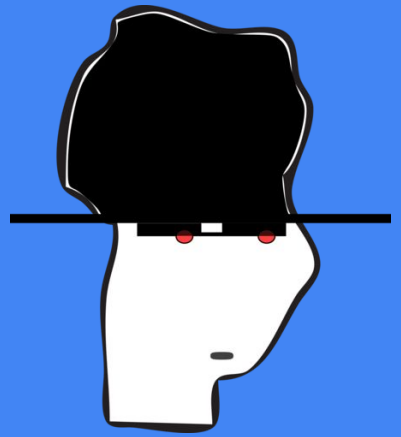
Seguridad Web

Introducción a OAUTH2.0



ATENCIÓN

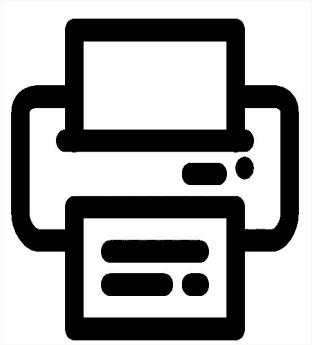
Agenda

- AuthN & AuthZ
- OAuth 2.0
- Attacks
  - Token/Code stealing
  - CSRF to link attacker account
  - Token impersonation

Autenticación(AuthN) vs Autorización(AuthZ)

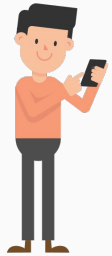
La autenticación es el proceso de verificar una identidad, es decir confirmar que una persona es quien dice ser. Normalmente, para constatar este hecho, el usuario usa algo que conoce para demostrar su identidad, como un usuario y una contraseña.

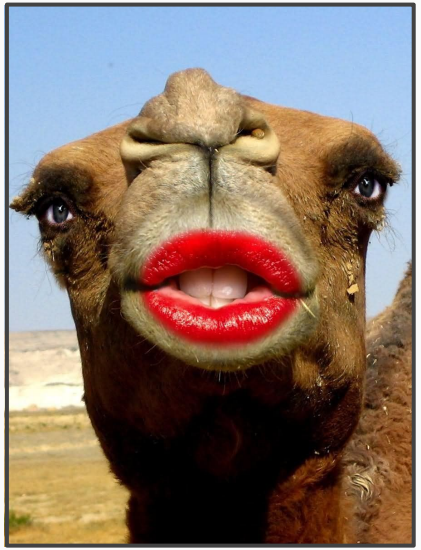
Por otro lado, **la autorización es el proceso de verificar lo que un usuario puede hacer**. Por ejemplo, un usuario puede añadir canciones en una lista compartida de Spotify, pero no puede eliminar dicha lista. La autorización ocurre después de que un usuario se haya autenticado.



Printing application

User





Resource to print which is protected

https://photos.google.com/photo/7sd778sfa87ad8sdf7

Accediendo a recursos protegidos de un usuario.

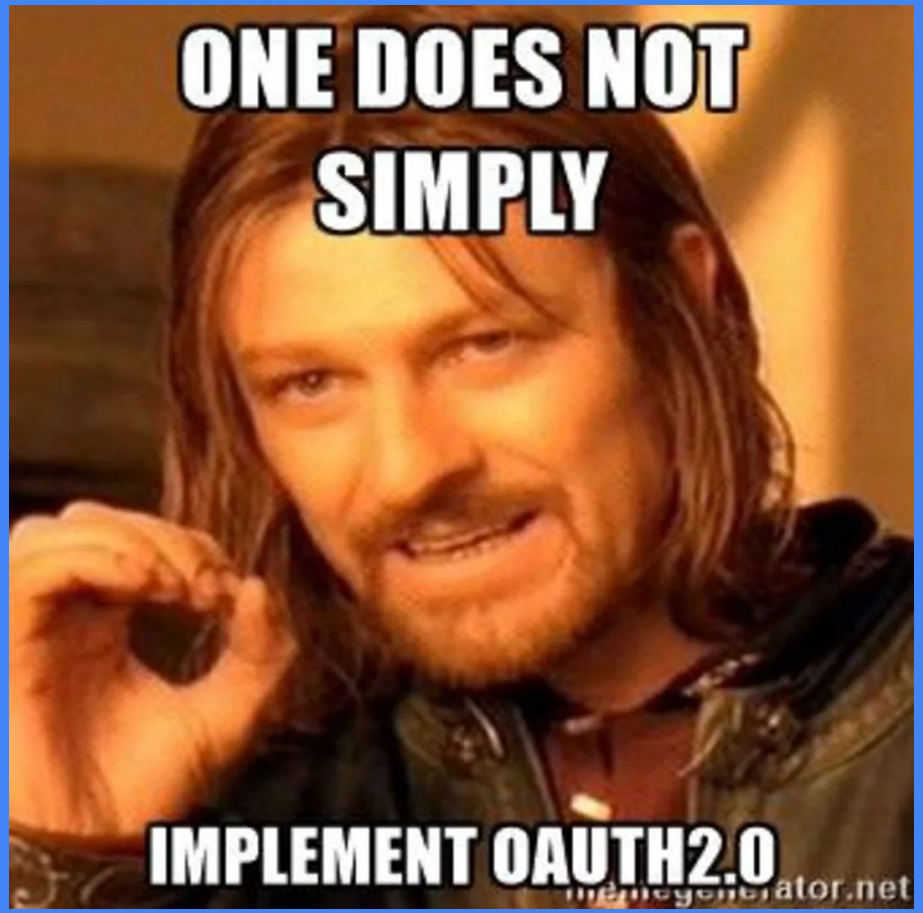
Métodos de AuthN & AuthZ antes de OAuth

- Compartir usuario y contraseña
  - Actuando exactamente como usuario final.
- Cookies
  - Cross-Site Request Forgery (CSRF o XSRF).
- API Keys
  - Dan acceso total a todas las operaciones que puede llevar a cabo la API.
  - No permiten identificar usuarios, identifican proyectos.

OAuth

- Authorization framework
- Permite a aplicaciones de terceros acceder de forma limitada a diferentes servicios.

OAUTH 2.0



hentication bypass on Airbnb via OAuth tokens theft

d on June 22, 2017 by Arne Swinnen

OR: Login CSRF in combination with an HTTP Referer header-based open redirect in Airbnb's OAuth login flow, could be abused to al OAuth access tokens of all Airbnb identity providers and eventually authenticate as the victim on Airbnb's website and mobile ion. This attack did not rely on a specific OAuth identity provider app configuration flaw (e.g. wildcards in whitelisted redirect_uri made it generic for all Airbnb's identity providers (Facebook & Google at the time of reporting). Airbnb fixed both the login ct issues and awarded a $5.000 bounty back in the summer of 2016.

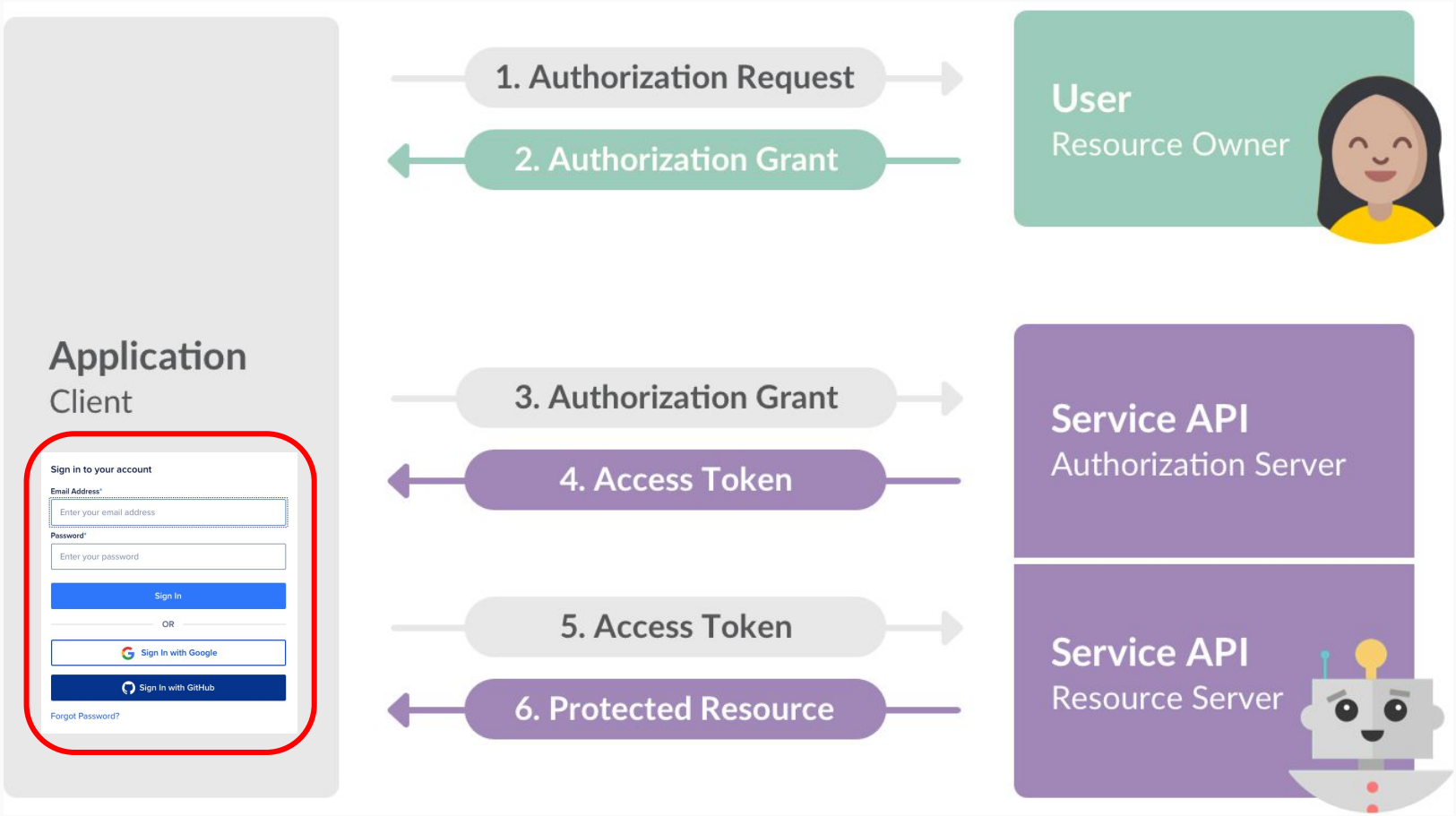
Facebook OAuth Framework Vulnerability	Amol Baikar (@AmolBaikar)	GitHub	OAuth flaw, Authorization bypass	$25,000
assing GitHub's OAuth flow	(@no…	Facebook	OAuth flaw	$55,000

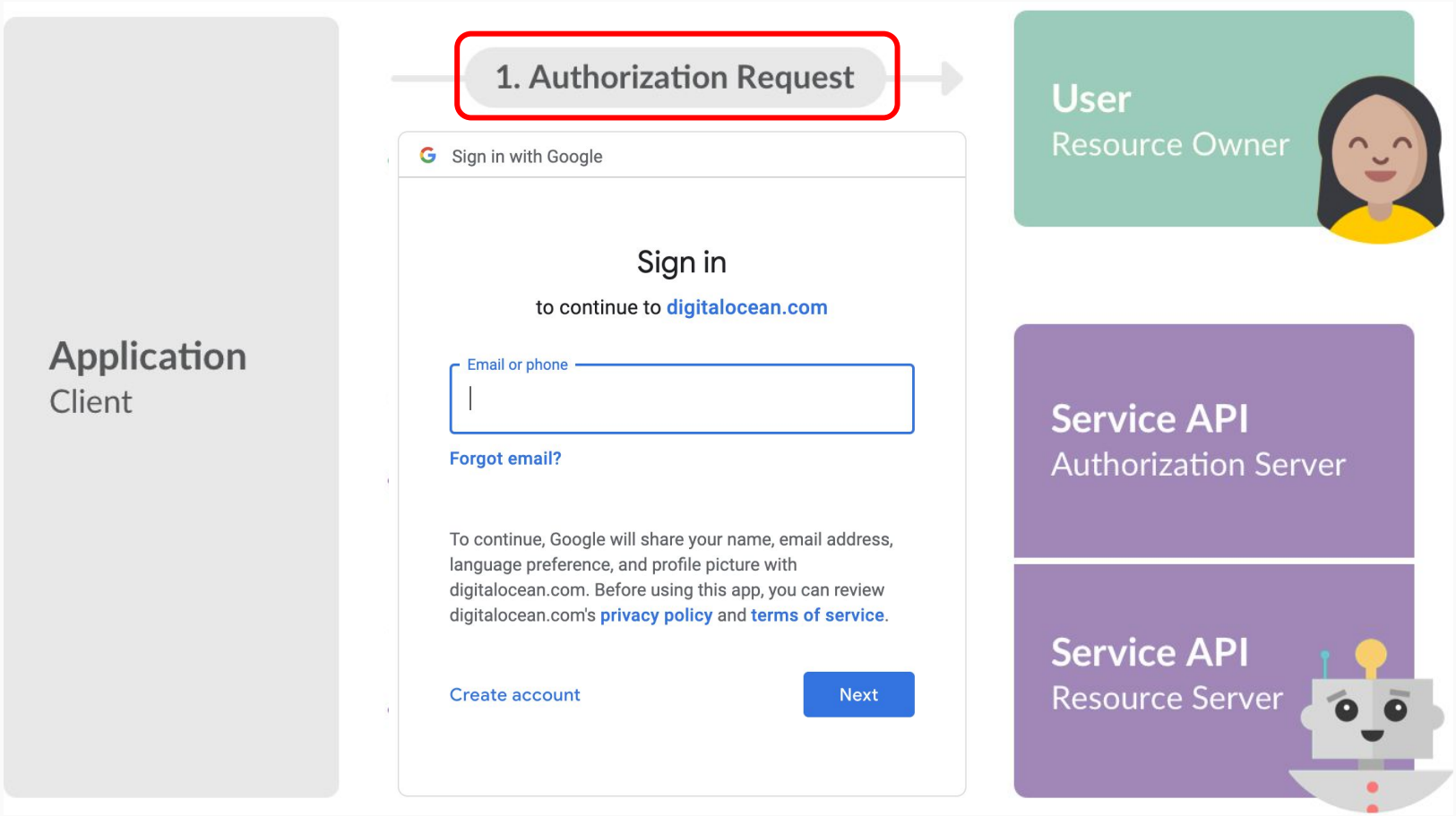
Flows

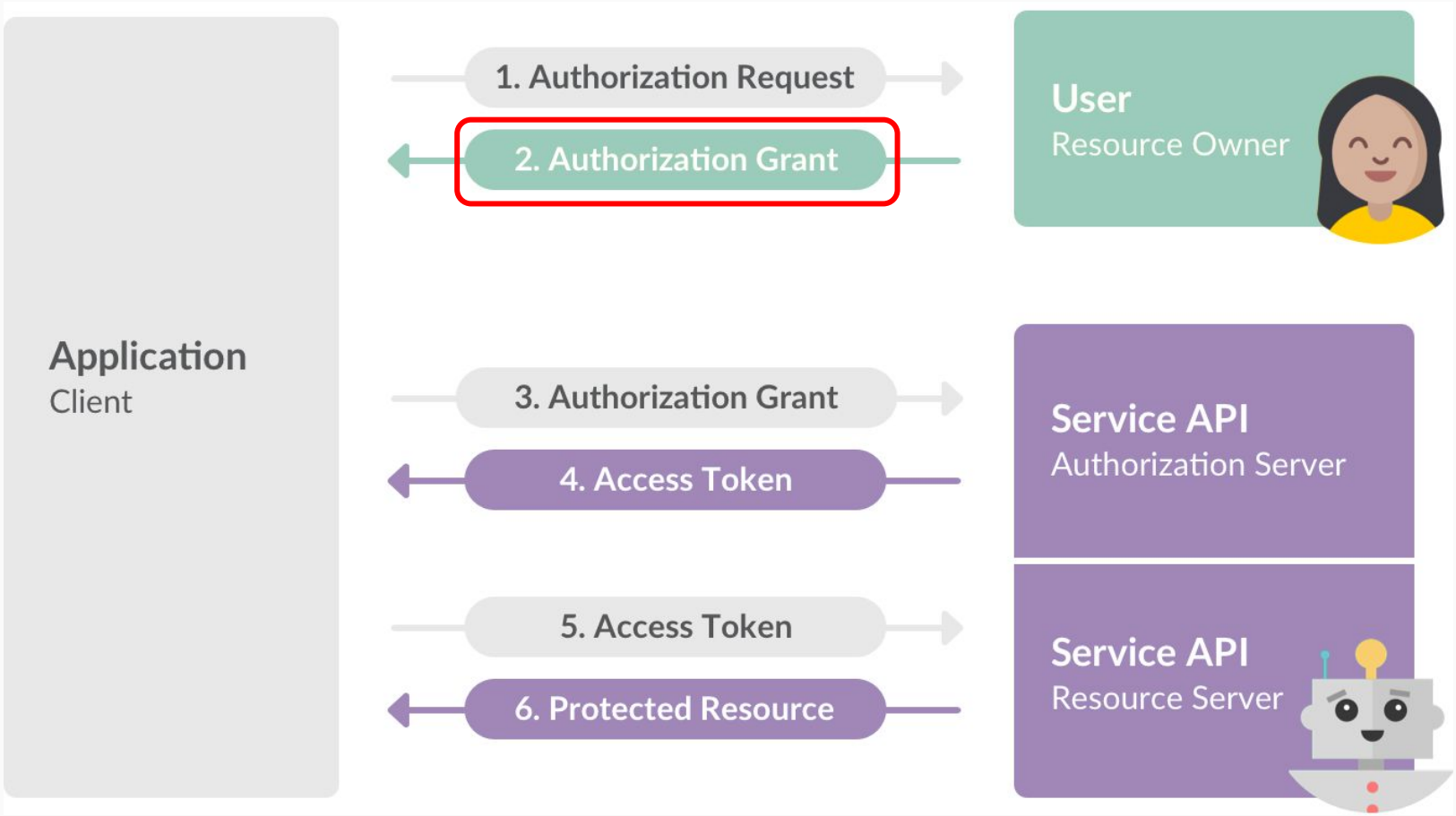
- Authorization Flow
- Implicit Flow
- Resource Owner Password Credentials Flow
- Client Credentials Flow
- Refresh Token Flow

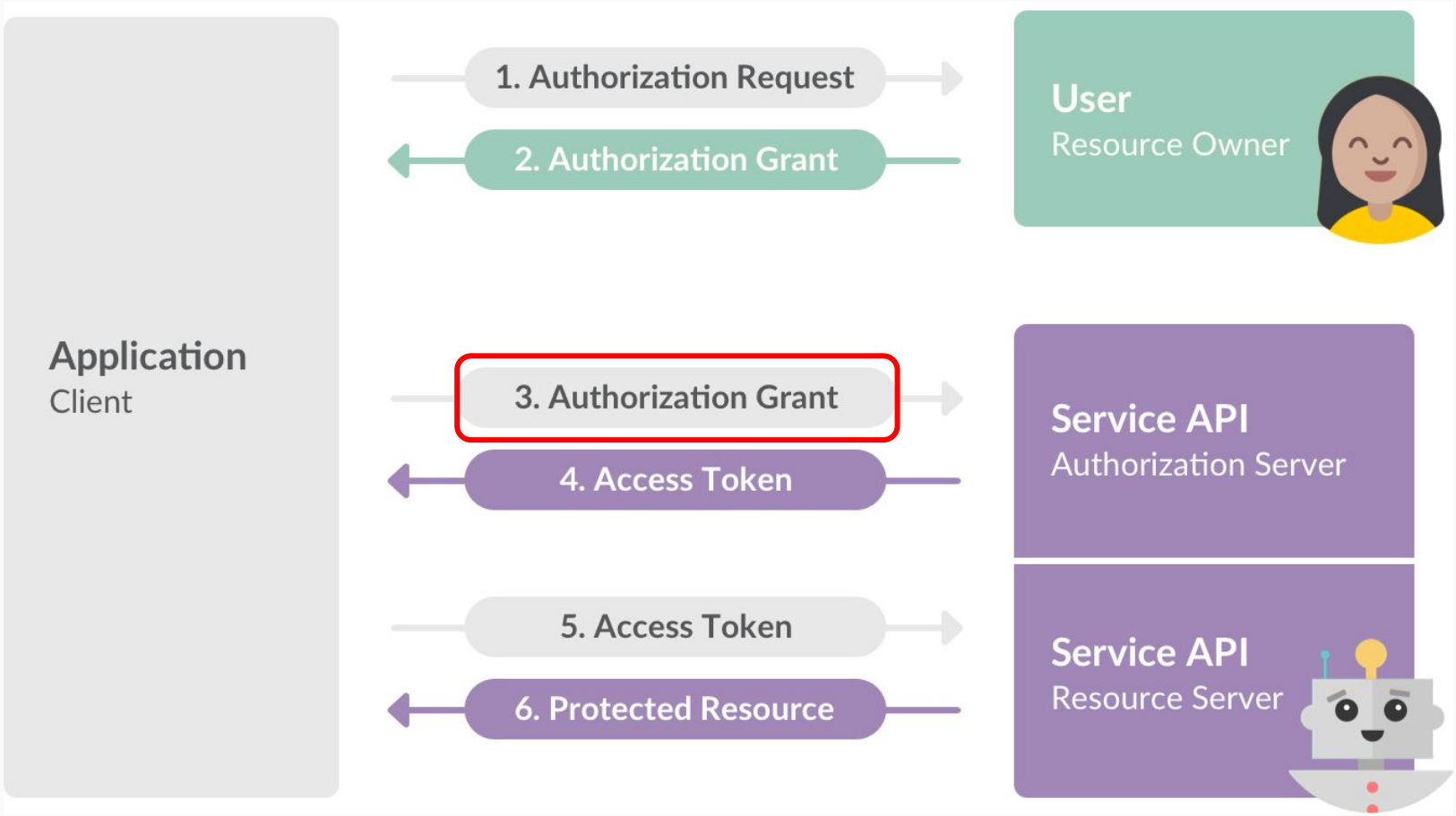
Authorization code flow

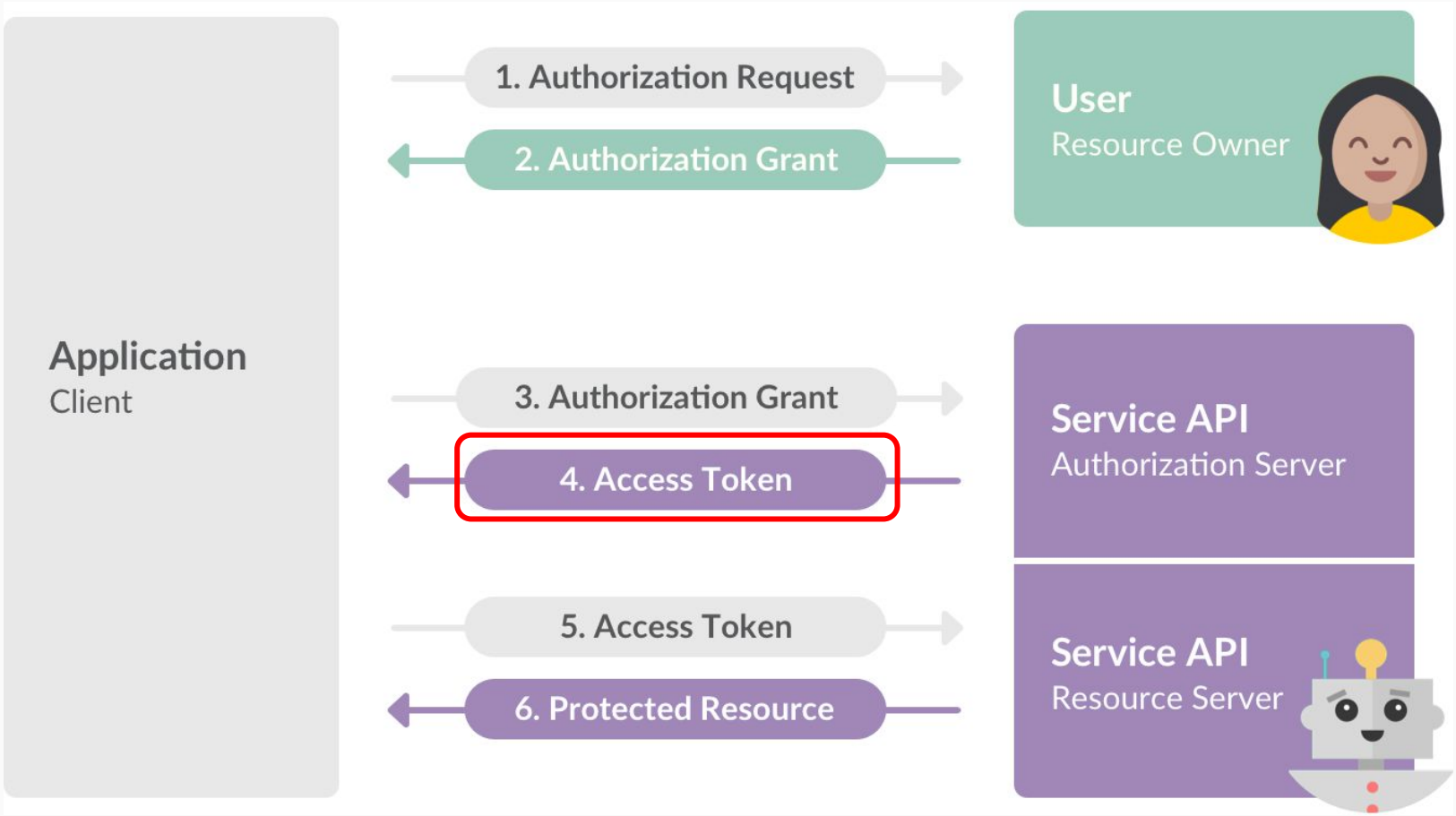
- El más completo, y por lo tanto el más seguro.
- Se utiliza con lo que se llaman confidential clients, que son aplicaciones que pueden guardar una contraseña (secreto).

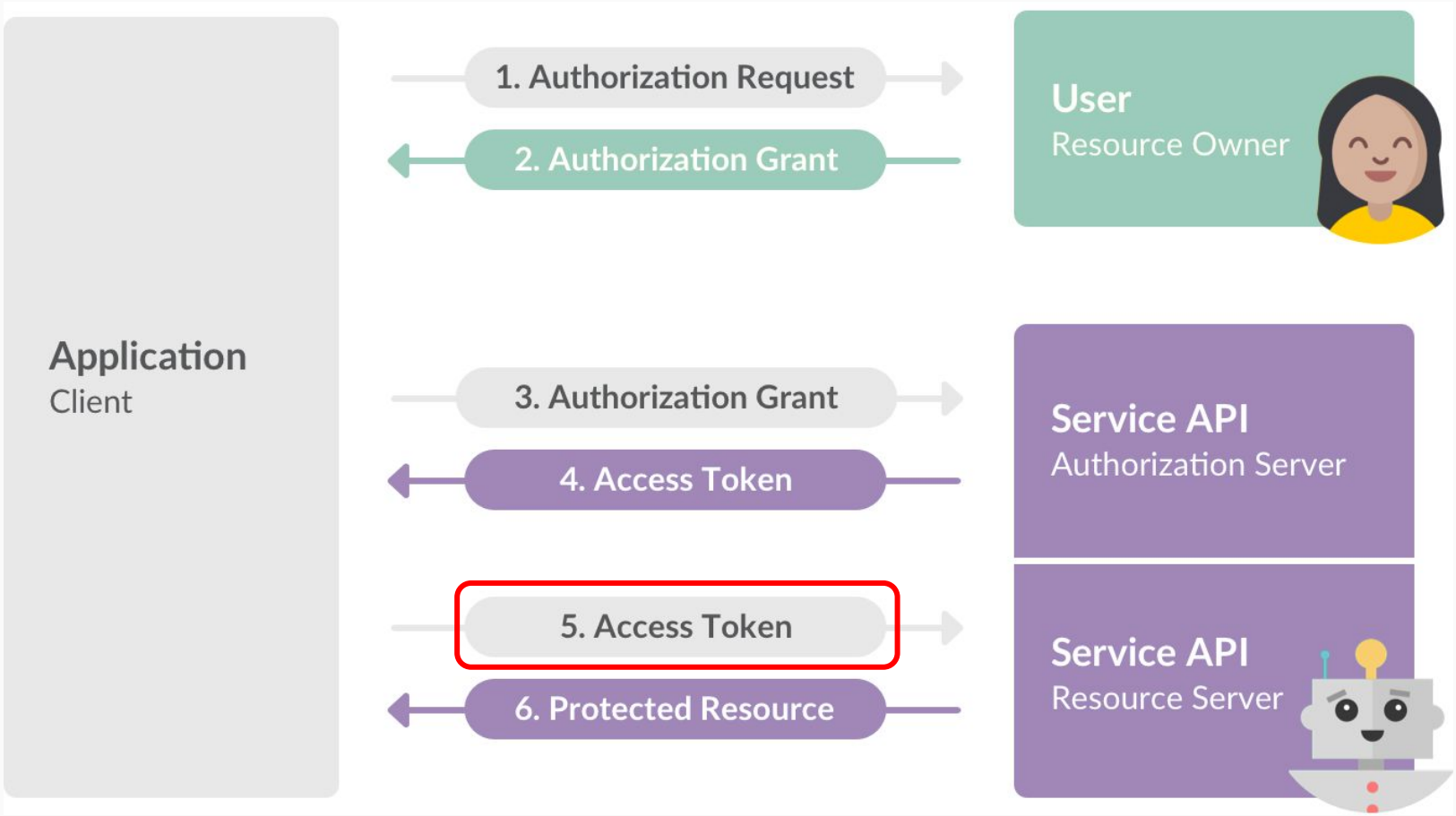

Application
Client
Sign in to your account
Email Address*
Enter your email address
Password*
Enter your password
Sign In
OR
Sign In with Google
Sign In with GitHub
Forgot Password?
1. Authorization Request
2. Authorization Grant
3. Authorization Grant
4. Access Token
5. Access Token
6. Protected Resource
User
Resource Owner
Service API
Authorization Server
Service API
Resource Server

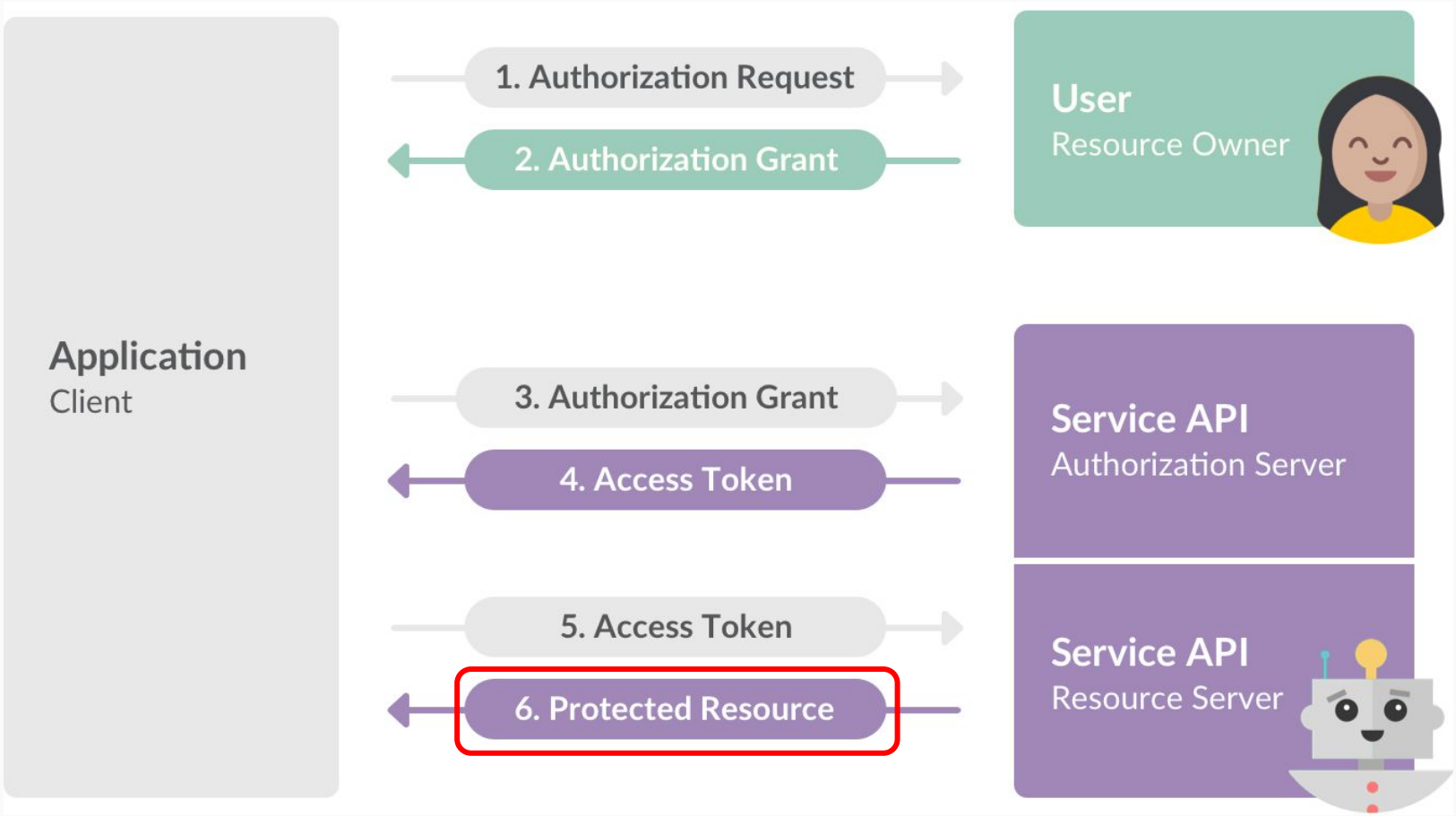

Application
Client
1. Authorization Request
Sign in with Google
Sign in
to continue to digitalocean.com
Email or phone
Forgot email?
To continue, Google will share your name, email address, language preference, and profile picture with digitalocean.com. Before using this app, you can review digitalocean.com's privacy policy and terms of service.
Create account
Next
User
Resource Owner
Service API
Authorization Server
Service API
Resource Server

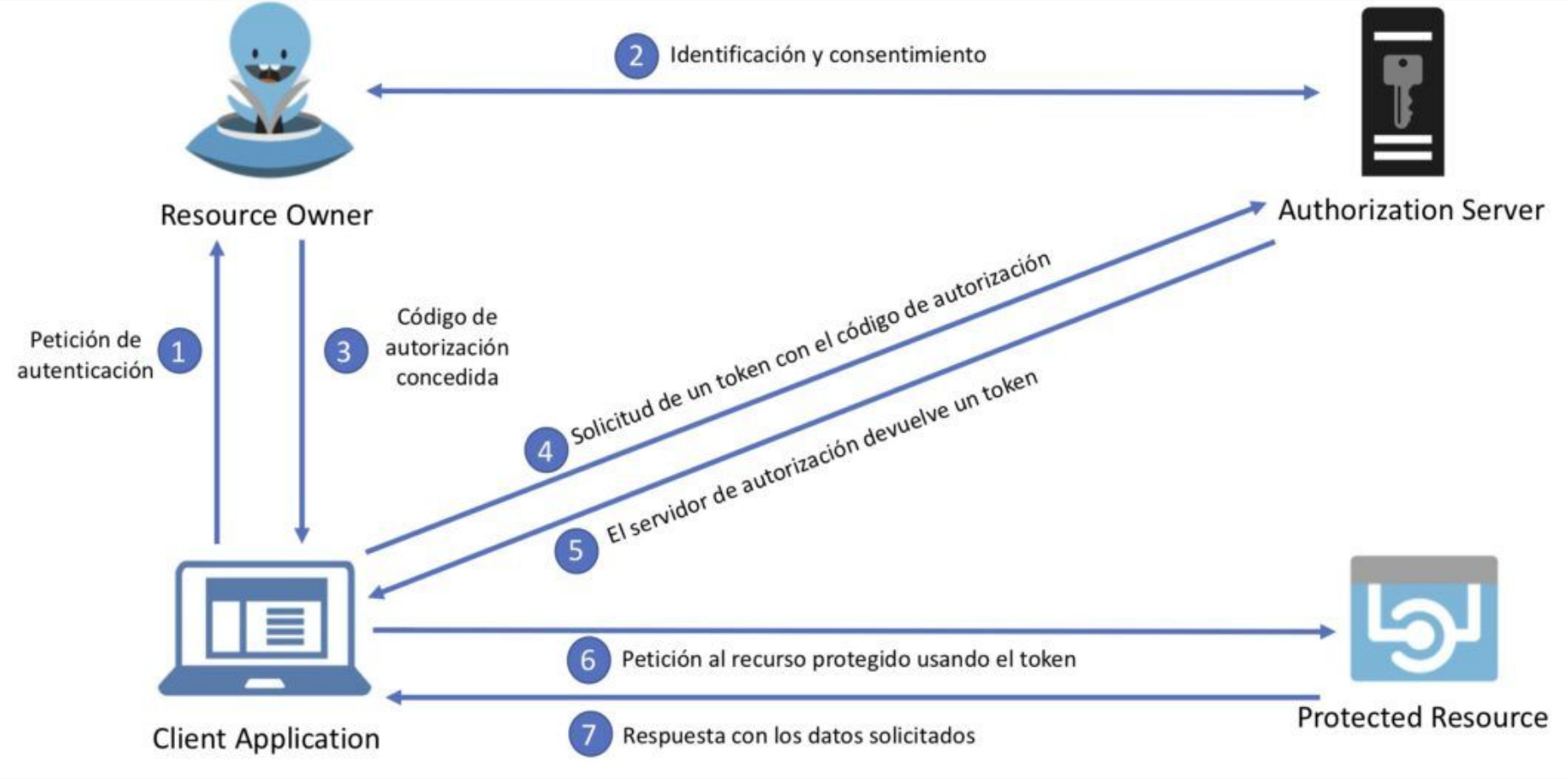

Application
Client
1. Authorization Request
2. Authorization Grant
User
Resource Owner
3. Authorization Grant
4. Access Token
Service API
Authorization Server
5. Access Token
6. Protected Resource
Service API
Resource Server


Application
Client
1. Authorization Request
2. Authorization Grant
User
Resource Owner
3. Authorization Grant
4. Access Token
Service API
Authorization Server
5. Access Token
6. Protected Resource
Service API
Resource Server


Application
Client
1. Authorization Request
2. Authorization Grant
User
Resource Owner
3. Authorization Grant
4. Access Token
Service API
Authorization Server
5. Access Token
6. Protected Resource
Service API
Resource Server


Application
Client
1. Authorization Request
2. Authorization Grant
User
Resource Owner
3. Authorization Grant
4. Access Token
Service API
Authorization Server
5. Access Token
6. Protected Resource
Service API
Resource Server

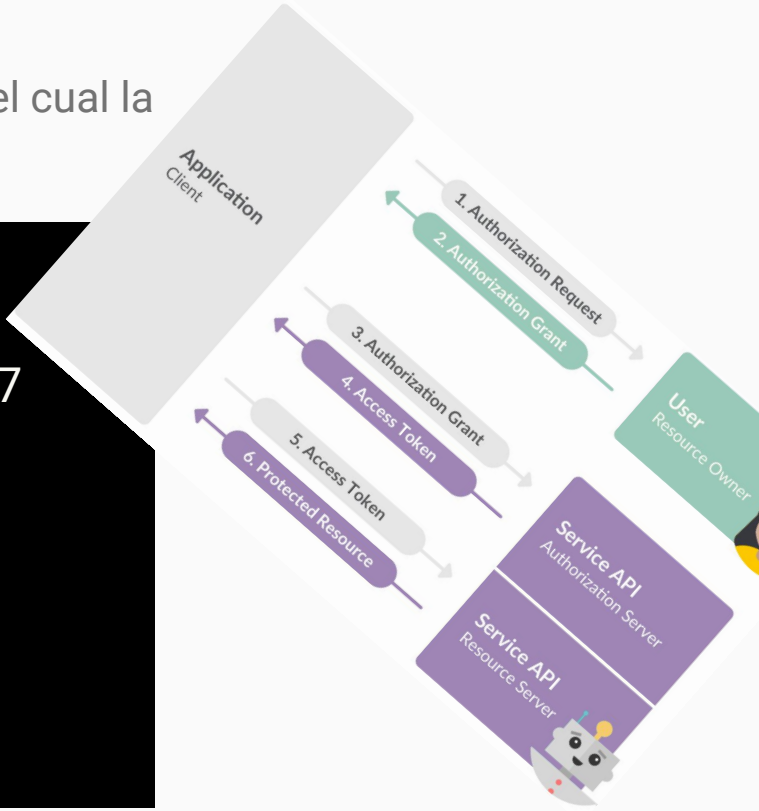

Application
Client
1. Authorization Request
2. Authorization Grant
User
Resource Owner
3. Authorization Grant
4. Access Token
Service API
Authorization Server
5. Access Token
6. Protected Resource
Service API
Resource Server



source: https://www.returngis.net/2019/04/oauth-2-0-openid-connect-y-json-web-tokens-jwt-que-es-que/

Primero se redirige al usuario al endpoint de autorización, el cual la aplicación cliente conoce, con una serie de parámetros:

```
https://authorization.server.com/authorize
?response_type=code
&client_id=213f6de8-f232-4854-8c20-80a9b385cca7
&redirect_uri=https://client.example.com/callback
&state=abc
&scope=returngis_api.read.api2.readAndWrite
```

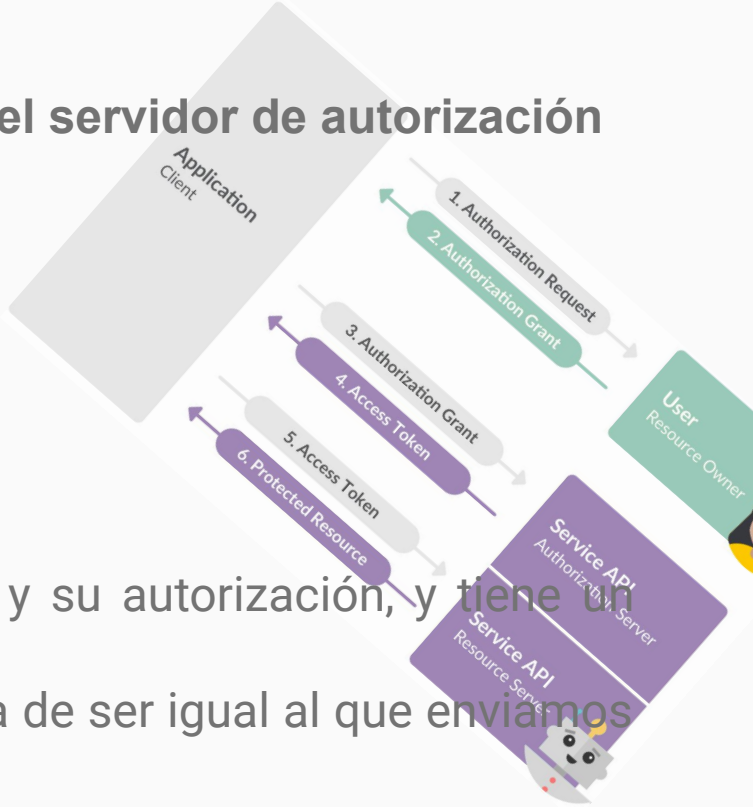


- **reponse_type**: este parár… de OAuth 2.0 vamos a seguir para recuperar el token, en este…
- **client_id**: se trata de un id… **strado** en el servidor de autorización.
- **redirect_uri**: cuando la au… mos especificar una URL de vuelta a nuestra aplicación. Adema… como parte del registro de nuestra aplicación cliente en el ser…
- **state**: es recomendado, p… rmar que la respuesta que recibimos por parte del servidor es … os que ningún malo ha cambiado la respuesta del servidor por …
- **scope**: se utiliza para dec… n». Son los **permisos** que se están pidiendo sobre nuestra AP… es a través de un espacio en blanco entre ellos.

- **reponse_type**: este parámetro es el que dice qué tipo de flujo de OAuth 2.0 vamos a seguir para recuperar el token, en este caso el valor debe ser **code**.
- **client_id**: se trata de un identificador de la aplicación cliente, **registrado** en el servidor de autorización. **redirect_uri**: cuando la autenticación del cliente finalice, necesitamos especificar una URL de vuelta a nuestra aplicación. Además, esta URL también **está guardada** como parte del registro de nuestra aplicación cliente en el servidor de autorización.
- **state**: es recomendado, pero no es obligatorio. Nos permite confirmar que la respuesta que recibimos por parte del servidor es lícita, de tal forma que nos aseguramos que ningún malo ha cambiado la respuesta del servidor por el camino.
- **scope**: se utiliza para decir el «para qué quiero esta autorización». Son los **permisos** que se están pidiendo sobre nuestra API. La forma de especificar varios scopes es a través de un espacio en blanco entre ellos.

Cuando el usuario se ha validado correctamente, el servidor de autorización responderá con lo siguiente:

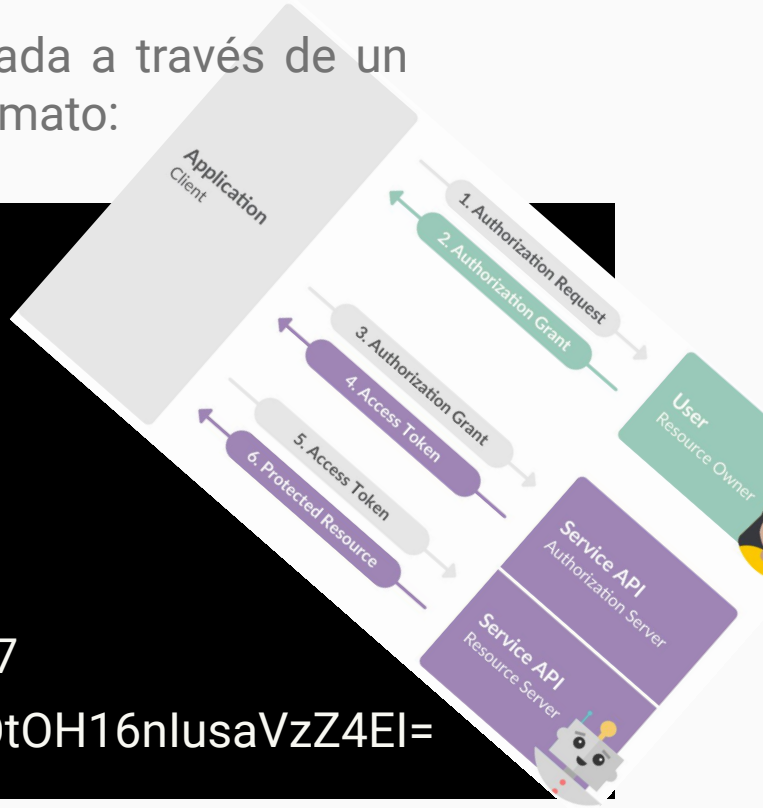
```
https://client.example.com/callback
?code=xxxxxxxxxxx
&state=abc
```



- Code representa el consentimiento del usuario y su autorización, y tiene un tiempo de vida bastante corto.
- Tenemos además el parámetro state que debería de ser igual al que enviamos en la primera petición.

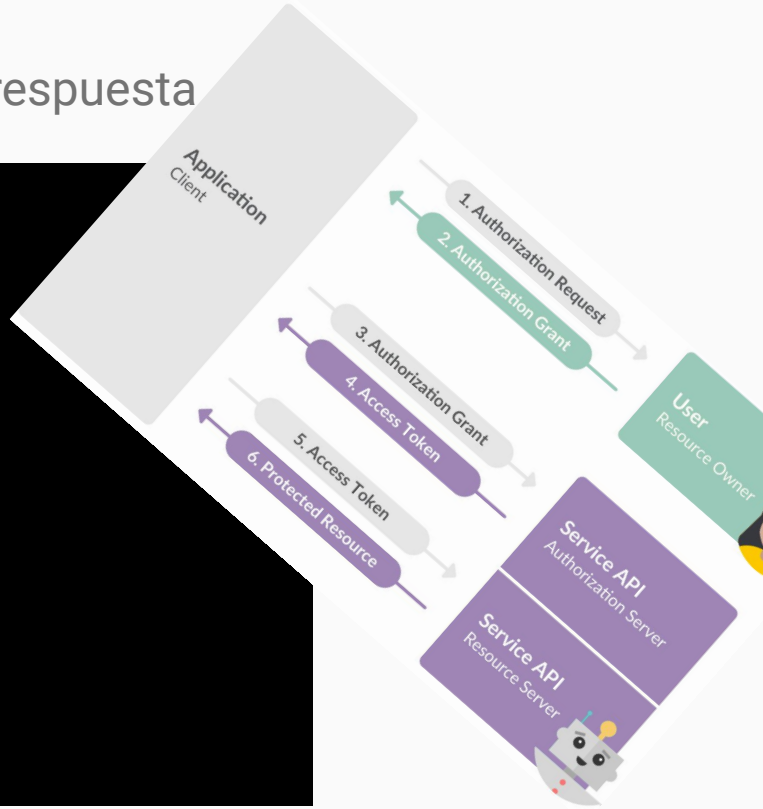
Con el código anterior la **aplicación** hará una llamada a través de un POST al servidor de autorización con el siguiente formato:

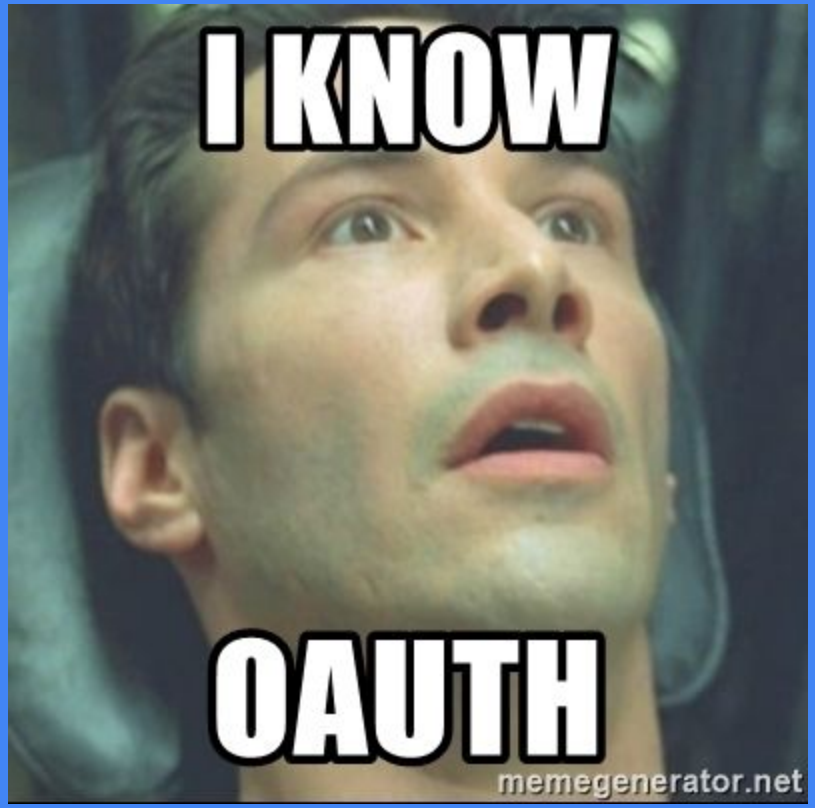
```
POST /token HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded

grant_type=authorization_code
&code=xxxxxxxxxxx
&redirect_uri=https://client.example.com/callback
&client_id=213f6de8-f232-4854-8c20-80a9b385cca7
&client_secrect=nH7TbHkgsjOWIAtb4NV78RQD5EOtOH16nIusaVzZ4EI=
```



Finalmente si todo va bien recibiremos la siguiente respuesta

```
HTTP/1.1 200 OK
Content-Type: application/json
{
   "access_token" : "una cadena muy larga",
   "token_type" : "Bearer",
   "expires_in" : 3600,
   "scope" : "returngis_api.read api2.readAndWrite"
  }
```




I KNOW
OAUTH
memegenerator.net

Playgrounds

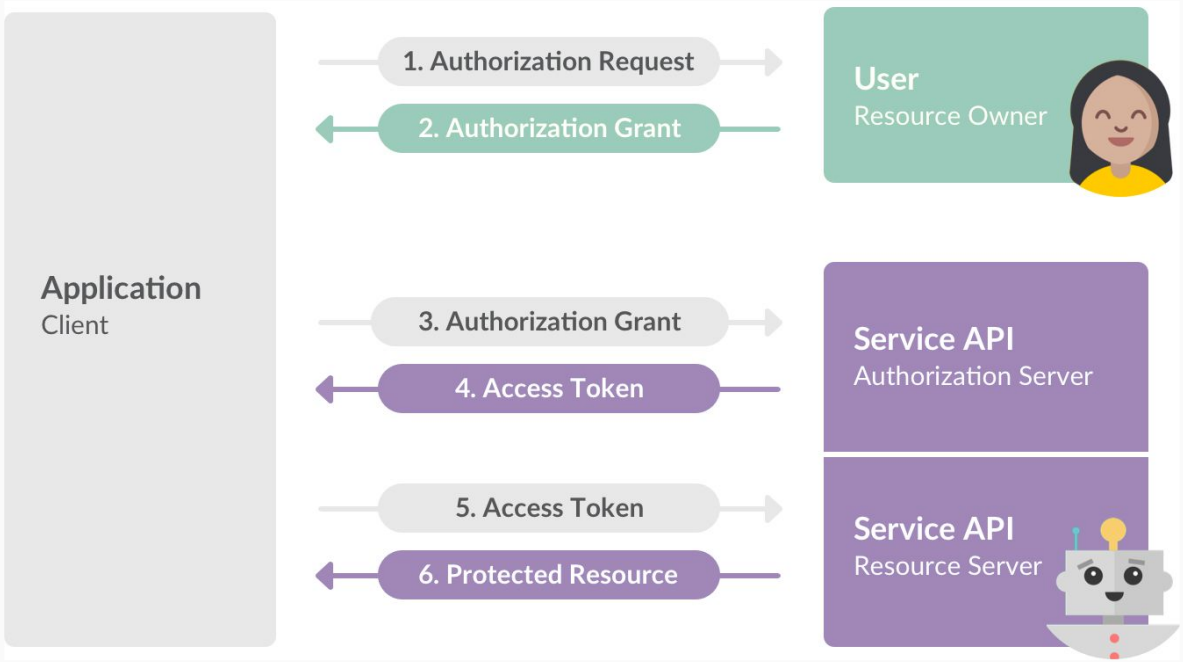
- https://developers.google.com/oauthplayground/
- https://www.oauth.com/playground/

Ataques en integraciones de OAuth 2.0

- Token/Code stealing
- CSRF (missing `state` parameter)
- Token impersonation
- Clickjacking

Token/Code stealing

1. Find the domain used in `redirect_uri`
2. Can you use `subdomains` in the `redirect_uri`?
3. Point the `redirect_uri` to a page.
    - open redirector to attacker's domain
    - xss which can be used in `redirec_uri` to pass `acces_token` to the attacker.
    - subdomain takeover (allowed subdomain in `redirect_uri`)
    - backtrack to a page which can be used to open (302)/xss
4. Use the stolen `access_token` to login.


Application
Client
1. Authorization Request
2. Authorization Grant
User
Resource Owner
3. Authorization Grant
4. Access Token
Service API
Authorization Server
5. Access Token
6. Protected Resource
Service API
Resource Server

CSRF

Objetivo

Conectar una cuenta de la víctima con una cuenta third party (i.e. Facebook) del atacante.

Hacer login con la cuenta third party en la cuenta de la víctima.

state=<ANTI_CSRF_TOKEN>

1. Check if `**state**` param in OAuth Authorization Link is validated?
2. Derive yourself a valid `**authorization_code**` link and do no use it.
3. Send this active `**authorization_code**` link to victim.
4. Your account will get connected with victim's account.
5. Login via your account.

1. Attacker initiates **`Connect Facebook`** flow

2. Attacker derives a valid **`authorization_code`** link and doesn't use it.

3. Victim is logged into example.com

4. Attacker send active **`authorization_code`** link to the victim account.

5. Attacker/s facebook account is connected to victim's account

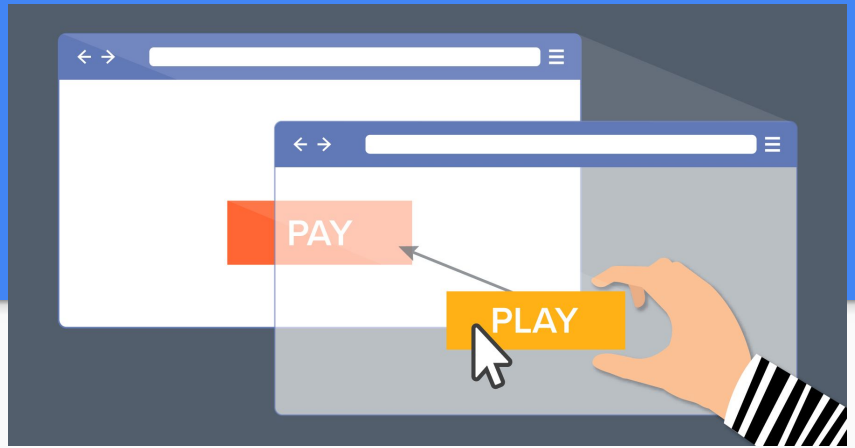
6. Login into victims account using **`Login with facebook`** (which is a facebook account owned by the attacker)

Token Impersonation

Objetivo

1. Usar un `access_token` de la víctima de cualquier otra third party app.
2. Usar el `access_token` que conseguimos en el paso anterior en otra third party app para lograr acceso desautorizado.

Clickjacking



Objetivo

1. Un sitio malicioso carga el sitio objetivo con un Iframe superpuesto a un conjunto de botones que están cuidadosamente puestos sobre botones importantes del sitio objetivo.
2. Cuando el usuario hace click en los botones visibles, está haciendo click en los botones en el sitio oculto.

Algunas contramedidas

- Registrar la URLs exactas que vamos a utilizar para nuestras callbacks.
- Deshabilitar el uso de iframes durante la autorización.
  - Para nuevos navegadores esto se puede forzar usando el header X-FRAME-OPTIONS.
  - Para navegadores más viejos se puede forzar mediante javascript pero podría no ser efectivos en todos ellos.
- Utilizar un fuerte parámetro `state` en el flujo y no olvidarse de validarlo correctamente.

- Understanding OAuth2 and Building a Basic Authorization Server of Your Own: A Beginner’s Guide
- Building a Basic Authorization Server using Authorization Code Flow
- OAuth 2.0, OpenID Connect y JSON Web Tokens (JWT) ¿Qué es qué?
- Diagrams And Movies Of All The OAuth 2.0 Flows
- Bypassing GitHub's OAuth flow
- The most common OAuth 2.0 Hacks
- The real impact of an Open Redirect vulnerability
- LevelUp 0x02 - Hacking OAuth 2.0 For Fun And Profit
- OAuth2.0 - RFC 6749
- Attacks OAuth2.0 - RFC 6819

- OAuth authentication bypass on Airbnb acquisition using 1-char Open Redirect
- Bypassing GitHub's OAuth flow
- Facebook CSRF bug which lead to Instagram Partial account takeover.
- Facebook OAuth Framework Vulnerability
- Stealing Facebook access_tokens using CSRF in device login flow
- Authentication bypass on Airbnb via OAuth tokens theft

PREGUNTAS?