

Authentication and Authorization

Basic Introduction for Web Apps

Authorization in Django

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 2 or 3 of these...

Something you _____

- *a username and password*

Something you _____

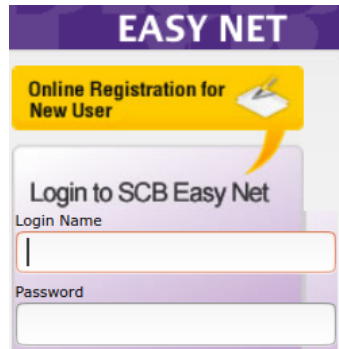
- *key card, registered mobile phone*

Something you _____

- *finger print, face*

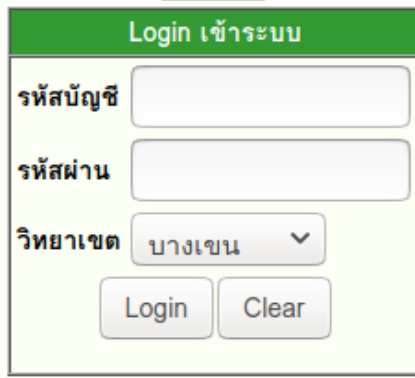
Username & Password

The oldest and one of the **worst** authentication methods.



A screenshot of the SCB Easy Net login interface. It features a purple header with 'EASY NET' in white. Below is a yellow banner for 'Online Registration for New User'. The main login area has a purple background with the text 'Login to SCB Easy Net'. It includes a 'Login Name' label and a text input field, and a 'Password' label and a password input field.

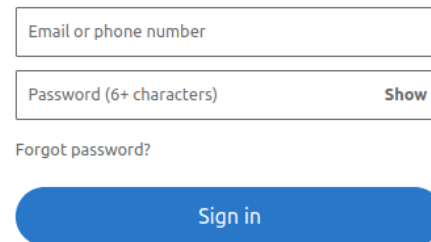
KU



A screenshot of the KU (Kuliah) login interface. It has a green header with 'Login เข้าสู่ระบบ'. Below are three input fields: 'รหัสบัญชี' (Account ID), 'รหัสผ่าน' (Password), and 'วิทยาเขต' (Campus) with a dropdown menu showing 'บางเขน'. At the bottom are 'Login' and 'Clear' buttons.

LinkedIn

Welcome to yo
professional
community



A screenshot of the LinkedIn login interface. It features a blue header with the LinkedIn logo. The main text says 'Welcome to yo professional community'. Below are two input fields: 'Email or phone number' and 'Password (6+ characters)' with a 'Show' button. A 'Forgot password?' link is below the password field. At the bottom is a large blue 'Sign in' button.

Two page design

Microsoft

Sign in

Email, phone, or Skype

No account? [Create one!](#)

[Can't access your account?](#)

[Sign-in options](#)

Back

Next

Microsoft

santaclaus

Enter password

Password

[Forgot password?](#)

Sign in

Username & Password

Why passwords are not very 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

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
 - Time-based OTP (Authenticator apps)
 - Challenge-response
 - FIDO
 - Web Auth *aka* Passkeys - a new standard from Google, Apple, and others
 - SQRL - a better Passkeys, but not widely used

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`

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:

$P(m)$

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

$P(p(m)) \rightarrow m$

$p(P(m)) \rightarrow m$

PKI = public key infrastructure

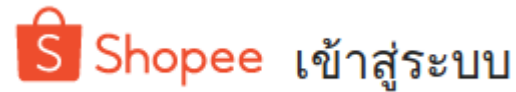
m = a *message* to encrypt or decrypt

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

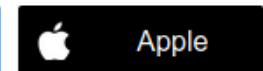
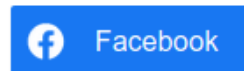


เข้าสู่ระบบ

ลืมรหัสผ่าน

เข้าสู่ระบบด้วย SMS

หรือ



เพิ่งเคยเข้ามาใน Shopee ใช่มั้ย? [สมัครใหม่](#)

OAuth providers



OAuth 2.0

OAuth is *really* about **granting access** to **resources**.
But, as a side effect, you confirm your identity.

Google.com

*"shopee.co.th wants access
to your name and email"*

Agree

Cancel

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

What Happened?

When you click "Login with Google", what happens?

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...

Signing in to other sites

 Signing in with Google


You use your Google Account to sign in to 3 sites and apps >


List of sites with access.

➡

Google Account sign-in prompts

Allow Google to offer a faster way to sign in with your Google Account on supported third-party sites ☒

 Atlassian

 JetBrains Account

Who has Access to your Google Account?

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

*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:

 **Atlassian**


REMOVE ACCESS

Has access to:

 Basic account info

See your personal info, including any personal info you've made publicly available

See your primary Google Account email address

Access given to: 

atlassian.com


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



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

Authorization Code Flow

server can keep a secret

Single-page app

Authorization Code Flow with PKCE
or Implicit Flow with Form Post

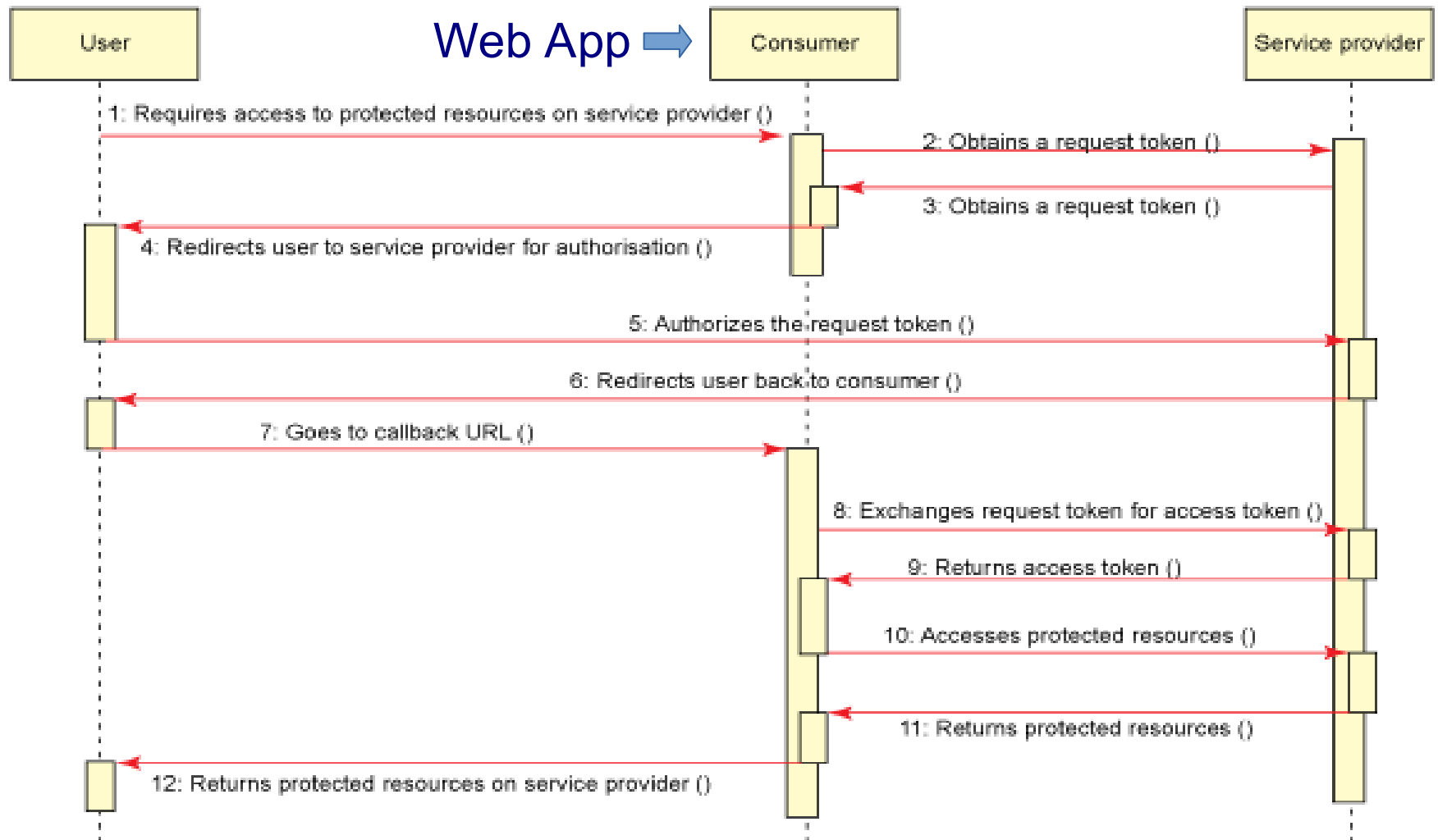
*Javascript in browser
cannot keep a secret*

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:

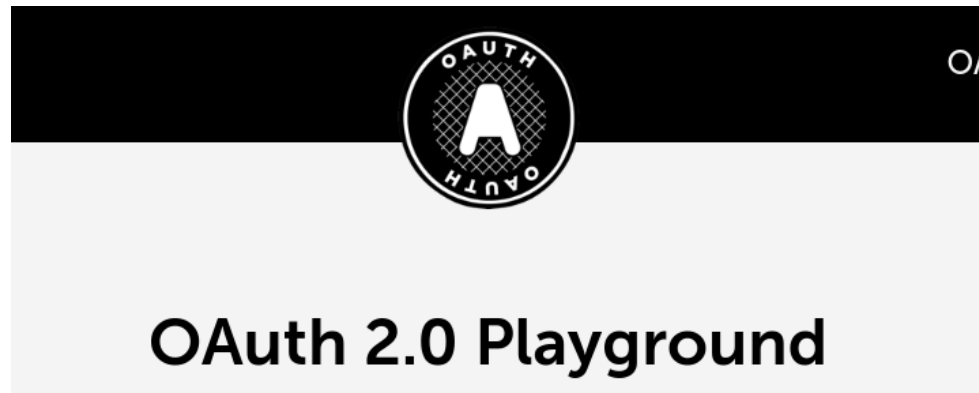
app. name and URL,
a callback URL,
requested scopes

server gives you:

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

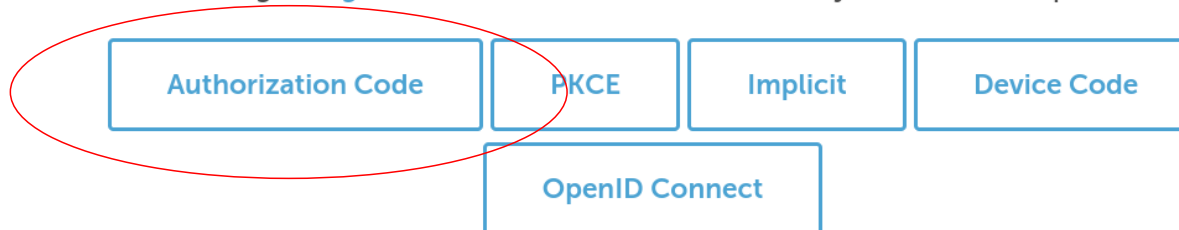
Exercise: OAuth 2.0 Playground

<https://www.oauth.com/playground/>
Choose "Authorization Code" Flow
and work through the exercise



Choose an OAuth flow

To begin, [register a client and a user](#) (don't worry, we'll make it quick)



What's Next?

OK, you can an authorization code or authorization token.

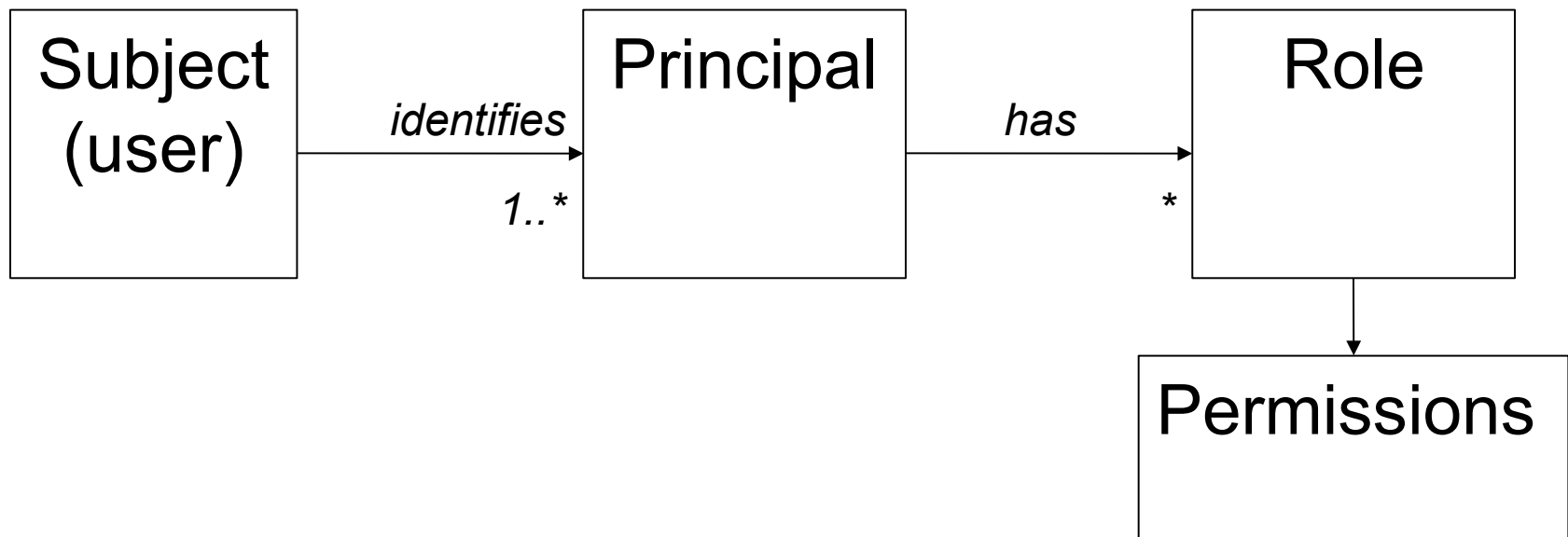
How do you use it?

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/`