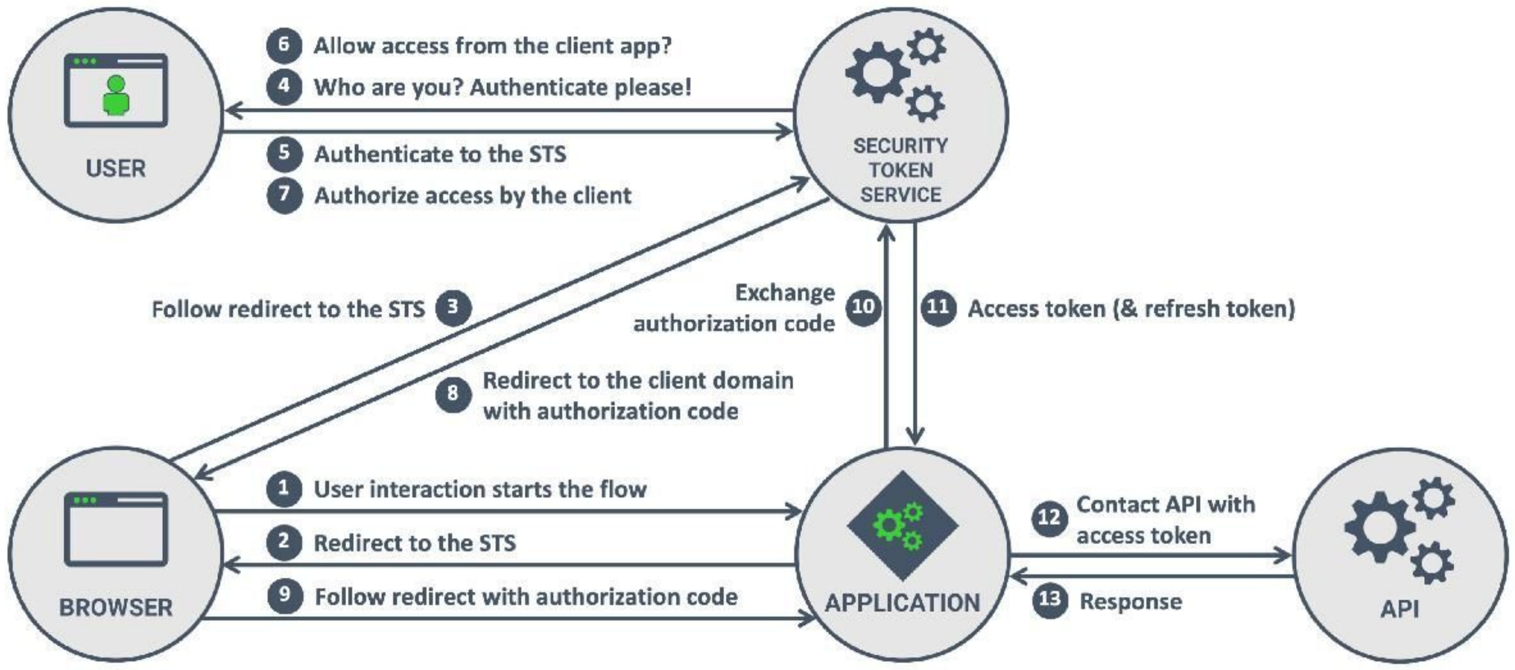
Pragmatic Web Security
Security for developers

SECURITY CHEAT SHEET
Version 2020.001

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.


USER
SECURITY TOKEN SERVICE
BROWSER
APPLICATION
API
6 Allow access from the client app?
4 Who are you? Authenticate please!
5 Authenticate to the STS
7 Authorize access by the client
Follow redirect to the STS 3
Exchange authorization code 10
11 Access token (& refresh token)
8 Redirect to the client domain with authorization code
1 User interaction starts the flow
2 Redirect to the STS
9 Follow redirect with authorization code
12 Contact API with access token
13 Response

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 1
  - The client includes the secret verifier in Step 10
- When using refresh tokens, apply additional protection
  - Rotate refresh tokens and act upon double use of a token
  - Invalidate refresh tokens for web applications when ...
    - – the user explicitly logs out of the security token service
    - – the user's session with the security token service expires
  - Invalidate refresh tokens when the user's password changes
- ☐ Include an audience in the flow and in the access tokens
  - This restricts who accepts the access token in Step 12
- 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

REFERENCES

- OAuth 2.0 threat model and security considerations
- OAuth 2.0 Security Best Current Practice
- The OAuth 2.1 Authorization Framework (draft)

RECOMMENDATIONS FOR BACKEND CLIENTS

- ☐ Use client authentication in Step 10
  - Prefer key-based authentication over shared client secrets
- ☐ Encrypt access tokens and refresh tokens in storage
  - Store the encryption keys using a secret management service
- ☐ Use proof-of-possession access/refresh tokens
  - Using sender-constrained tokens requires possession of a secret

RECOMMENDATIONS FOR FRONTEND WEB CLIENTS

- ☐ Use the *Authorization Code* flow with PKCE for new projects
  - The *Implicit* flow is not broken, but should be phased out
- Be careful with using refresh tokens in web applications
  - Do not use long-lived refresh tokens in the browser
  - Ensure that refresh tokens are protected (see on the left)
- Focus on preventing XSS vulnerabilities in the frontend
  - XSS results in the complete compromise of the client application
  - Avoiding the use of LocalStorage is not an XSS defense

RECOMMENDATIONS FOR NATIVE CLIENTS

- ☐ Use a system browser instead of an embedded browser
  - On mobile, use *SFSafariViewController* or *Chrome Custom Tabs*
- ☐ Encrypt access tokens and refresh tokens in storage
  - Store the encryption keys in a key store provided by the OS