

OAUTH 2.0 & OPENID CONNECT.



OAuth 2.0 was built to authorize, not to authenticate — **treating a successful token exchange as proof of identity, instead of validating OpenID Connect's purpose-built ID token, is the mistake underneath most real-world OAuth failures.**

- PKCE ON EVERY CLIENT
- STRICT REDIRECT URI VALIDATION
- SENDER-CONSTRAINED TOKENS
- ID TOKEN VALIDATION

01 THE MECHANISM

Authorization Code Interception

Without PKCE, an attacker who intercepts the authorization code mid-redirect can exchange it for a token themselves — **the code alone was never meant to be a strong enough secret**, which is exactly what PKCE was built to fix.

Bearer Token Replay

A standard OAuth access token is a bearer token — **whoever holds it can use it**, with no proof required that they're the client it was issued to. A token leaked through XSS or logs can be replayed with zero additional friction.

Consent Phishing

A malicious app can request broad, legitimate-looking OAuth permissions and trick a user into granting consent directly — **this bypasses passwords and MFA entirely**, needing only the victim's one-time click.

02 THE FLOW

The Hardened Authorization Flow

CONTAINED

1

Every authorization code exchange uses PKCE, so **an intercepted code is useless without the client's original secret verifier**.

2

The redirect URI is validated with an exact string match against a pre-registered value, closing the door on wildcard or open-redirect abuse.

3

Refresh tokens rotate on every use, so **a stolen, already-rotated refresh token being replayed is a detectable signal**, not a silent compromise.

The Legacy, Unhardened Flow

EXPOSED

1

Authorization codes are exchanged without PKCE, so **any party that intercepts the code in transit can redeem it for a token**.

2

Redirect URI validation accepts wildcard subdomains or partial matches, letting an attacker register a lookalike endpoint to capture codes and tokens.

3

Refresh tokens never rotate, so **a token leaked once remains valid and undetectable indefinitely**, with no signal that a copy is circulating.

03 CORE PILLARS & COVERAGE

PILLAR	WHAT IT COVERS
PKCE on Every Client	Requires a cryptographic proof tied to the original authorization request before a code can be exchanged for a token, protecting confidential and public clients alike.
Strict Redirect URI Validation	Matches the redirect URI by exact string comparison against a pre-registered value, rejecting wildcard, prefix, or open-redirect patterns.
Sender-Constrained Tokens	Binds an access token to the specific client holding a private key, such as through DPoP or mutual TLS, so a stolen bearer token alone isn't enough to use it.
ID Token Validation	Verifies the OpenID Connect ID token's signature, issuer, audience, and nonce on every login, rather than trusting a token just because it decodes.

04 DETECTION APPROACHES COMPARED

APPROACH	SUFFICIENT ALONE?	WHAT IT MISSES
Authorization Code Exchange Without PKCE	No	Leaves the code exchange vulnerable to interception, especially on public clients like mobile and single-page apps that can't hold a client secret.
Wildcard or Prefix-Matched Redirect URIs	No	Lets an attacker register a lookalike endpoint under the same allowed pattern and capture authorization codes or tokens directly.
Bearer Tokens With No Sender Binding	No	Treats possession of the token as sufficient proof, so a token leaked through XSS or logs can be replayed by anyone who finds it.
ID Tokens Accepted Without Full Validation	No	Opens the door to forged or replayed tokens, including algorithm-confusion attacks against improperly implemented JWT checks.
PKCE + Strict Redirect Validation + Sender-Constrained Tokens + ID Token Validation	Yes	Closes the interception, redirect, replay, and forgery paths together rather than any one alone.

OAUTH 2.0 & OPENID CONNECT.

AUTHORIZE · BIND · VALIDATE

05 PROGRAM OPERATIONS

OAuth 2.0 is an authorization framework, not an authentication protocol. Treating a successful token exchange as proof of identity, instead of using OpenID Connect's ID token, is a foundational mistake.

PKCE is no longer just for public clients. Current OAuth security guidance recommends it for every client type, since it protects against code interception regardless of client type.

A refresh token that never rotates is a standing liability. Rotation with reuse detection turns a leaked token from a silent, indefinite compromise into a one-time, detectable event.

Consent phishing doesn't need a stolen password. A user who clicks "Allow" on a malicious app's permission request hands over API access that MFA never had a chance to stop.

06 CONTROL MAPPING

OAUTH 2.0 SECURITY BCP (RFC 9700)

Current IETF guidance mandating PKCE for all clients and deprecating the implicit and password grant flows.

OPENID CONNECT CORE 1.0

The identity layer on top of OAuth 2.0, defining the ID token and the validation steps a relying party must perform.

RFC 9449 (DPoP)

The sender-constraining standard that cryptographically binds a token to the client holding a specific private key.

RFC 7636 (PKCE)

The proof-key mechanism that protects the authorization code exchange from interception.

07 WHAT THIS DOES NOT SOLVE

Consent Phishing & Malicious Apps

Protocol-level hardening doesn't stop a user from willingly granting a malicious app legitimate-looking permissions through social engineering.

Underlying Password Compromise

Securing the OAuth flow doesn't protect the identity provider's own login step, which still depends on the strength of its own authentication controls.

Overprivileged Scope Requests

Correctly implemented flows don't stop an application from requesting far broader scopes than it actually needs, expanding what a compromised token can do.

Client-Side Token Storage Mistakes

Protocol hardening doesn't fix a token stored insecurely in local storage or exposed via XSS on the client application itself.

08 MYTHS VS. REALITY

MYTH

"OAuth 2.0 authenticates the user."

REALITY

OAuth 2.0 is an **authorization framework** for delegated access — verifying identity requires OpenID Connect's ID token, a distinct and purpose-built mechanism.

MYTH

"PKCE is only necessary for mobile and single-page apps."

REALITY

Current best practice recommends PKCE for **every client type, including confidential server-side apps**, since it protects against code interception regardless of client type.

MYTH

"A stolen access token isn't a big deal since tokens expire quickly."

REALITY

A short-lived access token is still **immediately usable by whoever holds it** until it expires — sender-constraining, not just expiry, is what stops bearer replay.

MYTH

"Validating a JWT's signature is enough to trust an ID token."

REALITY

A relying party must also check the **issuer, audience, and nonce** — skipping those checks is how algorithm-confusion and replay attacks succeed even against signed tokens.

MYTH

"Consent phishing requires stealing a password."

REALITY

A malicious OAuth app only needs the user to **click "Allow" once** — no password or MFA prompt is ever involved.

09 SPEAK THE PROGRAM — KEY TERMS

PKCE (PROOF KEY FOR CODE EXCHANGE)

A mechanism that ties an authorization code to the **original request via a cryptographic verifier**, preventing an intercepted code from being redeemed by another party.

BEARER TOKEN

A token that grants access to **whoever presents it**, with no additional proof of ownership required, making leaked bearer tokens immediately usable by an attacker.

SENDER-CONSTRAINED TOKEN (DPoP)

A token cryptographically **bound to the specific client holding a private key**, so possession of the token alone isn't sufficient to use it.

ID TOKEN

An OpenID Connect JWT that **proves the user's authentication event** to the relying party, distinct from an OAuth access token, which only proves authorization.

CONSENT PHISHING

Tricking a user into **granting a malicious app OAuth permissions directly**, bypassing the need to steal a password or defeat MFA.

REFRESH TOKEN ROTATION

Issuing a new refresh token on every use and invalidating the old one, so a **replayed, already-used token signals theft** rather than going unnoticed.