Identity Providers

By Dylan Brassard

What is an IdP ?

Manages, maintains and creates Principals

Entity (user, service or device) that can be authenticated

Offers user authentication as a service

Federates identities

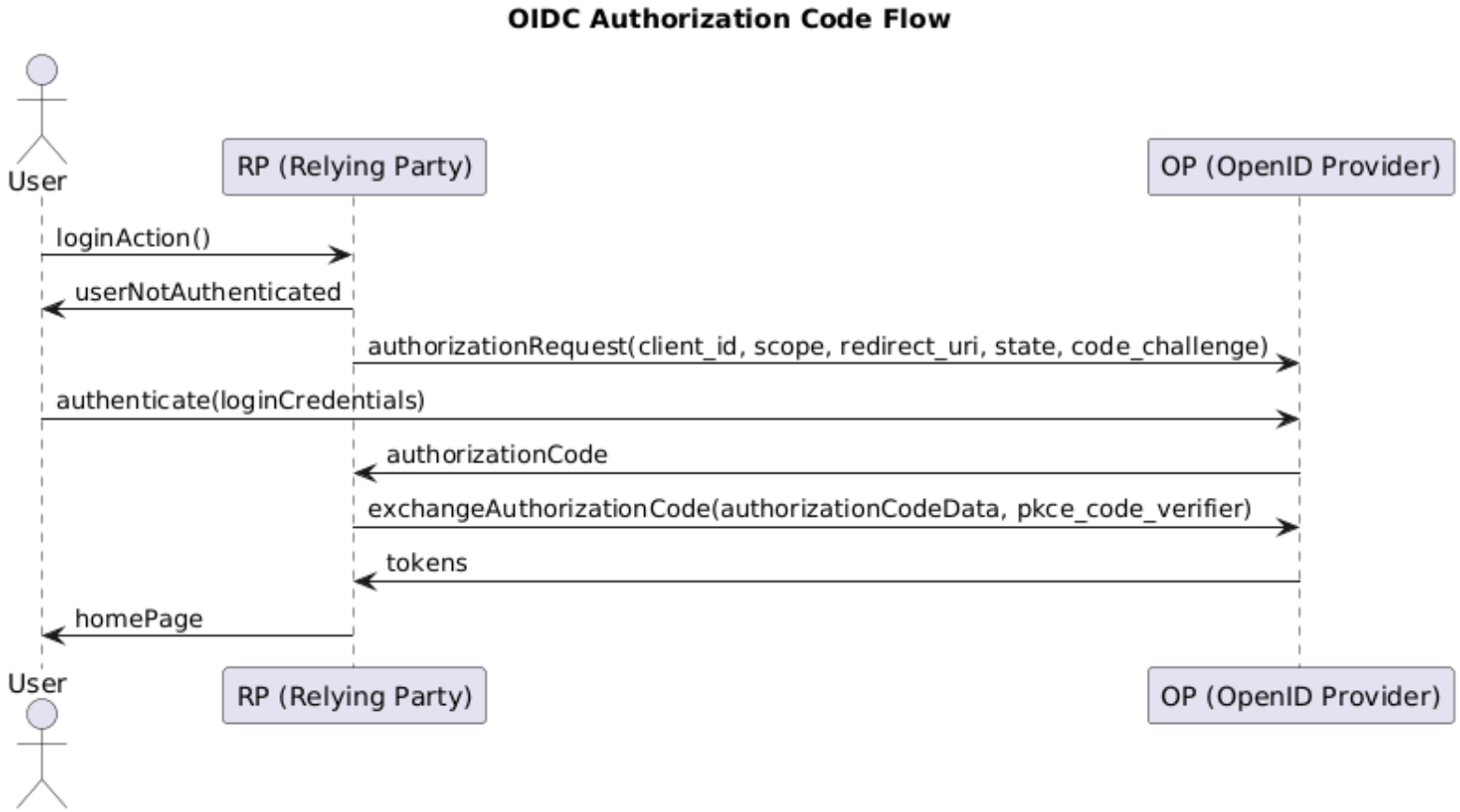
Issues security tokens (ID tokens, access tokens, etc.)

One identity for multiple applications

Ex: Google account to login in multiple apps

Basic Authorization Flow with OIDC.

[Plantuml]



What is OAuth2

- Grants access to a ressource
- NOT an authentication protocol -> Authorization protocol
- Uses **Access Token (**JWT or Opaque)
  - Opaque tokens are a random string that reference to stored data
- Uses an **authorization code** to get the access token
- Uses **scopes** which define the level of access to that ressource an entity has, to what and how is left to the application (Ressource Server)
  - Scopes are useful for fine grained control (read-only on specific data instead of coarse flat roles)

OAuth2 Endpoints

Endpoint	Purpose
/authorize	User login + obtain authorization code
/token	Exchange code for tokens
/userinfo	Get user profile
/revoke	Invalidate access/refresh tokens
/introspect	Check token validity
/device	Start Device Code Flow (user enters a short code or scans a QR code on another device to authenticate)
/end-session	Logout from the IdP (OIDC end-session)
/jwks	Public keys for verifying tokens
/.well-known/openid-configuration	Returns all OAuth2/OIDC endpoints, supported scopes, and metadata

What is OIDC ?

- Authentication protocol built on top of OAuth2
- Uses the **OpenID scope** to get the **ID Token**
- Using OIDC an IdP can authenticate with :
  - Username/password
  - Single-use code delivered out of band, e.g., by SMS or email
  - Code generated by App (OATH, TOTP or HOTP)
  - Biometric via App
  - Federated (e.g., Facebook, GoogleID)
- With SPAs (Single Page Applications) we use the **Authorization Code Flow with PKCE (*Proof Key for Code Exchange*)**, where only the authorization code is returned through the authorization endpoint, and all tokens are retrieved securely from the **token endpoint**.
  - PKCE = a random secret generated by the client (the *code_verifier*). Its hash (the *code_challenge*) is sent to the IdP when requesting the authorization code, and the original secret is later sent to the IdP when exchanging the code for tokens.

Example authorization request

GET

https://authentik.benmusicgeek.synology.me/application/o/authorize/?**response_type**=code&**state**=QVCkFWOGe2rUPDXkYdk-4Q&**code_challenge**=swj7ulxcWhX5ykZash2OUrcoBt5-l3S2eUQS3e1N3WA&**code_challenge_method**=S256&**client_id**=WEWyuT2JlOSVD3FIHT8UbKGI0T8UFjFgQ3fW4XQz&**scope**=openid profile&**redirect_uri**=https://jellyfin.benmusicgeek.synology.me/sso/OID/redirect/authentik

Parameters and what they mean

- **State** is used to protect against CSRF attacks by ensuring that the authorization response corresponds to a login request initiated by the user in the same browser session. This prevents unauthorized redirects to the callback URL—for example, an attacker cannot trick the victim into being logged into the attacker's account.

- The **code_challenge** is a hash of a secret (the code_verifier) generated on the frontend (RP). Later, during the token request, the RP sends the code_verifier to the IdP, which verifies that it matches the code challenge sent in the initial request.
- **response_type** indicates what the RP is requesting the IdP to return during the authorization step. In the Authorization Code Flow, response_type=code tells the IdP to return an authorization code, which the RP will later exchange for tokens.

What is SSO ?

- SSO stands for Single Sign-On, a method that allows users to access multiple applications with just one set of login credentials
- Requires
  - An **IdP** for a single login
  - An SSO protocol such as **OIDC** or **SAML**
- Allows
  - **MFA** (Multi-Factor Authentication)
  - Stronger passwords and better password enforcement
  - Central user management
- Example : https://authentik.benmusicgeek.synology.me/

What is Auth0 ?

IDENTITY PROVIDER (IDP)

IDENTITY & ACCESS MANAGEMENT (IAM)

SECURITY FEATURES LIKE MFA, PASSWORDLESS, SOCIAL LOGINS,, ANOMALY DETECTION

Why use an existing IdP ?

- **All critical features are built-in** (OAuth2, OIDC, SAML, MFA, passwordless, user management)
- **Battle-tested and security-hardened** (used by thousands/millions of apps)
- **No maintenance or infrastructure required** (patching, updates, scaling handled by vendor)
- **Protocols and standards implemented correctly** (PKCE, token rotation, OIDC spec compliance)
- **Wide support for login types and integrations** (social, enterprise, WebAuthn, LDAP, etc.)
- **Reduced security risk** — handled by teams specialized in authentication & identity

Auth0 integration example

Repository