OAuth Is Not Enough

Authorization Challenges for Autonomous AI Agents

(Working Draft — v0.3, May 25 2025)

Vivek Chopra

May 25, 2025

Abstract

Recent publications on AI Agents are anchoring on OAuth, specifically OAuth 2.1, for delegated access. The paper identifies key limitations of OAuth in this context, including coarse permission scopes, lack of dynamic policy enforcement, and insufficient support for multi-hop delegation of authority. It then proposes complementary mechanisms such as policy-as-code enforcement, enhanced token delegation flows, and secure AI agent architecture patterns to address these gaps. This approach aligns with emerging industry standards and research, maintaining compatibility with existing OAuth infrastructure, while extending it to ensure AI agents act within human-intended bounds. The analysis concludes that securing AI agents requires a layered authorization stack that extends beyond OAuth alone.

1 Introduction

AI agents are autonomous software systems, powered by large language models, and are increasingly tasked with performing actions on behalf of users, from managing calendars to executing transactions. While OAuth 2.0 has become the de facto standard for delegating limited access to third-party applications, it was not originally designed with non-human, autonomous agents in mind. OAuth allows a user (resource owner) to grant an application (client) scoped access to an API (resource server) using access tokens, typically with an explicit consent step. This framework works well for traditional apps and one-off user-approved tasks, but fully autonomous agents introduce new challenges.

One fundamental issue is that an AI agent (hereafter, "agent") may operate continuously and make complex decisions without fine-grained user oversight once an initial token is granted. The agent might chain together API calls or trigger actions at times and in contexts the user did not explicitly anticipate during consent. OAuth 2.0 scopes are coarse by necessity (e.g., read or write access to an entire calendar), and cannot easily encode context-specific constraints such as "only schedule meetings during work hours". In an autonomous setting,

this lack of contextual restriction is risky- an agent given broad scope may inadvertently or maliciously exceed the user's intent.

Another limitation is the difficulty in auditing and attributing actions. OAuth tokens typically convey the user's identity and permissions, but not which agent or process is using the token at any moment. If an agent behaves undesirably, current OAuth access tokens do not carry an identifier for the agent itself (only the client application and user). In multi-agent architectures or where an agent delegates tasks to sub-agents or external services, this absence of an agent identity in the delegation chain obscures accountability. Recent work proposes issuing an explicit "actor" token that binds the agent's own identity to each request, underscoring that standard OAuth alone is insufficient for comprehensive tracking and control.

Finally, OAuth's model of one-time user consent does not map cleanly to continuous or adaptive decision making by AI. Human users cannot be in the loop for every agent action, hence some form of dynamic oversight is needed to prevent runaway behaviors. The static nature of OAuth consent (a user grants X and Y permissions up front) lacks a mechanism for the agent to request additional permission on the fly or for policies to limit actions post-consent. For instance, if an agent is initially authorized to read emails, OAuth cannot prevent it from reading *all* emails. In practice, developers have started implementing ad-hoc solutions, like requiring re-consent for sensitive actions or embedding manual approval steps, but these are outside OAuth's core capabilities.

In summary, OAuth provides a necessary foundation for delegated authorization of AI agents, but it not sufficient - it alone cannot guarantee that autonomous agents will act safely and as intended in all cases.

2 Background

2.1 AI Agents as Operational Actors

As detailed earlier, Large Language Model (LLM) driven *AI agents* are increasingly deployed as long-running microservices that perform tasks formerly executed by human operators- for example, code review, automated remediation, or change management. These agents combine (i) an execution loop that receives goals or "prompts," (ii) tool-invocation plug-ins (e.g., REST calls, CLI wrappers), and (iii) persistent credentials that enable invocation of internal APIs without interactive log-in. From an access control perspective, each agent thus acts as a *virtual employee*: it holds service-account privileges while needing to remain subject to the policy boundaries ordinarily applied to the human on whose behalf it operates.

2.2 Model Context Protocol (MCP)

The *Model Context Protocol* (MCP) specifies a discovery interface through which an AI agent (or any MCP client) can use the core primitives it advertises.

Primitive	Description	Purpose for the Model
Tools	Executable functions such as `get_weather`, `query_db`, or `send_email`.	Allow the model to *act in the outside world* by invoking code, microservices, or third-party APIs.
Resources	Read-only data objects, e.g. documents, database records, code files, and similar artifacts.	Give the model *additional context* that it may read but *cannot* modify, supporting retrieval-augmented generation and grounding.
Prompts	Reusable instruction snippets containing variables (e.g., ``` ``Translate {text} to Spanish'' ```).	Provide consistent, pre-approved ways to steer the model's behavior or format its output.

Table 1: Core primitives advertised by an MCP server.

2.3 OAuth 2.1 in MCP

Under the current MCP Authorization draft (rev. `2025-03-26`), every server **shall** use **OAuth 2.1**—the updated bundle that rolls in PKCE, native-app guidance, stronger bearer-token rules, and drops the old implicit flow [7].

MCP tightens things further:

- **Dynamic Client Registration** (RFC 7591) is *encouraged*, so new agent instances can sign themselves up automatically;
- **Protected Resource Metadata** (RFC 9728) is *mandatory*, letting agents discover the right authorization and token URLs at run time; and
- **DPoP** (RFC 9449) is *optional*, offered as a way to bind each access token to an agent's public key [15].

Every MCP item—tool, data resource, or prompt—can be looked up through its own metadata endpoint (e.g., `/tools/list`). These listings *may* be JWT-signed as described in RFC 9728, so invoking any of them automatically goes through the same OAuth 2.1 security checks covered later.

Reference implementations- Auth0's MCP Gateway [13] and Stytch's Cloudflare Workers tutorial [14]- demonstrate how a Rich Authorization Request `authorization_details` object (RFC 9396) can accompany every agent call, enabling per-task least privilege even though the core MCP specification is currently silent on that semantic.

2.4 Related Work

Early examinations of OAuth limitations for service automation were presented by Hardt [1], Lodderstedt [4], and Campbell *et al.* in the Token-Exchange RFC [3]. Industry white papers on zero-standing-privilege and dynamic authorization (e.g., OPA technical reports [9]) raised similar least-privilege concerns. Recent analyses by Parecki [15] and Martinez [13] applied these ideas to MCP. The present work synthesizes these findings specifically for autonomous AI-agent deployments.

The paper analyzes residual security gaps and evaluates whether emerging OAuth extensions suffice or whether additional mechanisms-capability tokens, policy-as-code engines, or trusted-execution attestation are required.

3 Limitations of OAuth for Autonomous Agents

AI agents differ from traditional, tightly scoped service-to-service clients in fundamental ways, and those differences complicate a straightforward application of OAuth 2.0/2.1. Some of these limitation are:

3.1 Coarse scopes

Permission scopes exposed by most APIs are broad and binary; they cannot encode intent such as "read-only during office hours", or conditional rules such as time-of-day or rate limits tied to the token's use.

3.2 Lack of Agent Identity and Multi-Hop Delegation/Opaque delegation chains

A bearer or DPoP token authenticates the subject and client but not the specific agent instance. Hence, there is no standardized way to represent the agent's own identity or distinguish its actions from the user's. Also, if an agent uses an access token to call multiple downstream services on the user's behalf, those services see only the user's authority, not the chain of delegation. This undermines accountability: if the agent misbehaves or is compromised, it is difficult to pinpoint which agent (among possibly many) was responsible.

3.3 Static Consent vs. Dynamic Decision-Making

OAuth relies on a one-time consent and fixed scopes; it lacks native constructs for incremental or conditional approval during long-running tasks.

3.4 Dynamic Task boundaries.

An agent can shift from simple read-only lookups to high-impact actions (e.g. code deployment) within a single conversation, demanding fine-grained, on-the-fly changes in privilege.

3.5 Human-equivalent delegation.

Agents often operate 'on behalf' of a specific user while simultaneously holding their own service account rights, requiring composite or dual actor tokens that the core OAuth framework does not natively encode.

3.6 Prompt-driven control flow.

The agent's behavior is steered by natural-language instructions that can be altered or jailbroken, so semantic integrity must be enforced *after* the token has been issued.

Each of the limitations above manifests as security or governance risk when AI agents act unattended. Extensions such as Rich Authorization Requests (RAR, RFC 9396) or Token Exchange (RFC 8693) partially mitigate for Coarse scopes and Opaque delegation chains, yet they do not eliminate the need for an external policy layer, nor do they introduce fine-grained runtime checks.

4 Examples of OAuth limitations for AI Applications

The following subsections present examples showing how these discrepancies manifest as a limitation and describe the complementary mechanisms required to close the gap.

4.1 Delegated Authorization / Least-Privilege Delegation / OBO

Consider a coding agent, represented by a service account, that may hold read and write permissions for every source-code repository in the organization. A human developer, however, may possesses read, write, and maintain rights for only a subset of these repositories. When an instance of the coding agent operates on behalf of a developer, the resulting access token must enforce least-privilege delegation- that is, its scope is the intersection of the developer's entitlements and the agent's service entitlements. The base OAuth 2.0/2.1 specifications provide no construct for representing two distinct actors within a single token [1, 2].

4.2 Operational Guardrails

Consider a DevOps agent that performs nightly production deployments must observe organizational policies forbidding deployments on Fridays or weekends. Plain OAuth offers no mechanism to constrain an access token using calendar-based policies. Guardrails embedded within the agent (for example, a prompt instructing "never deploy on Friday") can be overridden; enforcement therefore

must reside in an external policy layer such as OPA or Cedar [9, 10], optionally augmented by an LLM-based evaluator that interprets higher-level intent.

4.3 Dynamic Authorization

Consider a customer-support agent that may begin a chat session with read-only access to a Customer Relationship Management (CRM) system. If a customer requests account deletion mid-session, write permission for the specific record becomes necessary for the duration of that task. Scopes are fixed at token issuance; extensions such as RAR (RFC 9396) or external policy engines therefore must be employed to issue short-lived, task-scoped tokens.

4.4 Instance-Bound Identity and Runtime Attestation

Consider a revenue-forecasting LLM image that may be authorized to query a finance API from production GPU nodes. Should the container be cloned and executed from an untrusted host with identical credentials, the API cannot distinguish the rogue instance. OAuth authenticates a client identifier, not a runtime instance. Binding an access token to a verified execution environment requires complementary measures—DPoP-bound tokens [5], confidential-compute attestation, or workload-identity systems such as SPIFFE/SPIRE [18, 19].

4.5 Prompt / Input Integrity

Consider a ChatOps deployment agent may receive the instruction *"Deploy commit `35e32b` to staging."* If a compromised message bus alters the payload to *"... to production,"* the downstream deployment API cannot detect the tampering because OAuth attests only transport security, not instruction integrity. A signed prompt manifest or hash commitment that the resource server verifies against an immutable audit log is required.

4.6 Non-Repudiable Audit and Provenance

Consider a scenario where fraudulent ledger entries are discovered months after quarter close, and that may be linked only to short-lived bearer or DPoP tokens, providing no connection to the model hash, prompt version, or runtime environment responsible for the calls. OAuth tokens are intentionally ephemeral and omit immutable provenance metadata; higher-layer constructs—such as message signatures coupled with a append-only audit store is necessary for forensic accountability.

5 Complementary Mechanisms for AI-Agent Authorization

OAuth alone cannot enforce contextual or adaptive controls that autonomous agents demand. This section surveys three complementary layers—*policy-as-code*, *delegation-aware tokens*, and *defence-in-depth architectures*- that together close the gaps identified earlier in the paper.

5.1 Policy-as-Code Enforcement

Declarative policy engines such as Open Policy Agent (OPA) and AWS Cedar allow resource servers or gateways to evaluate rich, attribute-based rules at request time [9, 10]. By treating the OAuth access token as just one input, a policy decision can incorporate additional attributes: request verb, resource label, time of day, or run-time risk score.

5.2 Delegation-Aware Tokens and Traceability

Even with policy gates, the *token* must expose who is acting on whose behalf. OAuth 2.0 Token Exchange (RFC 8693) supports "actor" (`act`) claims so that an AI agent can present a short-lived "on-behalf-of" (OBO) token that encodes `sub=Alice, act=AgentX, scope=calendar.read` [3]. Down-stream services can then differentiate between Alice acting directly versus Alice's agent, enabling selective throttling or revocation. Google's *Agentspace* reference design recommends pairing such OBO tokens with workload identity federation so that every container instance attests its provenance before token issuance [16]. Capability-style tokens such as *macaroons* further constrain use by embedding verifiable caveats (IP range, expiry, resource prefix), providing progressive attenuation as tasks are sub-delegated [8].

5.3 Defence-in-Depth Architectures for AI Agents

A secure deployment must layer multiple controls beyond token checks:

- **Sandbox and Isolation.** Agents run in locked-down sandboxes that proxy all egress through a firewall or service mesh; credentials never leave the enclave. Anthropic's MCP reference gateway illustrates this pattern by terminating OAuth at the edge and issuing per-tool capability tokens internally [12].
- **Just-in-Time (JIT) Privilege Escalation.** Tokens are scoped to a single sub-task and expire within minutes; escalation (e.g. from `read` to `write`) triggers an asynchronous user approval flow- Okta labels this *async authorization* [13].

- **Human-in-the-Loop Checkpoints.** For destructive actions (delete, transfer funds) an explicit secondary confirmation is required, delivered through the user's preferred channel (push, SMS, Slack).
- **Comprehensive Audit.** All agent calls are logged with (`sub`, `act`, model hash, prompt hash) and stored in an append-only ledger to support post-incident forensics. Google Agentspace recommends aligning these logs with IAM Access Transparency for cross-product correlation [16].

Industry guidance converges on this layered model: AI-safety researchers at OpenAI have outlined defence-in-depth practices for agent deployments [21, 22], while Google, Anthropic, and Okta likewise treat OAuth tokens as *necessary but not sufficient* credentials that must be wrapped by isolation, policy, and audit controls [16, 12, 13]. By combining these layers, the proposed architecture enforces the principle of least privilege even when agents evolve, sub-delegate, or when their prompts are jail-broken.

6 Conclusion

OAuth 2.0/2.1 provides a well-tested foundation for delegating authority and it will undoubtedly play a key role in AI agent ecosystems. However, an examination of autonomous AI agents reveals that OAuth alone is not sufficient to guarantee secure and aligned agent behavior. The protocol's intrinsic limitations, its static scope model and lack of built-in policy nuance mean that developers and organizations must augment it with additional mechanisms. The paper recommends combining it with complementary mechanisms to impose contextual and adaptive constraints on agent actions and ensure that AI agents remain trustworthy delegates of human intent.

Nomenclature

AI Artificial Intelligence

DPoP Demonstration of Proof-of-Possession (RFC 9449)

GNAP Grant Negotiation and Authorization Protocol (RFC 9635)

JIT Just-in-Time (authorization)

JWT JSON Web Token

LLM Large-Language Model

MCP Model Context Protocol

OPA Open Policy Agent

OBO On-Behalf-Of delegation (RFC 8693)

RAR Rich Authorization Requests (RFC 9396)

ZSP Zero-Standing-Privilege

References

[1] D. Hardt, *The OAuth 2.0 Authorization Framework*, RFC 6749, IETF, Oct. 2012. https://www.rfc-editor.org/rfc/rfc6749

[2] M. Jones and D. Hardt, *The OAuth 2.0 Authorization Framework — Bearer Token Usage*, RFC 6750, IETF, Oct. 2012. https://www.rfc-editor.org/rfc/rfc6750

[3] T. Campbell, S. Bradley, M. Jones, and A. Nadalin, *OAuth 2.0 Token Exchange*, RFC 8693, IETF, Jan. 2020. https://www.rfc-editor.org/rfc/rfc8693

[4] T. Lodderstedt, *OAuth 2.0 Rich Authorization Requests*, RFC 9396, IETF, Aug. 2023. https://www.rfc-editor.org/rfc/rfc9396

[5] J. Fett and T. Lodderstedt, *OAuth 2.0 Demonstrating Proof-of-Possession at the Application Layer (DPoP)*, RFC 9449, IETF, Jun. 2023. https://www.rfc-editor.org/rfc/rfc9449

[6] J. Richer, J. Bradley, A. Parecki, and J. Rischer, *Grant Negotiation and Authorization Protocol (GNAP)*, RFC 9635, IETF, Oct. 2024. https://www.rfc-editor.org/rfc/rfc9635

[7] D. Hardt *et al.*, "The OAuth 2.1 Authorization Framework," Internet-Draft draft-ietf-oauth-v2-1-12 (work in progress), Internet Engineering Task Force, 26 Apr. 2025. https://datatracker.ietf.org/doc/html/draft-ietf-oauth-v2-1-12

[8] A. Birgisson *et al.*, "Macaroons: Cookies with Contextual Caveats for Decentralized Authorization in the Cloud," in *Proc. NDSS*, Feb. 2014. https://research.google.com/pubs/archive/41892.pdf

[9] Cloud Native Computing Foundation, "Open Policy Agent," project website, accessed May 2025. https://www.openpolicyagent.org

[10] J. W. Cutler *et al.*, "Cedar: A New Language for Expressive, Fast, Safe, and Analyzable Authorization," arXiv:2403.04651, Mar. 2024. https://arxiv.org/abs/2403.04651

[11] Model Context Working Group, *Model Context Protocol — Basic Specification*, 26 Mar 2025. https://modelcontextprotocol.io/specification/2025-03-26

[12] Anthropic PBC, "Introducing the Model Context Protocol," company blog post, Dec. 2024. https://www.anthropic.com/news/model-context-protocol

[13] J. C. Martinez, "An Introduction to MCP and Authorization," Auth0 Blog, 7 Apr 2025. https://auth0.com/blog/an-introduction-to-mcp-and-authorization/

[14] Stytch, "Building an MCP Gateway on Cloudflare Workers," Technical Blog, Apr. 2025. https://stytch.com/blog/building-an-mcp-server-oauth-cloudflare-workers/

[15] A. Parecki, "Let's Fix OAuth in MCP," personal blog post, 3 Apr 2025. https://aaronparecki.com/2025/04/03/15/oauth-for-model-context-protocol

[16] S. Tiwary, "*Introducing Google Agentspace: Bringing AI Agents and AI-Powered Search to Enterprises*," Google Cloud Blog, 13 Dec. 2024. https://cloud.google.com/blog/products/ai-machine-learning/bringing-ai-agents-to-enterprises-with-google-agentspace

[17] C. Chu, "*PAM and Cloud Security: The Case for Zero Standing Privileges*," Cloud Security Alliance Blog, 22 Aug. 2024. https://cloudsecurityalliance.org/articles/pam-and-cloud-security-the-case-for-zero-standing-privileges

[18] Cloud Native Computing Foundation, *SPIFFE* (*Secure Production Identity Framework for Everyone*) *Specification, Version 1.1*, June 2023. https://spiffe.io/docs/latest/spiffe-about/overview/

[19] Cloud Native Computing Foundation, *SPIRE* (*SPIFFE Runtime Environment*) *Documentation*, accessed May 2025. https://spiffe.io/docs/latest/spire-about/

[20] Auth0 (Okta), "Secure "Human in the Loop" Interactions for AI Agents," Auth0 Blog, 11 Apr. 2025. https://auth0.com/blog/secure-human-in-the-loop-interactions-for-ai-agents/

[21] Y. Shavit *et al.*, "Practices for Governing Agentic AI Systems," OpenAI White Paper, Nov. 2024. https://cdn.openai.com/papers/practices-for-governing-agentic-ai-systems.pdf

[22] OpenAI, "A Practical Guide to Building Agents," OpenAI White Paper, Apr. 2025. https://cdn.openai.com/business-guides-and-resources/a-practical-guide-to-building-agents.pdf