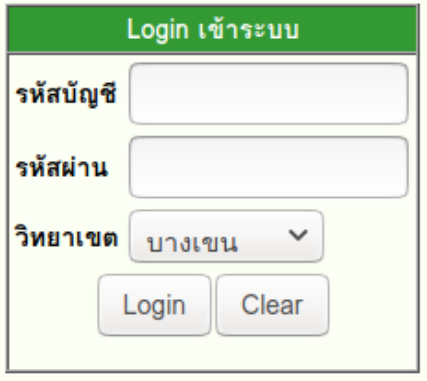
- *finger print, face*

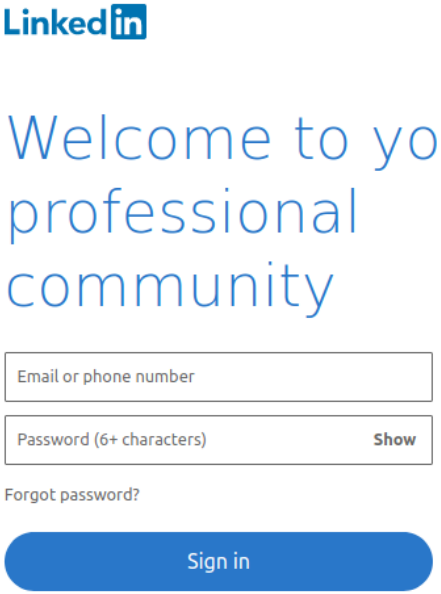
Username & Password

The oldest and one of the worst authentication methods.

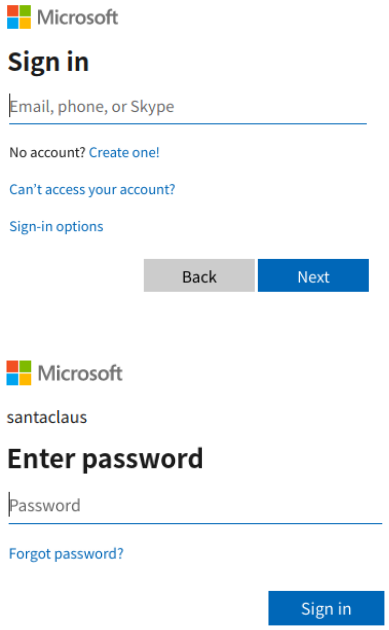


KU





Two page design



Username & Password

Why passwords are not secure:

- passwords can be stolen
- passwords can be guessed or "brute forced"
- vulnerable to man-in-the-middle & replay attack
- people reuse passwords or use weak passwords

Exercise: Have You Been Pwned?

Has your email address (and data) been stolen?

`https://haveibeenpwned.com/`

Has your **password** been seen in a data breach?

`https://haveibeenpwned.com/Passwords`

Key Observation about Passwords

- password is not using the *computational ability* of the user's device. It's just a fixed string.
- with just *a little computation ability* we can devise much more secure authentication protocols

Public-Private Key Algorithms

Public-private key pairs: Uses RSA (large prime numbers) or Elliptic Curves (Ecliptic Curve Cryptography)

Private key:
p(m)

Public key:
P(m)

m = a *message* to encrypt or decrypt

p(m) and P(m) are inverse:

P(p(m)) --> m

p(P(m)) --> m

PKI = public key infrastructure

Public-Private Auth Example

1. You connect to a server and give your username.
2. Server looks up your public key (P) and chooses a random message: **m1**
3. Server encrypts m1 with your public key:
   challenge = P(m1)
4. Server sends *challenge* to you and says:
   "*if this is really who you claim to be, then decrypt this challenge and send it back."*
5. You decrypt the challenge: m2 = p(*challenge*)
5. You *encrypt* and return a response = p(m2 + 1)
6. Server checks response: P(response) == m1 + 1 ??

OAuth & OpenID

Use a 3rd party to validate the user's identity

OAuth providers

OAuth 2.0

You choose "Google".

Shopee ***redirects*** you to Google (may open a pop-up):

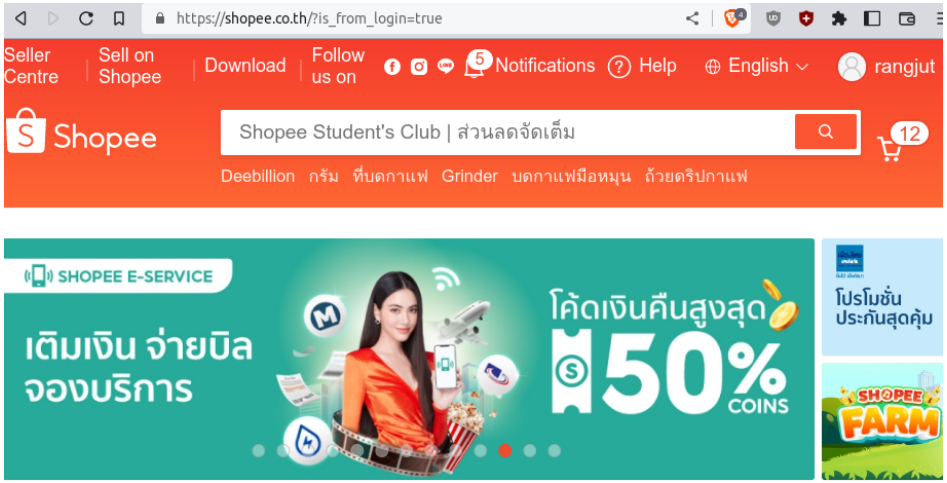
- tells Shopee who you are (grant access to your name & email),
  and **proves** that you have authenticated yourself to Google.

After You Approve...

You are *redirected* back to Shopee (the client).

What happened?

- Google gave your browser an "authorization code", & redirected the browser to Shopee "callback address"
- Shopee used the "authorization code" to get an "access token" to access your resources
- Shopee uses Google API and the "access token" to get your name and email address.



OAuth is for Authorization

OAuth is *really* about granting access to resources.

But, as a side effect, you confirm your identity.

What Happened?

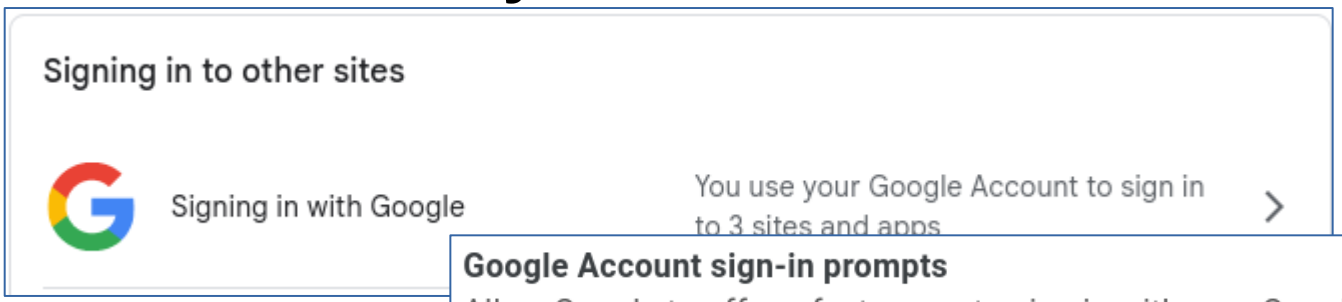
When you click "Login with Google",
what happens *behind the scene*?

presented in class

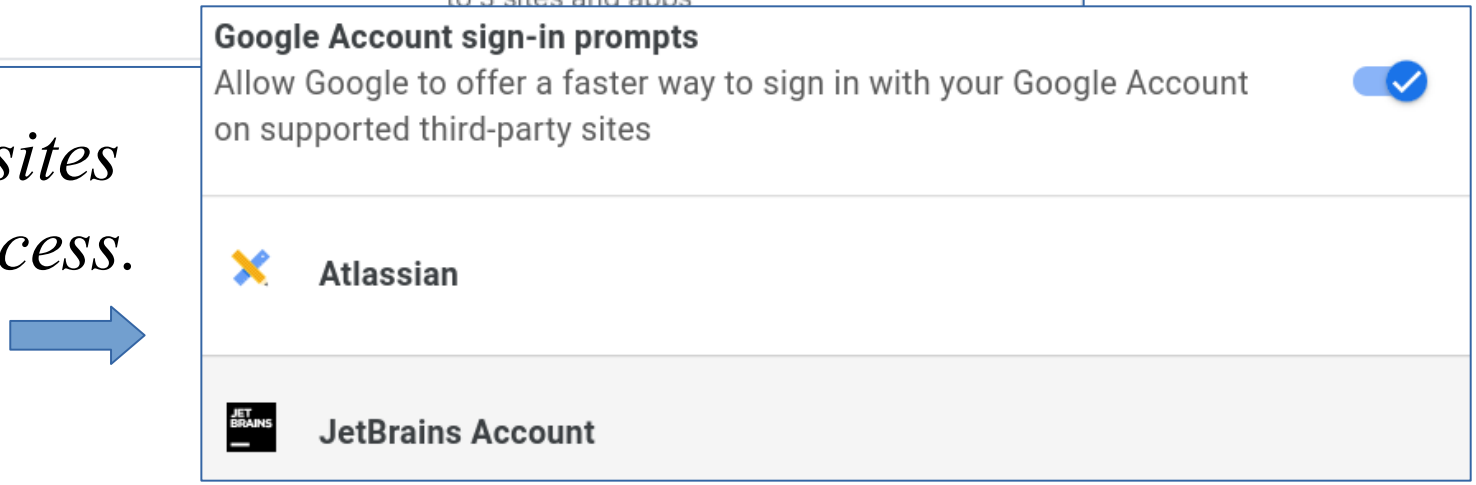
Who has Access to your Google Account?

1. Login to Google.
2. Visit https://accounts.google.com

   Chrome: Click your Photo -->
   "Manage Your Google Account"
3. Choose **Security**. Scroll down to...



List of sites with access.



Who has Access to your Google Account?

Does anyone have **more than 10 sites** with access to your Google account?

*Any sites that you **don't use anymore?***

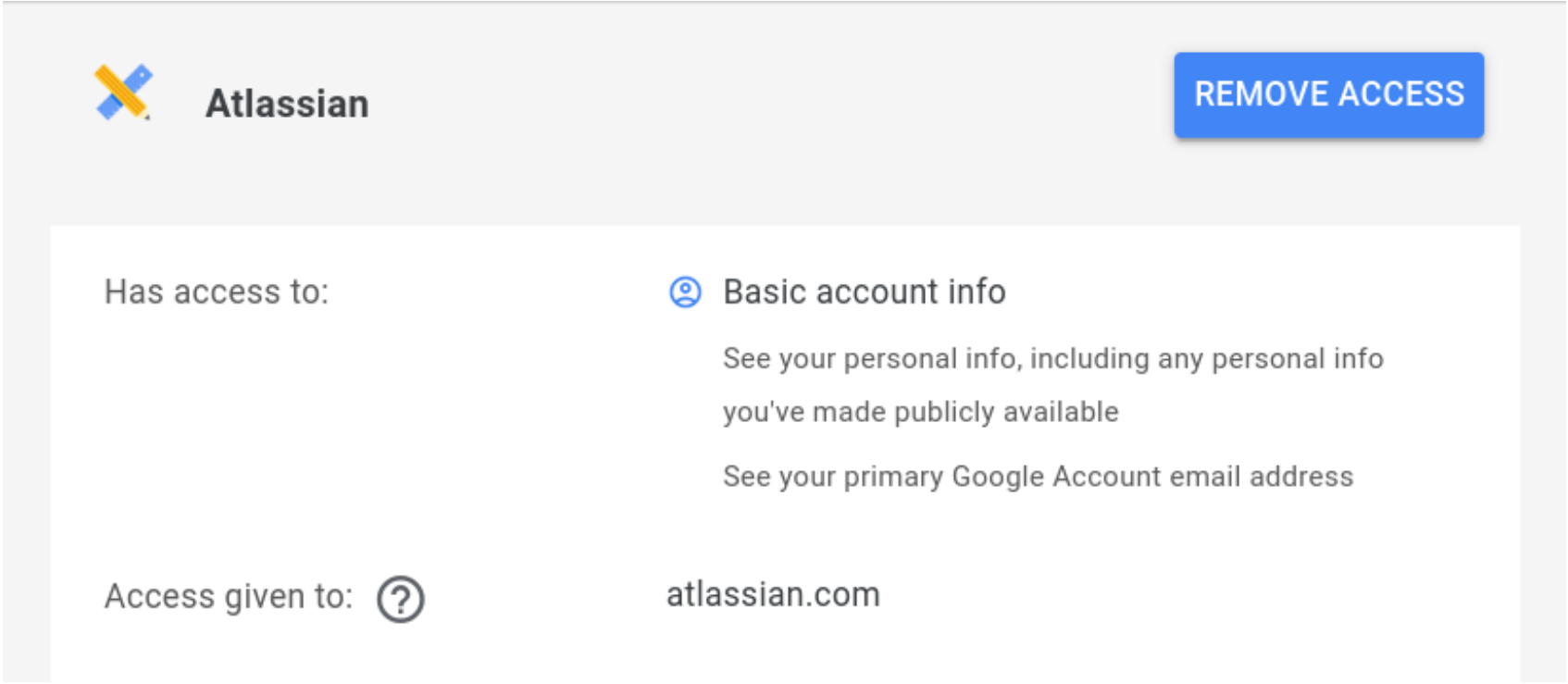
Delete them!

What Privileges (access) do sites have?

This is called the "scope".

When a site requests OAuth access to your account, it specifies what privileges (scope) it wants.

Click on a site name to view details:



Apps with Access to your Data

1. Go to My Account on Google.
2. Choose "Data & Privacy"
3. Choose "Third Party Apps with Account Access"

Are there any apps?

What apps?

Be Security Conscious

Supply Chain Attack: hackers target developer accounts.

If successful, they plant back doors or malicious code in the developer's source code.

Malicious PyPI packages with over 10,000 downloads taken down

By **Ax Sharma** December 13, 2021 06:54 AM 0

The Python Package Index (PyPI) registry has removed three malicious Python packages aimed at exfiltrating environment variables and dropping trojans on the infected machines.

These malicious packages are estimated to have generated over 10,000 downloads and mirrors put together, according to the researchers' report.

https://www.bleepingcomputer.com/news/security/malicious-pypi-packages-with-over-10-000-downloads-taken-down/

What You Should Do

Google Account

1. Review everything in the "Security" page on Google.
2. Use 2-Factor Authentication.

Github Account

1. Review everything in "Settings".
2. OAuth grants: Applications -> Authorized OAuth Apps
3. Consider using SSH and GPG keys
4. Use 2-Factor Authentication.

OAuth Use Cases

Server-side web app: The server-side has a "secret" that it uses when requesting access to a user's resources.

Single Page Web App: Javascript code running in web browser. Cannot keep a secret, so the flow is different.

Mobile App: uses a native mobile app as intermediary to grant OAuth access. Cannot keep a secret.

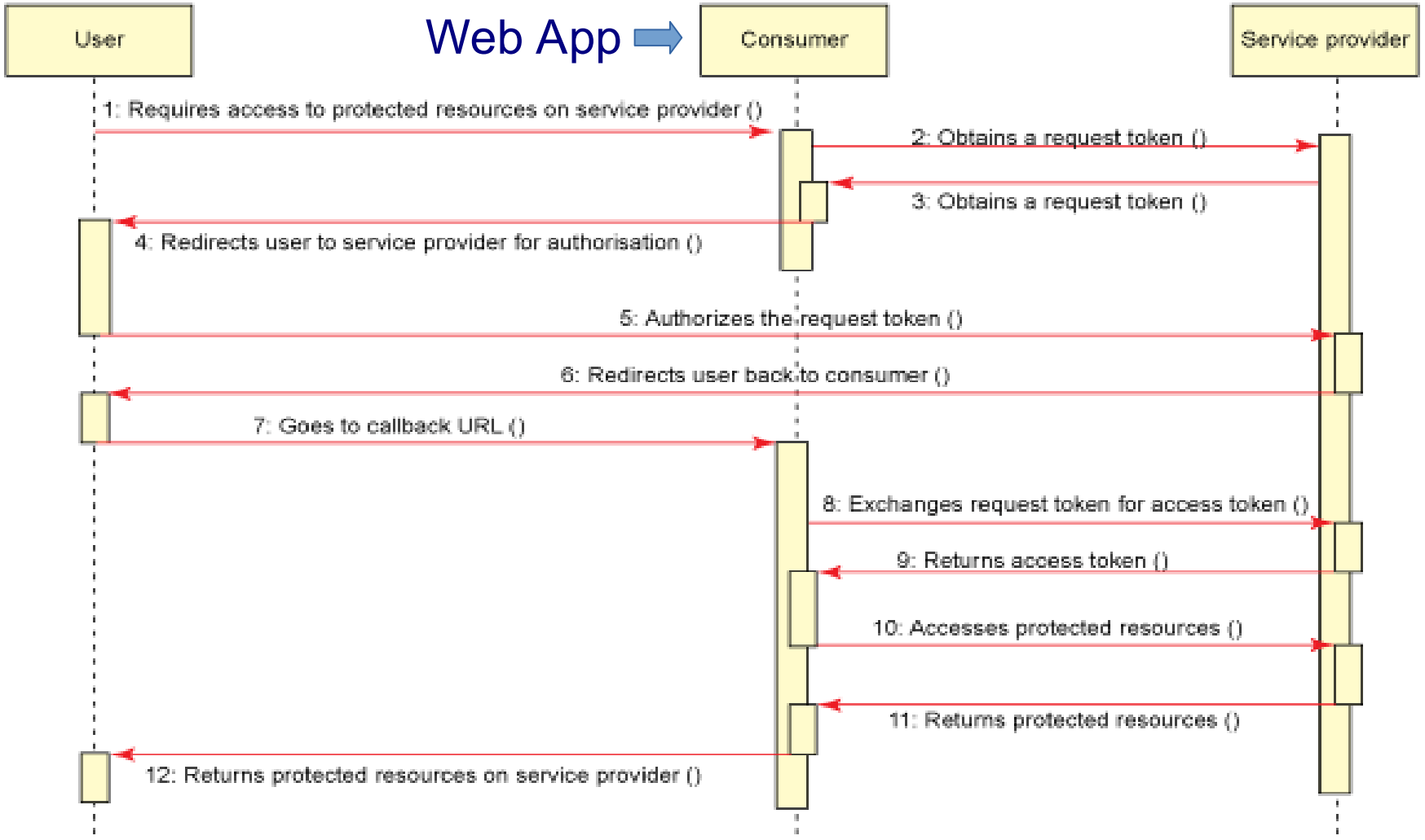
When you apply to Google, etc, for your app to use OAuth it is important to choose the correct "flow" or "grant type".

OAuth "Flows"

Use Case	OAuth Flow to Use
Server-side web app *server can keep a secret*	Authorization Code Flow
Single-page app *Javascript in browser* *cannot keep a secret*	Authorization Code Flow with PKCE or Implicit Flow with Form Post
Native Mobile app	Authorization Code Flow with PKCE

PKCE = Proof Key for Code Exchange

OAuth Flow for Server-Side Web Apps



This is the OAuth 1.0 flow, some names and parameters are different in OAuth 2.0

Step 0: Register your app

Go to the OAuth provider and request OAuth access.

"Register an application" using "Authorization Code" flow

give the server:

app. name and URL,

a callback URL,

requested scopes

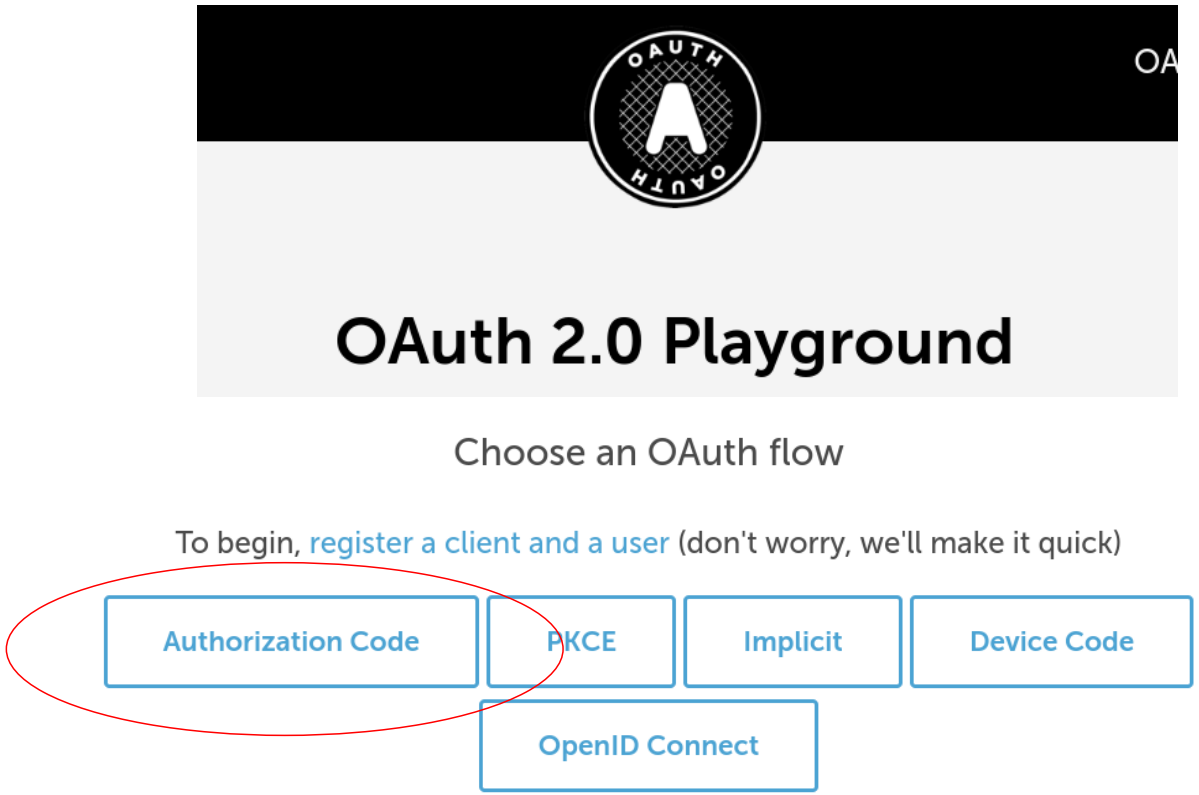
server gives you:

- client_id
- client_secret
- authorization URL - where you send the user's browser

Exercise: OAuth 2.0 Playground

`https://www.oauth.com/playground/`

Choose "Authorization Code" Flow
and work through the exercise



How to Register an App?

You need a developer account on an OAuth Authentication Server.

Need to provide:

- name & "home page" of the application
- OAuth "Flow" you want to use (not all providers)
- authorization callback URL
- scope(s) you want - access to what resources?

What the OAuth Provider Gives You

Client ID

Client Secret

Authorization URL - where to redirect "User Agent" (user's browser)

and from the provider documentation you learn:

Authorization Code Exchange URL - where to exchange an "authorization code" for an "access token"

Where to Register an App?

Github:

`https://github.com/settings/applications/new`

Google Developer Console:

`https://console.developers.google.com/`

`Django-allauth` docs list the OAuth "providers" and the Registration URLs. Also lists the "callback URL".

`https://django-allauth.readthedocs.io/en/latest/providers.html`

What's Next?

OK, you can an authorization code.

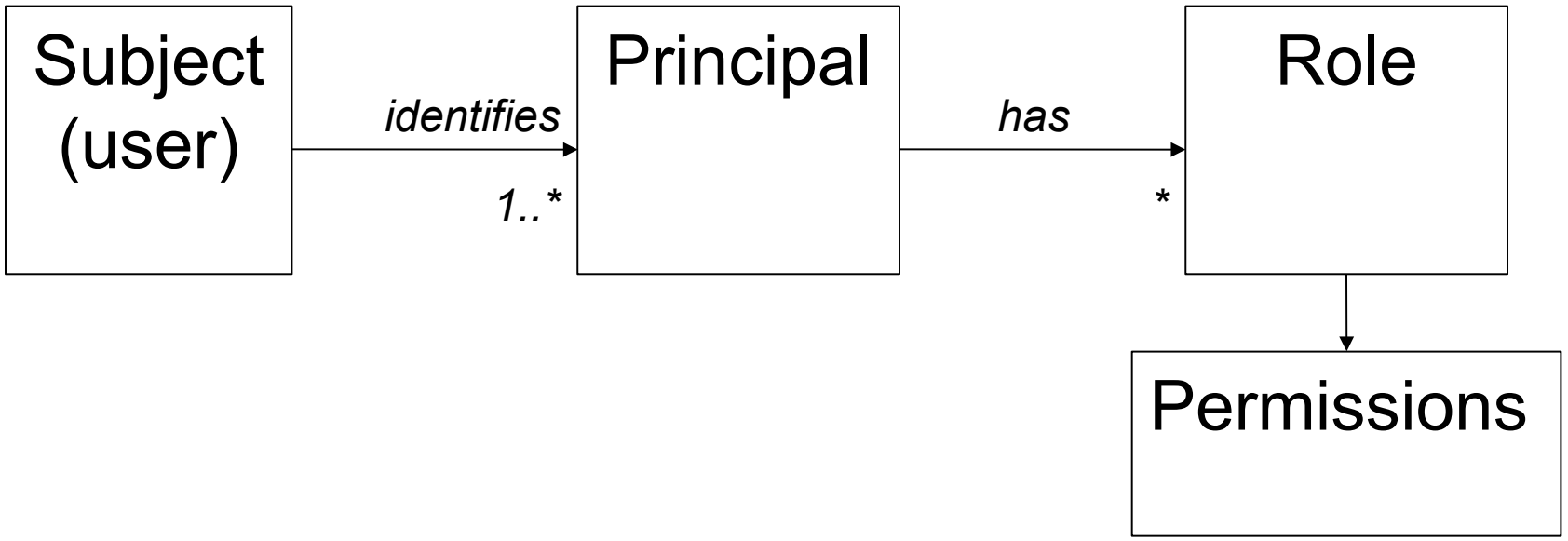
How do you use it?

Role Based Authorization

Permissions are based on the *roles* a user possesses.

A user may have many roles.

Example: “joe” has roles “voter” and “administrator”



Learn OAuth

`https://www.oauth.com`

Google OAuth Playground

`https://developers.google.com/oauthplayground/`