SECURING THE MODERN WEB APPLICATION

From Vulnerable SPA to FAPI 2.0-Grade Security with WSO2 Identity Server 7.x

A practical path from SPA + bearer tokens to a FAPI 2.0-aligned Backend-for-Frontend architecture, configured in WSO2 IS 7.x.

Cedric Guyomard | Security & IAM Architect

30-minute technical session

AGENDA

What we'll cover

01

The Problem

Why storing OAuth tokens in browser JavaScript increases risk.

02

The Solution

FAPI 2.0 principles combined with the Backend-for-Frontend (BFF) pattern.

03

In Practice

Step-by-step configuration in WSO2 Identity Server 7.x, with screenshots.

Audience: technical architects & developers, engineering managers, and API product owners.

01

The Problem

Why we need to rethink SPA security

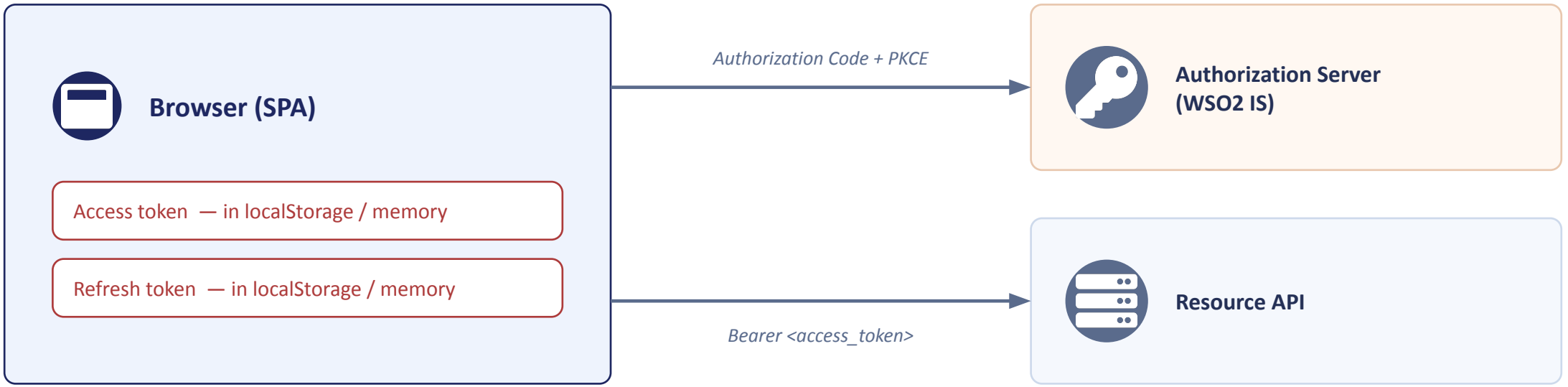
CONTEXT

Modern web apps: SPA + APIs, everywhere

A Single-Page Application (SPA) loads one page; JavaScript updates the interface without full page reloads.

It calls APIs for data. In this model, browser code also acts as the OAuth client and handles its tokens.

SPA



PKCE protects the code exchange. The access token, and sometimes a refresh token, still reaches browser JavaScript.

RISK

What can go wrong

RISK 01

XSS → token theft

XSS can steal accessible tokens or issue requests through the user's session.

RISK 02

No sender-constraint

A stolen bearer token works from anywhere — nothing proves the caller is the legitimate client.

RISK 03

Long-lived refresh tokens

A refresh token in the browser can prolong an attacker’s access if it is exposed.

RISK 04

Weak client authentication

A public SPA cannot keep a client credential, so it cannot strongly authenticate as a confidential client.

Why this matters to the business

Regulatory pressure

- FAPI originated in Open Banking and is used by multiple financial API ecosystems
- FAPI 2.0 maps to selected PSD2 RTS and PCI DSS technical controls, directly or in part
- This makes strong security profiles valuable, but FAPI alone does not establish compliance

Business exposure

- A token-theft incident can trigger breach response, forensics, and regulatory notification obligations
- Weak authentication architecture can create audit findings and delay partner integrations
- Identity incidents directly affect customer trust and operational cost

02

FAPI 2.0 & the BFF Pattern

Turning the protocol and the architecture into allies

FAPI: the Financial-grade API Security Profile

Born in Open Banking

Published by the OpenID Foundation's FAPI Working Group, originally to secure PSD2 / Open Banking APIs.

Now used far beyond finance

Designed for high-security APIs — the final FAPI 2.0 Security Profile is not limited to banking use cases.

A profile, not a new protocol

FAPI builds on OAuth 2.0 / OIDC — it removes the risky options and mandates the safe ones.

Two generations

FAPI 1.0 Advanced and FAPI 2.0 — the current, simplified security profile covered here.

THE STANDARD

FAPI 2.0: four controls to remember

CONTROL 01

PAR

Push the authorization request over an authenticated back channel. The browser sends client_id and request_uri.

AUTHORIZATION REQUEST

CONTROL 02

PKCE S256

A fresh verifier/challenge pair binds the token exchange to the authorization request that started the flow.

CODE BINDING

CONTROL 03

Confidential client

FAPI 2.0 requires confidential clients: authenticate with private_key_jwt or mutual TLS.

CLIENT AUTHENTICATION

CONTROL 04

Sender-constrained tokens

Every access token is bound to a client key via mTLS or DPoP; the Resource Server verifies possession.

PROOF OF POSSESSION

THE MISSING PIECE

FAPI 2.0 secures the protocol — but who holds the keys?

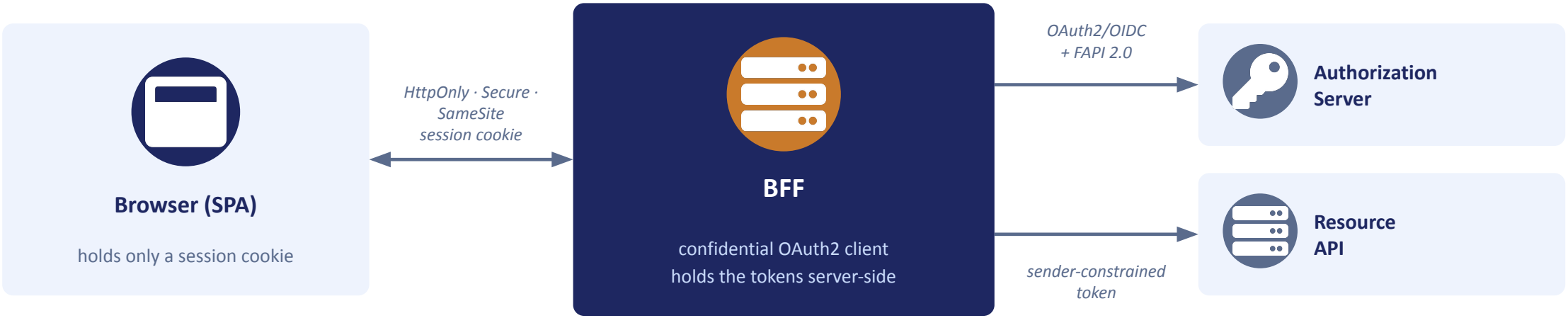
FAPI 2.0 requires a confidential client that can:

- authenticate itself with private_key_jwt or mTLS
- hold the key material used for sender-constraining

A pure browser SPA is a public client and is outside the scope of the FAPI 2.0 Security Profile. The issue is architectural: browser JavaScript cannot provide the same confidential-client trust boundary as a server-side component.

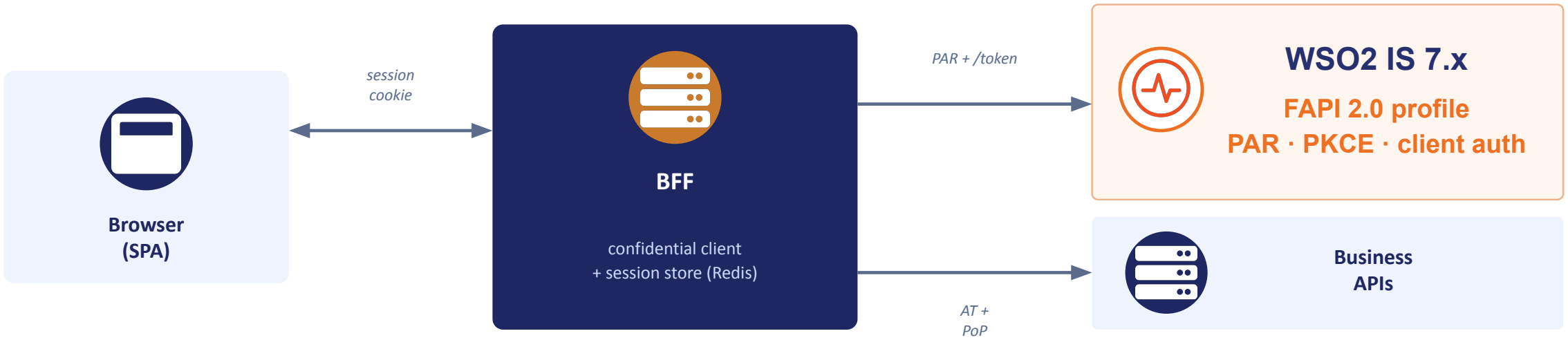
In this BFF architecture, the browser keeps an application session; the BFF holds OAuth tokens and keys server-side.

The Backend-for-Frontend (BFF) pattern



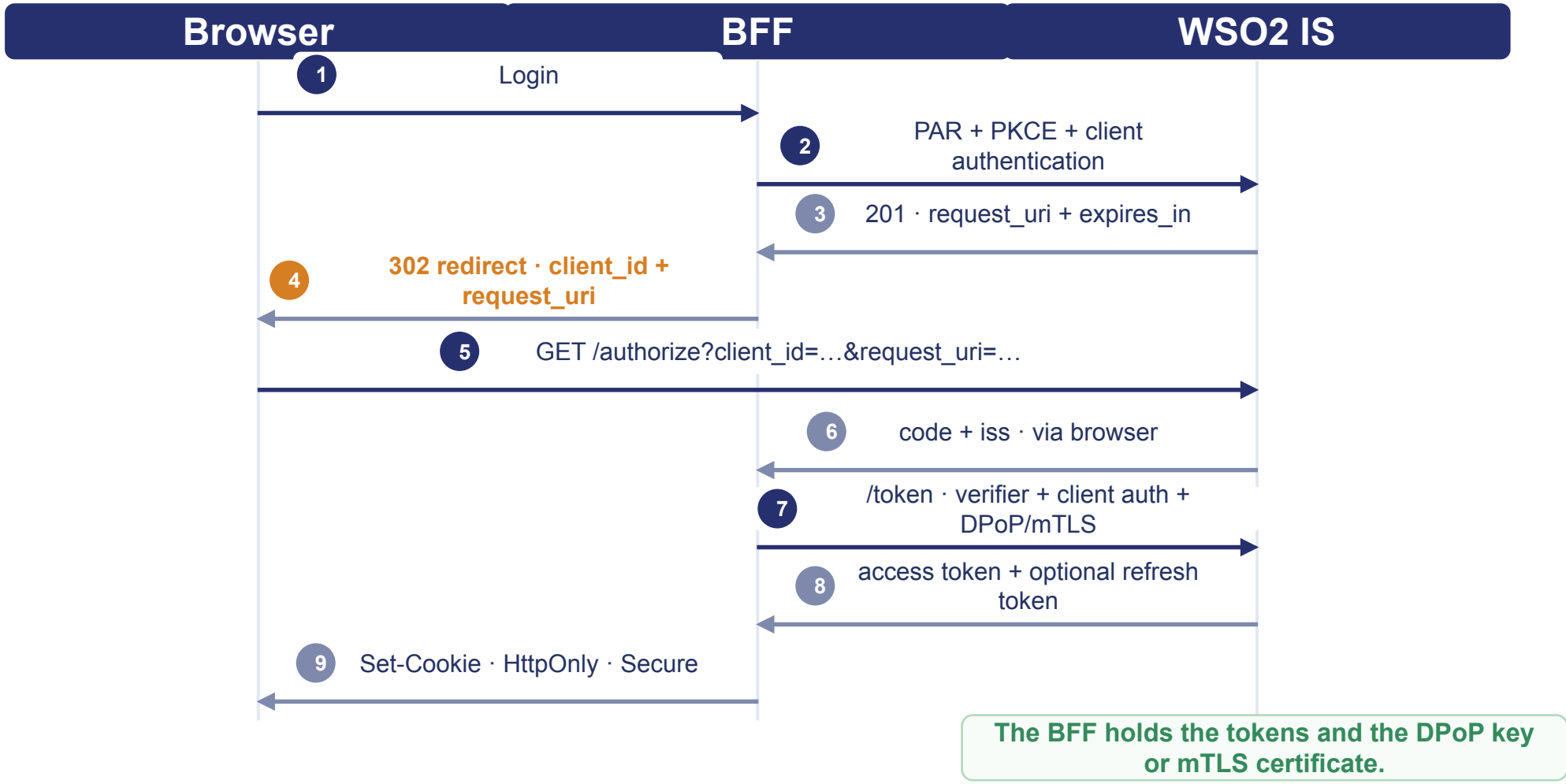
The browser never sees an access token, a refresh token, or a client secret.

BFF + WSO2 IS 7.x — FAPI 2.0 architecture

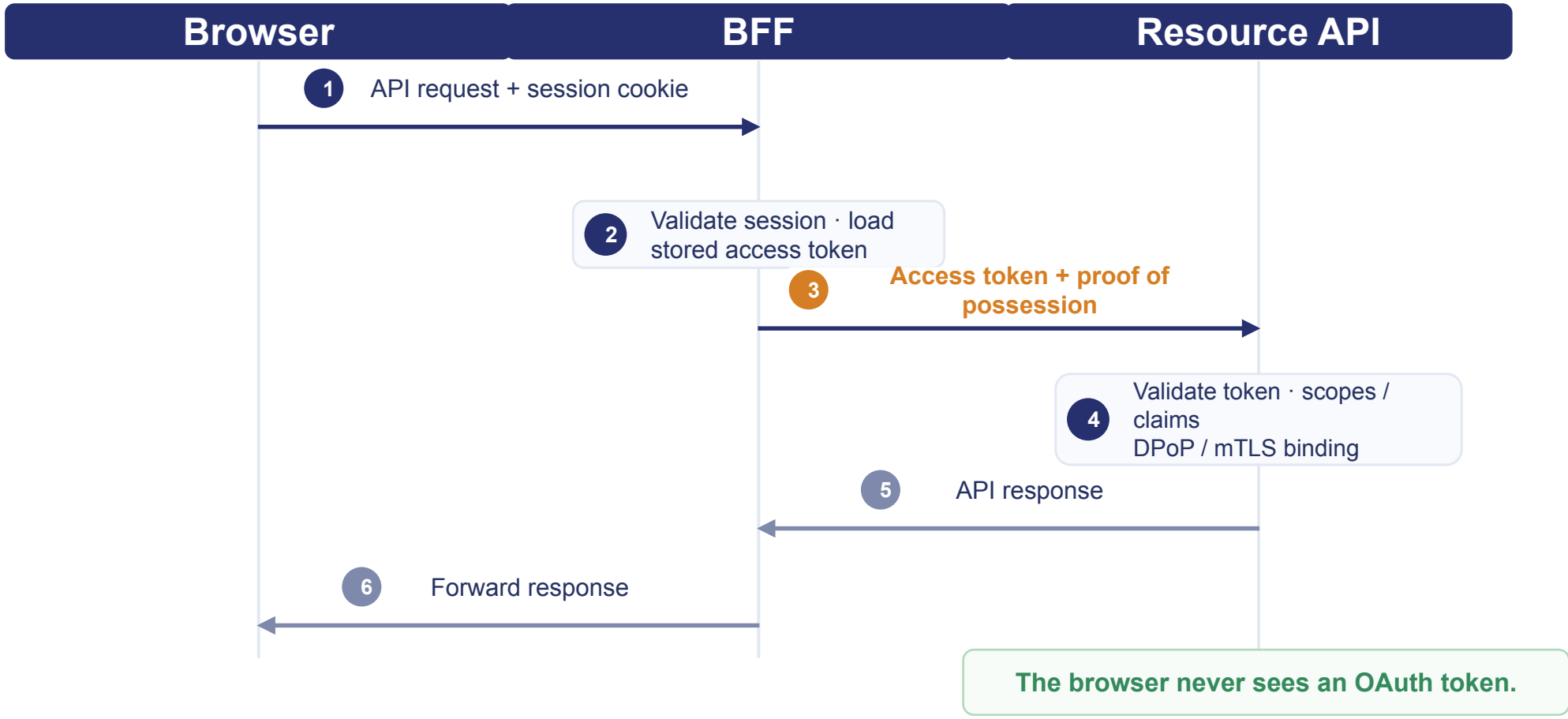


The browser uses an application session cookie. The BFF performs PAR and the code exchange at /token with WSO2, then calls the business API with a sender-constrained access token.

Flow 1 — Login, step by step



Flow 2 — Calling a protected API



OUTCOME

What we gained

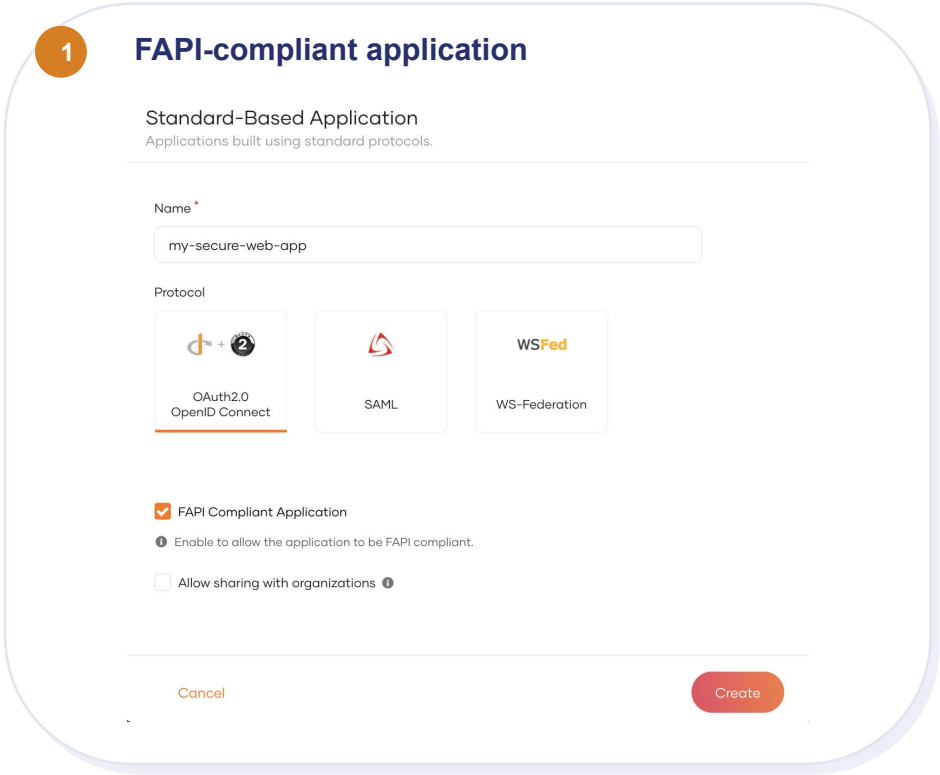
Risk	SPA	BFF + FAPI 2.0
Token exposure to XSS	High — tokens live in JS	Browser token theft reduced; XSS can still act through the user's session
Stolen token replay	Possible (plain bearer token)	The token alone is insufficient for replay
Client impersonation	Public client cannot strongly authenticate	Strong confidential-client auth (mTLS / private_key_jwt)
Authorization request tampering	Possible in the front channel	Authenticated PAR protects the request; client_id and request_uri cross the browser
Regulatory alignment	No dedicated financial-grade profile	FAPI 2.0 security-profile alignment

03

Configuring WSO2 Identity Server 7.x

From theory to a working FAPI 2.0 setup

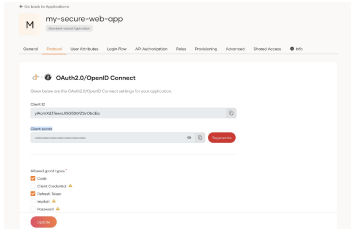
Register a FAPI-compliant application



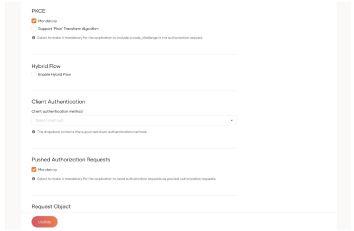
- Console › Applications › New Application › Standard-Based Application
- Protocol: OAuth2 / OpenID Connect
- Enable the “FAPI Compliant Application” option
- WSO2 restricts several settings to FAPI-compliant choices; complete the server configuration and verify the resulting behavior

PAR, PKCE & grant types

1 **Grant types**



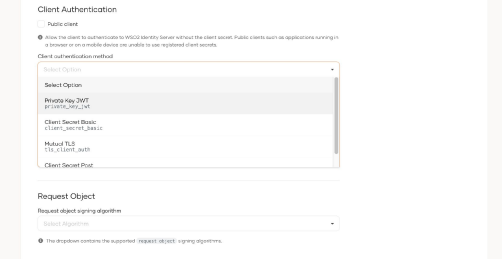
2 **PKCE & PAR**



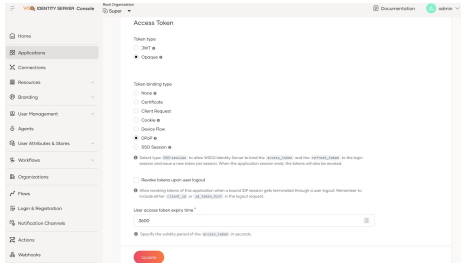
- For this web login flow: Authorization Code only; no Implicit or Hybrid response
- Enforce PKCE with S256 (plain is not allowed)
- Require client-authenticated Pushed Authorization Requests (PAR)

Sender-constraining tokens & client authentication

1 Client authentication



2 Access token & token binding



- FAPI 2.0 requires both client authentication and sender-constrained access tokens
- Client authentication: private_key_jwt or mTLS. Token binding: mTLS or DPoP
- WSO2 IS supports mTLS and certificate-bound access tokens
- From WSO2 IS 7.2, DPoP is supported natively
- The BFF holds tokens and keys; the Resource API verifies proof of possession

FAPI cryptography and TLS — WSO2 IS 7.x

Check FAPI JWT algorithms and enforce TLS 1.2+ wherever TLS terminates.

JWT

JWT signatures

Signed JWTs, including ID and JWT access tokens

FAPI: PS256, ES256 or EdDSA (Ed25519).
WSO2 example: PS256. Opaque access tokens are not JWTs.

ASYMMETRIC SIGNATURES

TLS

TLS transport

HTTPS channel protection

Enforce TLS 1.2 as the minimum.
Apply the policy where TLS actually terminates.

TLS 1.2 MINIMUM

VERIFICATION

Validate your setup

- Authorization requests are rejected unless created through authenticated PAR
- PAR requests without valid confidential-client authentication are rejected
- Authorization-code exchange fails without the correct PKCE code_verifier
- Authorization response contains iss and the BFF validates it before exchanging the code
- Resource API rejects a sender-constrained token when proof of possession is missing / invalid
- Run the OpenID Foundation FAPI 2.0 conformance suite for the target profile

SUMMARY

Key takeaways

TAKEAWAY 01

For architects & developers

Use a BFF to keep OAuth tokens and key material out of browser JavaScript; apply the FAPI 2.0 controls to that confidential server-side client.

ARCHITECTURE

TAKEAWAY 02

For managers

FAPI 2.0 gives a standards-based high-security profile and reduces common OAuth attack paths; regulatory compliance still depends on the applicable scheme and controls.

GOVERNANCE

TAKEAWAY 03

For API teams

Sender-constraining proves possession of the key bound to the access token and limits replay. Authorization itself still comes from scopes, claims and API policy.

API SECURITY

Questions?

From vulnerable SPA to FAPI 2.0-grade security — with WSO2 Identity Server 7.x

Cedric Guyomard | Security & IAM Architect