



Keeping your applications secure by evolving OAuth 2.0 and OpenID Connect

Alexander Schwartz | Keycloak Maintainer
FOSDEM (Brussels, BE) | 2025-02-01

The Epic Quest of Single Sign On



Share your identity and delegate resource access to selected services.

The Epic Quest of Single Sign On

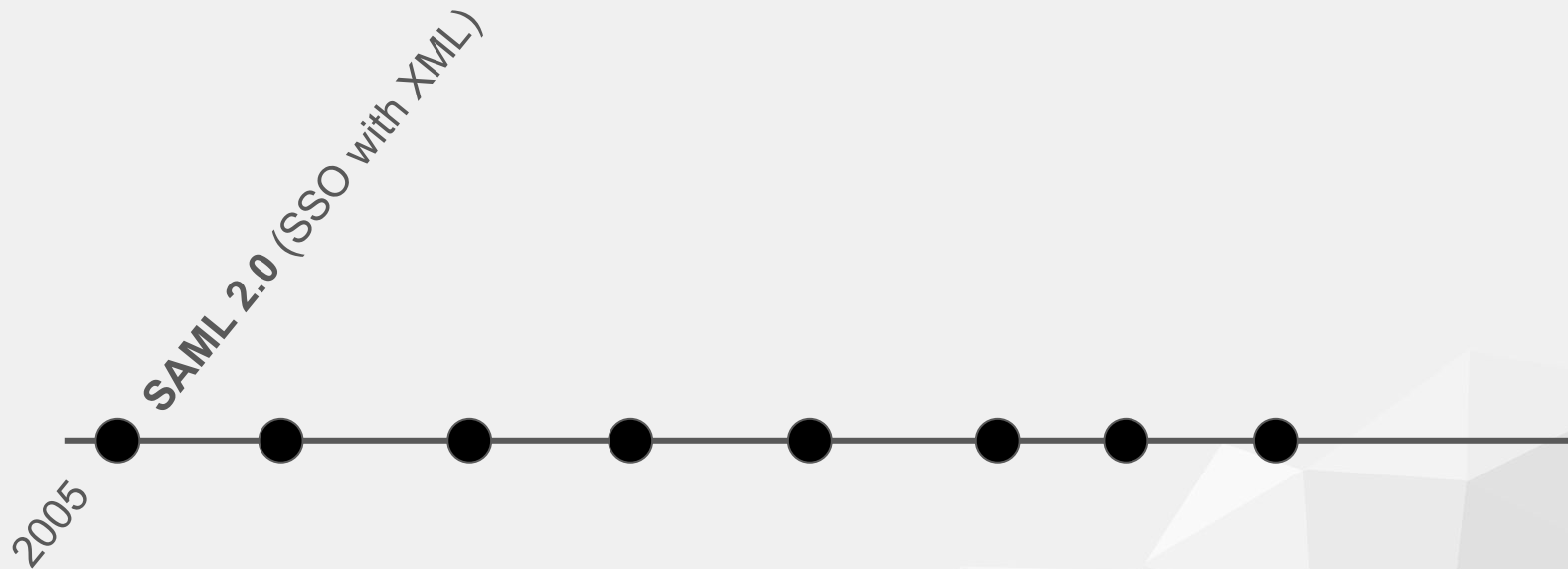


Share your identity and delegate resource access to selected services.

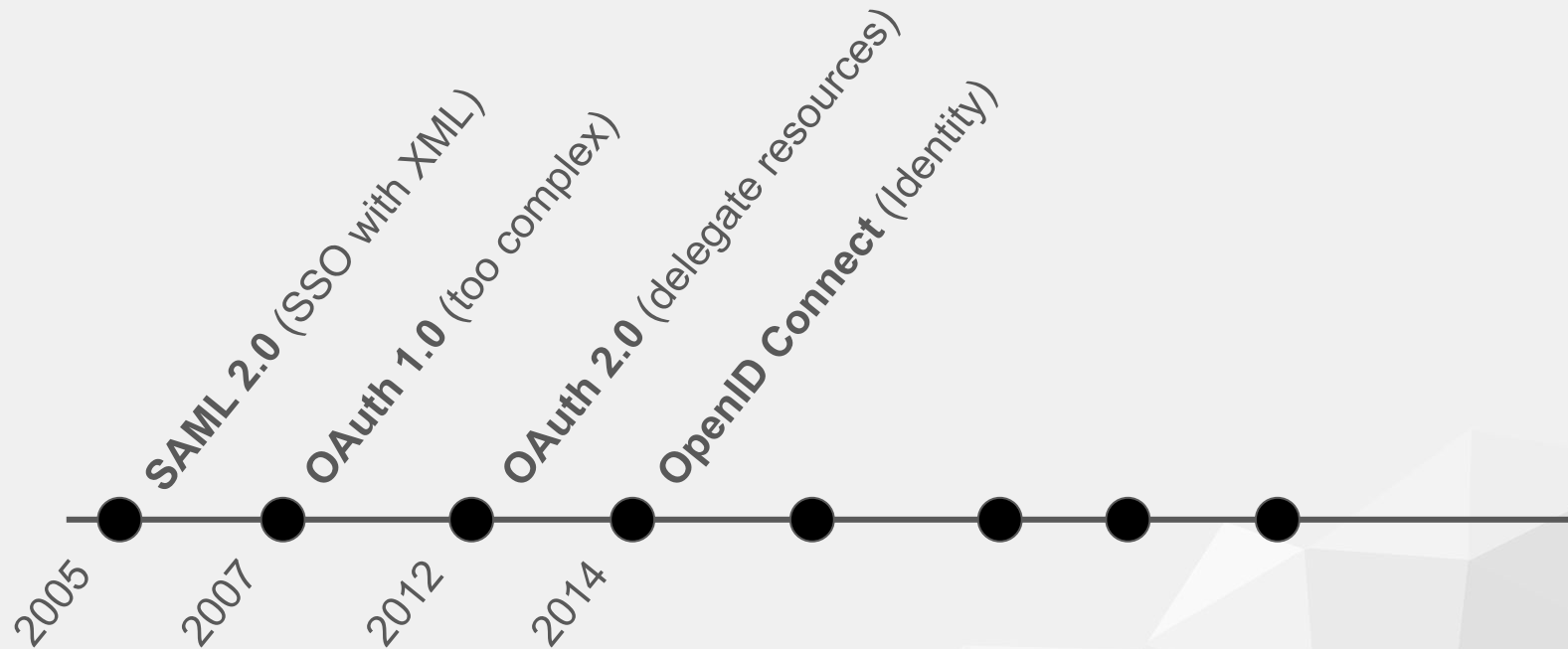


Keep your credentials secure.
Let applications operate on your data when permitted.

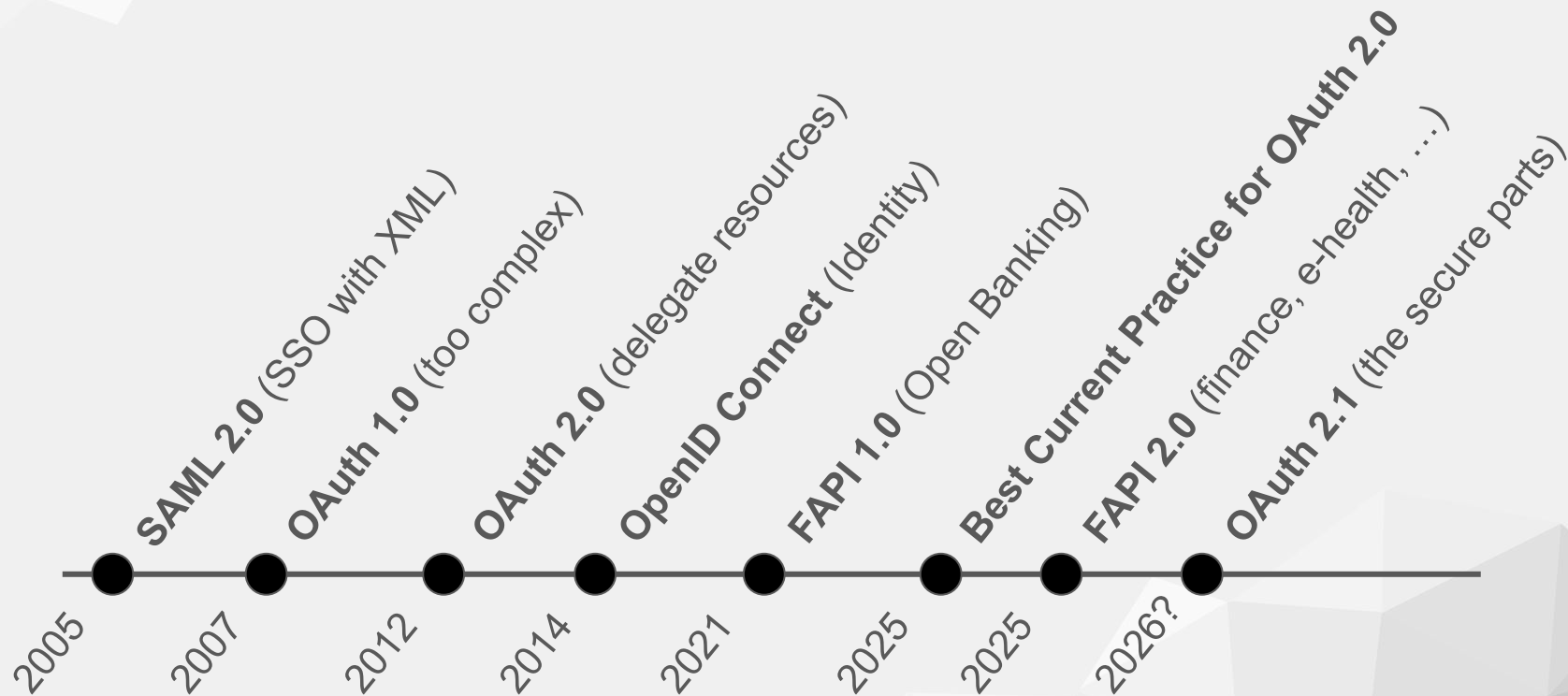
Evolution of Single Sign On and delegation



Evolution of Single Sign On and delegation



Evolution of Single Sign On and delegation



Security Assumptions FAPI 2.0

Attacker personas: what is assumed to be secure, and what can be compromised.

Assumed to work and not compromised/out of scope:



Transport layer security (TLS)



Sharing public keys (JWKS)



Browsers and Endpoints



Identities and session management

Attacker Models FAPI 2.0

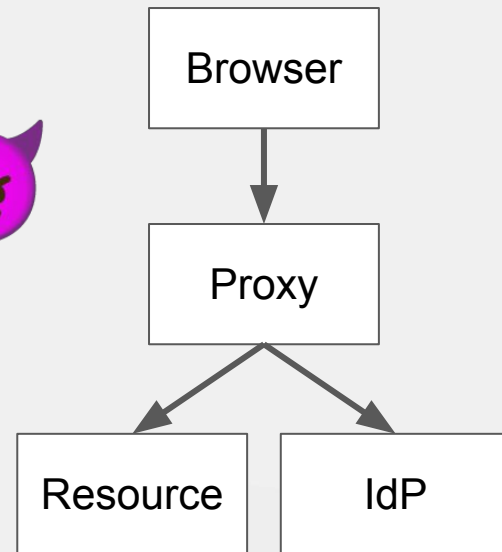
A1: Web attacker: Calls URLs on its own, makes users click links, but can't break encryption.

A2: Spying on the network, but can't break encryption.

A3a: Read authorization requests in the browser.

A5: Read proxy/resource owner request logs, but can't read responses.

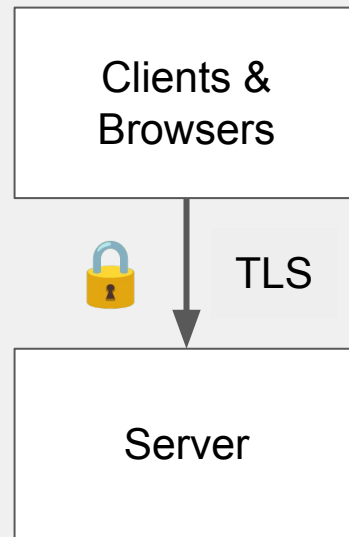
...



Secure your transport layer

- TLS 1.2+
- TLS certificate checks
- DNSSEC
- Secure TLS ciphers
- HSTS to avoid downgrading
- ...

TL;DR: Apply best practices to avoid breaking the out-of-scope assumptions earlier.



Simplified OAuth 2.0 Authorization Code Flow

```
GET authorization_endpoint + ?redirect_uri=...&prompt=login..."
```

```
GET redirect_uri "?...session_state=...code=..."
```

```
POST code and other parameters to token_endpoint
```

```
response with ID token, access token, refresh token, ...
```

Refreshing tokens and calling APIs

`POST refresh_token to token endpoint`

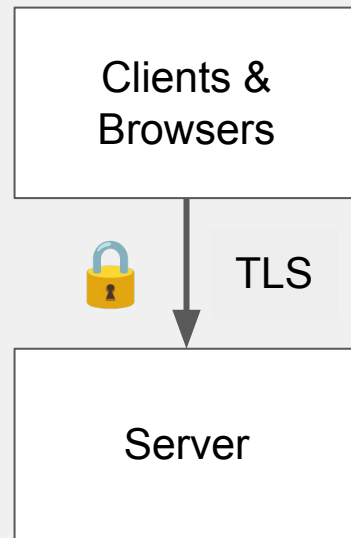
`response with ID token, access token, refresh token, ...`

`Call API endpoint with access token as "Authorization: Bearer ..." header`

`Receiving API response`

Use OAuth best practices

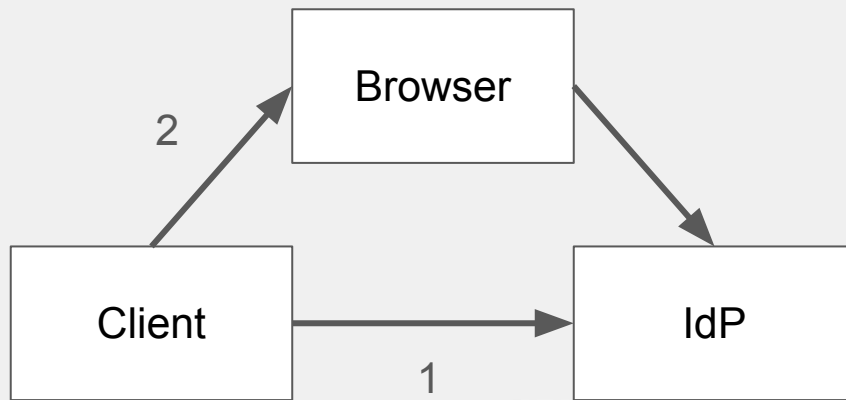
- TLS on all endpoints
- No resource owner password grant
- No wildcards in redirect URIs
- Private Key JWT Client Authentication (= no public clients)
- Pushed Authorization Requests (PAR)
- PKCE with S256
- Sender Constrained Tokens (mTLS or DPoP)



Use PAR for confidential auth flow parameters

Use Pushed Authorization Requests to prevent passing information via the URL to the browser. This ensures confidentiality and integrity. *Works only for confidential clients.*

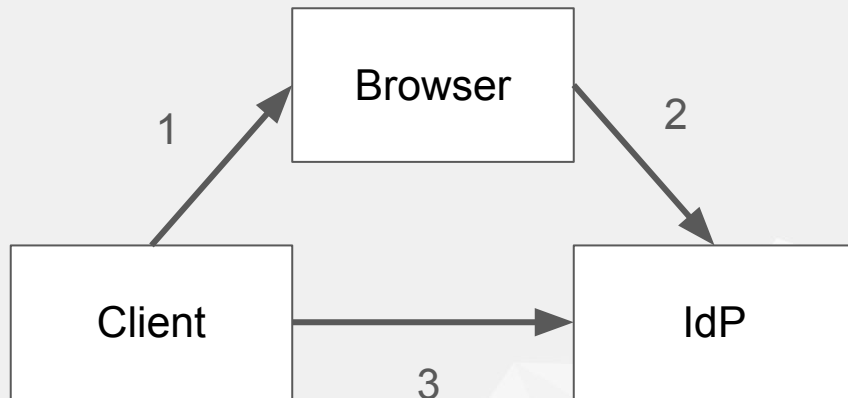
1. Send Information to IdP authenticated with client credentials and receive an ID
2. Forward it to the browser



Use PKCE to secure Authorization Code

Use Proof Key for Code Exchange to prevent others using the authorization code.

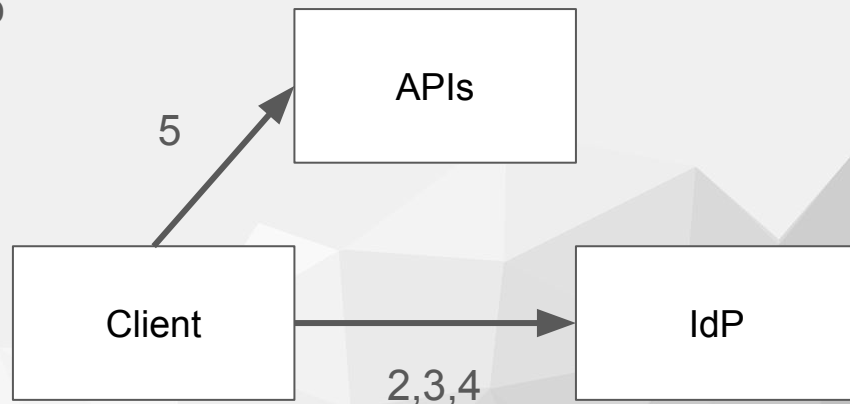
1. Send a code challenge at the start of the authorization code flow
2. Forward challenge to the IdP by the browser
3. Send a code verifier when requesting code-to-token exchange



Sender Constrained Tokens: DPOP

Use Demonstrating Proof-of-Possession (DPoP) with an ephemeral client key pair to secure all steps.

1. Create an ephemeral key pair (for SPAs use the WebCrypto API)
2. Use the authorization code flow with PKCE or DPOP to prevent spoofing
3. Use the DPOP key pair for the code-to-token flow to bind the access token (all clients) and refresh token (public clients) to the keypair to prevent misuse of stolen tokens
4. Use DPOP for all future token refreshes
5. Optional: Use DPOP for APIs (with “Authorization: DPOP ...”)



Sender Constrained Tokens: DPoP

HTTP method, URI, DPoP proof, access token and optional nonce need to align!

```
GET /protected-resource HTTP/1.1  
Authorization: DPoP <Access-Token>  
DPoP: <DPoP-Proof>
```

```
GET /protected-resource HTTP/1.1  
Authorization: DPoP <Access-Token>  
DPoP: <DPoP-Proof-with-Nonce>
```

```
HTTP/1.1 400 Bad Request  
DPoP-Nonce: <Nonce>  
  
{  
  "error": "use_dpop_nonce",  
  ...  
}
```

```
HTTP/1.1 200 OK  
  
...
```


Sender Constrained Tokens: mTLS

Bind all issued tokens to the the TLS client certificate.

1. Client connects to the IdP to acquire tokens. Tokens contain a hash of the client's certificate
2. All consumers of the access tokens can validate the hash as it is stored as a claim in the token.

Challenges:

- Only works for confidential clients
- Connecting to the IdP with mTLS
- All APIs that the client calls need to support mTLS

Keycloak is an Open Source Identity und Access Management System



First Commit 2013-07-02



Cloud Native Computing Foundation
Incubating project since April 2023



Apache License, Version 2.0



33k GitHub stars



Keycloak supports several standards

OpenID Connect, OAuth, SAML, ...

Including:

- **FAPI 2.0 Security Profile**
- **The OAuth 2.1 Authorization Framework (Draft)**

Demo: Let's enforce the FAPI 2.0 Security Profile

Keycloak Client Profiles

[Realm settings](#) > Client policies

Test

Realm settings are settings that control the options for users, applications, roles, and groups in the current realm. [Learn more](#)

☒ Enabled

Action ▾

- <
- General
- Login
- Email
- Themes
- Keys
- Events
- Localization
- Security defenses
- Sessions
- Tokens
- Client policies >

Profiles Policies

Configure via: ☒ Form view ☐ JSON editor

Create client profile


1 - 4 ▾ < >

Name	Description
fapi-2-security-profile Global	Client profile, which enforce clients to conform 'FAPI 2.0 Security Profile Final' specification.
fapi-2-message-signing Global	Client profile, which enforce clients to conform 'FAPI 2.0 Message Signing Final' specification.
fapi-2-dpop-security-profile Global	Client profile, which enforce clients to conform 'FAPI 2.0 Security Profile Final' using DPoP specification.

Keycloak Adding a client policy

[Realm settings](#) > [Client policies](#) > [Policy details](#)

My secure clients

 Enabled

Action ▼

Name *

My secure clients

Description

Apply FAPI 2.0 security profile to all clients that have a client-role "secure".

Save

Reload

Conditions ⓘ

[+ Add condition](#)

[client-roles](#) ⓘ 

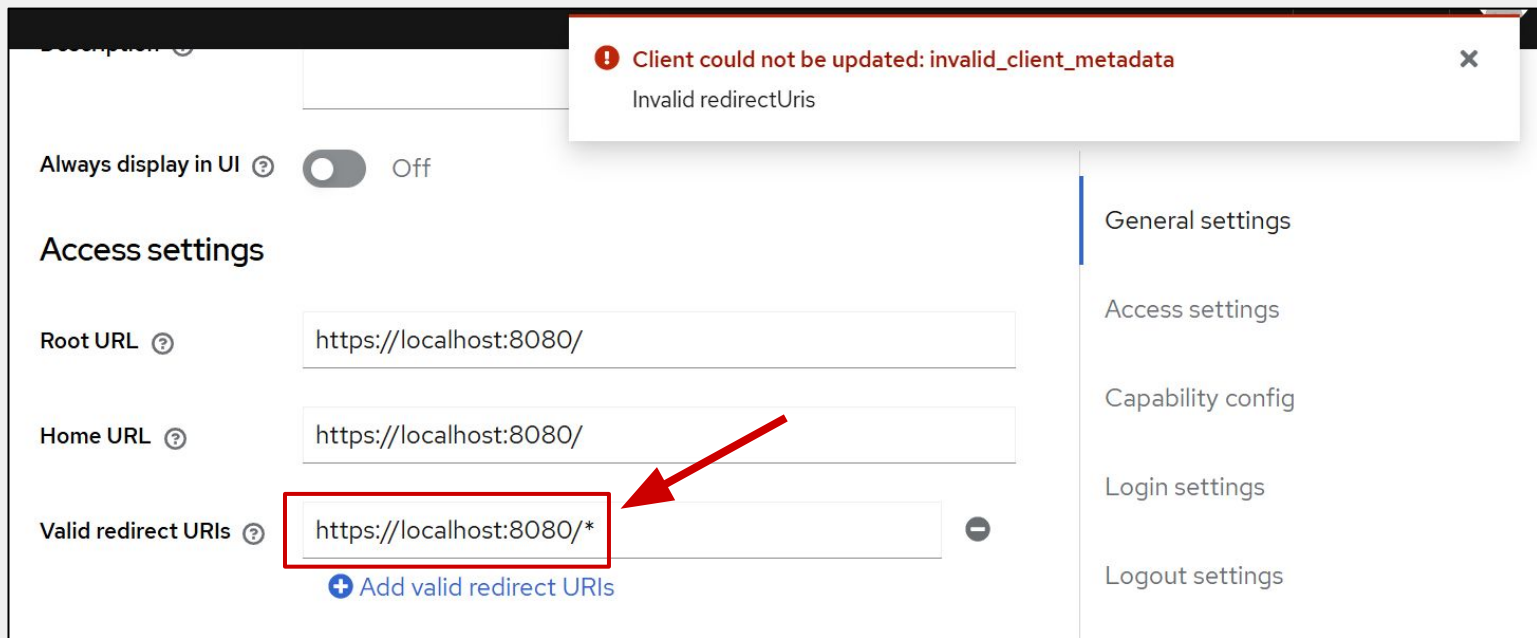
Client profiles ⓘ

[+ Add client profile](#)

[fapi-2-security-profile](#) ⓘ 

Validation of the policy in all actions

Policy is checked on client update, login, token issuing. See an example for client update below where a wildcard redirect URI is not allowed.



The screenshot displays the Keycloak Admin Console interface. At the top, a red error message box states: "Client could not be updated: invalid_client_metadata" with the sub-message "Invalid redirectUri". Below this, the "Access settings" section is visible. It includes a toggle for "Always display in UI" which is currently "Off". The "Valid redirect URIs" field contains the value "https://localhost:8080/*", which is highlighted by a red rectangle and a red arrow. To the right of this field is a minus icon. Below the field is a blue link that says "+ Add valid redirect URIs". On the right side of the console, a sidebar lists navigation options: "General settings", "Access settings", "Capability config", "Login settings", and "Logout settings".

Client could not be updated: invalid_client_metadata
Invalid redirectUri

Always display in UI ☐ Off

Access settings

Root URL

Home URL

Valid redirect URIs

[+ Add valid redirect URIs](#)

General settings
Access settings
Capability config
Login settings
Logout settings

Client-specific configurations

Some settings are also available on a per-client level.

[Clients](#) > Client details

Test OpenID Connect

Clients are applications and services that can request authentication of a user.

Settings

Roles

Client scopes

Sessions

Advanced

Events

General settings

Client ID ^{*} ?

test

Require PKCE ?

☒ On

PKCE Method ?

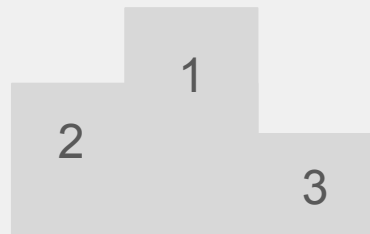
S256 ▼

Require DPoP bound tokens ?

☐ Off

OAuth 2.1 (draft) vs. OAuth 2.0

- No wildcard redirect URLs or URL fragments.
- No localhost and loopback addresses.
- Must use PKCE.
- No implicit grant.
- No Resource Owner Password Grant.
- No Bearer token in query parameters.
- ...



➡ Moving target as it is not released yet, but very much aligned with FAPI 2.0 and “Best Current Practice for OAuth 2.0 Security” (RFC 9700)

Tools to help you

- **mod_auth_openidc** / OAuth 2 and OpenID Connect for Apache 2.x httpd server
Supports FAPI 2.x (including DPoP)
https://github.com/OpenIDC/mod_auth_openidc
- **openid-client** / OAuth 2 and OpenID Connect Client API for JavaScript Runtimes
<https://github.com/panva/openid-client>
Certified for FAPI 2.0 (including DPoP)
- **Nimbus OAuth SDK** / Framework-agnostic OAuth 2 and OpenID Connect for Java
<https://connect2id.com/products/nimbus-oauth-openid-connect-sdk>

How to evolve OAuth security in your setup

1. Pick one security feature to enforce and educate developers (for example PKCE)
2. Wait for developers to complete their tests.
3. Update clients configurations in the IdP one-by-one.
4. Repeat for other features (like implicit grant, DPoP, etc.).



Security is a journey!

Brownouts to speed up the process

“deliberate introduction of temporary outages to a system, API or feature that is being phased out.”

1. Explain a set of best practices to your engineers and how to test them in staging.
2. Let IdP enforce best practices Monday between 9 and 10.
3. Set a deadline to enforce them permanently.
4. Repeat with the next set.



Keeping applications secure



Level up clients, IdPs and APIs.



Enforce policies to keep your organization aligned.



Share your successes and lessons learned with the community.

Case Studies

The Hitachi logo, consisting of the word "HITACHI" in a bold, black, sans-serif font, centered within a white rectangular box with a thin black border.

Hitachi Ltd. used Keycloak to make financial grade security easier

The OpenTalk logo, featuring the word "Opentalk" in a yellow, cursive script font, set against a dark blue rectangular background.

OpenTalk achieves versatile and compliant user authentication with Keycloak

The BRZ logo, with the letters "BRZ" in a white, bold, sans-serif font, centered on a solid red rectangular background.

BRZ migrated the Austrian Business Service Portal with 2M+ users to Keycloak

<https://www.keycloak.org/case-studies>

Links

- **Keycloak**
<https://www.keycloak.org/>
- **Case Studies**
<https://www.keycloak.org/case-studies>
- **KeycloakCon @ KubeCon EU**
<https://events.linuxfoundation.org/kubecon-cloudnativecon-europe/>
- **Using DPoP to use access tokens securely in your Single Page Applications @ FOSDEM 2026**
<https://archive.fosdem.org/2025/schedule/event/fosdem-2025-5370-using-dpop-to-use-access-tokens-securely-in-your-single-page-applications/>

Slides:




Kontakt



Alexander Schwartz
Keycloak Maintainer

alexander.schwartz@gmx.net
<https://www.ahus1.de>

 [@ahus1.de](https://twitter.com/ahus1)

 [@ahus1@fosstodon.org](https://fosstodon.org/@ahus1)