itverket
EN DEL AV OMEGAPOINT

OAuth2 and OpenID Connect

Eivind Jahr Kirkeby

> whoami

- > 30år
- > Fra Oslo
- > NTNU – Kybernetikk og Robotikk
- > Fullstack utviker
- > Blazor/React -> .NET -> SQL

Agenda

- Intro
- JWT
- Grant types (flows)
- BFF
- Logout
- Demo
- Cookies
- Refresh tokens

itverket
EN DEL AV OMEGAPOINT

Best practices

Workgroup: Web Authorization Protocol
Internet-Draft:
draft-ietf-oauth-security-topics-21
Published: 27 September 2022
Intended Status: Best Current Practice
Expires: 31 March 2023

T. Lodderstedt
yes.com
J. Bradley
Yubico
A. Labunets
Independent Researcher
D. Fett
yes.com

OAuth 2.0 Security Best Current Practice

Abstract

This document describes best current security practice for OAuth 2.0. It updates and extends the OAuth 2.0 Security Threat Model to incorporate practical experiences gathered since OAuth 2.0 was published and covers new threats relevant due to the broader application of OAuth 2.0.

https://oauth.net/2/oauth-best-practice/

itverket
EN DEL AV OMEGAPOINT

Authentication vs. Authorization

Who are you?

Validate the identification of the user

What are you allowed to do?

Check users' permissions to access data

itverket
EN DEL AV OMEGAPOINT

what is oauth2?

All Videos Images Books Shopping More Tools

About 16,400,000 results (0.58 seconds)

OAuth 2.0, which stands for “Open Authorization”, is **a standard designed to allow a website or application to access resources hosted by other web apps on behalf of a user**. It replaced OAuth 1.0 in 2012 and is now the de facto industry standard for online authorization.

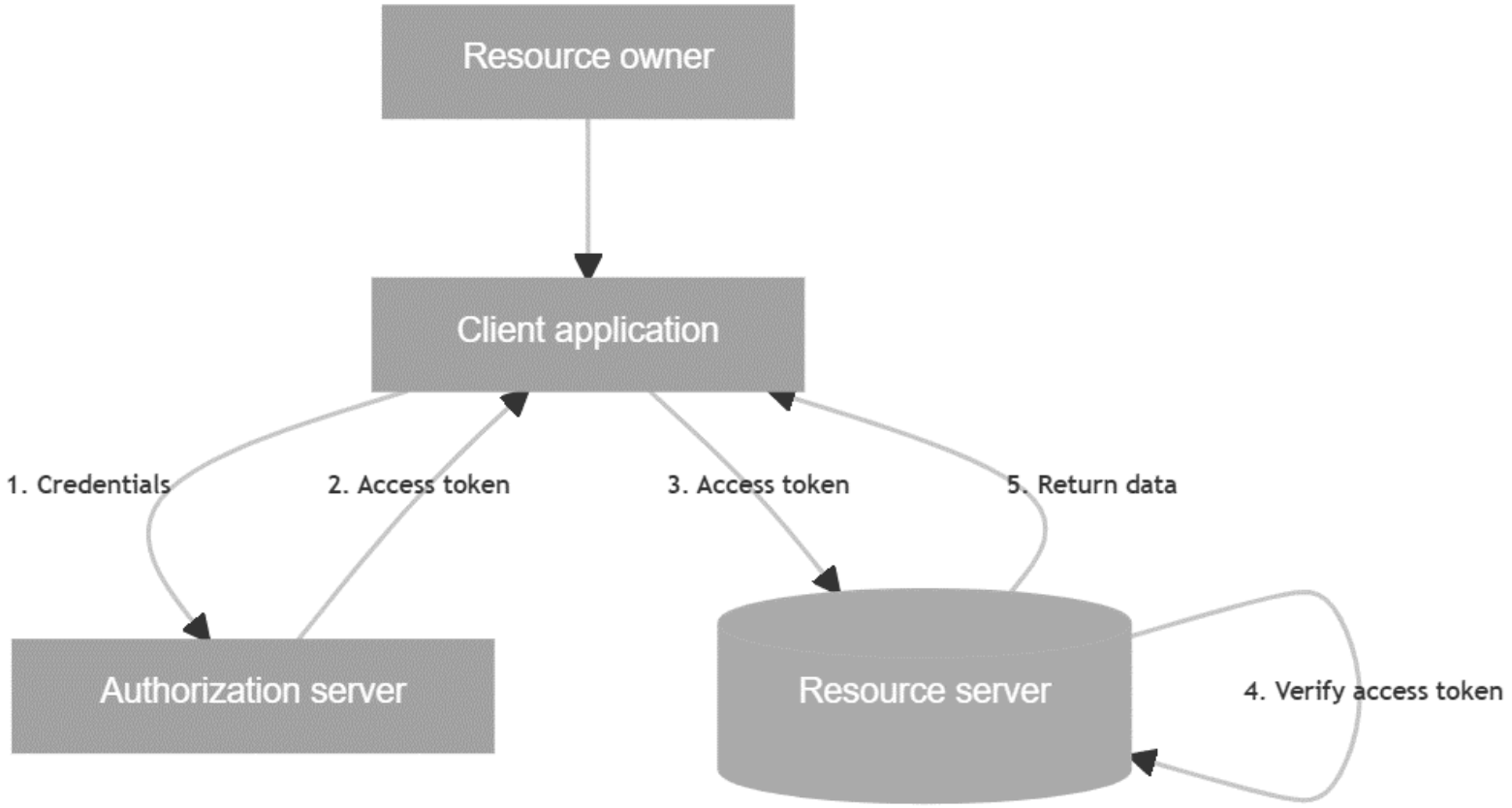
what is openid connect?

All Videos Images News Shopping More Tools

About 12,900,000 results (0.64 seconds)

OpenID Connect (OIDC) is **an open authentication protocol that works on top of the OAuth 2.0 framework**. Targeted toward consumers, OIDC allows individuals to use single sign-on (SSO) to access relying party sites using OpenID Providers (OPs), such as an email provider or social network, to authenticate their identities.

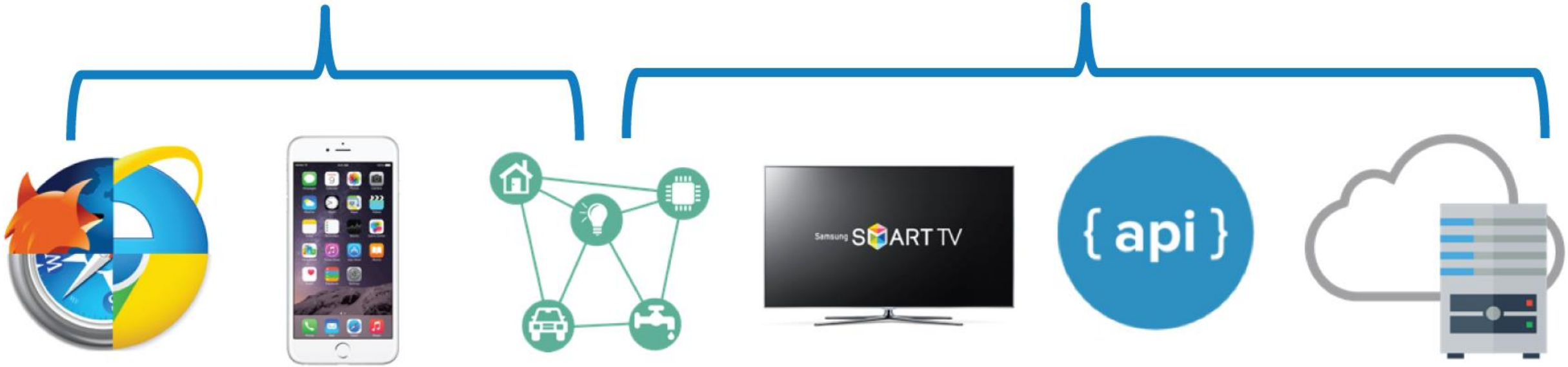
Overview



itverket
EN DEL AV OMEGAPOINT

Public
(Client Identification)

Confidential
(Client Authentication)



Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.7u0ls1snw4tPEzd0JTFaf19oXoOvQYtowiHEAZnan74
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYLOAD: DATA

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  tyk123
) secret base64 encoded
```

Signature Verified

SHARE JWT

itverket
EN DEL AV OMEGAPOINT

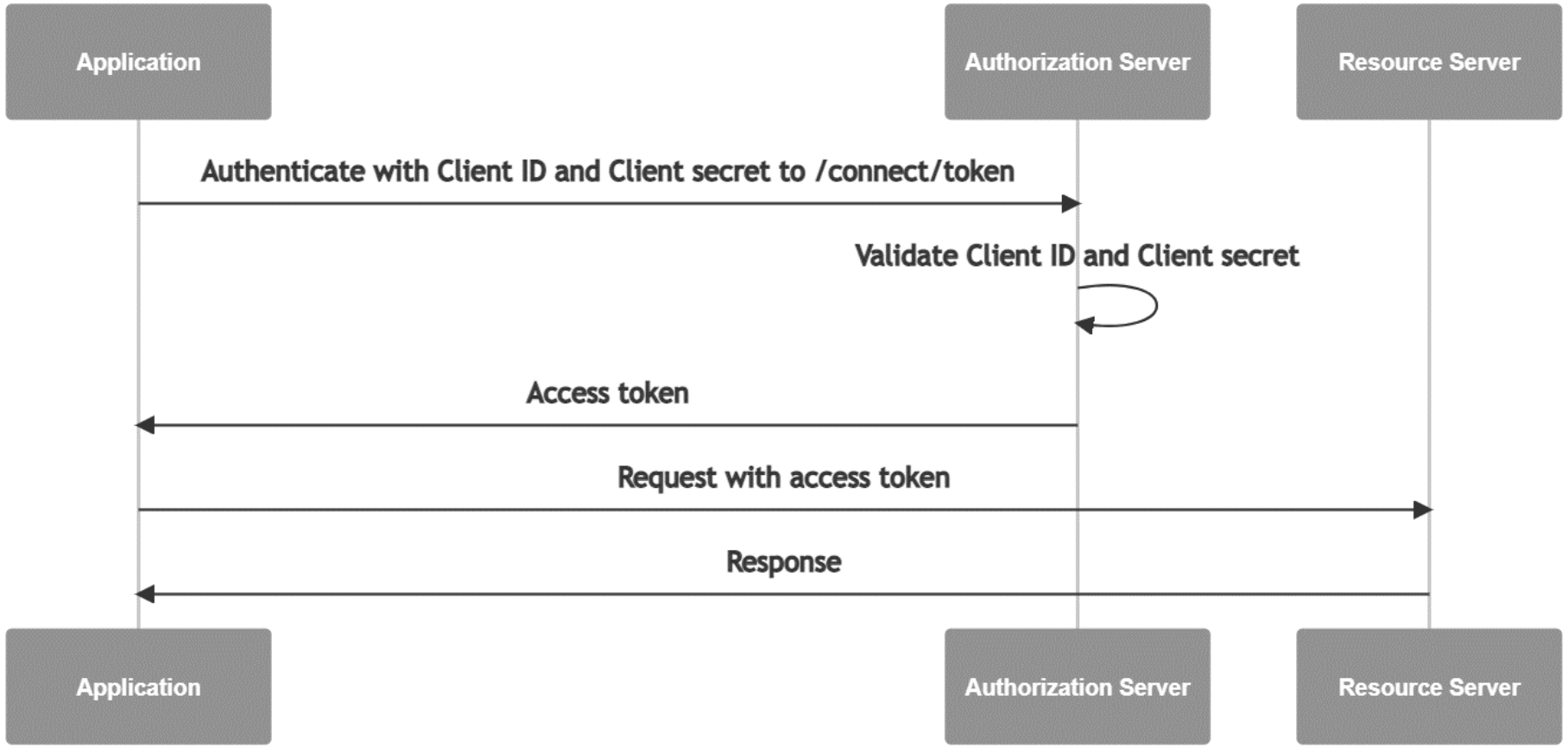
OAuth2 grant types (flows)

- Authorization code grant
- Client credential grant
- Device grant
- Implicit grant (deprecated)

itverket
EN DEL AV OMEGAPOINT

Client credential grant

Machine to machine
Not user bound



itverket
EN DEL AV OMEGAPOINT

2.1.2. Implicit Grant

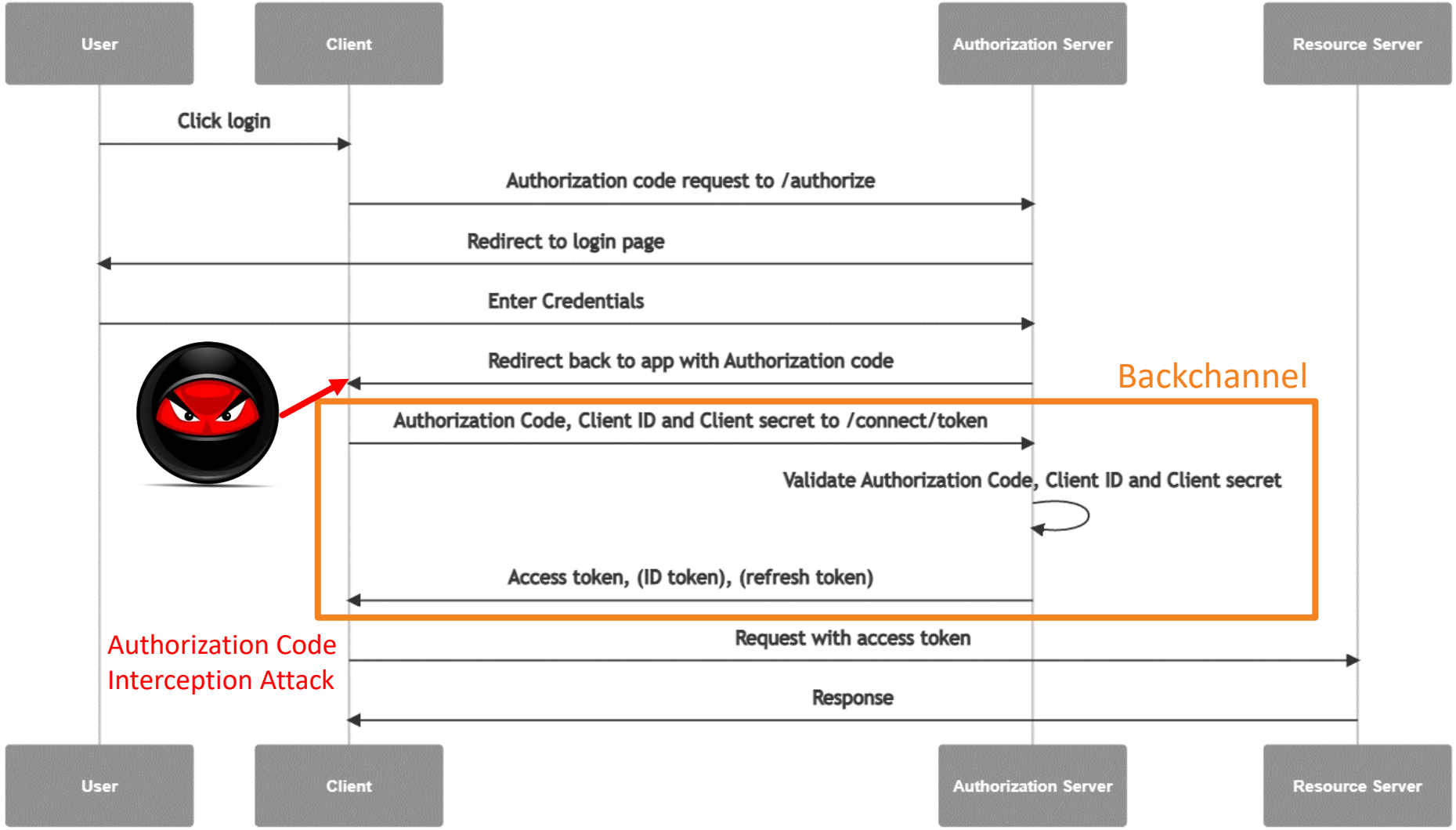
The implicit grant (response type "token") and other response types causing the authorization server to issue access tokens in the authorization response are vulnerable to access token leakage and access token replay as described in Section 4.1, Section 4.2, Section 4.3, and Section 4.6.

Moreover, no viable method for sender-constraining exists to bind access tokens to a specific client (as recommended in Section 2.2) when the access tokens are issued in the authorization response. This means that an attacker can use leaked or stolen access token at a resource endpoint.

In order to avoid these issues, clients SHOULD NOT use the implicit grant (response type "token") or other response types issuing access tokens in the authorization response, unless access token injection in the authorization response is prevented and the aforementioned token leakage vectors are mitigated.

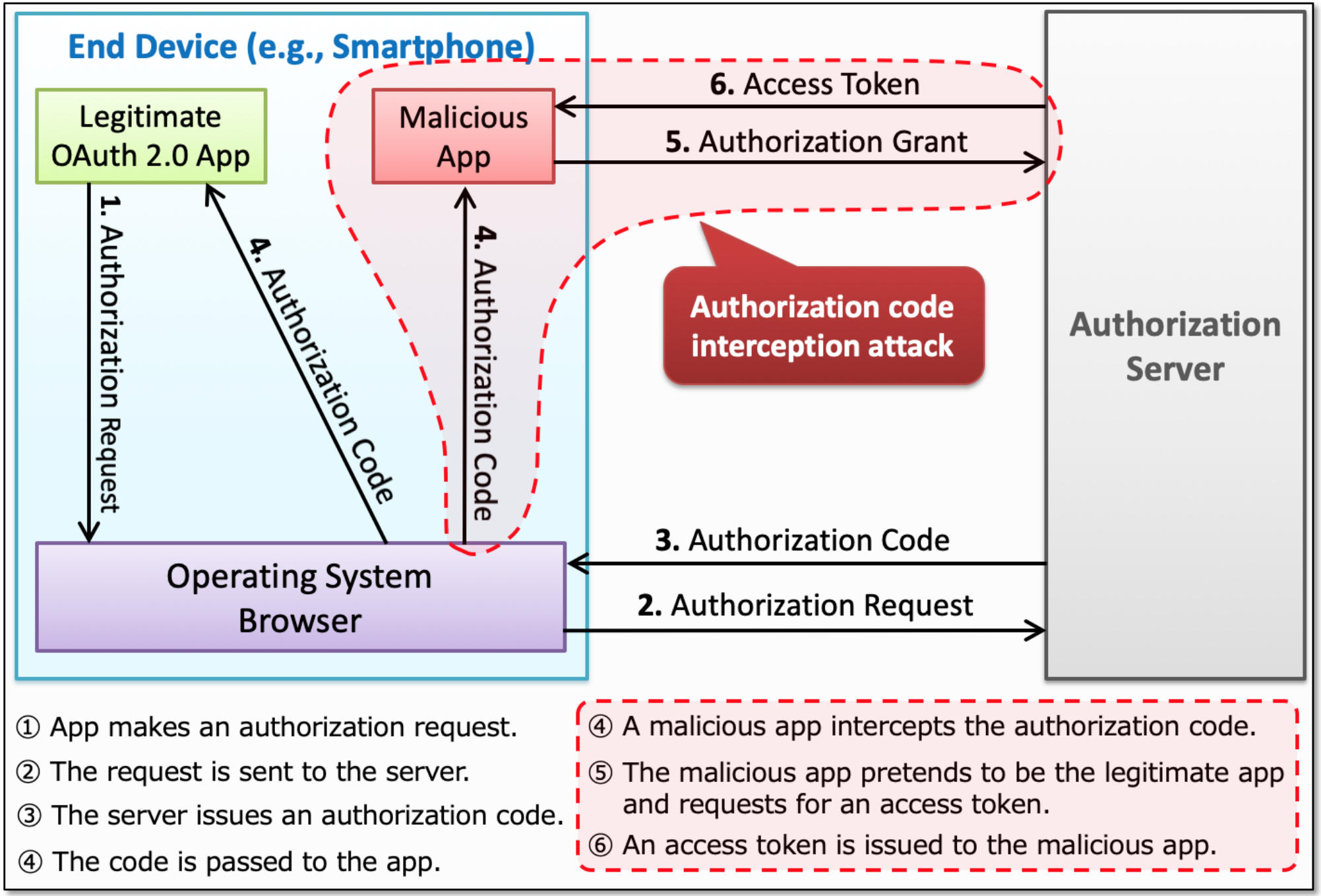
Clients SHOULD instead use the response type "code" (aka authorization code grant type) as specified in Section 2.1.1 or any other response type that causes the authorization server to issue access tokens in the token response, such as the "code id_token" response type. This allows the authorization server to detect replay attempts by attackers and generally reduces the attack surface since access tokens are not exposed in URLs. It also allows the authorization server to sender-constrain the issued tokens (see next section).

Authorization code grant



itverket
EN DEL AV OMEGAPOINT

Authorization Code Interception Attack



① App makes an authorization request.
② The request is sent to the server.
③ The server issues an authorization code.
④ The code is passed to the app.

④ A malicious app intercepts the authorization code.
⑤ The malicious app pretends to be the legitimate app and requests for an access token.
⑥ An access token is issued to the malicious app.

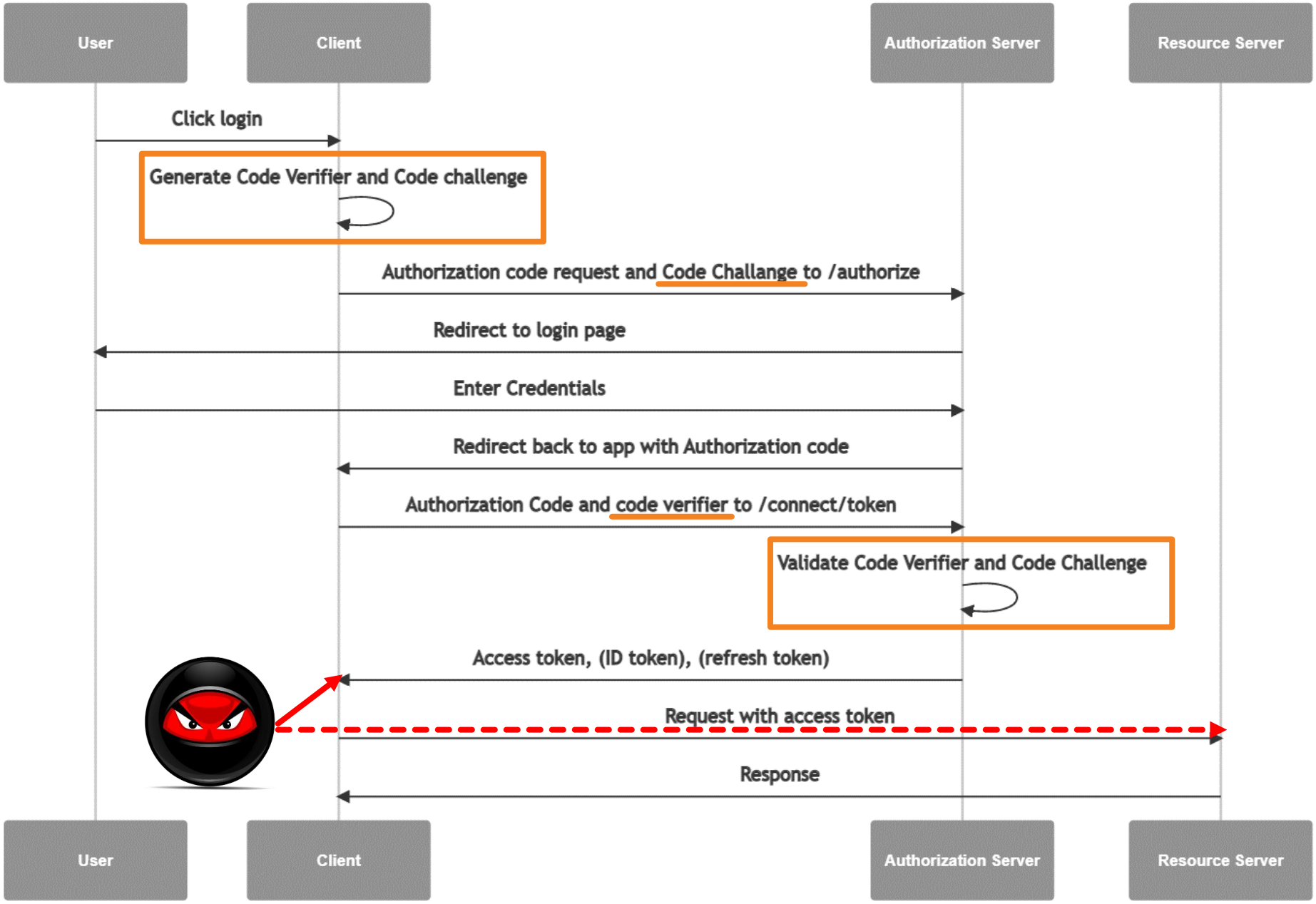
itverket
EN DEL AV OMEGAPOINT

2.1.1. Authorization Code Grant

Clients MUST prevent authorization code injection attacks (see Section 4.5) and misuse of authorization codes using one of the following options:

* Public clients MUST use PKCE [RFC7636] to this end, as motivated in Section 4.5.3.1.

* For confidential clients, the use of PKCE [RFC7636] is RECOMMENDED, as it provides a strong protection against misuse and injection of authorization codes as described in Section 4.5.3.1 and, as a side-effect, prevents CSRF even in presence of strong attackers as described in Section 4.7.1.



Authorization code grant with PKCE (Proof Key for Code Exchange)

Code Verifier:
random string

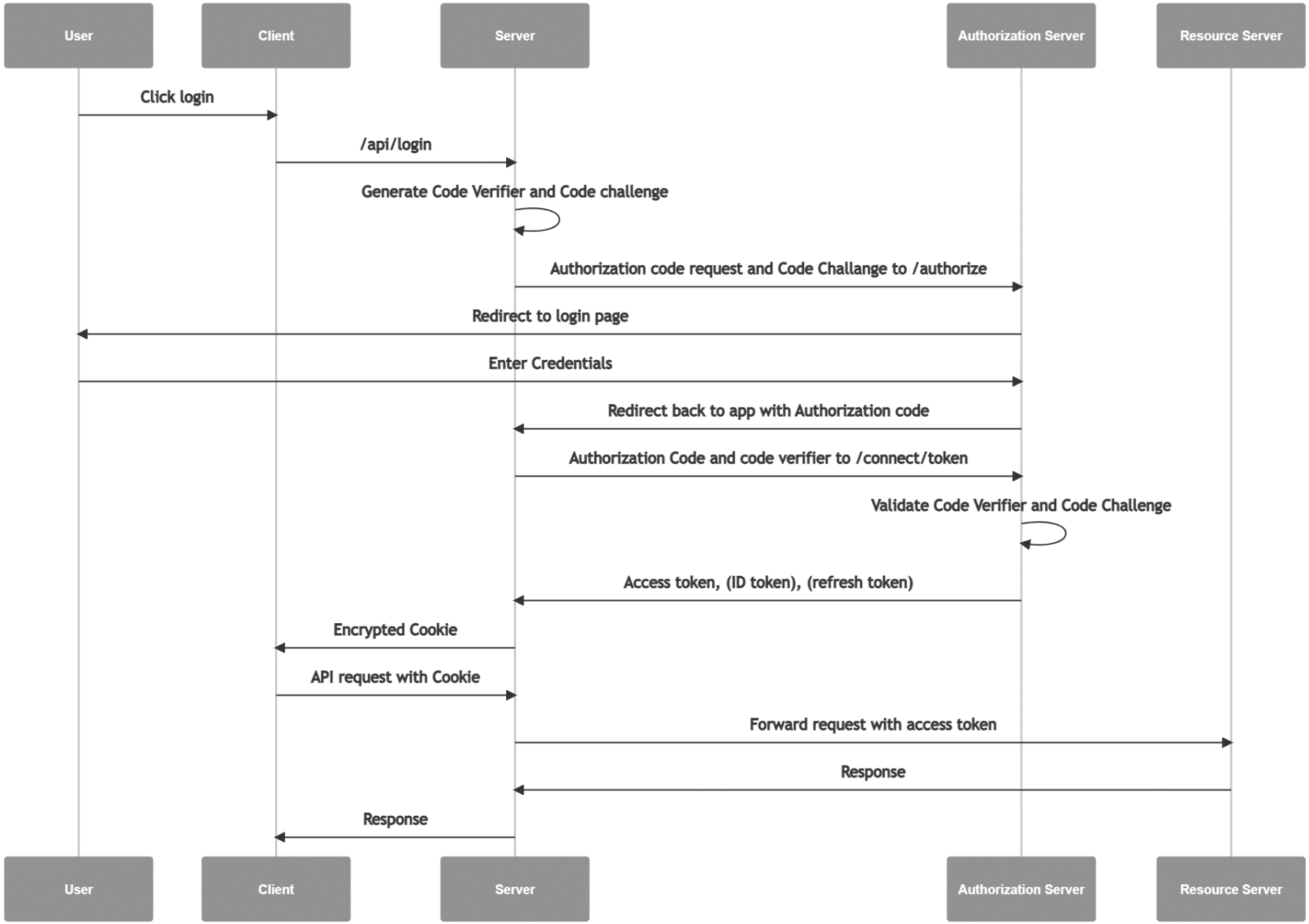
Code challenge:
base64(sha256(codeVerifier))

itverket
EN DEL AV OMEGAPOINT

BFF - Backend For Frontend

- Dedicated backend
- Handles communication with authorization server
- Cookie session with frontend
- Nothing remotely sensitive touches the client, ever

itverket
EN DEL AV OMEGAPOINT

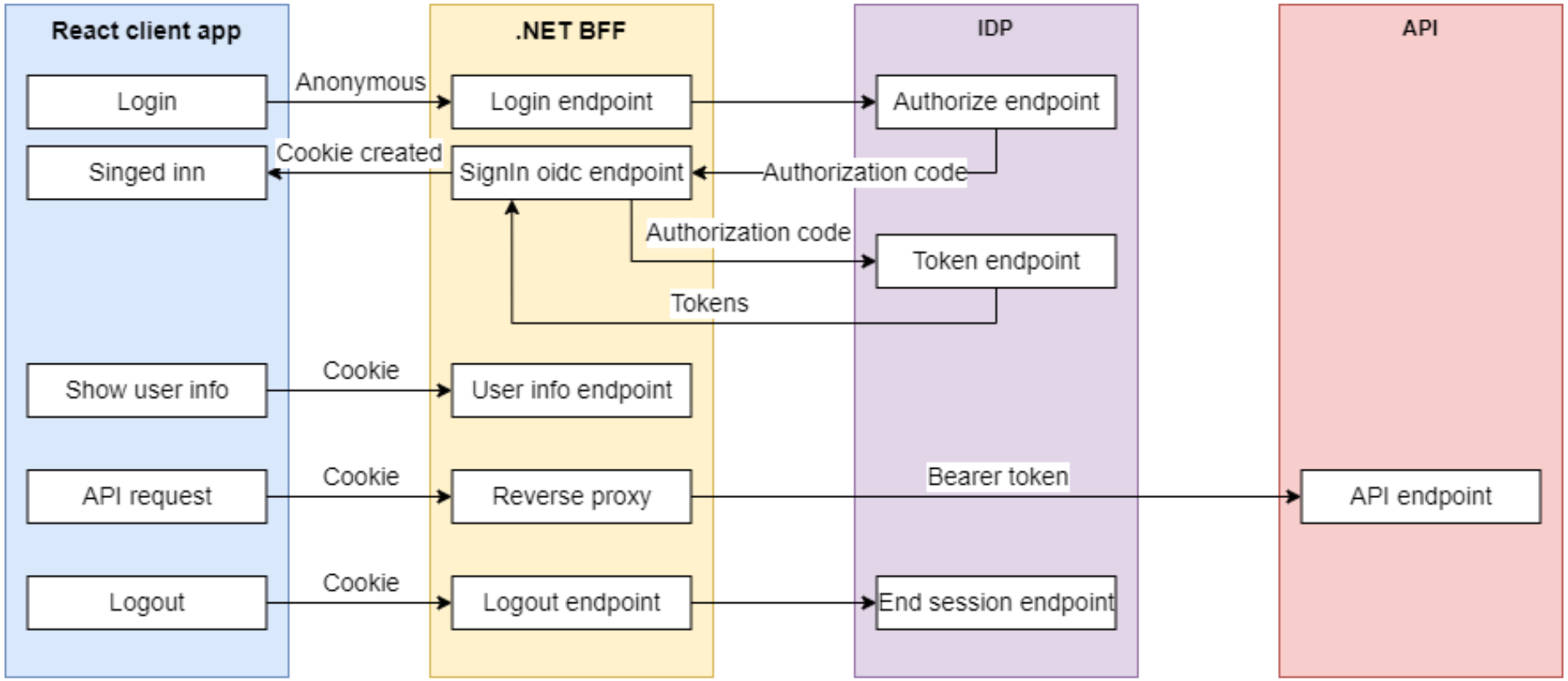


Authorization code flow with PKCE and BFF

This is best practice

itverket
EN DEL AV OMEGAPOINT

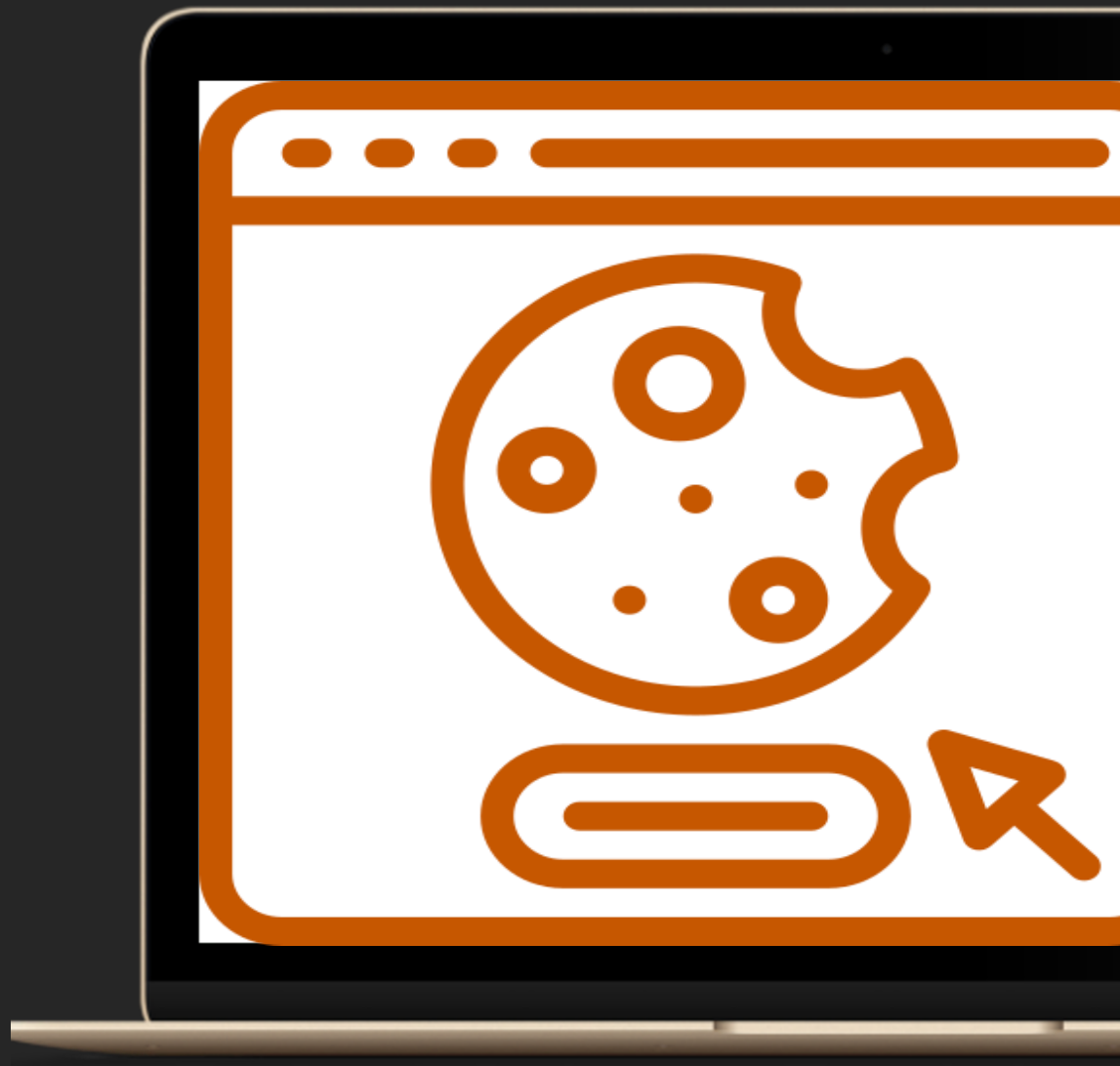
BFF Architecture



itverket
EN DEL AV OMEGAPOINT

Cookies

- Session bound
- Same site: Strict
- Http only
- Encrypted
- Sliding expiration



Logout

```
POST /connect/endsession HTTP/1.1
Host: authorization-server.com

id_token_hint=xxxxxxxx
&post_logout_redirect_uri=xxxxxxx
```

Refresh token

- Can be exchanged for a new access token
- Obtained by using the scope “offline access”
- Single use only
  - New refresh token should be returned when old one is used
- Longer lived that access token

Refresh token request

```
POST /oauth/token HTTP/1.1
Host: authorization-server.com

grant_type=refresh_token
&refresh_token=xxxxxxxxxxx
&client_id=xxxxxxxxxx
&client_secret=xxxxxxxxxx
```

Discovery enpoint

- Metadata about the Identity Provider (IDP)
- Open endpoint: /.well-known/openid-configuration
- https://github.com/ITverket-Academy/oidc/blob/main/Resources/discoverySampleResponse.json

itverket
EN DEL AV OMEGAPOINT

Do’s and Don'ts

- For users, Do use Authentication code flow with PKCE (Proof Key for Code Exchange)
- For applications, Do use Client credential flow
- Do use BFF (Backend For Frontend)
- Do protect against CSRF
- Do use content security policy (XXS protection)

- Don’t handle tokens in browser
- Don’t use implicit flow

Spørsmål?

Takk for meg