CS-068 · API Security

OAuth 2.0

A delegated authorization framework that lets apps access resources without handling user passwords.

Executive Summary

OAuth 2.0 is a framework for delegated authorization: it lets an application access a limited part of a user's resources on another service without ever seeing that user's password. Instead of sharing credentials, the user grants consent and the application receives a scoped, short-lived access token. Secure OAuth depends on using the right flow, limiting scopes, protecting tokens, and validating redirects.

What It Is

OAuth 2.0 is an authorization framework, not a login system by itself. It solves a specific problem: how can one application act on a user's behalf at another service without the user handing over their password. The user authorizes the application, an authorization server issues an access token that represents that grant, and the application presents the token to a resource server to access only what was permitted. The key pieces are the resource owner (the user), the client (the application requesting access), the authorization server (which authenticates the user and issues tokens), and the resource server (which holds the data). Access is limited by scopes, which describe exactly what the token may do. Because OAuth handles authorization rather than proving identity, an identity layer such as OpenID Connect is built on top of it when login is the goal.

Why It Matters

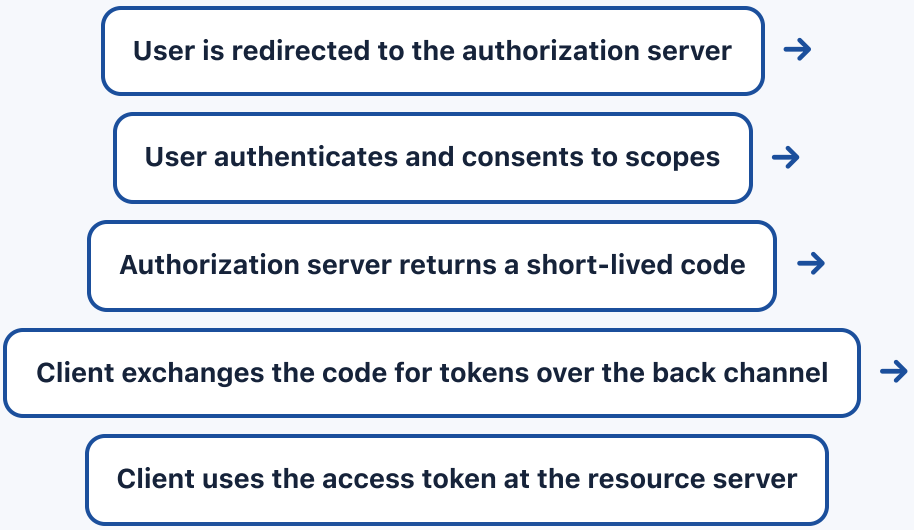
OAuth 2.0 is the backbone of connecting apps together and of most sign-in-with providers, so it governs access to enormous amounts of personal and business data. Done well, it removes the dangerous practice of sharing passwords across services and limits each application to the

narrow access it needs. Done poorly, it opens serious holes: a mishandled redirect can hand tokens to an attacker, an over-broad scope can grant far more access than intended, and a leaked token can be replayed. For engineers and security professionals, understanding OAuth flows and their failure modes is essential because the framework is flexible, and much of its security comes from choosing and configuring the right flow rather than from the framework enforcing safety automatically.

How It Works

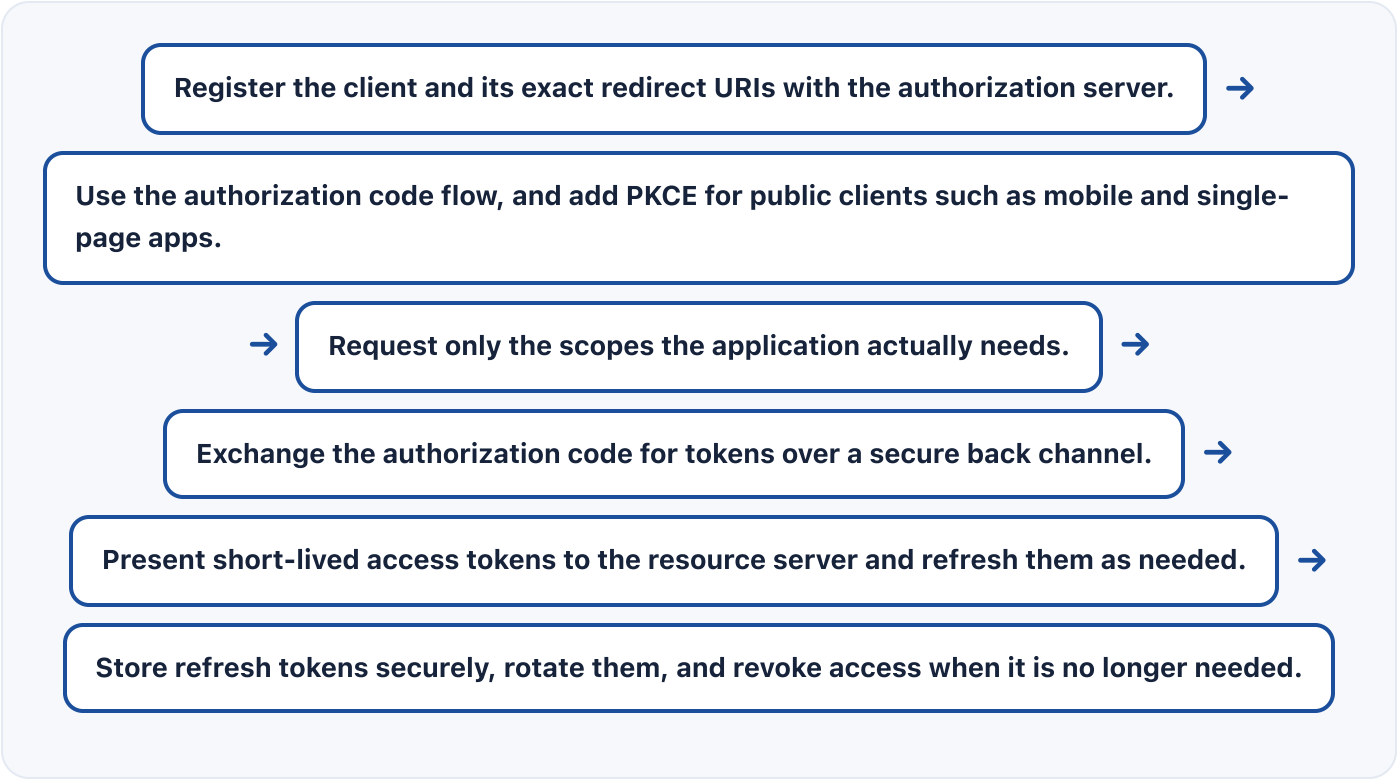
The recommended pattern is the authorization code flow. The client redirects the user to the authorization server, where the user authenticates and consents to specific scopes. The authorization server sends the user back to the client with a short-lived authorization code. The client then exchanges that code, over a back channel, for an access token and often a refresh token. The access token is presented to the resource server on each request and grants only the permissions in its scopes. For public clients such as mobile and single-page apps, a protection called PKCE binds the code exchange to the client that started it, preventing an intercepted code from being redeemed by an attacker. Access tokens should be short-lived; refresh tokens let the client obtain new access tokens without asking the user to sign in again, and they must be stored and rotated carefully.

Architecture Diagram



In the authorization code flow the user consents, the client receives a code, and the code is exchanged for a scoped access token.

Visual Workflow



Common Attacks

- Redirect URI manipulation that sends the authorization code or token to an attacker
- Authorization code interception on public clients that lack PKCE
- Cross-site request forgery on the authorization request when the state value is missing
- Token theft and replay when tokens are long-lived or stored insecurely
- Over-broad or escalated scopes granting more access than the user intended

Common Mistakes

- Using implicit or password-based flows instead of the authorization code flow with PKCE
- Allowing loose or wildcard redirect URIs instead of exact registered values
- Omitting the state parameter and leaving the flow open to cross-site request forgery
- Requesting broad scopes for convenience rather than the minimum needed
- Issuing long-lived access tokens with no rotation or revocation
- Treating a valid access token as proof of identity, which OAuth does not provide

Best Practices

- Prefer the authorization code flow with PKCE for both public and confidential clients
- Register exact redirect URIs and reject any that do not match
- Use and verify the state parameter to prevent cross-site request forgery
- Request the minimum scopes needed and review them over time
- Keep access tokens short-lived and rotate refresh tokens
- Use OpenID Connect on top of OAuth when you need to authenticate the user

Quick Checklist

- [x] Authorization code flow with PKCE in use for public clients
- [x] Redirect URIs registered exactly and matched strictly
- [x] State parameter set and verified on every authorization request
- [x] Scopes limited to the minimum required
- [x] Access tokens short-lived with refresh token rotation
- [x] Tokens stored securely and transmitted only over TLS

Recommended Tools

Authorization server
Authenticates users, obtains consent, and issues scoped tokens

Identity provider
Hosts OAuth and often OpenID Connect for delegated access and login

OAuth client library
Implements the flows and PKCE correctly in application code

API gateway
Validates access tokens and enforces scopes in front of resource servers

Industry Standards

OAuth 2.0 Authorization Framework (RFC 6749)

Proof Key for Code Exchange (PKCE, RFC 7636)

Defines the core roles, flows, and tokens

Protects the authorization code exchange for public clients

OAuth 2.0 Security Best Current Practice
Current IETF guidance on securing OAuth deployments

Career Relevance

OAuth 2.0 is essential knowledge for backend engineers, application security engineers, and API security engineers who build and secure integrations and sign-in features. Security engineers and penetration testers test OAuth flows for redirect and token flaws, and identity and AI governance professionals rely on OAuth concepts because it governs how applications and AI agents obtain delegated access to protected resources.

Interview Questions

- What problem does OAuth 2.0 solve, and how is it different from authentication?
- Walk me through the authorization code flow and where the token exchange happens.
- What is PKCE, and why is it important for mobile and single-page applications?
- Why must redirect URIs be registered exactly, and what happens if they are not?
- What is the role of the state parameter and of scopes in a secure OAuth flow?

Related Certifications

OWASP-aligned application security training

GIAC Web Application Penetration Tester (GWAPT)

Identity and access management vendor certifications (vendor-neutral concepts)

Further Reading

- RFC 6749: OAuth 2.0 Authorization Framework
- RFC 7636: Proof Key for Code Exchange
- OAuth 2.0 (official site)

Key Takeaways

- OAuth 2.0 delegates limited access without sharing the user's password.
- It handles authorization, not identity; use OpenID Connect on top when you need login.
- The authorization code flow with PKCE is the recommended, most secure pattern.
- Exact redirect URIs, the state parameter, and minimal scopes are core defenses.
- Keep access tokens short-lived and rotate refresh tokens.

FAQ

Is OAuth 2.0 a login system?

Why is PKCE recommended even for confidential clients?

What is the difference between an access token and a refresh token?

Related Careers

RELATED ROLES

Application Security Engineer

Api Security Engineer

Backend Engineer

Security Engineer

RELATED CERTIFICATIONS

OWASP-aligned application security training

GIAC Web Application Penetration Tester (GWAPT)

Identity and access management vendor certifications (vendor-neutral concepts)

CURRENT OPENINGS

Live roles are on the job board.

GUIDES & SALARY

Browse all jobs

Career guides | Role roadmaps | Salary data | Career tools

SUGGESTED LEARNING PATH

1. Ground the basics with CS-001 Cybersecurity
2. Study this sheet: **OAuth 2.0**
3. Go deeper: OpenID Connect
4. Go deeper: JSON Web Tokens (JWT)
5. Validate it: work toward **OWASP-aligned application security training**
6. Find the role: browse current openings

RELATED SHEETS

CS-069 OpenID Connect | CS-067 JSON Web Tokens (JWT) | CS-066 REST API Security | CS-062 Authentication | CS-063 Authorization