RIPE NCC
RIPE NETWORK COORDINATION CENTER

OAuth 2.0 Authentication

Adonis Stergiopoulos

Why OAuth 2.0?

What is OAuth 2.0?

OAuth 2.0 is a standard designed to access resources hosted by other web apps on behalf of a user

OAuth 2.0 provides a **standardised** and **secure** mechanism for applications to access our external APIs, without exposing user credentials.

- Security
- Flexibility
- Access Control

API Keys and OAuth 2.0

January 2025:

Introduction of API keys to authenticate updates in the RIPE Database

- Our aim is to offer solutions that enable third-party applications to securely integrate with the RIPE Database
- OAuth 2.0 is being implemented **as an alternative to using API keys** for authentication
- API keys and the OAuth 2.0 solution are complementary.

API Keys vs OAuth 2.0

Feature	API Keys	OAuth 2.0
Credentials	Managed by user	Managed by third-party applications
App identity	❌	✅
Scopes	✅	✅
Session lifetime	Lifetime is configurable up to 1 year	Access token is valid for 1 hour. Refresh token is valid for 365 days.

Authorisation Flows

Authorisation Flows

What are they?

Authorisation Flows are the process by which a **Client App** obtains authorisation from a **User** to access their protected data on a **Resource Server**.

OAuth 2.0 provides different flows depending on:

- Type of client
- Security requirements

User Interviews

1.	Ability to authenticate on behalf of other users
2.	Provide support for Web Applications
3.	Provide support for simple Command Line scripts
4.	Minimise the need for user intervention
5.	Provide support for ‘scopes’

Authorisation Flows

Authorisation Code Flow (with PKCE)

Recommended for: Web apps, Mobile apps and SPAs

- + Built-in security (*client_secret*, *redirect_uri,* PKCE)
- - Needs public URL for *redirect_uri*
- - Client Apps must support PKCE

Device Code Flow

Recommended for: Limited input devices (e.g. CLI)

- + Would work for CLI clients authenticating for themselves
- + Less development work for Client App
- - Less secure due to lack of *redirect_uri*
- - Vulnerable to phishing attacks

Client Credentials Flow (with Token Exchange)

Recommended for: Machine to machine communication

- + High level of flexibility in Token Exchange
- + Less development work for Client App
- - Not suitable for public clients
- - Significant RIPE NCC development required to ensure the Token Exchange is secure

User Requirements

	Authorisation Code Flow (PKCE)	Device Code Flow	Client Credentials Flow (with Token Exchange)
Web Applications	✅	✅	✅
Command Line Scripts	✅*	✅	✅
Authenticate on behalf of other Users	✅	❌	✅
Minimise User intervention	?	?	?
Support for scopes	✅	✅	✅

*some limitations are applicable

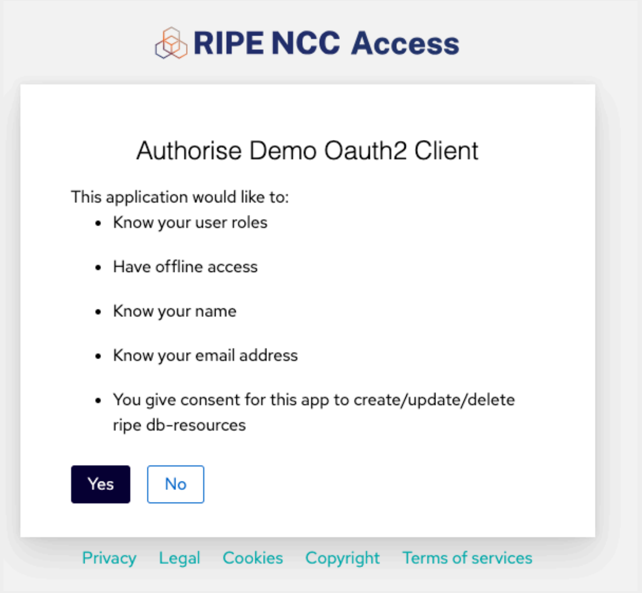
Authorisation Code Flow (with PKCE)

Recommended solution

- **Features:**
  - Supports both interactive and offline use
  - Sessions can be app-specific (Whois, RPKI, etc.)
- **User friendliness:**
  - Simply requires the User to click on a login button, provide their credentials and give consent for scope security
- **Development:**
  - Comes out of the box with Keycloak (SSO)
  - Exploring possibilities for command line scripts
- **Security:**
  - Offers PKCE for additional security

Next steps

- Phase 1 will be delivered in mid-2025


RIPE NCC Access

Authorise Demo Oauth2 Client

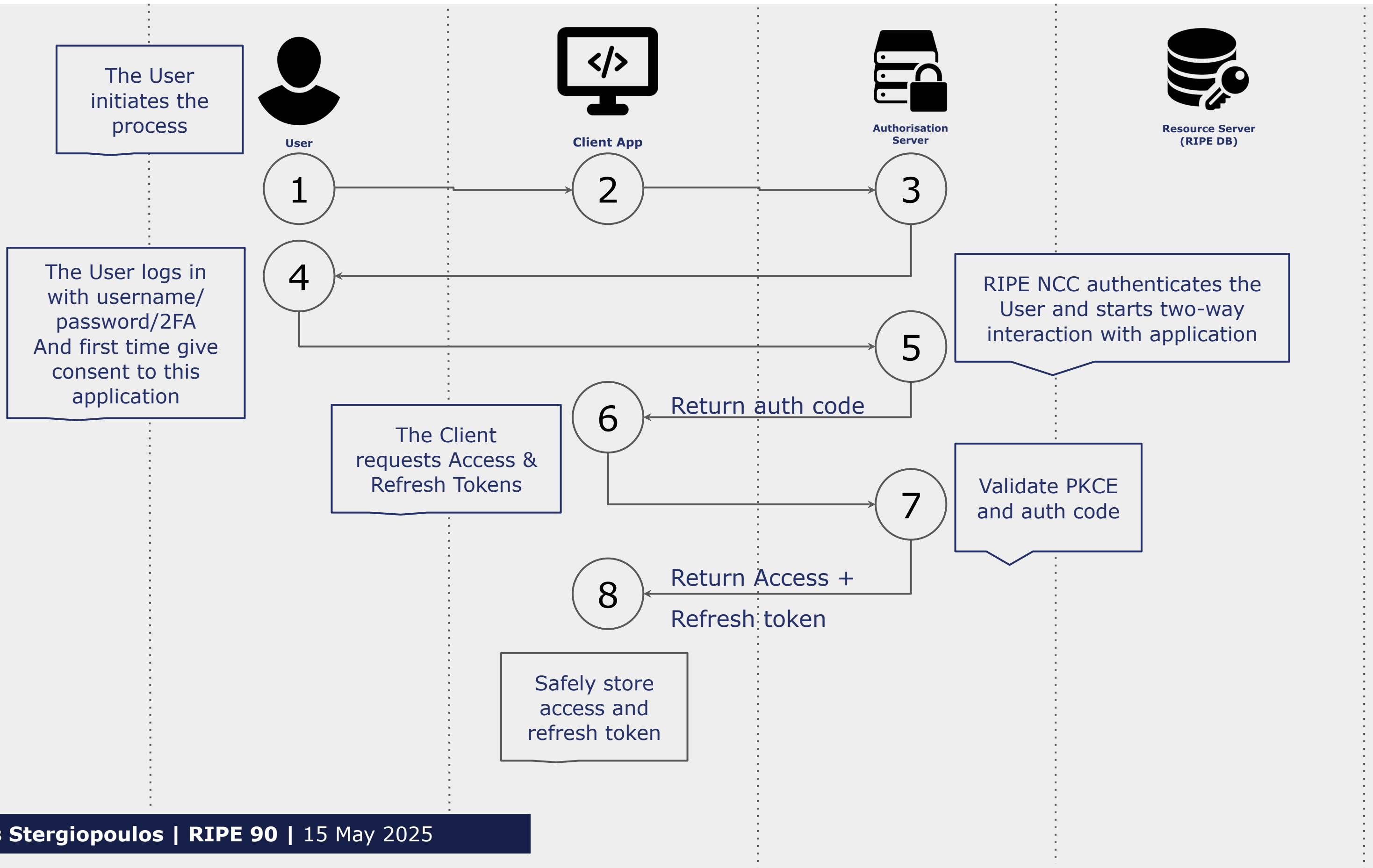
This application would like to:

- Know your user roles
- Have offline access
- Know your name
- Know your email address
- You give consent for this app to create/update/delete ripe db-resources

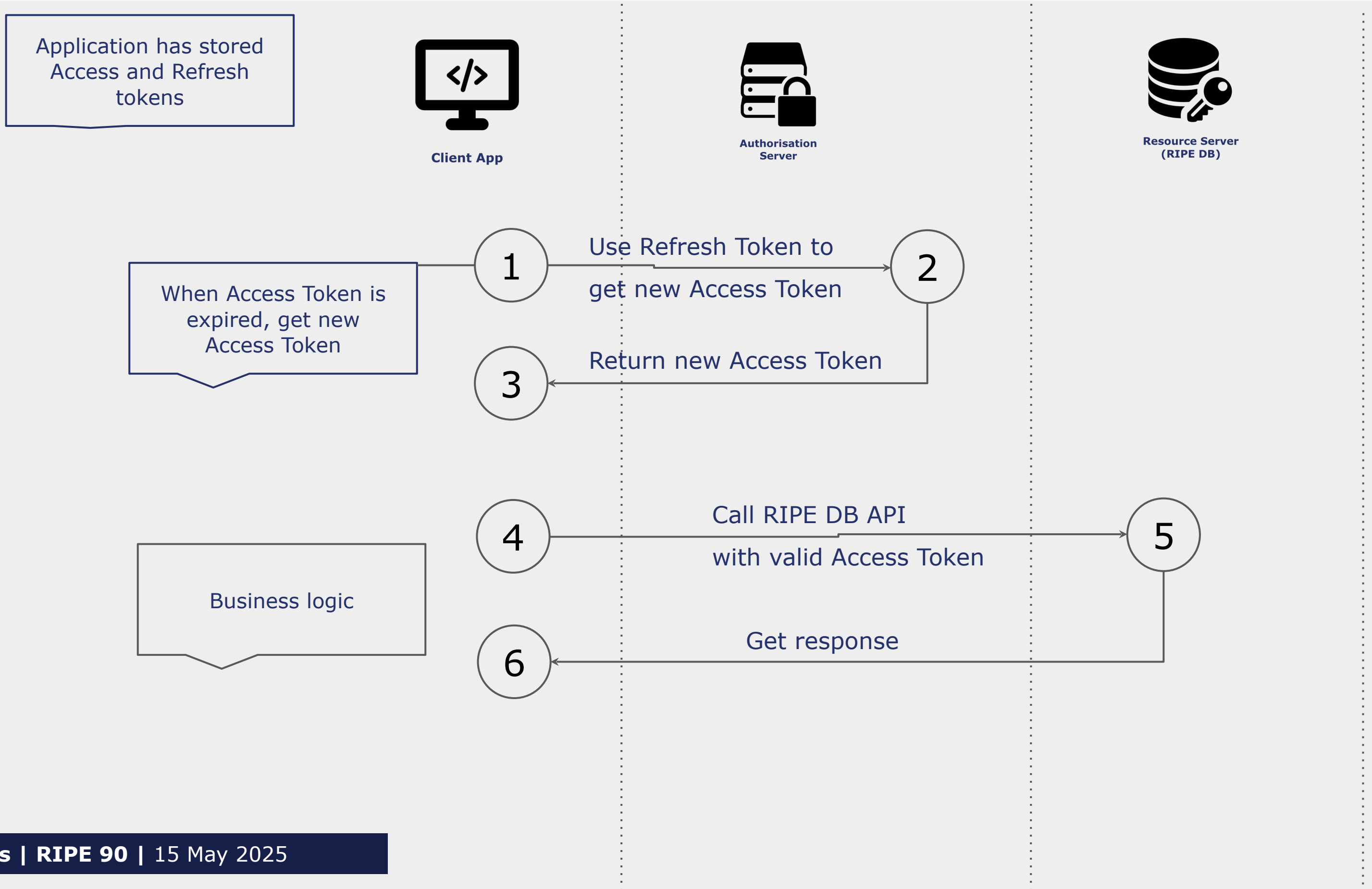
Preview of an OAuth 2.0 authorisation window

Authorisation Code Flow Architecture

Getting Tokens with Authorisation Code Flow (with PKCE)


The User initiates the process
User
Client App
Authorisation Server
Resource Server (RIPE DB)
1
2
3
4
The User logs in with username/ password/2FA And first time give consent to this application
5
RIPE NCC authenticates the User and starts two-way interaction with application
6
Return auth code
The Client requests Access & Refresh Tokens
7
Validate PKCE and auth code
8
Return Access + Refresh token
Safely store access and refresh token

Using OAuth 2.0 Tokens to Call the RIPE Database API


Application has stored Access and Refresh tokens
Client App
Authorisation Server
Resource Server (RIPE DB)
When Access Token is expired, get new Access Token
1
Use Refresh Token to get new Access Token
2
3
Return new Access Token
4
Call RIPE DB API with valid Access Token
5
Business logic
6
Get response

Tokens

Tokens in OAuth 2.0

OAuth 2.0 defines two main type of tokens:

Access tokens
(default)

- Short-lived credential
- Grants access to protected resources
- Is sent with each API request (in http header)
- Proposed lifetime: 1 hour

Refresh tokens
(optional)

- A long lived credential
- Used to obtain a new access token when that expires.
- Is NOT sent to normal API request (stored in a safe location instead)
- Proposed lifetime: 365 days

Expiration Time of Tokens

The expiry time of a token has no standards in OAuth 2.0. It’s always a trade-off between security and usability.

	Short Expiry (high security)	**Long Expiry** (high usability)
Pros	Stolen token can only be used for a short period.	Users remain logged in for extended periods, reducing the risk of disruptions if a page refresh fails
Cons	More network traffic for generating access tokens.	If token is stolen it can be used longer and it’s harder to detect that it’s stolen.

Share Your Feedback

- We need your input on the different types of use cases
- We are looking for volunteers to test our proposed Authorisation Code Flow solution
- Book a demo with us while in Lisbon or online from next week

Questions & Comments ?

astergiopoulos@ripe.net

RIPE NCC
RIPE NETWORK COORDINATION CENTER

THANK YOU!