OAUTH DECISION TREE

2-Legged vs. 3-Legged OAuth: *Which Flow Do You Need?*

The difference comes down to one thing: who authorizes access.

START HERE

Is there a resource owner separate from your client who needs to approve this *access?*

YES → **3-Legged OAuth**
Authorization Code Flow

NO → **2-Legged OAuth**
Client Credentials Flow

Mobile or single-page app?

YES → **Add PKCE**

- → Generate a random code verifier
- → Hash and send it at auth start
- → Prove the same client completes it

Mandatory in OAuth 2.1

e.g. Mobile app or SPA: Google, GitHub

NO → **Implement Auth Code Flow**

- → User grants explicit permission
- → Exchange auth code for short-lived token
- → Validate state param, prevent CSRF

15–60 min token TTL

e.g. Web app: user's calendar or Drive

Can you eliminate static secrets?

YES → **Adopt Workload Identity Federation**

- → Trust derived from the environment
- → Cryptographic proof, no stored secret
- → Removes the bootstrap problem

Eliminates credential sprawl

e.g. GitHub Actions to AWS via OIDC

NOT YET → **Inject Credentials Securely**

- → Store in vault, inject at runtime
- → Scope credentials to each session
- → Rotate often, limit exposure windows

Minimizes exposure windows

e.g. Microservice to DB, session-scoped

THE REAL GOAL

Stop managing secrets better and eliminate them entirely. Workloads authenticate via cryptographically verified environment attributes, with no rotation burden and no persistent attack surface.

THE THREAT BOTH FLOWS SHARE

Long-lived credentials are the common vulnerability. A 2-legged token valid for hours or a 3-legged token sitting in the browser both create windows attackers can exploit. Short TTLs are non-negotiable.

REGARDLESS OF FLOW, ALWAYS:

Verify JWT signatures · **Enforce TLS** · **Scope to minimum permissions** · **Audit every issuance** · **Keep TTLs sub-hour**