CS 419: Computer Security

Week 11: Recitation

Third-Party Authorization: OAuth

The Problem

I don’t want to give an application my Google account login and password just so the application can access my contacts

- Because then it can access my email, calendar, photos, etc.

Can we provide a way for one application to access services from another (possibly run by a different company) without users having to share their login credentials?

What is OAuth 2?

An authorization framework to allow a user to give an application *limited* access to resources on another service

OAuth is Open Authorization – not *authentication*

- **Authentication:** validate your identity
- **Authorization:** provide access to requested resources

It allows users to authorize access to *specific* data for a *limited time* without sharing login credentials

Designed to work over HTTP

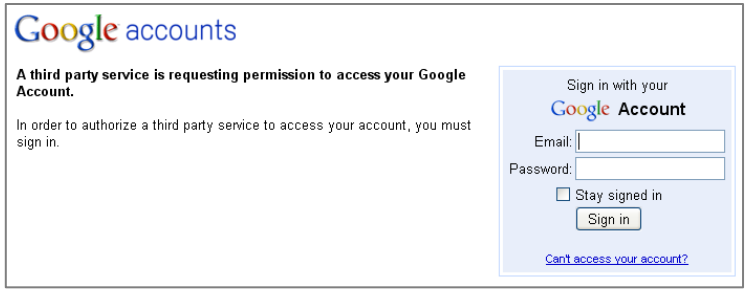
The Main Ideas: **Redirection** and **Access Tokens**

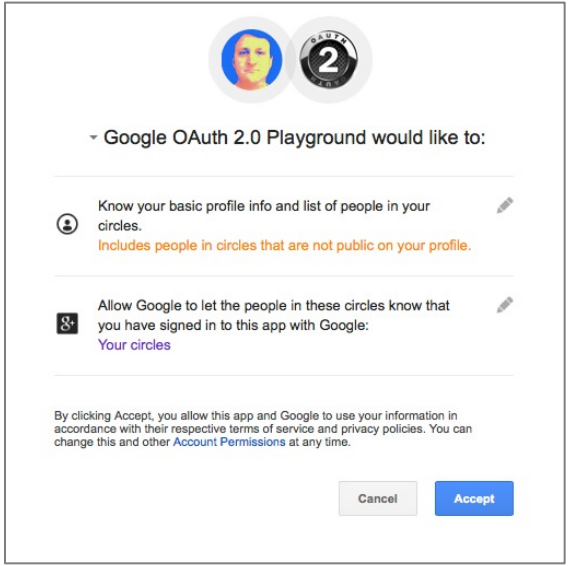
Redirection

Redirect users to an authorization service so they can approve access requests.
After that, redirect the request back to the service that wants the access

Access token: a small piece of data that contains information about:

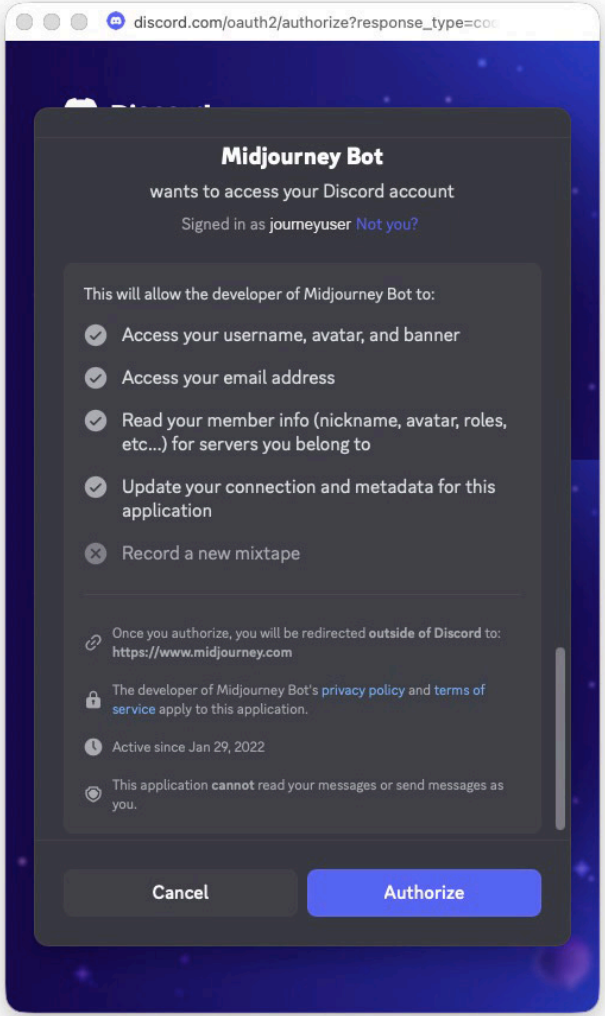
- The user
- The resource the token is for
- Rules for data sharing
- Generated by the authorization service and then sent to set up a session between the client and the service that has the data (APIs) that the client wants to access





Sample Interaction

- I want to access midjourney.com (an AI image generation service)
- Midjourney:
  - Wants to access my Discord information
  - Sends me to Discord's authorization service:
    discord.com/oauth2/authorize
- Discord:
  - Presents me with information telling me:
    - Discord wants access to certain data and interfaces
    - An itemized list of what it wants
  - If I approve, Discord redirects back to Midjourney
- Midjourney gets an access token as a response



Oauth Example: Participants

The key players:

Resource owner
The user who owns the data and grants permission

Client

Client
The application requesting access

Authorization Server

Authorization Server
System that authenticates the user and issues access tokens

Resource Server

Resource Server
Service that holds the user's data

The Setup

Before an application (client) can request access tokens from an OAuth server, it needs to register with it

- Example: a note-taking app registers with the Google OAuth service

- The application provides information
  - Application name, type (mobile, web, desktop)
  - Homepage URL
  - Redirect URI (callback URL to receive the authorization code)
  - List of permissions (called **scopes**) that it will request from users (e.g., "*access contacts*", "*access username*")

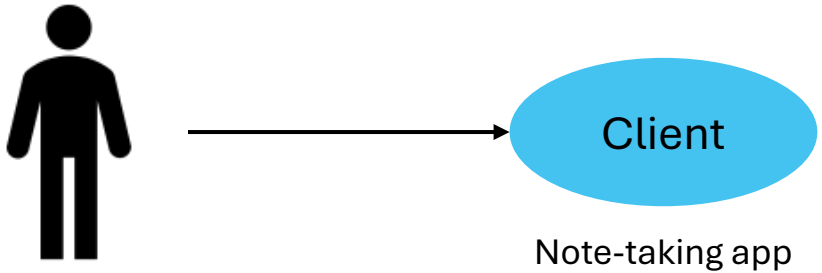
- The OAuth server provides:
  - A unique **Client ID**: `123456789.apps.googleusercontent.com`
  - A **client secret** between the OAuth server & client: `GOCSPX-randomlyGeneratedSecret`

Setting up a client ID & shared secret ensures that only the registered client will be allowed to exchange an authorization code for an access token

Step 1: User initiates login

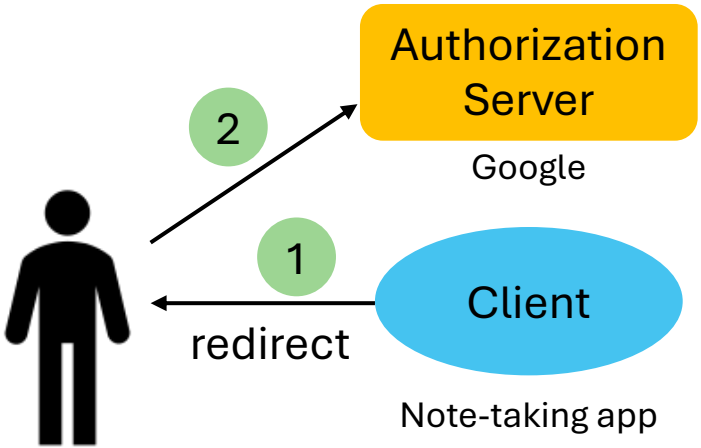
Example: a user wants to sign in to an online note-taking app using their Google account

- User clicks "sign in with Google" on the note-taking app



Step 2: Redirect to Authorization Server

- App redirects the user to Google's Authorization Server along with:
  - Client ID (issued to the app by Google)
  - Redirect URI (where Google should send the user after authorization)
  - Scope (what data the app wants from the service, e.g., *email*, *profile*)
  - Response type (indicating the app wants an authorization code)

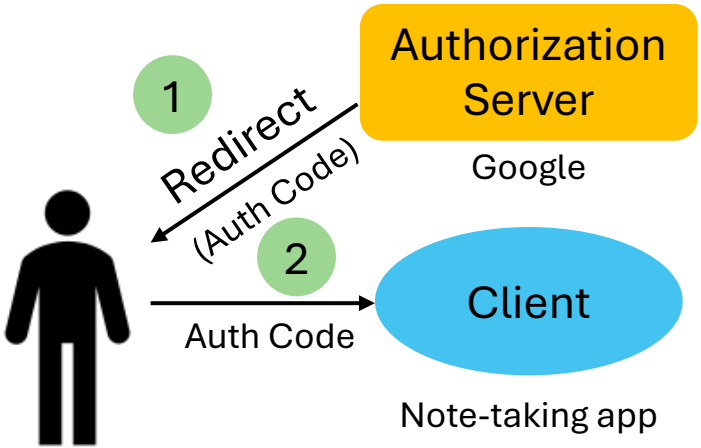


Sample message to the authorization service

```
GET https://accounts.google.com/o/oauth2/auth?
    client_id=CLIENT_ID
    &redirect_uri=REDIRECT_URI
    &response_type=code
    &scope=email profile
```

Step 3: User Grants Permission

- Google asks the user, "*Allow this app to access your profile and email?*"
- If the user agrees, Google grants an authorization code and redirects the user back to the app's Redirect URI

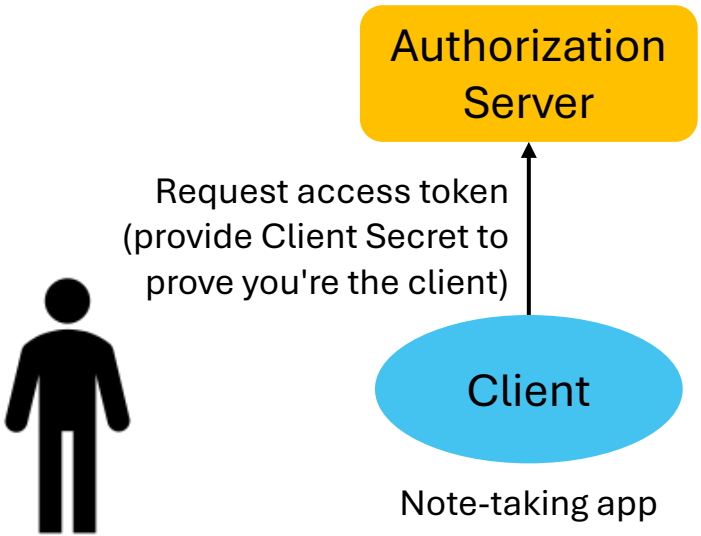


Sample message from OAuth to the app

```
https://yourapp.com/callback?code=AUTH_CODE
```

Step 4: Exchange Authorization Code for Access Token

- App sends a request to Google's authorization server to exchange the authorization code for an access token



Sample message from the app asking for a token

```
POST https://oauth2.googleapis.com/token
Content-Type: application/x-www-form-urlencoded

    client_id=CLIENT_ID
    &client_secret=CLIENT_SECRET
    &code=AUTH_CODE
    &grant_type=authorization_code
    &redirect_uri=REDIRECT_URI
```

Why Can't OAuth Just Send an Access Token?

Why does OAuth exchange an authorization code for an access token?
Why not simply have the OAuth service return the access token?

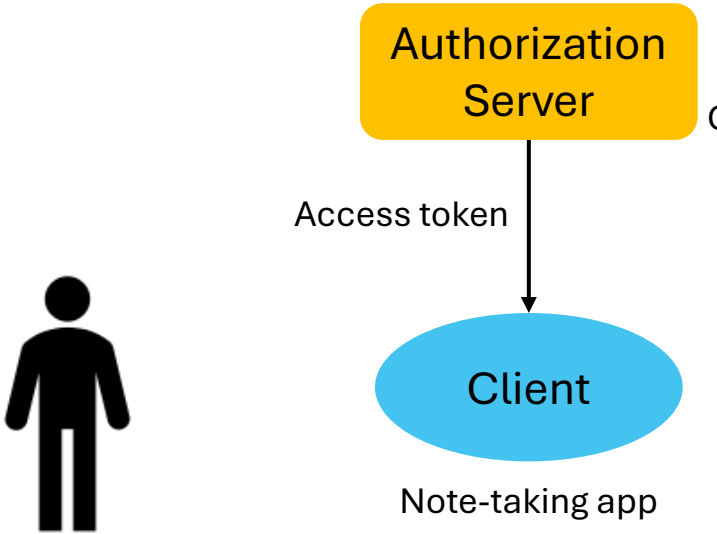
1. **Prevents exposure of access tokens in the URL**
   - URLs are often logged by browsers/proxies/servers
   - Malicious actors might intercept the access token
   - The authorization code has a short lifetime AND is useless without the client secret

2. **Keeps access tokens confidential**
   - Ensures that the access token is obtained in a secure backend exchange – direct communication over a secure (encrypted) connection

3. **Prevents unauthorized token issuance**
   - When exchanging the authorization code for an access token, the client provides:
     - Authorization code, client ID (public data), client secret (private data), redirect URI (must match original request)
     - This ensures only a pre-registered applications with the correct client secret can get the access token

Step 5: Get the Access Token

- If the request is successful (valid), Google responds with an access token
  - A time-limited permission to connect to the service

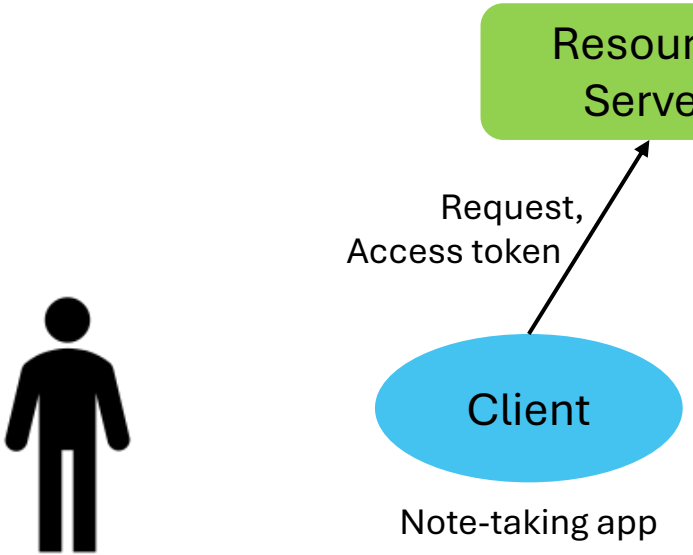


Sample access token from OAuth

```
{
  "access_token": "ACCESS_TOKEN",
  "expires_in": 3600,
  "token_type": "Bearer",
  "refresh_token": "REFRESH_TOKEN"
}
```

Step 6: App Uses Access Token to Fetch User Data

- App makes an API request to the service, giving it the access token
  - Request user info (for example)

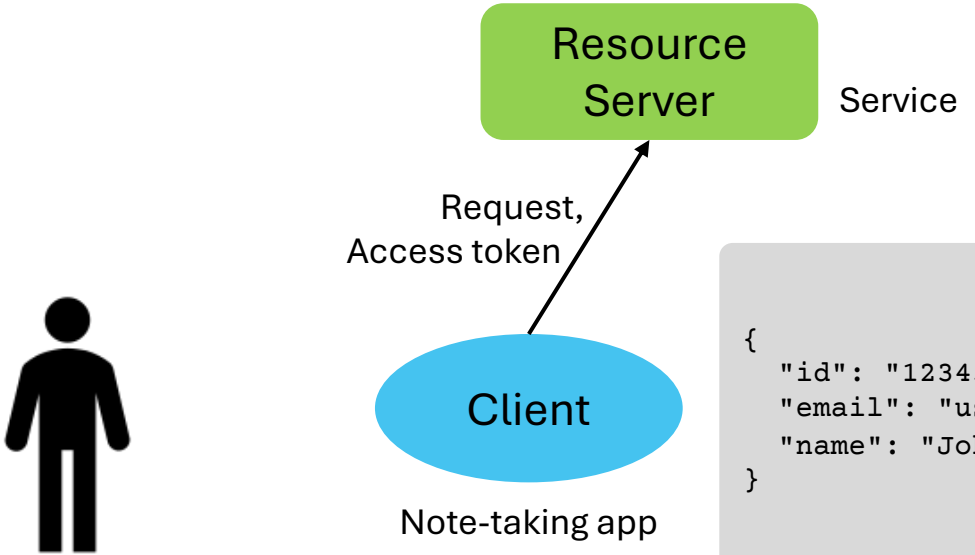


Sample service request

```
GET https://www.googleapis.com/oauth2/v1/userinfo
Authorization: Bearer ACCESS_TOKEN
```

Step 6: Service Responds to the Request

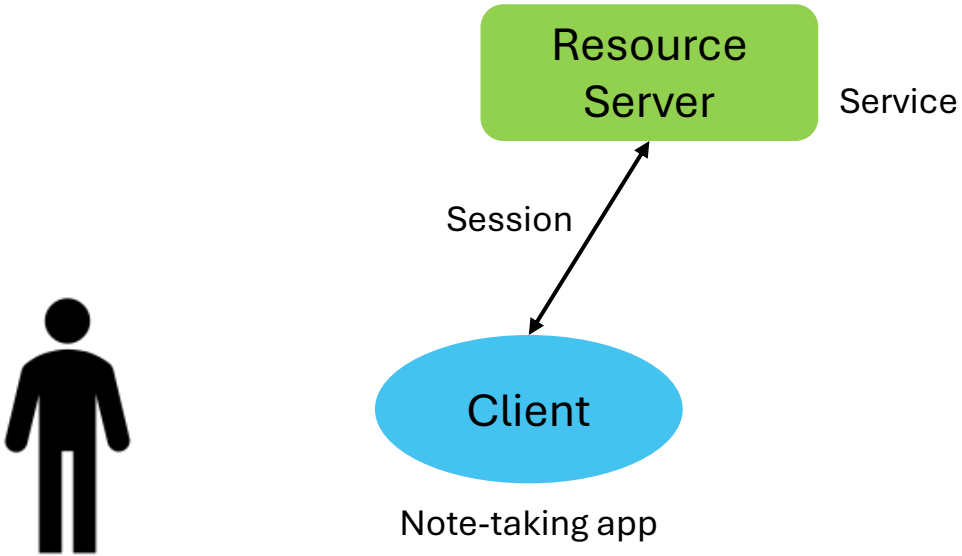
- The service responds with the requested data
  - The user's profile data, in this case




```
{
  "id": "123456789",
  "email": "user@example.com",
  "name": "John Doe"
}
```

Step 7: App logs in the User

- The note-taking app creates a session for the user using the retrieved info



Key Benefits of OAuth 2.0

- Security
  - The user's credentials are never shared with the third-party app.
- Limited Access
  - The app gets only the data it needs, as specified in the scope.
- Token Expiration & Refreshing:
  - The app can request a new access token without making the user log in again.

The End