Web identity: OAuth2 and OpenIDConnect

Brendan McCollam
University of Chicago / Globus

About me

- Originally from Chicago, now outside London
- Python developer at Globus

THE UNIVERSITY OF CHICAGO

Argonne NATIONAL LABORATORY

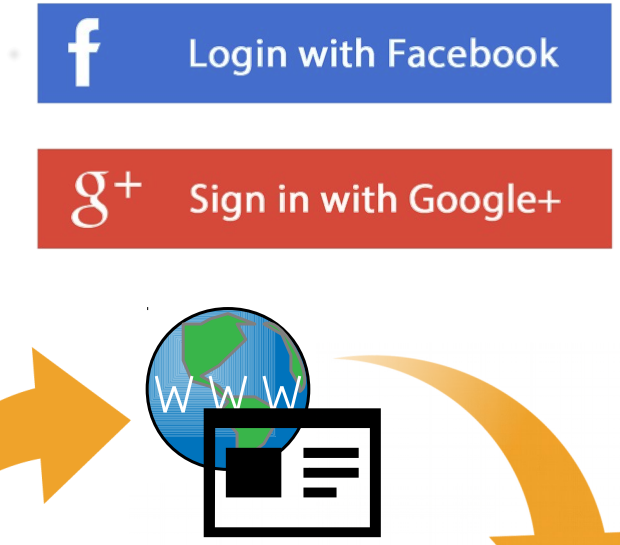
- Currently working on authentication platform using Oauth2 and OpenIDConnect

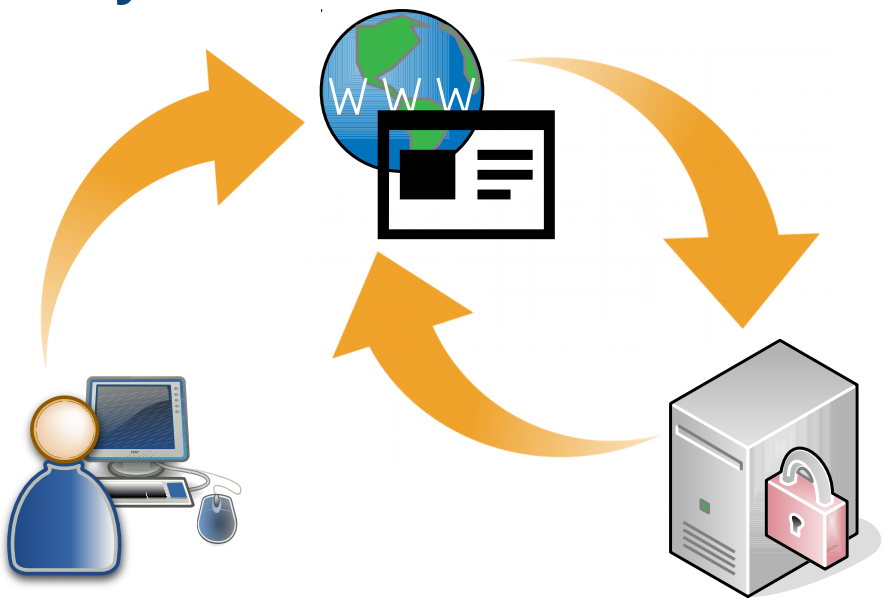
User:quintheislander / Pixabay / CC0

© User:Colin / Wikimedia Commons / CC BY-SA-4.0

What are we talking about?

- "Log in with…"
- Single sign on
- Federated Identity
- Web Identity





Why would you want that?

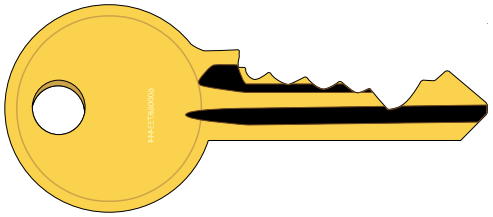
- Convenience
- Platform
- Security
- Sheer laziness

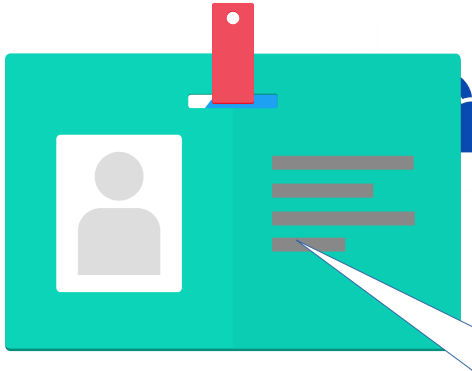
Terminology

- Authorization Protocols
  - OAuth 1.0
  - OAuth 2.0
- Authentication Protocols
  - OpenID
  - OpenID 2.0
  - OpenIDConnect

Not backward-compatible

Extends





- Name
- Department
- Expiration

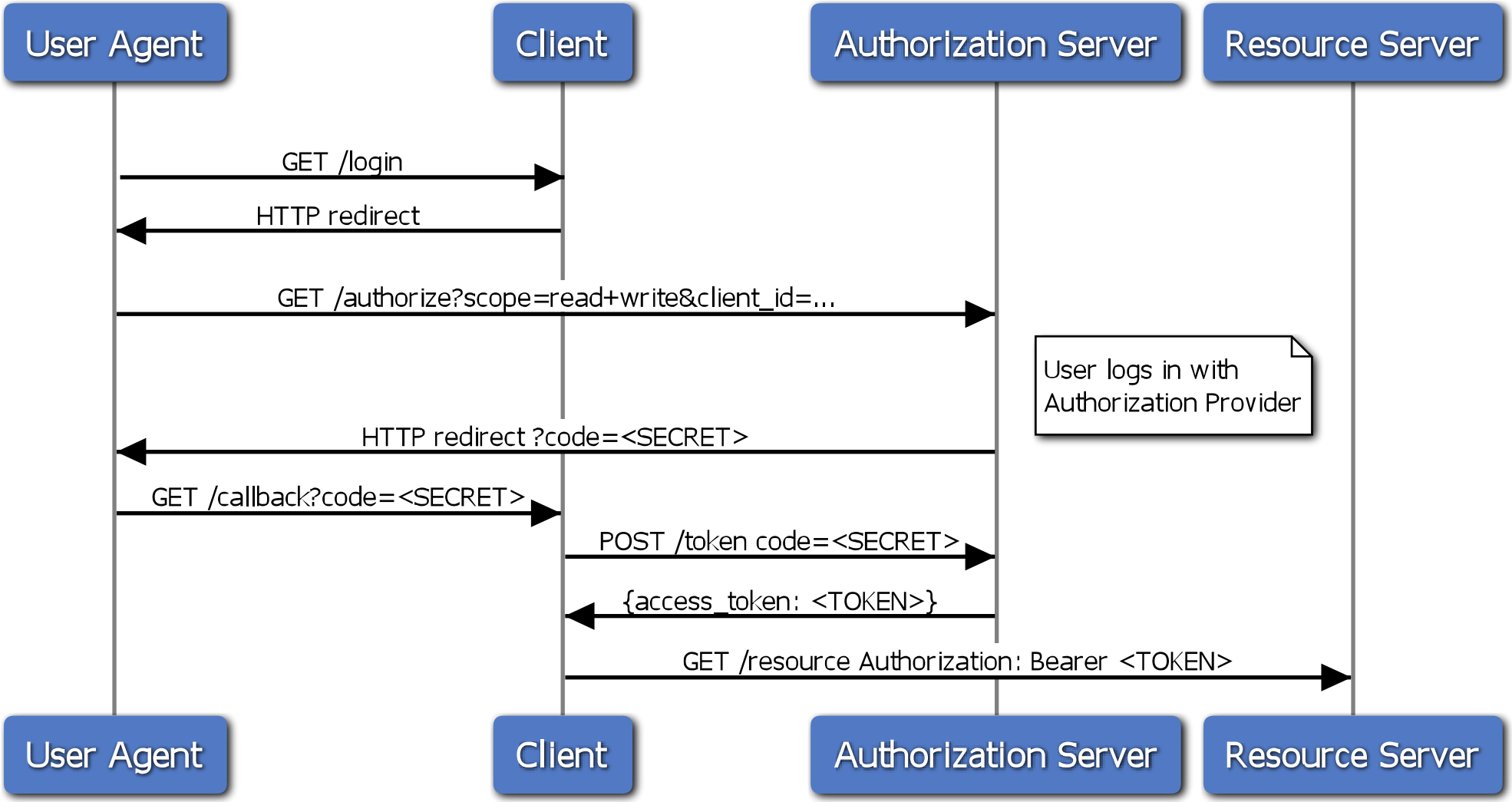
Google Calendar

OneDrive

OAuth 2.0

- How do you let users log in without seeing their passwords?
- Defined in RFC 6749
  - access_token: our "key"
  - scope: "what the key unlocks"
  - A "framework", not a specification
- Widespread adoption
- Python libraries

Authorization Code Grant



Other stuff in OAuth2

- Refresh tokens for long-term access
- Implicit grant for "public" client
- Client credentials grant
- Username/password grant (legacy)
- OAuth2 extensions
  - SAML Profile for OAuth 2.0 Client Authentication (RFC 7522)
  - OpenID Connect

OpenID Connect

- Set of extensions to OAuth2
- Adds the missing authentication layer
- Enabled via special "openid" scope
- Returns id_token
  - JSON Web Token (JWT)
  - contains "claims"
    - given_name
    - email
    - address
    - Whatever else the Authentication Server feels like

Password login

```
from flask import (Flask, redirect, url_for,
    request, session, render_template)
from flask_sqlalchemy import SQLAlchemy
from passlib.hash import bcrypt
```

```
app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = <DB_URI>
app.secret_key = "keep this secret"
db = SQLAlchemy(app)

class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String, unique=True)
    password_hash = db.Column(db.String)
```

```
@app.route('/')
def index():
    if 'authed_user' not in session:
        return redirect(url_for('login_get'))
    return render_template('index.html.jinja2')

@app.route('/login', methods=['GET'])
def login_get():
    return render_template('login.html.jinja2')
```

```
@app.route('/login', methods=['POST'])
def login_post():
    un = request.form['username']
    user = (User.query
                .filter_by(username=un)
                .one())
    pw = request.form['password']
    if bcrypt.verify(pw, user.password_hash):
        session['authed_user'] = user.username
        return redirect(url_for('index'))
    else:
        return "Login error."
```

Password login

```
@app.route('/login', methods=['POST'])
def login_post():
    un = request.form['username']
    user = (User.query
            .filter_by(username=un)
            .one())
    pw = request.form['password']
    if bcrypt.verify(pw, user.password_hash):
        session['authed_user'] = user.username
        return redirect(url_for('index'))
    else:
        return "Login error."
```

Replacing with OAuth2

```
from flask import Flask, request, redirect, session, url_for
from requests_oauthlib import OAuth2Session

app = Flask(__name__)

CONFIG = {
    'client_id': "<your client key>",
    'client_secret': "<your client secret>",
    'auth_url': 'https://auth.globus.org/v2/oauth2/authorize',
    'token_url': 'https://auth.globus.org/v2/oauth2/token',
    'scope':['urn:globus:auth:scope:ap
    'redirect_uri': 'https://myapplicat
}
```

High-level client library built on oauthlib & requests
github.com/requests/requests-oauthlib

You'll get these when you register your client application

This is your application's callback url that will recieve the authorization code

You can get these values from your Authorization Provider.
In this example, we're using Globus Auth.

Replacing with OAuth2

```
@app.route('/login', methods=['GET'])
def login():
    provider = OAuth2Session(
                client_id=CONFIG['client_id'],
                scope=CONFIG['scope'],
                redirect_uri=CONFIG['redirect_uri'])
    url, state = provider.authorization_url(CONFIG['auth_url'])
    session['oauth2_state'] = state
    return redirect(url)
```

Replacing with OAuth2

```
@app.route('/callback', methods=['GET'])
def callback():
    provider = OAuth2Session(CONFIG['client_id'],
                             redirect_uri=CONFIG['redirect_uri'],
                             state=session['oauth2_state'])
    token_response = provider.fetch_token(
                         token_url=CONFIG['token_url'],
                         client_secret=CONFIG['client_secret'],
                         authorization_response=request.url)

    session['access_token'] = token_response['access_token']
    session['access_token_expires'] = token_response['expires_at']

    api_url = 'https://transfer.api.globusonline.org/v0.10/task_list'
    transfers = provider.get(api_url)

    return redirect(url_for('index'))
```

Replacing with OAuth2

```
@app.route('/')
def index():
    if 'access_token' not in session:
        return redirect(url_for('login'))

    headers = {'Authorization': 'Bearer ' + session['access_token']})
    api_url = 'https://transfer.api.globusonline.org/v0.10/task_list'
    transfers = requests.get(api_url, headers=headers)

    return render_template('index.html.jinja2',
                           transfers=transfers.json())
```

Adding OpenIDConnect

```
from random import SystemRandom

import requests
from jose import jwt

random = SystemRandom()

keys = requests.get('https://auth.globus.org/jwk.json').json()

OIDC_CONFIG = {
    'jwt_pubkeys': keys,
    'scope': ['openid', 'email', 'profile'],
    'expected_issuer': 'https://auth.globus.org',
    'algorithm': 'RS512'
}

CONFIG.update(OIDC_CONFIG)
```

Adding OpenIDConnect

```
@app.route('/login', methods=['GET'])
def login():
    provider = OAuth2Session(client_id=CONFIG['client_id'],
                             scope=CONFIG['scope'],
                             redirect_uri=CONFIG['redirect_uri'])
    nonce = str(random.randint(0, 1e10))

    url, state = provider.authorization_url(CONFIG['auth_url'],
                                             nonce=nonce)
    session['oauth2_state'] = state
    session['nonce'] = nonce
    return redirect(url)
```

Adding OpenIDConnect

```
@app.route('/callback', methods=['GET'])
def callback():
    provider = Oauth2Session(CONFIG['client_id'],
                             redirect_uri=CONFIG['redirect_uri'],
                             state=session['oauth2_state'])
    response = provider.fetch_token(...)
    session['access_token'] = response['access_token']
    id_token = response['id_token']
    claims = jwt.decode(id_token,
                        key=CONFIG['jwt_pubkeys'],
                        issuer=CONFIG['expected_issuer'],
                        audience=CONFIG['client_id'],
                        algorithms=CONFIG['algorithm'],
                        access_token=response['access_token'])
    assert session['nonce'] == claims['nonce']
    session['user_id'] = claims['sub']
    session['user_email'] = claims['email']
    session['user_name'] = claims['name']
    return redirect(url_for('index'))
```

Adding OpenIDConnect

```
def callback():
    … # Same as Oauth2
    id_token = response['id_token']
    claims = jwt.decode(id_token,
                        key=CONFIG['jwt_pubkeys'],
                        issuer=CONFIG['expected_issuer'],
                        audience=CONFIG['client_id'],
                        algorithms=CONFIG['algorithm'],
                        access_token=response['access_token'])
    assert session['nonce'] == claims['nonce']
    session['user_id'] = claims['sub']
    session['user_email'] = claims['email']
    session['user_name'] = claims['name']

    return redirect(url_for('index'))
```

Server Implementation

- Fewer libraries
  - OAuthlib: https://github.com/idan/oauthlib
  - PyOIDC: https://github.com/OpenIDC/pyoidc
- Things to think about
  - How to authenticate users
  - Consent from users for data release
  - Token format

Questions?

- Slides and code: brendan.mccoll.am/pycon2017
- Work for Globus: globus.org/jobs
- Useful libraries:
  - requests_oauthlib: requests-oauthlib.readthedocs.io
  - OAuthlib: oauthlib.readthedocs.io
  - PyOIDC: github.com/rohe/myoidc
  - Globus SDK: globus-sdk-python.readthedocs.io