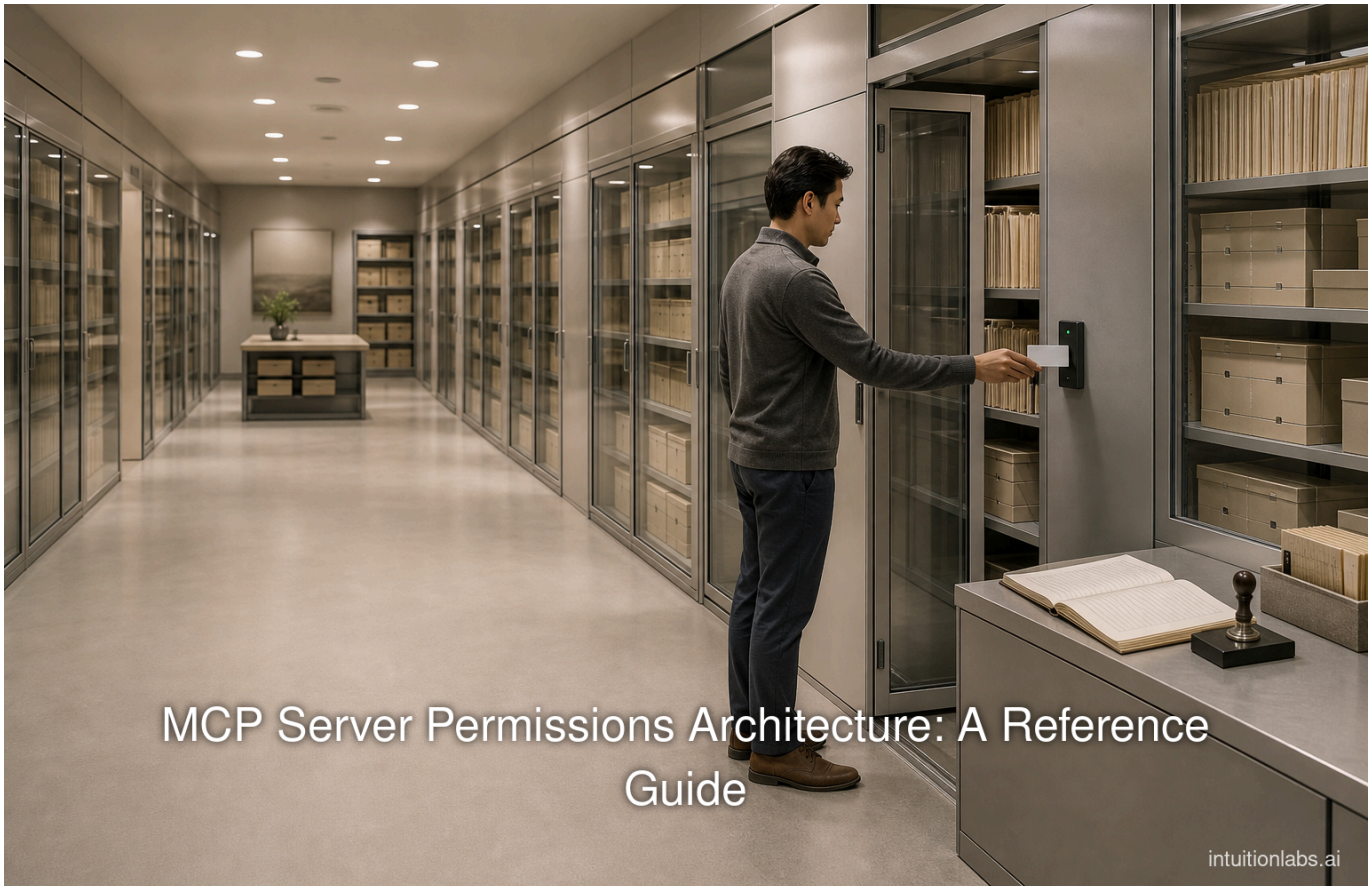
MCP Server Permissions Architecture: A Reference Guide

IntuitionLabs.ai · Adrien Laurent · 2026-09-05 · mcp · model context protocol · mcp security · mcp permissions · oauth 2.1 · tool poisoning · least privilege · ai agent security · access control · anthropic



Executive Summary

The Model Context Protocol (MCP) is an open standard, created at Anthropic and open-sourced in November 2024, that lets large language model (LLM) applications connect to external tools and data through a uniform client-server interface (anthropic.com) (anthropic.com). Its permissions architecture is not one mechanism but a layered set of primitives: an optional OAuth (Open Authorization) 2.1 authorization framework for remote servers, recommended human oversight of model-invoked Tool calls, deprecated Roots as advisory filesystem context rather than access control, and deprecated Sampling alongside Elicitation (The 2026-07-28 Specification). As of July 28, 2026, the official MCP release reported that its Tier 1 SDKs were seeing close to half a billion downloads a month (The 2026-07-28 Specification), and in December 2025 Anthropic donated the protocol's governance to the Agentic AI Foundation (AAIF), a directed fund under the Linux Foundation (anthropic.com) (linuxfoundation.org).

The specification's authorization requirements have changed substantially across four dated revisions. The 2025-03-26 update added the first OAuth 2.1 framework where none had existed (modelcontextprotocol.io); the 2025-06-18 revision reclassified MCP servers as OAuth resource servers and required Resource Indicators under Request for Comments (RFC) 8707 to bind tokens to a single server (modelcontextprotocol.io); and the 2025-11-25 revision added OpenID Connect Discovery and formalized governance under the AAIF (modelcontextprotocol.io). Authorization itself, however, remains optional in the specification (modelcontextprotocol.io), and the U.S. National Security Agency (NSA) has documented the practical consequence: MCP "currently lacks support for exchanging Role Based Access Control (RBAC) permissions at instantiation" (media.defense.gov), making cross-deployment access rules non-interoperable.

Major MCP client platforms, including Claude Code, Visual Studio Code (VS Code), and GitHub Copilot, document permission enforcement at a harness or a sandbox (code.claude.com) (code.visualstudio.com).

For organizations designing new MCP connectors, a permissions design can bind each access token to one server and a narrow scope, use confirmation for state-changing calls, and document its review process before granting write access.

Introduction and Background

The Model Context Protocol (MCP) is an open standard for connecting large language model (LLM) applications to external tools, data sources, and services through a uniform client-server interface. Anthropic open-sourced MCP in November 2024, announcing at launch that "Today, we're open-sourcing the Model Context Protocol (MCP)" (anthropic.com). The protocol was created at Anthropic by David Soria Parra and Justin Spahr-Summers (anthropic.com). In the year that followed, MCP grew from a single-vendor release into what its own maintainers describe as one of the fastest-adopted integration standards in the artificial intelligence (AI) ecosystem: by December 2025, Anthropic reported "There are now more than 10,000 active public MCP servers" and "97M+ monthly SDK downloads across Python and TypeScript" (anthropic.com) (anthropic.com).

That growth is also the reason a permissions architecture matters. Every MCP server is, by design, a bridge between a probabilistic language model and a concrete system, be it a filesystem, a customer relationship management (CRM) database, or a regulated document repository. IntuitionLabs.ai, a life-sciences and AI consultancy, frames the resulting design requirement in its own technical guidance for regulated-content connectors: an MCP integration should function as "one standardized connector that works with any compatible client and server, replacing the need for custom integrations" rather than as a bespoke, unaudited channel between a model and sensitive content (intuitionlabs.ai). A companion IntuitionLabs analysis comparing MCP to Anthropic's Claude Skills framework similarly notes that "MCP servers have defined permissions on what resources they can access", and that configuring those permissions correctly, not merely enabling the connection, is the operative security question for any deployment (intuitionlabs.ai).

This report is a companion to that earlier comparison, addressing what it does not: how MCP's permission and authorization model is actually specified, how it has changed across protocol revisions, what independent security research has found as of September 2026, and how teams can design MCP connectors that enforce least-privilege, auditable, read and write boundaries. That donation, described above, placed MCP's governance under a multi-

stakeholder foundation rather than a single company's roadmap. The sections below examine the protocol's permission primitives, enterprise enforcement patterns, and a reproducible method for scoping a new connector to the minimum access it needs.

What MCP Permissions Govern: Roles, Transports, and Trust Boundaries

MCP defines three architectural roles: a **host** application (such as Claude Desktop, Claude Code, or an integrated development environment, IDE) that embeds an LLM, one or more **MCP clients** that the host instantiates to maintain a single connection each, and **MCP servers** that expose Tools, Resources, or prompts to those clients. Whether a permission boundary exists between a client and a server, and how it is enforced, depends heavily on transport. For local integrations connecting over standard input/output (STDIO), the specification instructs implementers that "Implementations using an STDIO transport SHOULD NOT follow this specification", meaning STDIO servers instead inherit credentials from the local environment (a system account, an environment variable, an operating-system keychain) rather than negotiating a token (modelcontextprotocol.io). For remote, network-reachable servers, the 2025-06-18 revision of the specification formally casts the MCP server as an OAuth 2.1 resource server and the MCP client as an OAuth 2.1 client, stating that "A protected MCP server acts as an OAuth 2.1 resource server" (modelcontextprotocol.io).

Authorization itself, however, is not mandatory. The specification is explicit on this point: "Authorization is OPTIONAL for MCP implementations. When supported" (modelcontextprotocol.io), leaving many self-hosted and internal deployments to implement access control outside the protocol entirely. The NSA's Cybersecurity Information Sheet on MCP, published in May 2026, identifies the practical consequence of that design choice: "MCP currently lacks support for exchanging Role Based Access Control (RBAC) permissions at instantiation", so two organizations can both claim MCP compliance while enforcing entirely different, and non-interoperable, access rules (media.defense.gov).

Where the protocol does define permission mechanics, they cluster around four primitives: **Tools** (model-invocable actions for which the specification recommends a human able to deny invocations (MCP Tools specification)), **Roots** (deprecated client-provided informational filesystem context, not an access-control mechanism (MCP Roots specification)), **Sampling** (a reversed channel letting a server request a completion from the client's model), and **Elicitation** (a server's ability to request structured input from the user, bounded against collecting sensitive information). The Open Worldwide Application Security Project (OWASP) frames the underlying risk these primitives exist to contain as "Excessive Agency": "The root cause of Excessive Agency is typically one or more of" excessive functionality, permissions, or autonomy, three failure modes mapping directly onto tool scope, data access, and unsupervised action (genai.owasp.org).

The MCP Permission Model: Core Architectural Components

The OAuth 2.1 Authorization Layer

When an MCP deployment does implement authorization, the mechanics are precise. Servers must expose OAuth 2.0 Protected Resource Metadata under RFC 9728 and respond to unauthenticated requests with a specific header: "MCP servers MUST use the HTTP header WWW-Authenticate when returning a 401" (modelcontextprotocol.io), so clients can automatically discover the correct authorization server rather than depend on out-of-band configuration. Clients, in turn, must implement RFC 8707 Resource Indicators, meaning every token request "MUST identify the MCP server that the client intends to use the token with" (modelcontextprotocol.io), binding the resulting token to one specific server rather than to the user's account as a whole. Servers must reciprocate by validating that binding: the specification states that "MCP servers MUST NOT accept or transit any other tokens" (modelcontextprotocol.io) issued for a different resource, closing the door on a token obtained for one MCP server being replayed against another. The same audience-binding logic underlies the specification's prohibition on token passthrough: "The MCP server MUST NOT pass through the token it received" (modelcontextprotocol.io), a rule aimed squarely at the confused-deputy pattern in which a server forwards a client's credential to a downstream application programming interface (API) it was never scoped for. Where an MCP server itself acts as a proxy in front of a third-party identity provider, the specification requires that it "MUST obtain user consent for each dynamically registered client before forwarding" (modelcontextprotocol.io) a request onward. The NSA has separately noted a gap the specification leaves open: "MCP servers rely on bearer tokens, as defined in OAuth 2.1, without specifying protocol-level token lifecycle management" (media.defense.gov), leaving refresh, revocation, and reuse control to individual implementations.

Tool Annotations, Human Review, and Consent

Tools are the primitive most directly exposed to model-initiated action, and the specification defines descriptive annotations for tools: "clients MUST consider tool annotations to be untrusted unless they come from trusted servers", and the JavaScript Object Notation (JSON) schema itself repeats that clients "should never make tool use decisions based on ToolAnnotations received from untrusted servers" (modelcontextprotocol.io) (modelcontextprotocol.io). Practically, this means a tool's declared `readOnlyHint` or `destructiveHint` annotation is a hint for a client's user interface, not a security boundary a client can rely on for automatic authorization decisions; the schema even defaults `destructiveHint` to true, so an undeclared tool is treated as capable of destructive updates unless a trusted server states otherwise (modelcontextprotocol.io). For trust and safety, the specification recommends that a human can deny tool invocations; it does not mandate a particular review interface (MCP Tools specification): "there SHOULD always be a human in the loop with the ability to deny tool invocations" (modelcontextprotocol.io).

Roots: Scoping Filesystem and Resource Access

Roots are a deprecated way for a client to provide filesystem context to a server, most commonly for local filesystem or code-repository workflows. They are informational guidance, not an access-control mechanism: the protocol does not enforce that a server stays within them. A deployment that needs an enforced read boundary

must implement it through filesystem or operating-system controls and server-side authorization, rather than treating Roots as containment. Clients that still support Roots must validate root URIs and should obtain user consent before exposing them (MCP Roots specification).

Sampling and Elicitation Boundaries

Sampling is deprecated in protocol version 2026-07-28, and new implementations should not adopt it. The released stateless protocol uses Multi Round-Trip Requests (MRTR) for server-to-client needs such as sampling and elicitation, removing the need for held-open bidirectional streams (The 2026-07-28 Specification). **Elicitation** remains a way for a server to request structured user input during a call.

Permission Design Considerations

Permission enforcement is optional or client-defined, so deployment teams should define access boundaries, review procedures, and validation appropriate to their own environment.

Access Scope and Token Boundaries

Least-privilege configuration limits tool access to the minimum necessary. Token-audience binding and no-passthrough rules set boundaries for how credentials are used across services.

Sampling and Data Boundaries

Sampling and elicitation choices should be assessed within each deployment's data-handling and access-control design.

Table 1 summarizes neutral deployment controls for permission design.

Table 1: Permission Design Considerations

Design Consideration	Deployment Control
Tool access	Scope tools to the minimum function set and require confirmation for state-changing calls.
Configuration review	Record approved configuration and review changes before extending access.
Data boundaries	Define output handling, read scope, and logging practices for the deployment.
Token handling	Use audience-bound tokens and avoid forwarding received tokens to downstream systems.

Enterprise Implementation Patterns Across MCP Clients and Platforms

Because the specification leaves much of permission enforcement optional or client-defined, the practical security posture of an MCP deployment depends heavily on which host application and which server platform an organization chooses. Major MCP clients have converged on broadly similar patterns while differing in enforcement point and default posture.

Anthropic's own tooling enforces permissions at the harness, not the model. Claude Code's documentation states that "Permission rules are enforced by Claude Code, not by the model." (code.claude.com), and its permission rules support globs scoped to individual servers or, using the wildcard `mcp__*`, a rule that "matches every MCP tool across all servers" (code.claude.com). Deny rules take precedence over allow rules by design, "so a deny rule can't carry allowlist exceptions" (code.claude.com), preventing a broad deny from being quietly reopened by a narrower allow. Anthropic's Messages API applies a parallel model for developers connecting directly to remote MCP servers, letting them "Enable all tools, allowlist specific tools, or denylist unwanted tools" (platform.claude.com); Anthropic's own guidance adds that "Denylisting write or destructive tools is recommended when building read-only assistants" (platform.claude.com).

Microsoft's VS Code layers four independent, revocable trust boundaries (workspace, extension publisher, MCP server, and network domain), each of which "requires your explicit consent before it is trusted, and you can revoke trust at any time" (code.visualstudio.com). Its permission model spans a documented risk gradient up to an agent mode, Autopilot, that VS Code's own documentation says "auto-approves all tools and drives the agent to continue working until the task is complete" (code.visualstudio.com), and its security guidance specifically recommends that organizations "Grant tool and terminal permissions at the session level rather than workspace or user level. This limits the duration of elevated trust." (code.visualstudio.com). For local STDIO servers, VS Code additionally supports operating-system-level sandboxing so that "the server can only access the file system paths and network domains that you explicitly permit" (code.visualstudio.com), enforcing the boundary at the kernel rather than relying on the model to respect it.

GitHub Copilot's enterprise controls operate at the organization level, offering administrators a binary choice between an "Allow all" mode ("No restrictions. All MCP servers can be used." (docs.github.com)) and a "Registry only" mode ("Only servers from the registry may run." (docs.github.com)). GitHub notes the registry toggle is a public-preview convenience, and that defining an allowlist in a dedicated enterprise configuration file is the generally available alternative (docs.github.com). Microsoft Azure App Service takes a token-scoping approach for remote servers, requiring administrators to configure protected resource metadata so that "This setting configures the PRM to include the required scope" (learn.microsoft.com), letting Microsoft Entra ID issue narrowly scoped tokens rather than broad account credentials.

Table 2 below compares the enforcement point, granularity, and default posture of six MCP client and platform implementations discussed above, as documented as of September 2026.

Table 2: Permission Enforcement Across MCP Clients and Platforms

Platform	Enforcement Point	Granularity	Default Posture	Notable Control
Claude Code	Harness, outside the model	Per-server or all-MCP-tools glob	Deny takes precedence over allow	Enforced by the harness, not the model
Anthropic Messages API MCP connector	API request configuration	Per-tool allowlist or denylist	Configurable; all tools enabled unless restricted	Denylisting write tools recommended for read-only use (platform.claude.com)
VS Code	Client user interface plus OS sandbox	Per-server, per-session, per-tool	Consent required per trust boundary	Kernel-level filesystem and network sandboxing

Platform	Enforcement Point	Granularity	Default Posture	Notable Control
GitHub Copilot (Enterprise)	Organization policy	Per-registry or enterprise-wide allowlist	Admin-selected (Allow all or Registry only)	Enterprise-wide allowlist via managed-settings.json (docs.github.com)
Azure App Service	OAuth or Entra ID token issuance	Per-scope, declared in protected resource metadata	Scope required for authorization	Protected resource metadata declares required scope (learn.microsoft.com)

The pattern across *Table 2* is that every major platform has independently arrived at the same conclusion the specification itself only recommends: enforcement belongs outside the model, at a harness, an API gateway, an operating-system sandbox, or a token-issuing proxy, precisely because tool descriptions and model behavior are not reliable enforcement points on their own.

Designing a Least-Privilege MCP Connector: A Reference Method

The preceding sections describe what the specification requires and how platforms enforce it; this section sets out a reproducible method for scoping a new MCP connector, drawing on identity-vendor implementation guidance alongside the protocol's own primitives.

Step 1: Separate identity scopes from tool scopes. Auth0's technical guidance on the 2025-06-18 authorization update emphasizes that a correctly issued token should be "tightly scoped and only valid for that specific MCP server" (auth0.com), not a general-purpose credential the server happens to also use for MCP traffic. Identity platform WorkOS's developer guide states the underlying principle generally: "Each client or user is granted only the specific permissions (scopes) they need." (workos.com), and that "The MCP server enforces scopes and permissions defined by the authorization server or by its own policy." (workos.com), meaning enforcement logic must exist at the server regardless of what the identity provider issues.

Step 2: Scope beneath the underlying API, not at its level. Identity platform Descope's MCP documentation makes the case for finer granularity than most backend APIs natively expose: MCP-level scopes can be "finer-grained than the permissions the downstream services behind your MCP server actually expose" (docs.descope.com), letting an administrator grant, for example, read-only search over a document store whose underlying API only distinguishes authenticated from unauthenticated callers. Descope's documentation recommends designing explicitly for "Granular tool access (e.g., read-only vs. write actions)" (docs.descope.com) at the tool level, not only at the connection level.

Step 3: Bind every token to one server, and never forward it. This restates the specification's own audience-binding and no-passthrough rules described above: a connector that receives a credential scoped to itself must not present that same credential to a second downstream system, even one it trusts, without minting a new, narrower token for that hop.

Step 4: Make destructive actions require confirmation, not just annotation. When defining a deployment policy, a connector's client-side integration should gate any state-changing tool call behind an explicit human confirmation step when its deployment policy requires it, consistent with the specification's guidance to keep a human able to deny tool invocations (MCP Tools specification).

Step 5: Build testable deployment controls. Access scoping can be documented alongside deployment controls. A minimum control set can include configuration review at server registration and updates, validation that write-scoped calls are denied unless explicitly authorized, and output-handling rules for fields returned to model context. The NSA's guidance states that “those access paths should be explicitly denied at runtime” (media.defense.gov).

Data Analysis and Evidence

Ecosystem Scale and Growth

MCP's adoption metrics come almost entirely from two self-reported sources, Anthropic and the Linux Foundation, both dated to the December 2025 AAIF announcement; no independent survey was located, so the figures below are vendor-reported, not third-party-audited. A year after the November 2024 launch described in the Introduction, Anthropic's Chief Product Officer characterized the trajectory as: "When we open sourced it in November 2024, we hoped other developers would find it as useful" (linuxfoundation.org), pointing to a Claude connector directory that had grown to "Claude now has a directory with over 75 connectors (powered by MCP)" (anthropic.com). The Linux Foundation's own press release independently restates the industry-wide headline figure, noting "more than 10,000 published MCP servers now covering everything from developer tools" (linuxfoundation.org), and confirms the December 9, 2025 date directly: "the nonprofit organization enabling mass innovation through open source, today announced the formation of the" AAIF that same day (linuxfoundation.org).

GitHub's REST API provides the current number of users who have starred a repository.

The Authorization Timeline

Table 3 traces how the specification's authorization requirements changed across four dated revisions, the clearest evidence available that permissions architecture in MCP is a moving target rather than a fixed baseline.

Table 3: MCP Protocol Revision Timeline and Authorization Capabilities

Revision Date	Key Authorization Change	Source Quote
2025-03-26	First formal OAuth-based authorization framework added to the specification	"Added a comprehensive authorization framework based on OAuth 2.1"
2025-06-18	MCP servers reclassified as OAuth 2.1 resource servers; RFC 8707 Resource Indicators required	"Classify MCP servers as OAuth Resource Servers, adding protected resource metadata"
2025-11-25	OpenID Connect Discovery 1.0 added; governance structure formalized under SEP-932	"Enhance authorization server discovery with support for OpenID Connect Discovery 1.0"
2026-07-28 (current)	Released authorization hardening: RFC 9207 issuer validation and a shift from Dynamic Client Registration to Client ID Metadata Documents	"A set of authorization hardening changes including RFC 9207 issuer validation and a formal shift away from Dynamic Client Registration (DCR) toward client metadata documents (CIMD)." (The 2026-07-28 Specification)

Each revision in *Table 3* closed a specific, named gap rather than making a general security pass: 2025-03-26 introduced authorization where none had existed, 2025-06-18 responded to token-scoping ambiguity with formal audience binding, and 2025-11-25 added identity federation and a durable governance process now housed at the AAIF. The published 2026-07-28 revision continues that pattern by shipping issuer validation for authorization responses and formally deprecating Dynamic Client Registration in favor of Client ID Metadata Documents (The 2026-07-28 Specification).

Case Studies and Real-World Examples

The following examples illustrate how the components described above can compose into a working access boundary.

Illustrative Permission-Inheritance Pattern

An illustrative permission-inheritance pattern can limit an AI agent's effective permissions to those of the requesting user. The design goal is that an AI agent's effective permissions never exceed the requesting user's own: "can only access, search, and retrieve documents that the authenticated user is authorized to see" (intuitionlabs.ai), with "No escalation of privileges, no bypassing of access controls." (intuitionlabs.ai) as an explicit design constraint rather than an assumed property of the underlying platform. The integration describes avoiding server-side retention: "Document content is processed in real time and not stored, cached, or retained by the MCP server itself." (intuitionlabs.ai). As an adjacent advisor rather than the platform vendor, IntuitionLabs' role in this pattern is one of deployment configuration: "IntuitionLabs helps you choose the right deployment model based on your security posture, data residency requirements, and IT governance policies." (intuitionlabs.ai).

(Hypothetical Example): A Read-Scoped Literature-Search Connector for Enterprise Research

Consider a research team that wants an LLM assistant to search internal literature summaries, protocol drafts, and public guidance documents, without ever letting the assistant modify, delete, or export them. A least-privilege connector for this case would expose exactly one Tool, `search_literature`, annotated `readOnlyHint: true` (understanding these annotations are advisory, not a security boundary), and enforce its read boundary through a read-only mount plus server-side access controls. A Root, if supplied, would be deprecated advisory context only, not the security control (MCP Roots specification). Its OAuth token would carry one custom scope, `literature.read`, bound to that server per RFC 8707, with no `literature.write` or `literature.delete` scope ever issued, so a compromised token has no destructive action available regardless of what the model is instructed to do. Testable controls before production access: scan the tool manifest for hidden Unicode characters, run a fixed adversarial prompt-injection suite against the tool description, and confirm in staging that no write-scoped call succeeds even when issued directly against the server.

Implications and Future Directions

The authorization changes discussed here shipped in the published 2026-07-28 revision: clients must validate the authorization-server issuer before redeeming an authorization code, and Dynamic Client Registration is deprecated in favor of Client ID Metadata Documents. The revision also has a stateless protocol core; its Multi Round-Trip

Requests (MRTR) support mid-call input without legacy held-open bidirectional streams (The 2026-07-28 Specification).

Governance, not just protocol content, is also now a moving variable.

For research-intensive organizations, including the life-sciences sponsors and contract research organizations IntuitionLabs advises, the practical implication is that a permissions architecture chosen today should be treated as provisional. A connector built against the 2025-06-18 authorization rules should assess and adopt the published 2026-07-28 authorization requirements, including SEP-2350 scope step-up; the revision also formally shifts the standard away from Dynamic Client Registration toward Client ID Metadata Documents. IntuitionLabs' own guidance for regulated-content integrations, discussed above, already reflects this posture: choosing a deployment model is treated as an ongoing configuration decision revisited as the standard matures, not a one-time setup step.

Frequently Asked Questions (FAQs)

What is the MCP permission model? MCP's permission model is a set of primitives rather than a single mechanism: an optional OAuth 2.1 authorization layer for remote servers; tool guidance recommending a human able to deny invocations; deprecated Roots, which provide advisory filesystem context rather than access control; and Elicitation. Sampling is also deprecated, and the current stateless protocol uses Multi Round-Trip Requests for server-to-client needs (The 2026-07-28 Specification). Authorization itself remains optional in the specification, as described in the section above.

What is Model Context Protocol security architecture? Architecturally, MCP separates a host application, one or more clients, and one or more servers, with the server treated as an OAuth resource server whenever authorization is implemented. The NSA has noted this leaves role-based access control exchange unstandardized across deployments (media.defense.gov).

What are MCP data access controls? OAuth scopes can constrain token-authorized actions. Roots are deprecated advisory filesystem context and do not constrain a server to particular locations; enforced filesystem boundaries require platform controls and server-side authorization. Clients that support Roots must validate root URIs and should obtain user consent before exposing them (MCP Roots specification).

What are MCP server access control best practices? Independent guidance converges on least privilege: OWASP recommends limiting the tools an agent may call to the minimum necessary (genai.owasp.org), and a 2025 academic survey states the same principle operationally (arxiv.org).

How to scope MCP tool permissions? Scoping should happen beneath the underlying API's own permission model, not only at the connection level. Descope's documentation notes that MCP-level scopes can be made finer-grained than the API they front (docs.descope.com), while Auth0 recommends tokens scoped to one specific MCP server (auth0.com).

What is MCP data governance for research tools? Governance in practice depends on permission inheritance and non-retention, as detailed in the Case Studies section above: agents inherit only the requesting user's own access rights, and no document content is cached by the MCP server itself.

What are MCP authorization boundaries? The clearest boundary is audience binding: a server must reject tokens issued for a different resource, and the specification separately prohibits token passthrough to downstream systems, both detailed in the Core Architectural Components section above.

What does secure MCP server design for enterprise research look like? Enterprise platforms enforce permissions outside the model wherever possible, as the Claude Code and VS Code examples above illustrate.

Conclusion

MCP's permissions architecture includes specification primitives and an optional authorization layer. Effective deployments can enforce permissions outside the model through harnesses, sandboxes, scoped tokens, and confirmation policies for state-changing actions.

For teams building or procuring MCP connectors, practical choices include binding tokens to a single server and narrow scope, scoping beneath the underlying API rather than the connection level, requiring confirmation for destructive actions, and validating access controls before granting write access.
