CircleNight di luglio 2020

Guida completa di OAuth 2.0

STEFANO SLOBODIUK

Programma

Storia del protocollo.

Tutto su OAuth 2.0.

Dimostrazione.

OAuth è un protocollo di rete aperto e standard, progettato specificamente per lavorare con l'Hypertext Transfer Protocol (HTTP). Essenzialmente consente l'emissione di un token di accesso da parte di un server autorizzativo ad un client di terze parti, previa approvazione dell'utente proprietario della risorsa cui si intende accedere.

https://it.wikipedia.org/wiki/OAuth

Storia del protocollo.

Storia del protocollo

- OAuth è un protocollo di rete aperto e standard, lavora in HTTP
- Access delegation utilizzato nelle grandi compagnie (Amazon, Google, Facebook, Microsoft…)
- Tale protocollo permette l’autorizzazione di API di sicurezza con un metodo standard e semplice in varie situazioni (mobile, web, app per pc)
- Il protocollo OAuth 1.0 è stato pubblicato come RFC 5849 nell'aprile 2010.
- Un'evoluzione di OAuth 1.0, OAuth 2.0 è descritta nel documento RFC 6749 dell'ottobre 2012.
- L’idea di base è quella di autorizzare terze parti a gestire documenti privati senza condividere la password.
- Problemi in OAuth 1.0: sicurezza e phishing

Storia del protocollo

- Senza OAuth, il modo più semplice per permettere l'accesso a terze parti, è quello di fornire username e password 😱 Le ex-applicazioni dipendenti da Twitter lo facevano!
- Problema: salvataggio delle credenziali plaintext, completo accesso all'account (anche al cambio password 😈)
- Una volta capite le difficoltà, sono state realizzate delle architetture proprietarie (Google: AuthSub, Yahoo: BBAuth…) ognuna incompatibile con l'altra
- Nel 2007: gruppo OpenID coinvolge Google e AOL
- OAuth 2.0 è l'evoluzione degli errori commessi, mettendo maggiormente in chiaro alcuni aspetti del protocollo

Tutto su OAuth 2.0.

Tutto su OAuth 2.0.

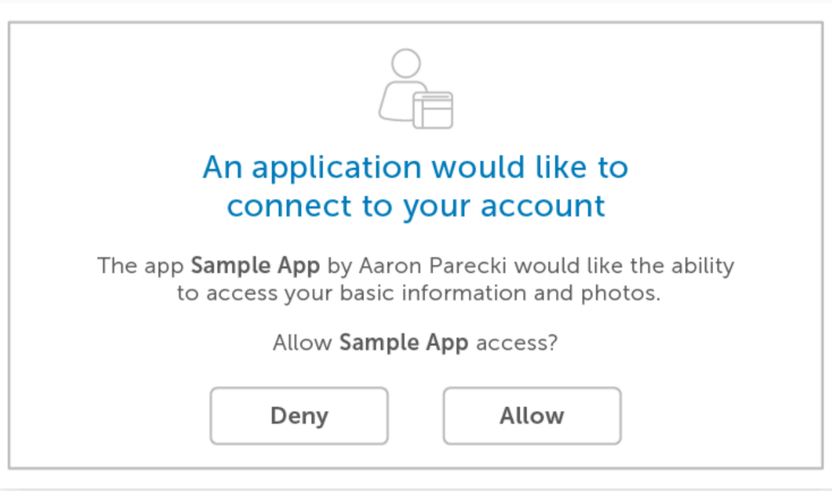
- I ruoli su OAuth sono:
  - The Third-Party Application: “Client"
  - The API: "Resource Server”
  - The Authorization Server
  - The User: "Resource Owner”

Tutto su OAuth 2.0.

- Creazione della app
  - Normalmente viene richiesto inserire il nome dell'applicazione, logo, privacy policy…
  - Inoltre viene richiesto di inserire il **Redirect URI**
    - Il Redirect URI aiuta a prevenire alcuni attacchi. Dovrebbe essere HTTPS, per evitare intercettazioni
  - **Client ID** e **Secret**: Il Secret è opzionale e viene fornito solo per alcuni **Grant**.
    - Il Client ID viene considerata informazione pubblica (può essere usato in JS). Il secret viene usato per es. da app lato server. Per js o app esistono delle soluzioni alternative

Tutto su OAuth 2.0.

- Autorizzazione (Authorization)
  - Il primo passaggio di OAuth è ottenere l'autorizzazione dall'utente.
  - Per app web o mobile, si intende la pagina di accesso
  - Grant types per ogni caso d'uso
    - **Authorization Code**: app web server, app mobile
    - **Password**: accesso first-party-app con username/password
    - **Client credentials**: accesso app diretto (no utente)
    - ~~**Implicit**: per client senza secret,~~ superato da **Authorization Code con PKCE**

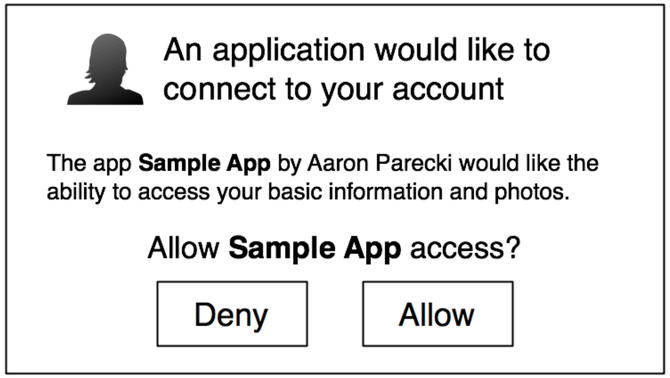


Tutto su OAuth 2.0.

- **App web server**
  - Scritto in server side, il secret è inaccessibile per l'utente finale.
  - Authorization: tasto login con
    https://authorization-server.com/auth?response_type=code&client_id=CLIENT_ID&redirect_uri=REDIRECT_URI&scope=photos&state=1234zyx
    - **response_type=code**: richiedo un authorization code
    - **client_id**: Il client_id dell'applicazione
    - **redirect_uri**: L'URI di redirect dopo l'authorization
    - **scope**: eventuali richieste di accesso ad API
    - **state**: stringa random generata da client

Tutto su OAuth 2.0.

- **App web server 2**
  - Utente reindirizzato al Authorization Server
  - Se permette l'accesso viene rediretto a:
    https://example-app.com/cb?code=AUTH_CODE_HERE&state=1234zyx
    - **code**: l'authorization_code fornito dal server
    - **state**: il valore random generato prima
  - Ora il client verificherà se **state** corrisponde.



Tutto su OAuth 2.0.

- **App web server 3**
  - Ottenere Access Token
    - Il client richiederà via POST
      POST https://api.authorization-server.com/token
      grant_type=authorization_code&
      code=AUTH_CODE_HERE&
      redirect_uri=REDIRECT_URI&
      client_id=CLIENT_ID&
      client_secret=CLIENT_SECRET
      - **grant_type=authorization_code**: Il tipo di Grant usufruito è di tipo authorization_code
      - **code=AUTH_CODE_HERE**: codice ricevuto precedentemente
      - **redirect_uri=REDIRECT_URI**: Identico al URI fornito precedentemente
      - **client_id=CLIENT_ID**: il client_id dell'applicazione
      - **client_secret=CLIENT_SECRET:** siccome stiamo parlando di un app server-side, viene fornito anche il secret

Tutto su OAuth 2.0.

- **App web server 4**
  - Risposta dal Server:
    - {
  "access_token":"RsT5OjbzRn430zqMLgV3Ia",
  "expires_in":3600
}

Tutto su OAuth 2.0.

- **Single-Page Apps**
  - Sono applicazioni che hanno vita interamente nel browser. Siccome il codice sorgente è visibile, il secret non può essere utilizzato.
  - Utilizzeremo l'estensione PKCE.
  - *Precedentemente si utilizzava l'accesso implicit:*
    It was originally designed to protect mobile apps, but its ability to prevent authorization code injection makes it useful for every OAuth client, even web apps that use a client secret. (*https://oauth.net/2/pkce/*)

Tutto su OAuth 2.0.

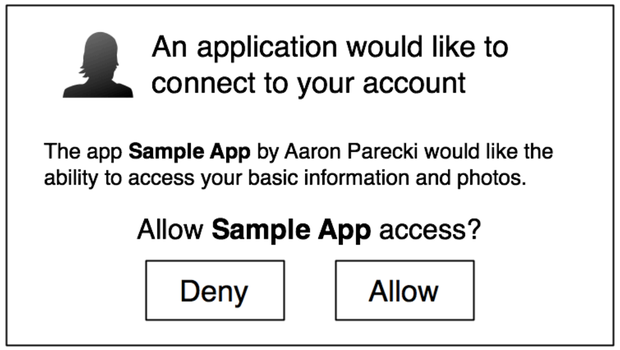
- **Single-Page Apps 2**
  - Authorization:
    https://authorization-server.com/auth?
    response_type=code&client_id=CLIENT_ID&redirect_uri=REDIRECT_URI&scope=photos&state=1234zyx&code_challenge=CODE_CHALLENGE&code_challenge_method=S256
    - **response_type=code**: Richiedo un authorization_code al server
    - **client_id**: il client ID
    - **redirect_uri**: URI di reindirizzamento
    - **scope**: eventuali scope
    - **state**: la stringa generata casualmente
    - **code_challenge**: base64(SHA256(altra stringa casuale))
    - **code_challenge_method=S256**: metodo di hashing usato

VERRANNO CREATE 2 VARIABILI:

CODE_VERIFIER= RANDOM STRING 43-128 CHARS
CODE_CHALLENGE= BASE64(SHA256(CODE_VERIFIER))

Tutto su OAuth 2.0.

- **Single-Page Apps 3**
  - Una volta premuto Allow:
    https://example-app.com/cb?code=AUTH_CODE_HERE&state=1234zyx
    - **code**: authorization_code fornito dal server
    - **state**: valore state precedentemente fornito
  - Verifico la correttezza di state.



Tutto su OAuth 2.0.

- **Single-Page Apps 4**
  - Ottenere access token:
    POST https://api.authorization-server.com/token
    grant_type=authorization_code&
    code=AUTH_CODE_HERE&
    redirect_uri=REDIRECT_URI&
    client_id=CLIENT_ID&
    code_verifier=CODE_VERIFIER
    - **grant_type=authorization_code**: dichiaro di voler utilizzare questo grant
    - **code=AUTH_CODE_HERE**: L'authorization_code ricevuto
    - **redirect_uri=REDIRECT_URI**: Lo stesso URI fornito all'inizio
    - **client_id=CLIENT_ID**: Il client ID
    - **code_verifier=CODE_VERIFIER**: Il codice creato precedentemente
  - L'authorization server eseguirà l'hash del code_verifier e lo verificherà col challenge_code inviato prima.

Tutto su OAuth 2.0.

- **<u>Mobile Apps</u>**
  - Sfatiamo un mito: da un'app "compilata" riesco ad estrapolare il codice! Pertanto il "secret" non è più tanto segreto.
  - *Precedentemente le guide ufficiali consigliavano di usare il grant Implict, ora superato da…*
  - PKCE flow, di nuovo!

Tutto su OAuth 2.0.

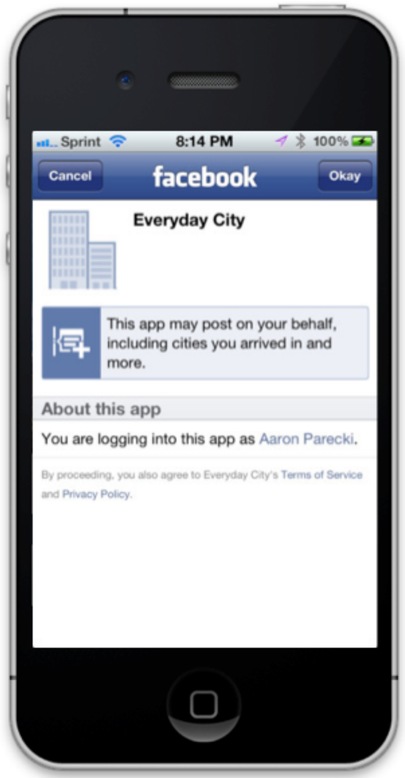
- **Mobile Apps 2**
  - Authorization (con URI personalizzata, ipotizzando che l'app ufficiale Facebook sia installata):
    fbauth2://authorize?response_type=code&client_id=CLIENT_ID&redirect_uri=REDIRECT_URI&scope=email&state=1234zyx
    - **response_type=code**: Richiedo un authorization_code al server
    - **client_id=CLIENT_ID**: Il client ID
    - **redirect_uri=REDIRECT_URI**: URI di redirect, ad es. fb00000000://authorize
    - **scope=email**: Gli scopes
    - **state=1234zyx**: Stringa random generata dall'applicazione
  - Se il Server supporta PKCE (ed è fortemente consigliato), aggiungere i seguenti parametri:
    - **code_challenge=XXXXXXX**: Il code_challenge generato
    - **code_challenge_method=S256**: Metodo di hashing usato

Tutto su OAuth 2.0.

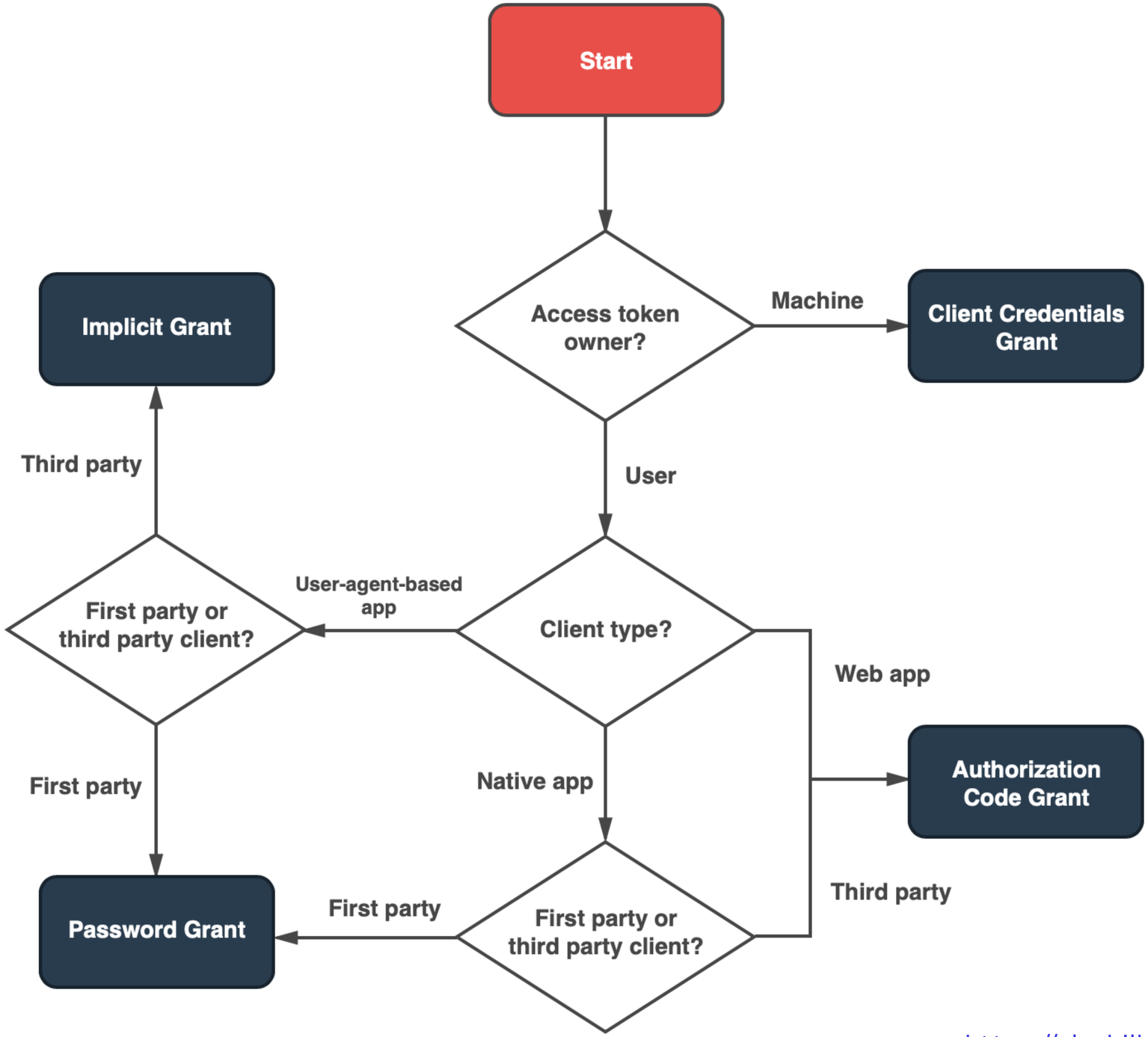
- **Mobile Apps 3**
  - Se il dispositivo non ha la app nativa, bisogna ripiegare su browser.
  - Le direttive sconsigliano di utilizzare un *embedded browser view*, dato che l'utente non può verificare se sta inserendo dati nel sito reale o phishing.
  - Su iOS è possibile usare "SafariViewController": lancia un browser all'interno dell'app, mostra la Address bar e condivide gli stessi cookie dell'originale browser di Safari.
  - https://facebook.com/dialog/oauth?response_type=code&client_id=CLIENT_ID&redirect_uri=REDIRECT_URI&scope=email&state=1234zyx
  - I parametri sono sempre gli stessi, in più se si fa uso del PKCE flow, aggiungere i dati mancanti.

Tutto su OAuth 2.0.

- **Mobile Apps 4**
  - Ottenere Access token
    - Una volta premuto Approve, si verrà reindirizzati a:
      fb00000000://authorize?code=AUTHORIZATION_CODE&state=1234zyx
    - L'applicazione mobile verificherà lo state e può procedere a scambiare l'authorization_code per un access_token.
    - POST https://api.authorization-server.com/token
      grant_type=authorization_code&
      code=AUTH_CODE_HERE&
      redirect_uri=REDIRECT_URI&
      client_id=CLIENT_ID&
      code_verifier=VERIFIER_STRING
    - I parametri sono gli stessi che abbiamo analizzato qualche slide fa.



OBSOLETO



https://alexbilbie.com/guide-to-oauth-2-grants/

Tutto su OAuth 2.0.

- **Password grant**
  - Utilizzato per scambiare la coppia username/password con un access_token
  - Va necessariamente usato nelle first-party app.
  - POST https://api.authorization-server.com/token
    grant_type=password&
    username=USERNAME&
    password=PASSWORD&
    client_id=CLIENT_ID
    - **grant_type=password**: Richiedo il grant Password
    - **username=USERNAME**: il nome utente
    - **password=PASSWORD**: la password dell'utente
    - **client_id=CLIENT_ID**: Il client ID

Circleton App

Project overview

Repository

Files

Commits

Branch

Tags

Collaboratori

Graph

Confronta

Issues 0

Richieste di merge 0

CI / CD

Operations

Packages & Registries

Analytics

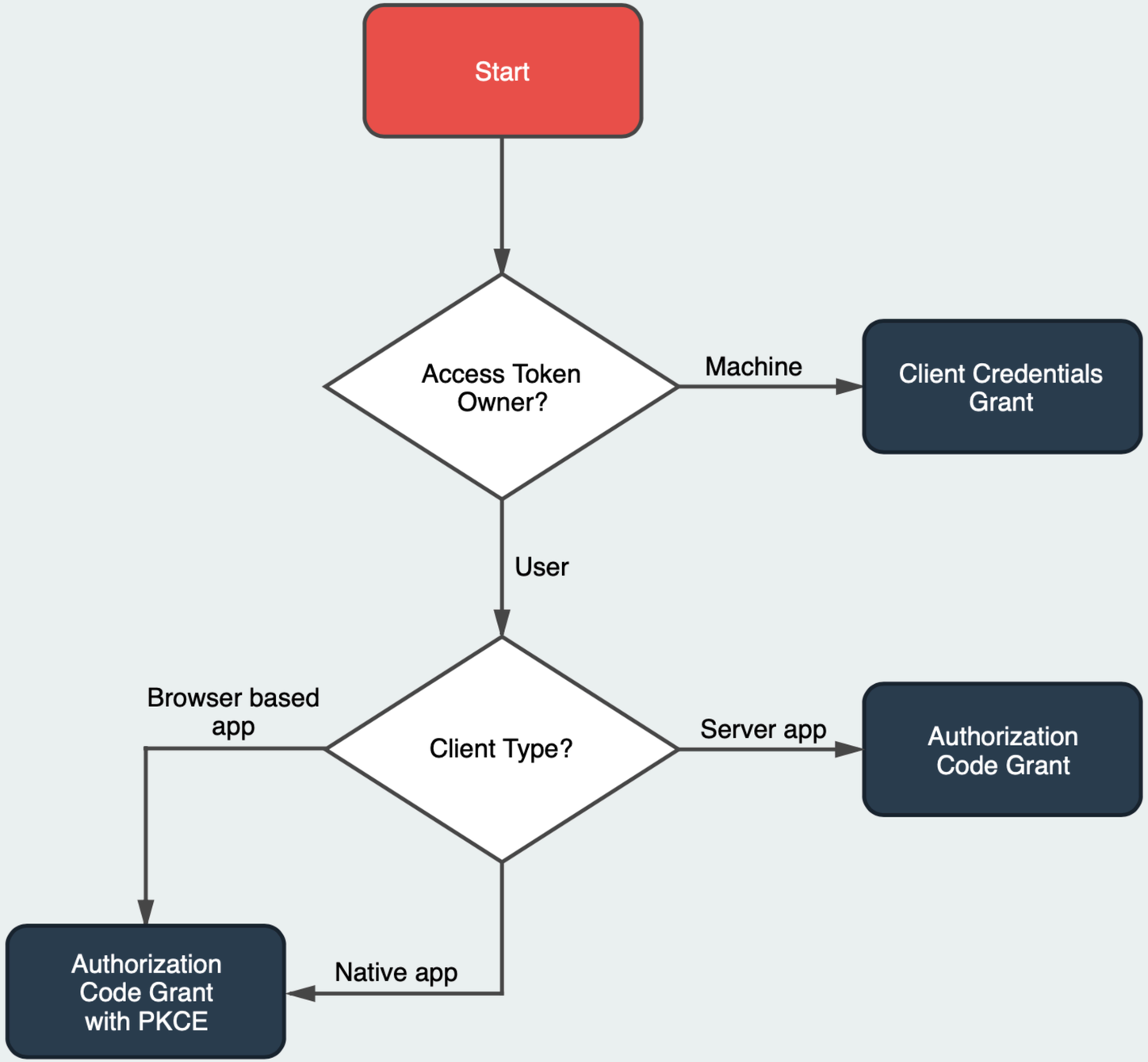
Membri

```
64      public void login(Button buttonLogin, EditText userEditText, EditText passwordEditText)
65      {
66          this.buttonLogin = buttonLogin;
67          this.userEditText = userEditText;
68          this.passwordEditText = passwordEditText;
69
70          disableButtonLogin();
71
72          JSONObject parameters = new JSONObject();
73          try
74          {
75              parameters.put("grant_type", "password");
76              parameters.put("client_id", BuildConfig.CLIENT_ID);
77              parameters.put("client_secret", BuildConfig.CLIENT_SECRET);
78              parameters.put("username", userEditText.getText().toString().trim());
79              parameters.put("password", passwordEditText.getText().toString());
80          }
81          catch (Exception e)
82          {
83              e.printStackTrace();
84          }
85          RequestQueue requestQueue = Volley.newRequestQueue(getContext());
86          JsonObjectRequestRestRaw jsonObjectRequestRestRaw = new JsonObjectRequestRestRaw(Request.Method.POST, ApiRest.OAUTH_T
87          requestQueue.add(jsonObjectRequestRestRaw);
88      }
89
90      private void disableButtonLogin()
91      {
92          buttonLogin.setEnabled(false);
93          userEditText.setEnabled(false);
94          passwordEditText.setEnabled(false);
95
96          if(android.os.Build.VERSION.SDK_INT < android.os.Build.VERSION_CODES.JELLY_BEAN)
97          {
98              buttonLogin.setBackgroundDrawable(ContextCompat.getDrawable(getContext(), R.drawable.form_button_default_loading)
99          }
100         else
101         {
102             buttonLogin.setBackground(ContextCompat.getDrawable(getContext(), R.drawable.form_button_default_loading));
```

https://gitlab.marstefo.ovh/circleton/app/-/blob/master/app/src/main/java/com/circleton/controller/Login/LoginService.java#L75

Tutto su OAuth 2.0.

- **Client credentials grant**
  - Si tratta di un accesso non legato ad un utente, ad esempio per ottenere le statistiche utenti.
  - POST https://api.authorization-server.com/token
    grant_type=client_credentials&
    client_id=CLIENT_ID&
    client_secret=CLIENT_SECRET
    - La risposta sarà un access token come visto in precedenza.



https://oauth2.thephpleague.com/authorization-server/which-grant/

Tutto su OAuth 2.0.

- **Eseguire richieste autenticate**
  - curl -H "Authorization: Bearer RsT5OjbzRn430zqMLgV3la" \
    https://api.authorization-server.com/1/me
  - Da fare via HTTPS (senza mai ignorare i certificati non validi), essendo infatti l’unico modo per non essere mai intercettati o altrimenti vanificare il circuito di sicurezza realizzato.

Tutto su OAuth 2.0.

- https://aaronparecki.com/oauth-2-simplified/
- https://www.oauth.com

Dimostrazione.

CircleNight di luglio 2020

Grazie per l’attenzione.

Guida completa di OAuth 2.0

STEFANO SLOBODIUK