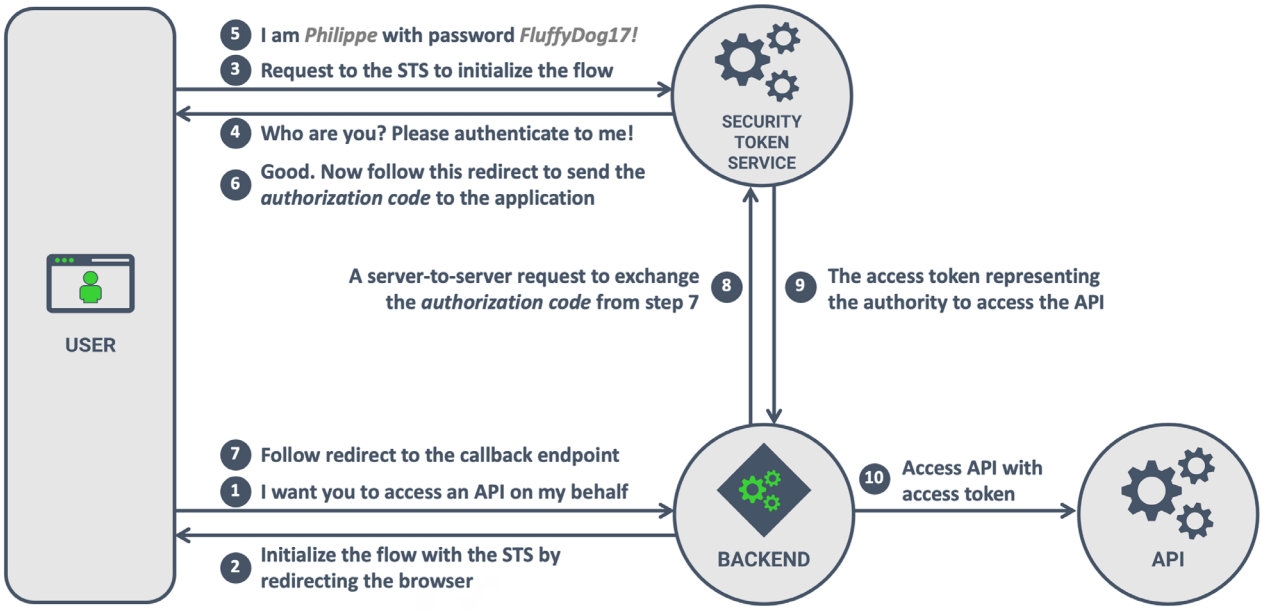
Pragmatic Web Security
Security for developers

SECURITY CHEAT SHEET
Version 2025.001

OAUTH 2.0 BEST PRACTICES FOR DEVELOPERS

OAuth 2.0 is an elaborate framework, which continuously evolves to address current needs and security considerations. The framework is even evolving into a consolidated OAuth 2.1 specification. This cheat sheet offers an overview of current security best practices for developers building OAuth 2.x client applications.

THE *AUTHORIZATION CODE* FLOW FOR OAUTH



GENERAL RECOMMENDATIONS

- [ ] Use the *Authorization Code* flow in every redirect scenario
- [ ] Always use *Proof Key for Code Exchange* (PKCE)
  - The client includes a challenge based on a secret in Step 2
  - The client includes the secret verifier in Step 8
- Restrict the capabilities of bearer access tokens
  - Keep the lifetime of access tokens as short as possible
  - Use scopes to restrict the permissions associated with a token
- Use sender-constrained tokens instead of bearer tokens
  - Sender-constrained access tokens are bound to a secret key
  - The client proves possession of the key when using the token
  - Can be implemented with mTLS (RFC 8705) or DPoP (RFC 9449)
- Consider Resource Indicators to minimize access tokens
  - This reduces the authority of an access token to a single API
  - The client obtains an access token per API with the refresh token

RECOMMENDATIONS FOR NATIVE CLIENTS

- [ ] Use a system browser instead of an embedded browser
  - On iOS, use *ASWebAuthenticationSession* to run the flow
  - On Android, use *Chrome Custom Tabs* to run the flow
- [ ] Encrypt access tokens and refresh tokens in storage
  - Store the encryption keys in a key store provided by the OS
- Keep an eye on the draft spec for first-party applications
  - This spec aims to enable in-app authentication for 1st party apps

RECOMMENDATIONS FOR BACKEND CLIENTS

- [ ] Use key-based client authentication in Step 8
  - Key-based client authentication avoids the use of a shared secret
  - Can be implemented with mTLS (RFC 8705) or JWT (RFC 7523)
- [ ] Encrypt access tokens and refresh tokens in storage
  - Store the encryption keys using a secret management service
- Use Pushed Authorization Requests (PAR) (RFC 9126)
  - The client pushes the config to the server at the start of the flow
  - PAR doesn't expose the authorization request to the frontchannel

RECOMMENDATIONS FOR FRONTEND WEB CLIENTS

- [ ] Do not use OAuth 2.0 directly in frontend clients
  - Using OAuth 2.0 in a frontend increases the attack surface
  - Review the threat model in the current draft spec
- [ ] Use a Backend-for-Frontend (BFF) to handle OAuth 2.0
  - The BFF acts as a backend client to the authorization server
  - The BFF uses a session to keep track of the user's tokens
  - The BFF proxies API calls and attaches the user's access token
- Make the use of a BFF the default for frontends
  - Build or use a BFF library to simplify BFF deployment
  - BFFs do not contain business logic and can easily be reused
  - A well-designed BFF library only needs a few lines of config