**OAuth, Authorization, and Password Risk**

Joshua Coman  
The University of Arizona Global Campus  
INT-499 Capstone in Information Technology  
Instructor: John Russel  
September 2, 2025

GitHub:

**Introduction**

OAuth reduces password exposure by replacing shared credentials with scoped access tokens. It enables third-party access with user consent without giving the client the user’s password. This paper explains the aim of OAuth, the problem it solves, real-world use, risks from misuse as an authentication method, how OAuth 2.0 differs from OAuth 1.0, the roles in OAuth 2.0, and how OpenID Connect (OIDC) adds a standards-based login layer. Thesis: OAuth is a delegation framework that protects passwords through token-based access, and OIDC adds the missing identity layer for login.

**Purpose of OAuth and the password problem it solves**

OAuth enables delegated access to protected APIs with time-bound, scope-bound tokens instead of shared passwords. A user can grant an app access to a subset of data or actions, then revoke that grant without changing a password. This model limits blast radius, supports least-privilege access, and removes password storage from downstream apps (Hardt, 2012; Jones & Hardt, 2012).

**Real-world use**

Common patterns include: a photo print app that pulls albums from a cloud photo service; a CI job that reads a private Git repo with a short-lived token; a fitness app that posts new runs to a training platform after the user grants a scope such as write: workouts. Each case uses bearer tokens over TLS with server checks for scope and token life (Jones & Hardt, 2012; Hardt, 2012).

**Risks from using OAuth for “login” without OIDC**

OAuth covers authorization, not identity. If a site treats an access token as proof of identity, it can open paths to token substitution, mix-up, or a confused-deputy outcome. The safer pattern uses OIDC’s ID Token for identity and keeps the OAuth access token for API access. Current guidance favors the authorization code flow with PKCE and strict redirect URI enforcement to cut code interception and mix-up risk (Lodderstedt et al., 2025; Sakimura et al., 2023; Jones, Hardt, & Sakimura, 2015).

**OAuth 2.0 vs. OAuth 1.0**

Function and crypto: OAuth 1.0 signs each request (HMAC-SHA1, RSA-SHA1, or PLAINTEXT). OAuth 2.0 uses bearer tokens with TLS for channel protection, which reduces client work and fits web, mobile, desktop, and device apps (Hammer-Lahav, 2010; Jones & Hardt, 2012).

Terminology and flows: Both define resource owner, client, and resource server patterns, yet OAuth 2.0 formalizes grant types and scope use across web, native, and SPA clients. The code flow with PKCE is the default for public clients; the implicit flow has fallen out of favor (Hardt, 2012; Jones et al., 2015; Lodderstedt et al., 2025).

Security posture:  Best-current-practice guidance deprecates weaker modes and sets guardrails: use code + PKCE, sender-constrained tokens where possible, strict redirect handling, and tight scope design (Lodderstedt et al., 2025; Jones et al., 2015).

**Roles in OAuth 2.0 and their duties**

* Resource Owner: the user who grants access.
* Client: the app that requests a token and uses it to call an API.
* Authorization Server: issues tokens after user consent and policy checks.
* Resource Server: hosts the API and enforces scope and token checks.
* This split lets one service handle consent while another enforces data access (Hardt, 2012).

**OAuth 2.0 and OpenID Connect**

OpenID Connect adds an identity layer to OAuth 2.0. The authorization server issues an ID Token for login and an access token for API access. The client verifies the ID Token’s signature and claims, then uses the access token for resource calls. In short: OAuth grants access; OIDC confirms identity (Sakimura, Bradley, Jones, de Medeiros, & Mortimore, 2023).

**Conclusion**

OAuth replaces password sharing with scoped tokens and moves access control into a standard grant process. This shift reduces password spread, supports least-privilege design, and enables revocation without a password reset. Where login is in scope, OIDC supplies an identity layer that works with OAuth tokens yet avoids the traps that arise when an authorization framework gets pressed into service as a login system (Hardt, 2012; Sakimura et al., 2023; Lodderstedt et al., 2025).

**References**

Hammer-Lahav, E. (2010). The OAuth 1.0 protocol (RFC 5849). Internet Engineering Task Force. https://www.rfc-editor.org/rfc/rfc5849

Hardt, D. (2012). The OAuth 2.0 authorization framework (RFC 6749). Internet Engineering Task Force. https://www.rfc-editor.org/rfc/rfc6749

Jones, M., & Hardt, D. (2012). OAuth 2.0 bearer token usage (RFC 6750). Internet Engineering Task Force. https://www.rfc-editor.org/rfc/rfc6750

Jones, M., Hardt, D., & Sakimura, N. (2015). Proof key for code exchange by OAuth public clients (RFC 7636). Internet Engineering Task Force. https://www.rfc-editor.org/rfc/rfc7636

Lodderstedt, T., Bradley, J., & Fett, D. (2025). Best current practice for OAuth 2.0 security (RFC 9700). Internet Engineering Task Force. https://www.rfc-editor.org/rfc/rfc9700

Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., & Mortimore, C. (2023, December 15). OpenID Connect core 1.0 (incorporating errata set 2). OpenID Foundation. [https://openid.net/specs/openid-connect-core-1\_0.html](https://openid.net/specs/openid-connect-core-1_0.html?utm_source=chatgpt.com)