# OAuth 2.0 and OpenID Connect

Fabian Hauck

# OAuth 2.0 vs. OpenID Connect

**OAuth 2.0:** “The OAuth 2.0 authorization framework enables a third-party application to obtain limited access to an HTTP service [...]” - RFC 6749

- Example use case: Authorize a printer to access a cloud storage with photos

**OpenID Connect:** Is an identity layer on top of OAuth 2.0 that enables clients to verify the identity of a user - openid.net

- Example use cases: Single Sign-On, Insurance verifies the identity of a user with a bank

# Public vs Confidential Clients

**Confidential Clients:**

- Can keep a client secret to authenticate to the authorization server
- For example: client implemented on a secure server

**Public Clients:**

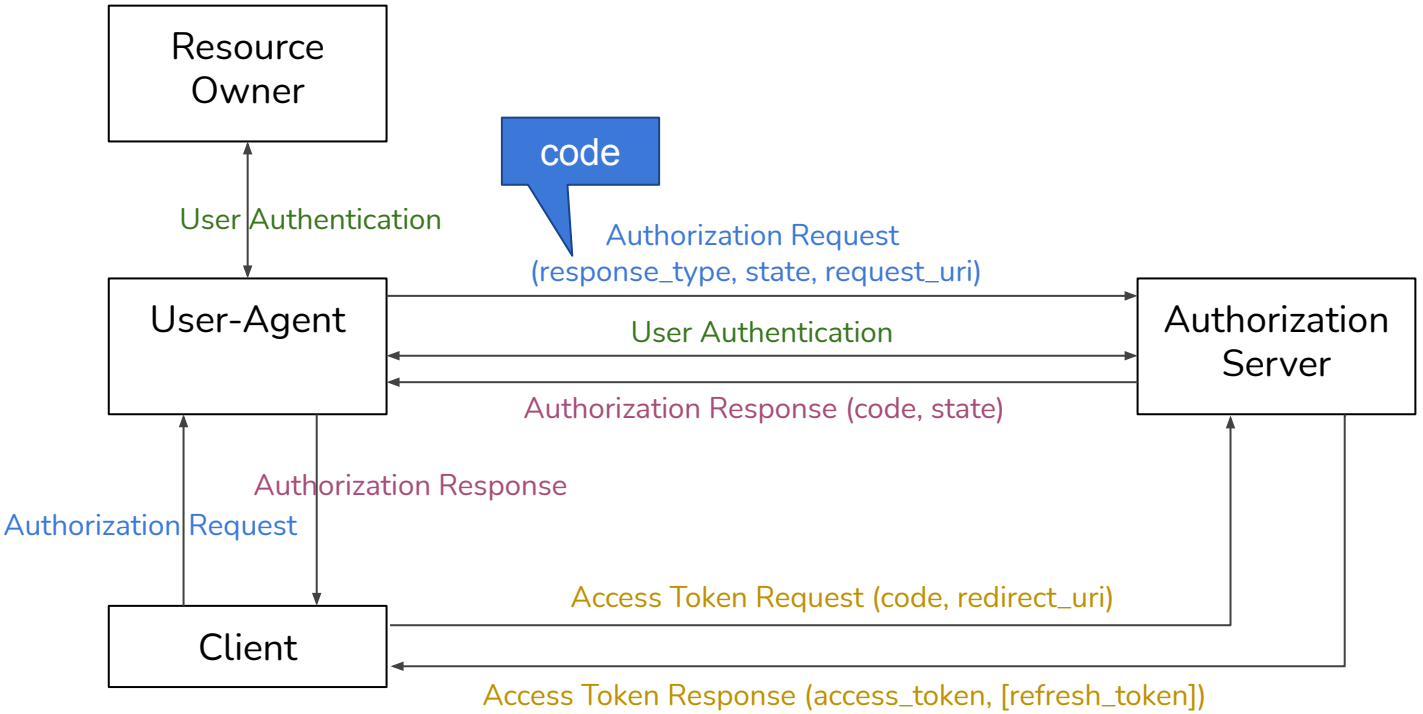
- Can not keep a client secret
- For example: native apps, web browser-based applications

Further information about client types can be found in RFC 6749

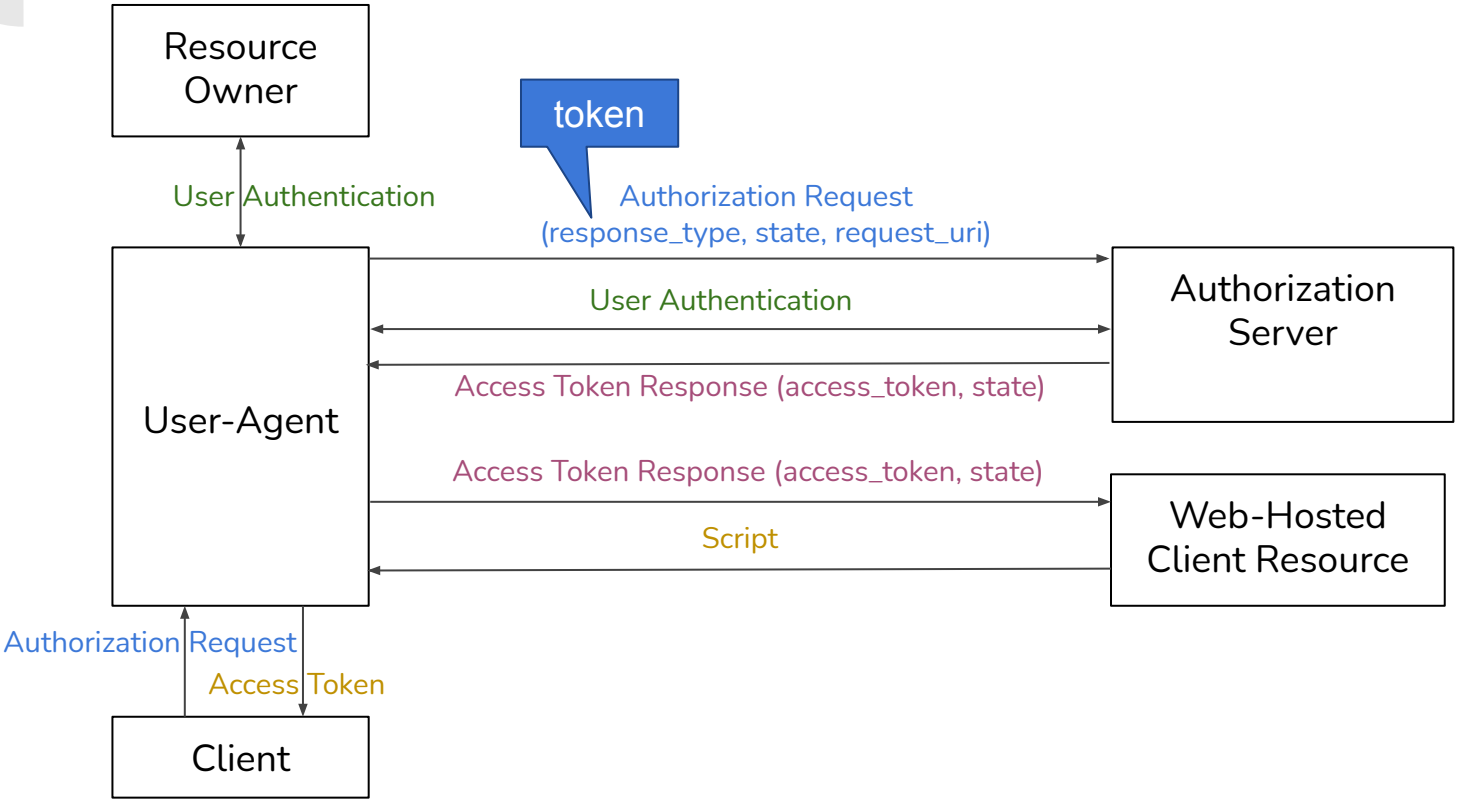
# OAuth 2.0 Flows

- Authorization Code Grant
- Implicit Grant

# Authorization Code Grant



# Implicit Grant



# Attacks

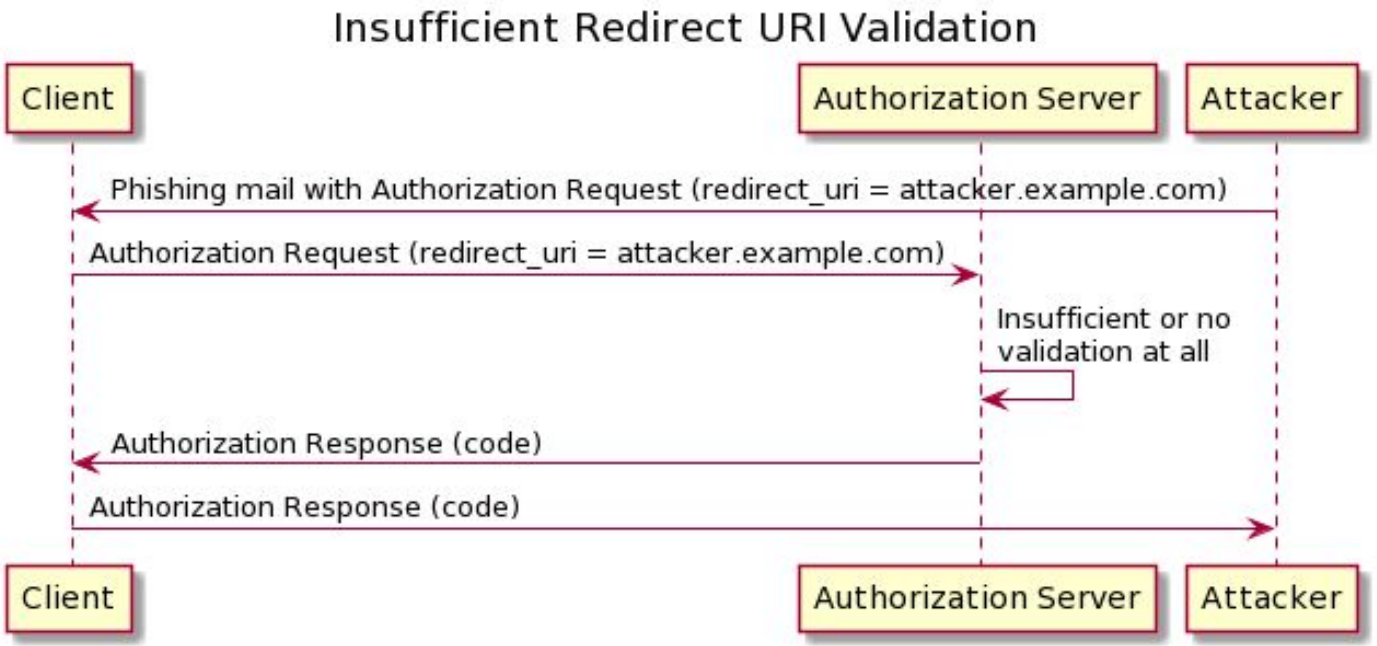
- Insufficient Redirect URI Validation
- Code Leakage

# Insufficient Redirect URI Validation

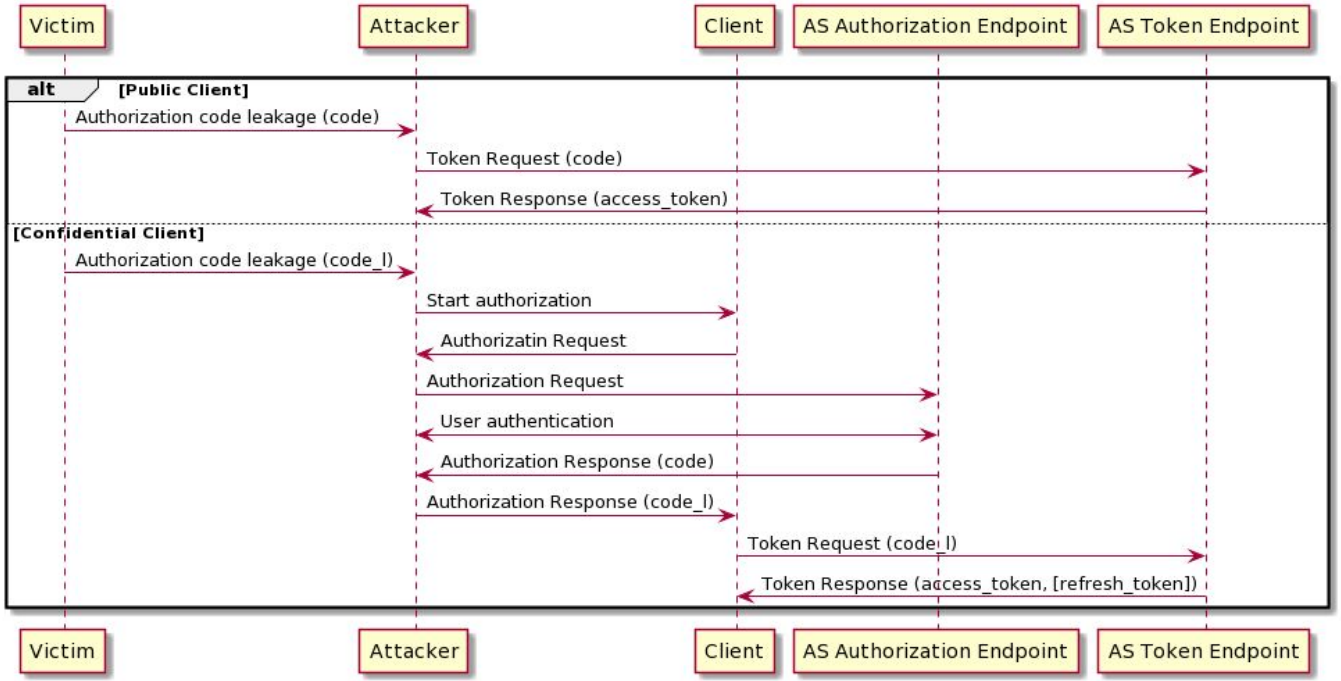
- During client registration the client defines one or more redirect URIs for the authorization response
- If those redirect URIs do not get validated at the AS an adversary can create an authorization request which redirects to a site the attacker owns
- Even if the AS validates the redirect URI but allows pattern like "*https://*.somesite.example/**" the attacker could still be able to redirect to his site by setting the redirect URI to "*https://attacker.example/.somesite.example*"
  → this does obviously depend on how the AS validates the redirect URI

Further information can be found in the OAuth 2.0 Security BCP

# Insufficient Redirect URI Validation



# Code Leakage



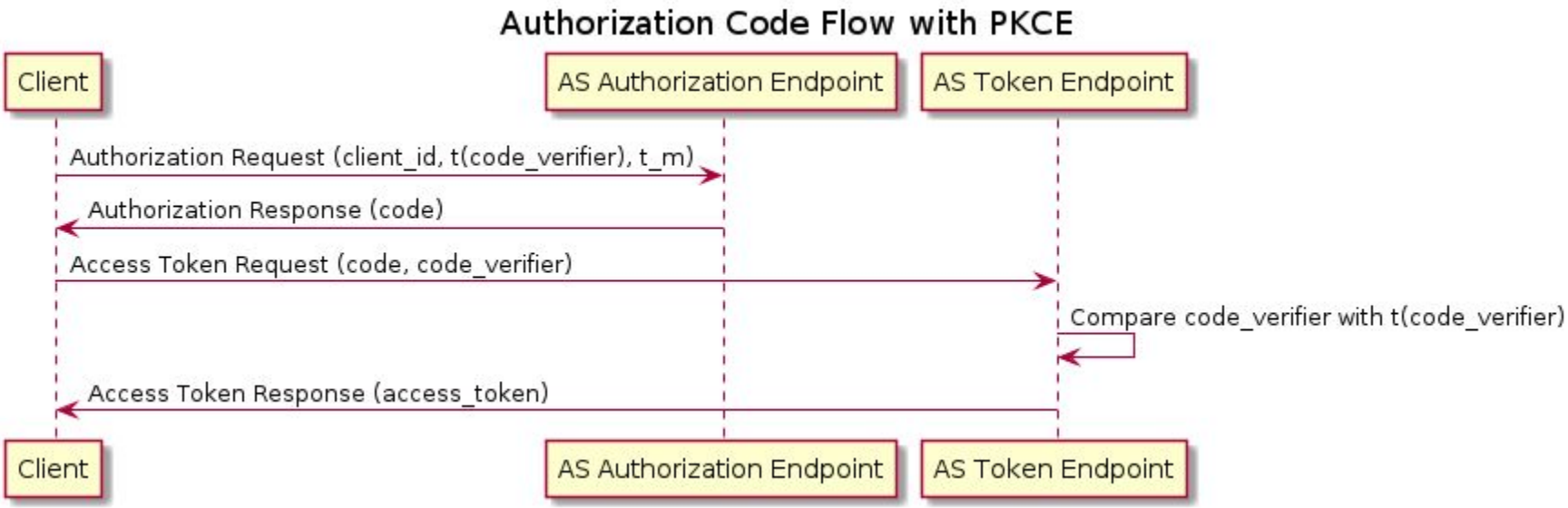

Further information can be found in the [OAuth 2.0 Security BCP]

# Live Demo

Authorization Code stealing in an insecure usage of the AppAuth-Android library. The source code can be found on GitHub.

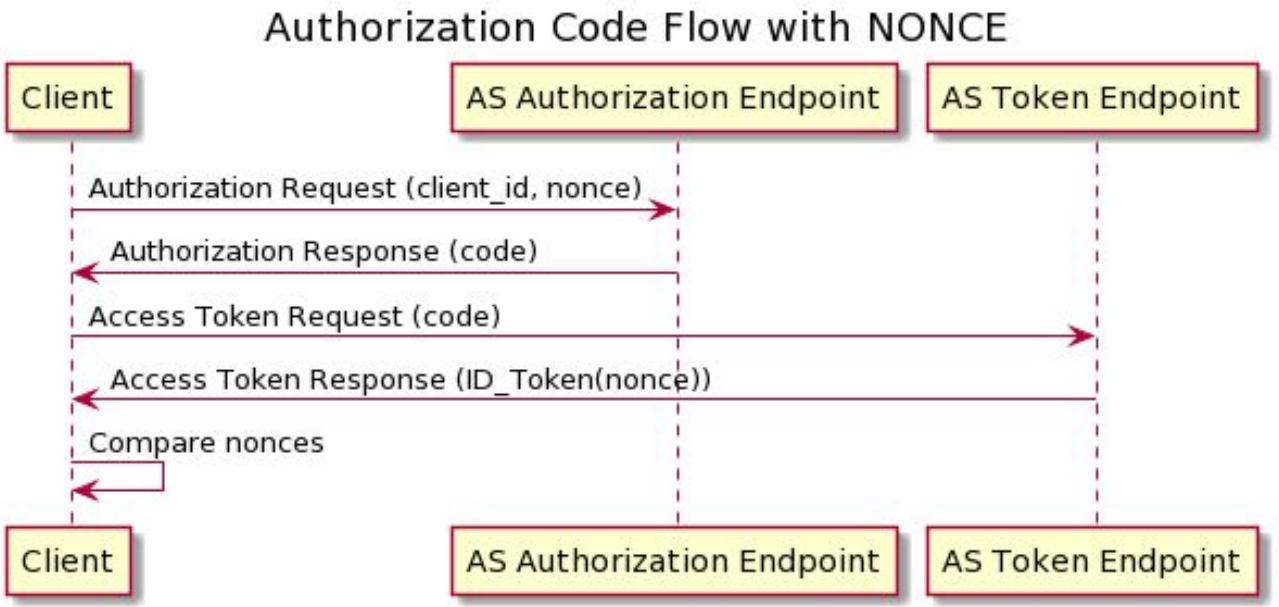
# Security Mechanisms

# Proof Key for Code Exchange (PKCE)




Further information can be found in the [RFC 7636](RFC 7636).

# Nonce (OpenID Connect)



# Outlook

# App2App Authorization Flows

- The next big thing in OAuth are App-to-App flows on mobile devices because they improve the user experience
- But compared to the web redirections the redirections between apps are much more susceptible to hijacking
- This is why it is important to use Android App Links respectively Universal Links on iOS
- If Android App Links can not be used on Android the Intent scheme should be used → this is supported in all major browsers

Further information can be found in the Improving OAuth App-to-App Security blog post

# OAuth 2.1

- Authorization Code flows have to use PKCE
- Redirect URIs have to be compared by exact string matching
- The Implicit flow and the Resource Owner Password Credentials flow were removed from the specification
- Refresh token must either be bound to the client or refresh token rotation must be used

Further information can be found in the OAuth 2.1 draft.

# Penetration Testing Guide

- Check if the appropriated flow is used (most times probably Authorization Code flow)
- Check for insufficient redirect URI verification
- Check whether PKCE and Nonce () is used and correctly verified
- If no PKCE is used verify that the ‘state’ parameter is used for CSRF protection
  → further information can be found in the OAuth 2.0 Security BCP
- Make sure that the authorization code cannot be reused
- Verify that all secrets (client_secret, state, none, pkce_verifier) have a sufficient high entropy and are not leaked through any channel
- On mobile: Check the security of the redirections (look at Improving OAuth App-to-App Security guide)

# Additional Resources

- https://maxfieldchen.com/posts/2020-05-17-penetration-testers-guide-oauth-2.html