Runlayer

Who's Driving?

Delegation and the Confused Deputy Problem for AI Agents

Hi!

Alvaro Inckot

Founding Identity Engineer

The Valet Key

You hand your car key to a valet

Valet key = trunk locked, mileage capped

OAuth scopes = the valet key of APIs

But the valet key still opens too many doors.

And you can't make a narrower copy for the valet's assistant.

The Confused Deputy

Classic CS problem — Norm Hardy, 1988

A program with legitimate authority is tricked into misusing it on behalf of a less-privileged actor

The Agentic Version

User grants a scoped OAuth token to an AI agent (MCP client, plugin, copilot)

Even with proper scopes, the token doesn't encode why it was granted

A prompt injection or chained sub-agent can use send_email for exfiltration — within scope

The agent is the confused deputy

Why This Is Acute Now

MCP adoption is exploding — standardized tool-use protocol for LLMs

Agents aren't just reading data — they're writing emails, creating PRs, moving money

The Mismatch: Human Consent vs. Agent Autonomy

OAuth assumes the user is present at consent time and understands scopes

Agents act autonomously, often chaining tools across services

The user approved "read email" — the agent piped content to a code interpreter that called an external API

MCP Adopted OAuth 2.1 — And It's Not Enough

MCP spec (Nov 2025) mandates OAuth 2.1 + OIDC for remote servers
But the spec acknowledges a proxy confused deputy scenario:

— A gateway uses a single client identity with the upstream AS

— Multiple agents register dynamically with the gateway

— The upstream AS caches consent after the first authorization

— Result: Agent B inherits Agent A's consent

What's Missing from OAuth 2.1

No delegation chain

No attenuation

No intent binding

No audit trail of intent

The Gap, Illustrated

The flow that OAuth can't prevent:

1. User tells Agent: "Summarize my recent emails"
2. Agent calls MCP Server: read_emails (user's Bearer token)
3. MCP Server calls Upstream API: Read emails — Token has full email scope
4. Agent calls MCP Server: send_email (same token!)
5. MCP Server calls Upstream API: Send email — Nothing prevents this

Real Attack Surfaces

— Prompt injection

— Tool confusion

— Sub-agent escalation

— Session bleed

OAuth 2.1 secures the pipe. You still need to secure the driver.

The Landscape Is Moving Fast

A wave of coordinated activity across bodies and vendors:

- NIST (Feb 2026) — formal push to accelerate agent identity & authorization adoption
- IETF — three active drafts in parallel:
  - OBO for AI Agents — new OAuth parameters for delegation
  - AIMS / Agent Auth — 8-layer framework; labels static API keys an antipattern (yet 53% of MCP servers still use them)
  - Attenuating Tokens — JWTs that narrow per hop, never expand
- AAIF (Linux Foundation, Dec 2025) — industry coalition forming around agentic AI

What the MCP Spec Does Today

OAuth 2.1 + OIDC mandate for remote servers

Token validation on every tool invocation, not just at session setup

Protected Resource Metadata to advertise capabilities

Security best practices: per-client consent, no cookie reuse

But: no built-in delegation chain, no built-in standardized tool-level scoping, no multi-hop attenuation

Identity Vendors Are Converging Too

WorkOS — M2M tokens with agent identity primitives

Auth0 / Okta — token exchange for OBO patterns

Google — service account delegation

Microsoft Entra — on-behalf-of flow

Common direction: explicit delegation, scoped short-lived tokens, machine identity as first-class

Emerging Patterns — Convergence

— Delegation as first-class object

— Intersection of permissions

— Per-resource session binding

— Attenuation, not escalation

— Audit at every boundary

The Gap That Remains

Standards are drafts, not RFCs

MCP spec is thin on delegation

Somebody has to build the layer between "OAuth token" and "tool call"

We're building this at Runlayer

Sits between AI agents and upstream MCP servers (tools, resources, APIs)

Every tool call passes through the gateway — it's the enforcement point

Three Identity Modes

Human User

authenticates via SSO, gets their policies

Agent (Standalone) — M2M

M2M credentials, agent's own policies only

Agent (On-Behalf-Of) —

acts for a specific user, permissions = intersection of both

Delegation as a First-Class Entity

A delegation is a managed object, not a token property

Records: who delegated, who received delegation, when it starts, when it expires

Can be revoked instantly — takes effect on the next request

Every lifecycle event emits an audit event

The OBO Token Exchange

Based on RFC 8693 (OAuth 2.0 Token Exchange)

— Agent presents its own credential + claims about which user it's acting for

— Gateway validates: agent is active, user is active, active delegation exists

— Issues a short-lived OBO token encoding

— On every subsequent request re-validates delegation

— If revoked between issuance and use — access denied immediately

Policy Intersection — The Confused-Deputy Kill Switch

How it works:

1. Tool Call arrives at Policy Engine
2. Policy Engine evaluates Agent's policies
3. Policy Engine evaluates User's policies
4. If Agent allows AND User allows — ALLOW
5. If either denies — DENY

No ambient authority, no privilege escalation through delegation

Session Grants — Binding Credentials to Services

Problem:

Delegation says "agent may act for user" — but whose OAuth credentials for which upstream service?

User X grants Agent Y permission to use X's OAuth session for Server Z

Precondition: grantor must already have a valid OAuth session

Grants can be personal or shared (org-wide)

The Four Properties You Need

1. Scoped

2. Auditable

3. Revocable

4. No ambient authority

Four independent validation layers. Defense in depth.

Mapping to Emerging Standards

Explicit delegation chains — IETF OBO draft

Attenuation — Attenuating Authorization Tokens draft

Short-lived credentials — AIMS framework

Per-resource binding and revocability — NIST guidance

No static keys — machine identity + token exchange

What to Do Monday

1. Audit your agent tokens—
do they carry delegation context or just user scopes?

2. Model delegation explicitly—
a managed record, not a token property

3. Intersect permissions—
agent AND user policies must both allow

4. Bind credentials per-service—
don't let one token access everything

5. Log everything—
who delegated, who acted, which credential, which tool

Thank you.

Runlayer