Application Security for API Microservices; Key Principles and Flows

Ionut Balosin

www.IonutBalosin.com | ionut.balosin@gmail.com | @IonutBalosin

Agenda

Introduction

Authentication and Authorization

- OpenID Connect
- OAuth 2.0
- Identity, Access and Refresh Tokens

API and Microservices Security

- Token introspection
- JSON Web Key Set (JWKS)
- Role-Based Access Control (RBAC)

Introduction

00

Ionut Balosin

- Principal Software Architect @ Raiffeisen Bank International
- Technical Trainer
- Security Champion
- Oracle ACE Associate
- Blogger
- Speaker

www.IonutBalosin.com | ionut.balosin@gmail.com | @ionutbalosin

My Training Catalogue

- Software Architecture Essentials
- Java Performance Tuning
- Designing High-Performance, Scalable, and Resilient Applications
- Application Security for Java Developers

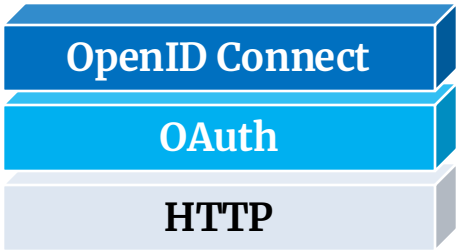
Training figures: 90+ sessions | 1000+ trainees | 1400+ hours | 12+ clients | 5+ countries
Conference figures: 40+ sessions | 15+ countries

www.IonutBalosin.com/training

Authentication and Authorization

01

Authentication and Authorization

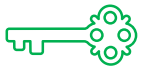


Authentication (i.e., AuthN)

Authorization (i.e., AuthZ)

OICD Token

Access Token



Refresh Token

Open Authorization

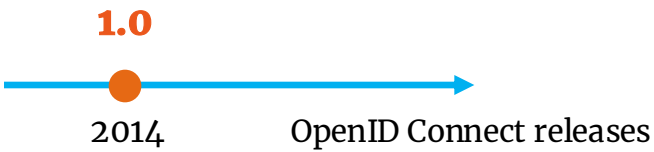
Open Authorization (**OAuth**) is a standard that allows a client (e.g., an application, a system) to **access resources** from another system on behalf of a user, without sharing passwords. It's the industry standard for **secure authorization**.

Notes:

- **OAuth 2.0** is the current official version.
- **OAuth 2.1** is still in draft with no official release date set. It builds on OAuth 2.0 by incorporating best practices and removing insecure flows.

OpenID Connect

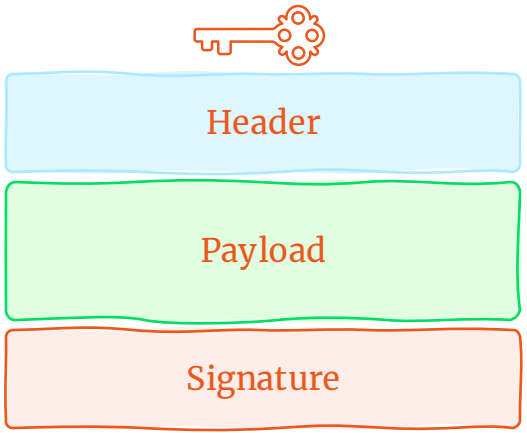
OpenID Connect (**OIDC**) is an **identity authentication protocol** built on top of OAuth 2.0 that standardizes the process for **authenticating users and authorizing their access** to digital services.



Type	OAuth 2.0 Grant	OpenID Connect Flow	Use Case
Client Credentials	Client Credentials Grant	Client Credentials Flow	Machine-to-machine (M2M) such as CLIs, daemons, backend services
Password *(deprecated)*	Password Credentials	N/A	Highly trusted apps (but in general considered insecure)
Implicit *(deprecated)*	Implicit Grant	Implicit Flow	Single-page apps (SPAs), client-side apps
Authorization Code (with PKCE) *(out of scope)*	Authorization Code Grant	Authorization Code Flow	Single-page apps (SPAs), web apps, server-side apps requiring high security
Device Authorization *(out of scope)*	Device Code Grant	Device Authorization Flow	Devices with limited input capabilities (e.g., smart TVs, IoT)

JSON Web Token (JWT)

JSON Web Token (JWT) is part of the JSON Object Signing and Encryption (**JOSE**) family of protocols. The most common format for JWTs is **JSON.**



Token Claims:

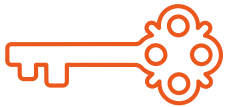
alg (signing algorithm), *typ* (token type),etc.

sub (subject), *exp* (expiration), *iat* (issued at),
sn (surname), *givenName* (first name), *email*, *roles*, *permissions*,etc.

RSASHA256 (
  base64UrlEncode(*header*) + "." + base64UrlEncode(*payload*)
)

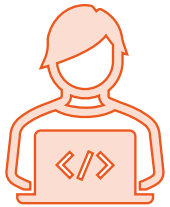
The **Signature** is added by the Identity Provider (IdP) or Authorization Server based on the Header and Payload to ensure the integrity and authenticity of the token.

JSON Web Token (JWT) Types



Access Token

- Short-lived (e.g., 5 to 30 min)



Identity Token

- Short-lived (e.g., 5 to 30 min)

Refresh Token

- Long-lived (e.g., days to weeks)

Demo Time

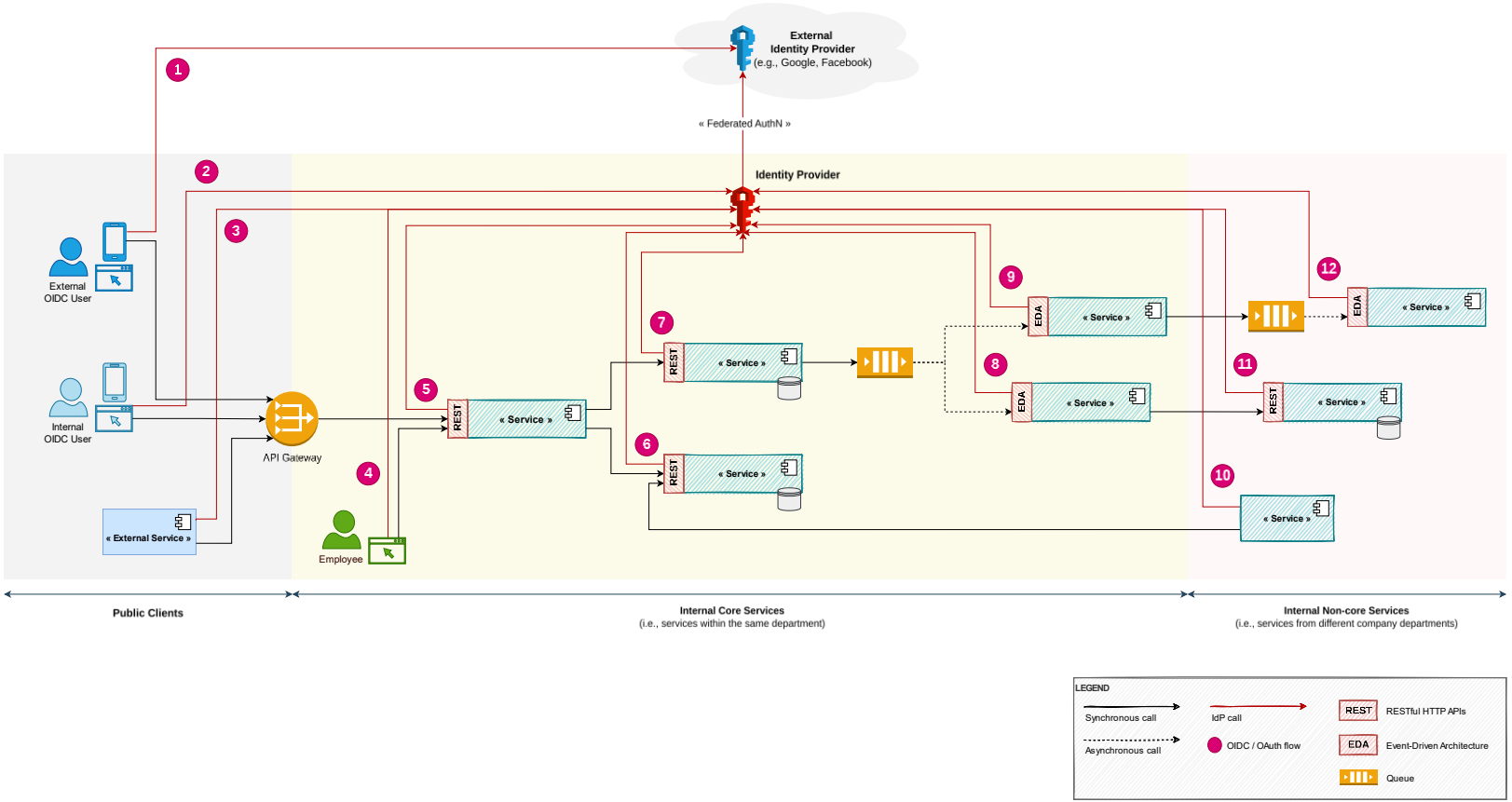
Open the file **02-PRACTICE-Authentication-and-Authorization.md** and follow the steps in the **Hands-on Demo** section.

Source: [https://github.com/ionutbalosin/java-application-security-practices]

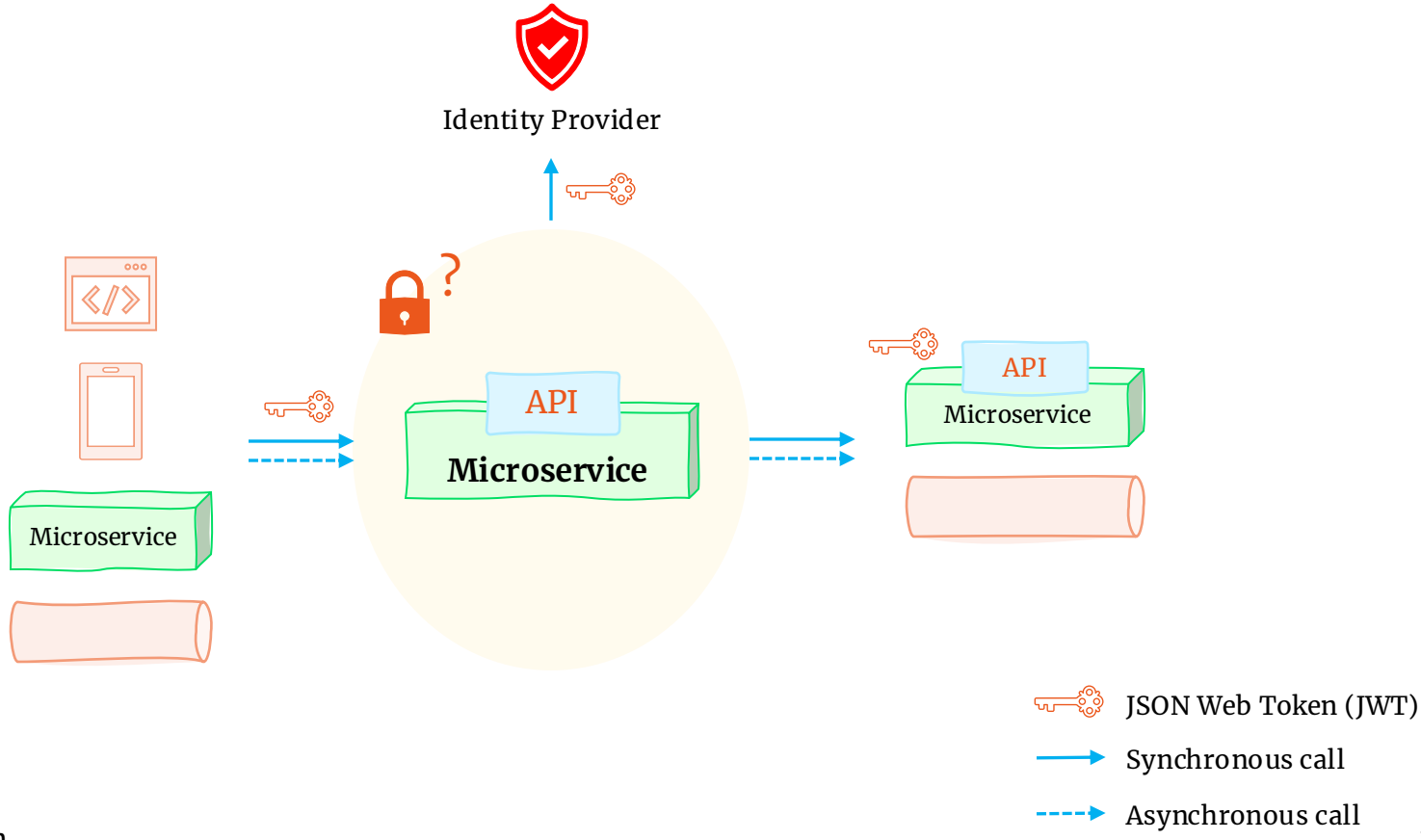
API and Microservices Security

02

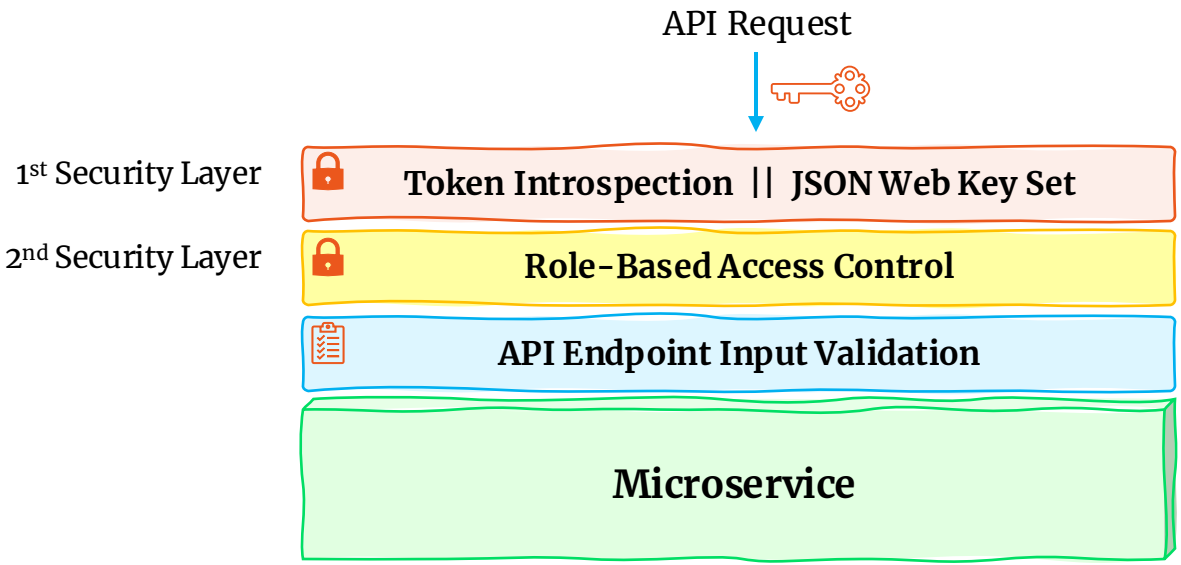
A Comprehensive Secure Software Architecture Blueprint



Streamlined Microservice Interactions



Key Security Layers for API Microservices



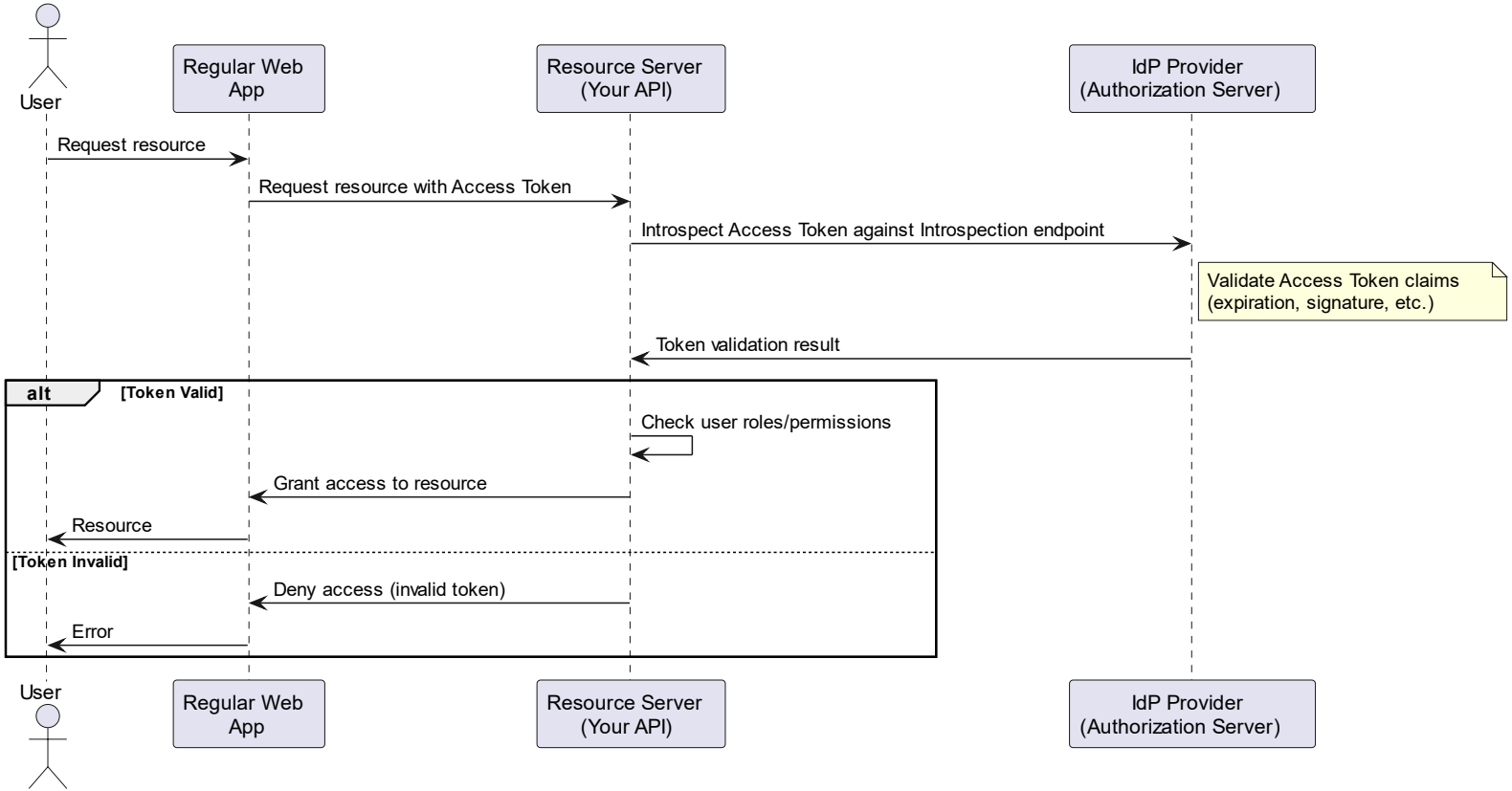
Token Introspection

Token introspection is used to verify the validity of an access token against the introspection endpoint provided by the authorization server before granting access to protected resources.

It is mainly used at critical points of the application (entry points and critical resources) because it is a very accurate way to determine if a token is still valid (and has not been revoked on the server) compared to JWKS.

Nevertheless, it also introduces higher latency since it requires a call to the Identity Provider for each resource request.

Token Introspection



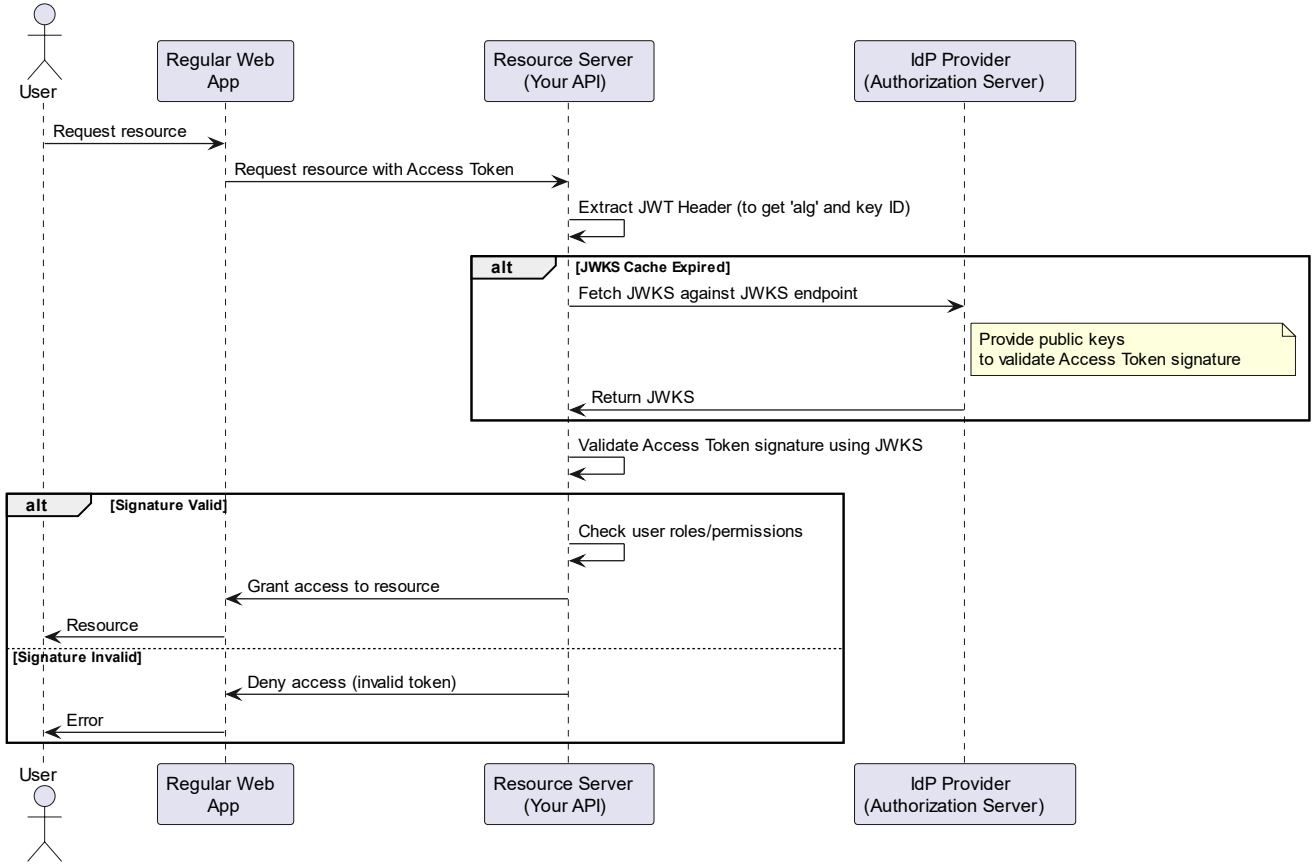
JSON Web Key Set (JWKS)

JSON Web Key Set (**JWKS**) is a set of public keys exposed by an authorization server that allows clients to retrieve these keys to verify the signatures of incoming JWTs.

JWKS is ideal for environments where latency is critical, as it allows the resource server to locally validate the signature of a JWT without needing to make a round trip to the authorization server.

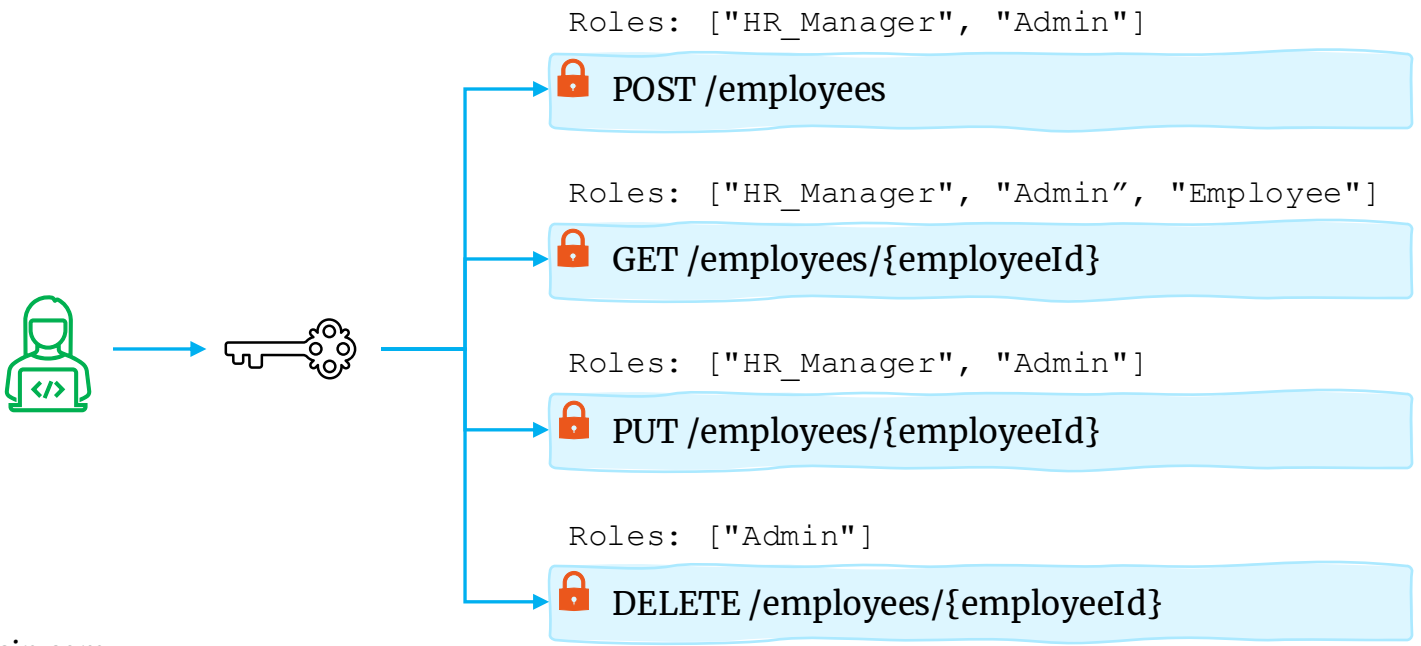
While JWKS is less accurate than token introspection in verifying a token's status, it remains adequate for many applications.

JSON Web Key Set (JWKS)



Role-Based Access Control (RBAC)

Role-Based Access Control (**RBAC**) is a security model that allows access to resources based on roles, ensuring that each entity requesting access has the minimum necessary roles to perform its designated functions.



Demo Time

Open the file 03-PRACTICE-API-and-Microservices-Security.md and follow the steps in the **Hands-on Demo** section.

Source: [https://github.com/ionutbalosin/java-application-security-practices]

Thank You

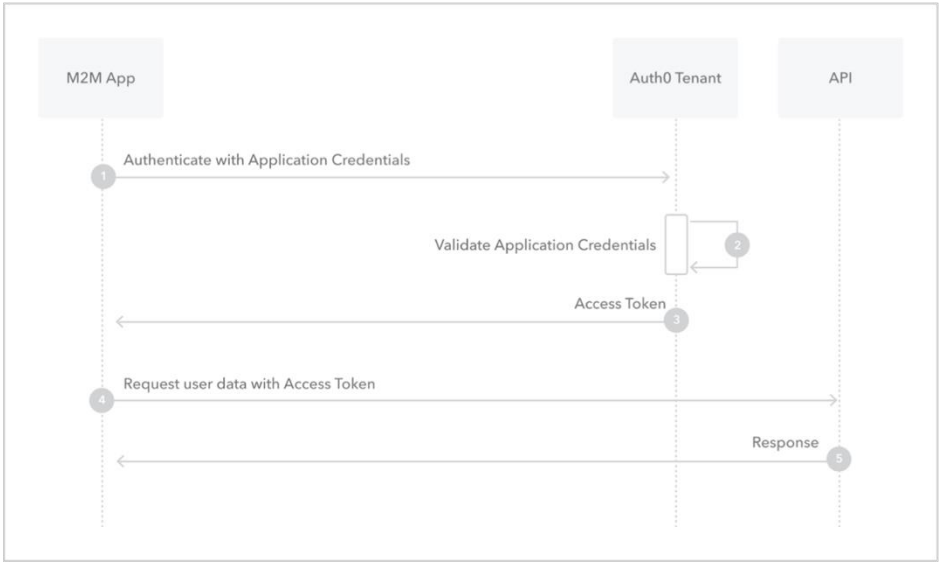
Further Readings

Websites/Blogs

https://ionutbalosin.com
https://github.com/ionutbalosin/java-application-security-practices
https://owasp.org/www-project-top-ten
https://auth0.com/docs/get-started/authentication-and-authorization-flow
https://cheatsheetseries.owasp.org/cheatsheets/HTTP_Headers_Cheat_Sheet.html
https://cheatsheetseries.owasp.org/cheatsheets/Deserialization_Cheat_Sheet.html#java
https://cheatsheetseries.owasp.org/cheatsheets/XML_External_Entity_Prevention_Cheat_Sheet.html
https://owasp.org/www-community/vulnerabilities/XML_External_Entity_(XXE)_Processing
https://nvd.nist.gov/vuln-metrics/cvss
https://snyk.io/blog
https://contrastsecurity.com/blog

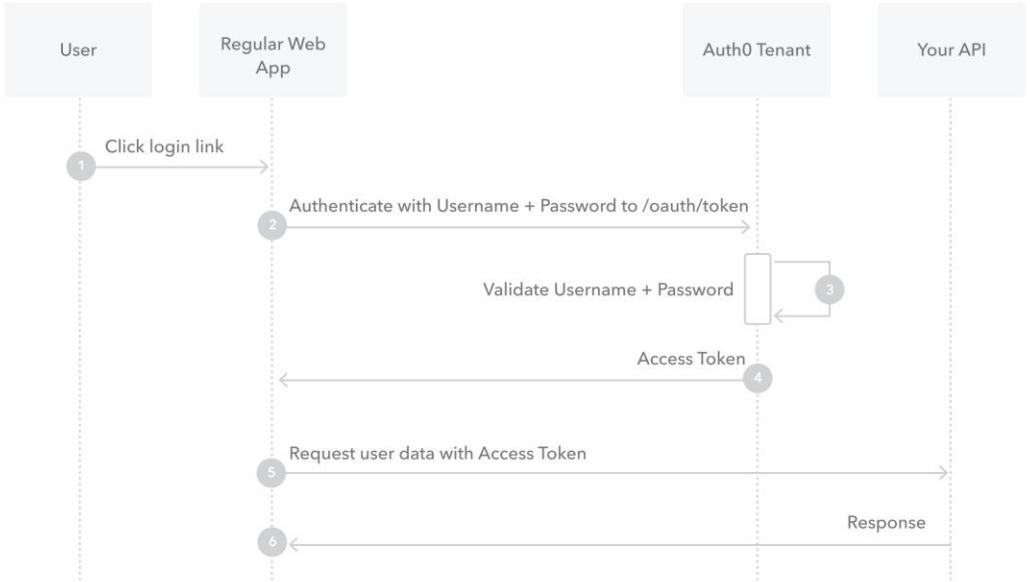
Appendix

Client Credentials Flow



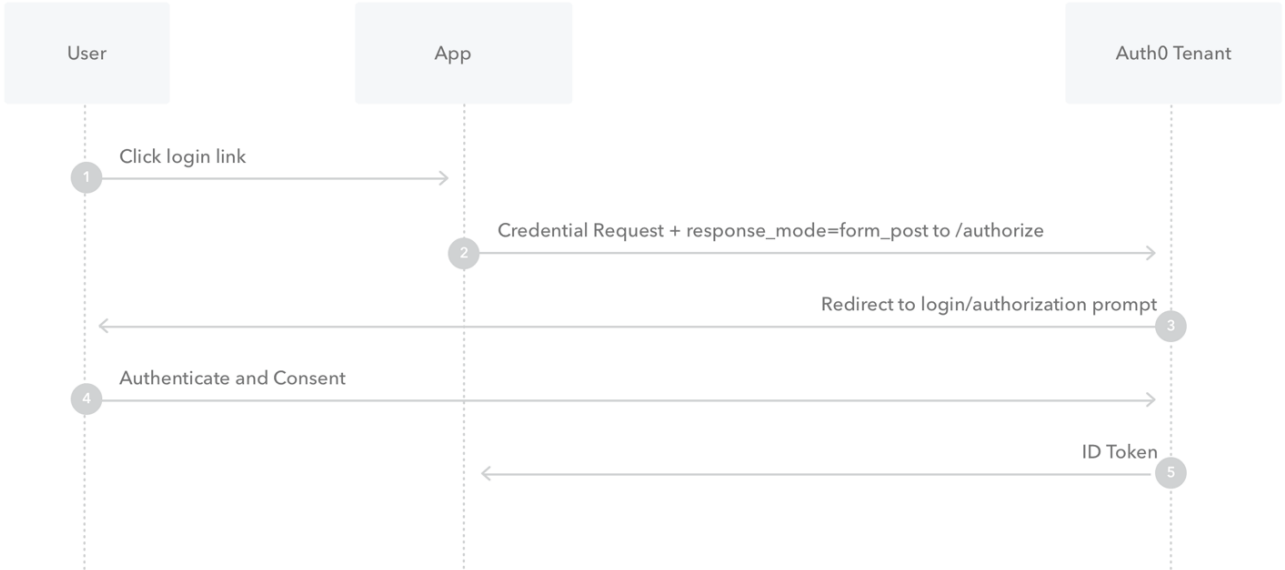
Source: [https://auth0.com/docs/get-started/authentication-and-authorization-flow/client-credentials-flow]

Password Flow (deprecated)



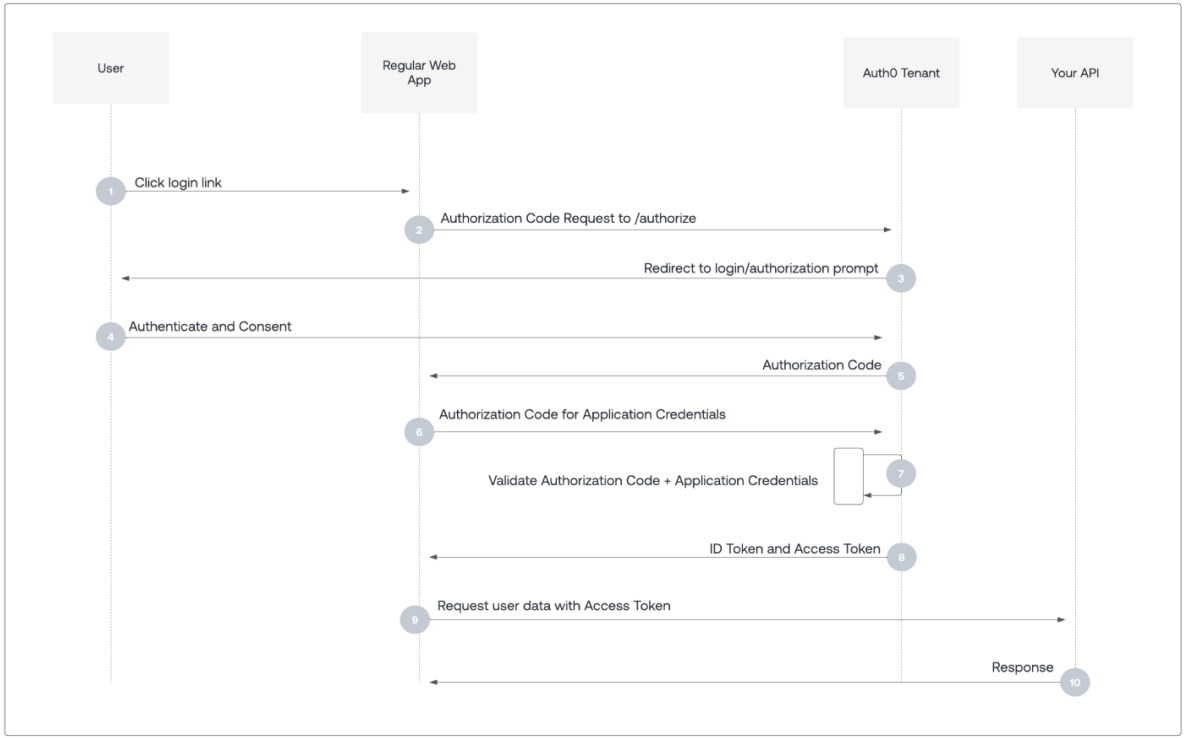
Source: [https://auth0.com/docs/get-started/authentication-and-authorization-flow/resource-owner-password-flow]

Implicit Flow (deprecated)



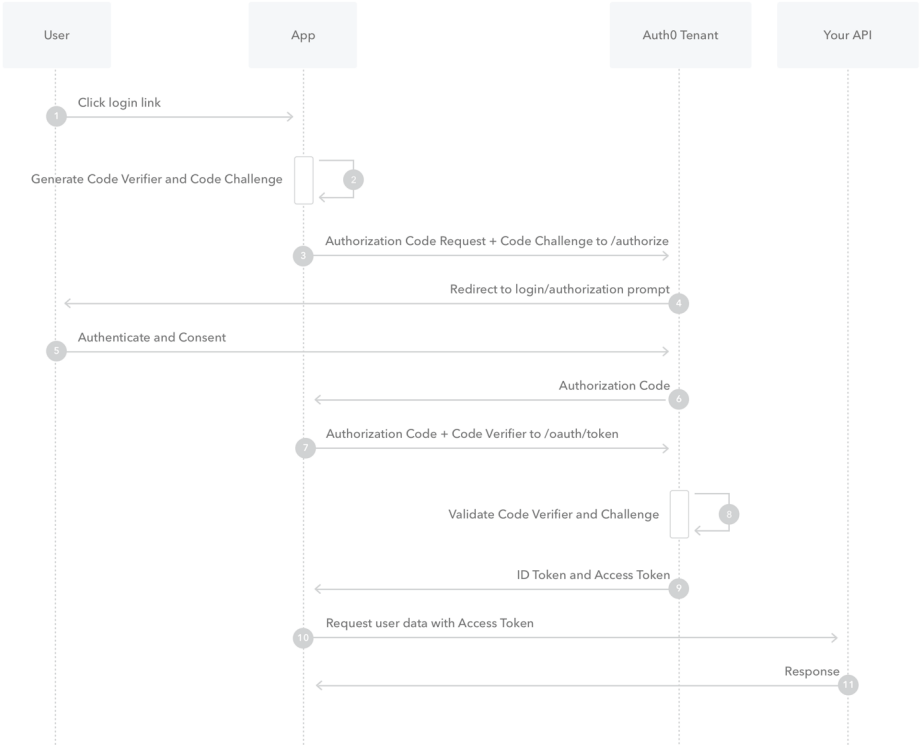
Source: [https://auth0.com/docs/get-started/authentication-and-authorization-flow/implicit-flow-with-form-post]

Authorization Code Flow



Source: [https://auth0.com/docs/get-started/authentication-and-authorization-flow/authorization-code-flow]

Authorization Code Flow with PKCE



Source: [https://auth0.com/docs/get-started/authentication-and-authorization-flow/authorization-code-flow-with-pkce]

Authorization Code Flow with PKCE

Example of generating and using the **Code Verifier** and the **Code Challenge**:

```
// A randomly generated 32-byte sequence
Code Verifier = l93m6y2x0rP5YfS2sb5K5D5L5zvFq8p1

// Derived from the code verifier by applying the SHA-256 hashing algorithm and Base64 URL encoding
Code Challenge = Base64URL(SHA256(Code Verifier)) = 0dfy4j2i93eFLQz8EIZ4Pl5sg7N09E6F
```

OIDC Token

OIDC Tokens (**ID Tokens**) contain identity information about the authenticated user (e.g., name, email, profile picture, language preferences, etc.).

An OIDC token (specifically the ID Token) is issued after a successful authentication process as part of the OpenID Connect protocol.

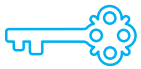
ID Tokens are commonly formatted as JSON Web Tokens (JWT).

ID Tokens typically have a short lifespan (usually 5 to 15 minutes).

OIDC Token Payload Example

```
{
  "typ": "ID",
  "exp": 1730032142,
  "iat": 1730031242,
  "iss": "http://idp.keycloak.com/realms/master",
  "aud": "demo_public_client",
  "sub": "df277362-4ee9-45ed-bb72-c25643d124b7",
  "azp": "demo_public_client",
  "sn": "Doe",
  "given_name": "John",
  "email": "john.doe@keycloak.com",
  "preferred_language": "eng",
  "profile_picture": "http://ionutbalosin.com/JohnDoe.png"
}
```

Access Token



Access tokens contain roles or permissions (often encoded in claims) that authorize access to protected resources.

Access tokens are issued by the authorization server after a successful authentication and authorization process, often as part of the OAuth 2.0 protocol.

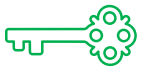
Access tokens are commonly formatted as JSON Web Tokens (JWT).

Access tokens typically have a short lifespan (usually 5 to 30 minutes, sometimes up to an hour).

Access Token Payload Example

```
{
  "typ": "Bearer",
  "exp": 1730032142,
  "iat": 1730031242,
  "iss": "http://idp.keycloak.com/realms/master",
  "sub": "df277362-4ee9-45ed-bb72-c25643d124b7",
  "azp": "demo_public_client",
  "roles": [
    "HR_Manager",
    "Admin",
    "Employee"
  ],
  "sn": "Doe",
  "given_name": "John",
  "email": "john.doe@keycloak.com"
}
```

Refresh Token



Refresh tokens are used to obtain new access tokens after the original access token has expired, without requiring the user to re-authenticate.
They are primarily used in client-facing applications to maintain user sessions and prolong connectivity without forcing frequent re-authentication.

Refresh tokens are issued by the authorization server alongside access tokens as part of the OAuth 2.0 authorization process.

Refresh tokens are commonly formatted as JSON Web Tokens (JWT).

Refresh tokens have a longer lifespan than access tokens (generally days, weeks, or even months), but they can also be revoked by the authorization server.

Refresh Token Payload Example

```
{
  "typ": "Refresh",
  "exp": 1730034842,
  "iat": 1730031242,
  "iss": "http://idp.keycloak.com/realms/master",
  "aud": "http://idp.keycloak.com/realms/master",
  "sub": "df277362-4ee9-45ed-bb72-c25643d124b7",
  "azp": "demo_public_client"
}
```