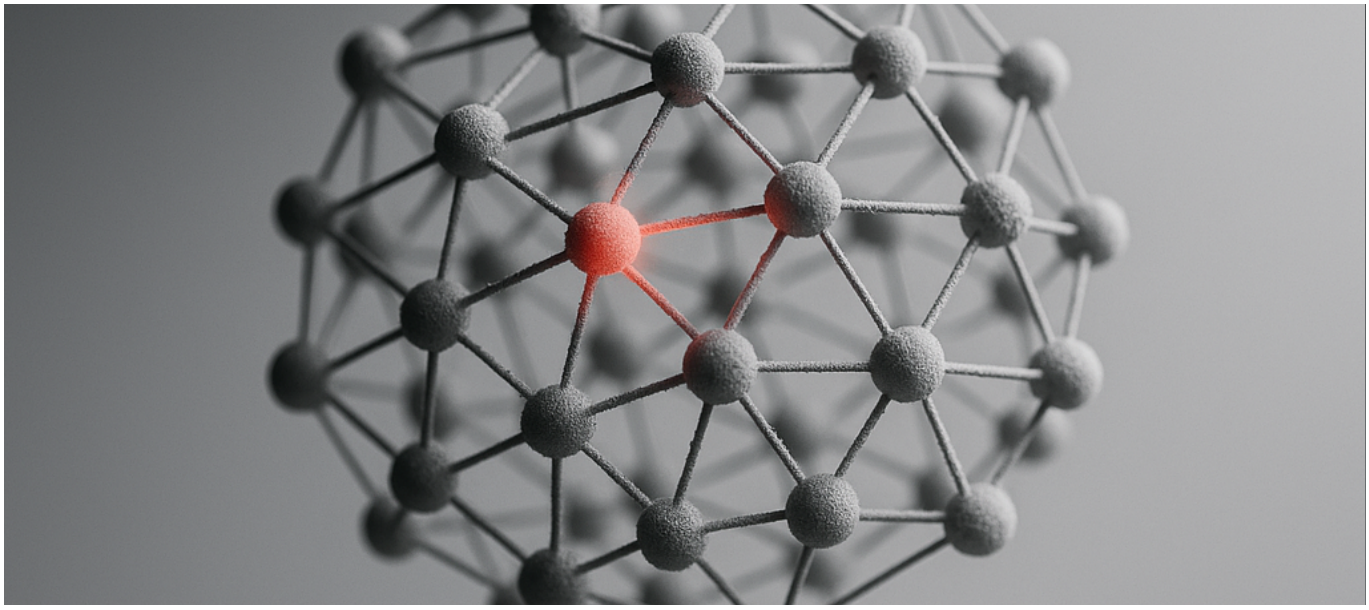

# MCP, Explained: How the Model Context Protocol Actually Works — and Why 91.8% of Servers Ship Without Auth

Every major AI vendor adopted MCP within five months of its release. Then researchers scanned 21,000 public MCP servers and found 91.8% had no authentication at all.

## Why this matters right now

The Model Context Protocol is no longer a research curiosity you can wait out. Anthropic introduced it in November 2024 as an open standard for two-way connections between AI models and external tools. Microsoft added support to Copilot Studio on **March 19, 2025** and made it a first-class standard across Windows, GitHub and Azure AI Foundry at Build on May 19, 2025. OpenAI adopted it on **March 26, 2025**, starting with the Agents SDK and moving to ChatGPT Desktop

and the Responses API. Google DeepMind committed Gemini models and SDK on **April 9, 2025**.

Five months, three of the largest AI platforms on earth. That is not standards-body pacing — that is a land rush.

The bill is now arriving as CVEs. Ten MCP server CVEs landed in NVD in a single day on August 9, 2026, dominated by SSRF and path traversal. Days later, CVE-2026-75149 in Marimo notebooks was published — MCP commands executable in edit mode before any cell is run, CVSS v4 8.7, affecting every version before 0.23.15.

Here is the part worth internalizing before we go further: **the spec is not the problem. The spec has been patched. The deployments have not.**

## The concept: a driver model where the driver writes the instructions

Think of MCP as a driver model. Before it, every AI application that wanted to read your database, hit your issue tracker, or list files on disk needed a bespoke integration — an N×M problem where N is the number of AI clients and M is the number of tools. MCP collapses that into N+M: a tool exposes itself once as an MCP *server*, and any MCP *client* can consume it.

The protocol carries tools (functions the model can invoke), resources (data the model can read), and prompts (templates). The model sees tool names, descriptions and parameter schemas, then decides what to call.

That last sentence contains the entire security story. In a classic driver model, the kernel decides what the driver may do. In MCP, a language model decides what to call based on *text supplied by the tool provider*. The description field is not documentation — it is input to the decision-maker.

Invariant Labs named the consequence on April 6, 2025: Tool Poisoning Attacks. Hidden malicious instructions embedded in tool descriptions and metadata, invisible to the human in the UI, fully visible to the model.

# How it actually works: transports, tokens, sessions

## Transport layer

Locally, MCP runs over stdio — the client spawns the server as a subprocess and talks over pipes. Remotely, the 2025-03-26 revision consolidated transports and introduced Streamable HTTP, de-emphasizing the earlier dual-endpoint HTTP+SSE pattern.

This matters for a reason most integration guides skip: the local stdio case has no network attack surface, and the remote HTTP case has all of it. Many teams develop against stdio, ship against HTTP, and never revisit their threat model.

## Authorization

The 2025-06-18 revision did the structural work. It classified MCP servers explicitly as **OAuth 2.x Resource Servers** — a server is no longer a thing that happens to check an API key, it is a thing with a defined role in a token-issuance flow.

The key addition: clients must include an RFC 8707 `resource` parameter when requesting tokens, binding each access token to a specific MCP server.

That is the confused-deputy mitigation, and it is worth spelling out. Without resource binding, a token minted for MCP server A is a bearer credential that server B can replay against A. Server B does not need to compromise anything; it just needs you to have connected to it. Resource indicators make the token useless outside its intended audience.

The same revision added structured tool output, elicitation, resource links, removed JSON-RPC batching, and shipped a dedicated security best practices document. A spec that has to add a security best-practices document nineteen months after launch is telling you where the design pressure was.

## Sessions — where the SDKs got it wrong

CVE-2026-52869 in the MCP Python SDK is the cleanest illustration of the gap between spec and implementation. SSE and stateful Streamable HTTP transports

routed requests *solely by session_id*, without verifying the authenticated principal. Guess or steal a session identifier and you inherit someone else's session. Patched in v1.27.2.

CVE-2026-34742 in the MCP Go SDK is the local-transport mirror image: DNS rebinding against localhost HTTP MCP servers running without authentication. A malicious website invokes your local MCP tools on your behalf. The server thought "localhost only" was an access control. It is not.

**Two SDKs, two languages, two variants of the same mistake: treating an identifier or a network location as an identity.**

# The deployment numbers, and what they actually mean

The spec-level fixes exist. Almost nobody is using them.

An internet-wide scan found **21,000 MCP servers reachable on the public internet, with 91.8% lacking OAuth authentication**. Then the detail that should stop you: **687 tool instances exposing shell execution without access controls.** Not "read a database." Shell.

The registry data is arguably worse, because registry servers are the ones users trust by default. An audit of the official MCP registry tested 518 servers; **214 of them — 41% — had no authentication at all and responded to `tools/list` without credentials**. An unauthenticated `tools/list` is a free reconnaissance endpoint: it hands an attacker the complete inventory of callable functions and their parameter schemas before a single exploit attempt.

Two of the August 9 CVEs remain instructive: CVE-2026-19367 (LudusMCP 1.0.24, CVSS 6.3, remotely exploitable) and CVE-2026-19339 (alibabacloud-dataworks-mcp-server 1.0.43, CVSS 6.3) — both unpatched. A 6.3 is not dramatic on its own. A 6.3 on a component wired directly into an agent that also has shell access is a different conversation.

## Clients are not clean either

An arXiv systematic analysis of MCP security empirically evaluated seven major

MCP clients and found significant issues across most of them, attributed to insufficient static validation and parameter visibility for tool metadata.

“Parameter visibility” is the polite phrase for: the human approving a tool call often cannot see what the model is actually about to send. That is tool poisoning viewed from the other end of the wire.

## Trade-offs and limits

The N+M collapse is real and valuable. Write one server, reach every compliant client. That is why adoption was measured in months rather than years, and I do not think that was a mistake by the vendors — a fragmented tool-integration landscape would have been worse. What the docs underplay:

**The spec secures the channel, not the semantics.** RFC 8707 resource indicators stop token replay. They do nothing about a tool whose description contains instructions the user never sees. OAuth confirms *who* is calling. It says nothing about whether the model was manipulated into making the call.

**Authentication is optional in practice.** The spec defines how to do OAuth properly. It cannot force a developer shipping a side-project server to a public registry to implement it. The 41% registry figure is the measured cost of that gap.

**Transport migration left debris.** The 2025-03-26 consolidation de-emphasized HTTP+SSE, but SSE code paths persisted in SDKs — and CVE-2026-52869 landed squarely in those SSE and stateful Streamable HTTP paths. Deprecation is not deletion.

**Tool composition multiplies blast radius.** An agent with one filesystem tool and one HTTP tool has an exfiltration primitive that neither tool has alone. The permission model is per-tool; the risk is per-combination.

**Local does not mean safe.** CVE-2026-34742 (DNS rebinding) and CVE-2026-75149 (Marimo executing MCP commands in edit mode before any cell runs) are both local-context bugs. Binding to 127.0.0.1 is not authentication.

**The mitigation layer is immature.** Prompt-injection filtering for tool calls is where the ecosystem’s countermeasures currently sit. I looked at one option in detail in StackOne Defender 0.8.2 in practice — useful, not a boundary you can lean

your architecture on. For how these injection paths behave once they reach a production assistant, CVE-2025-32711 “EchoLeak” is the reference case: untrusted text reaching a model that holds credentials.

## My take

**My take:** MCP is a good protocol with a distribution problem. The security data is not an indictment of the design — it is an indictment of the default.

The spec authors did the right work in the right order. Consolidate transports (March 2025), then define the authorization role and bind tokens to resources (June 2025). RFC 8707 resource indicators are the correct fix for confused-deputy, not a workaround. That is competent standards engineering under time pressure.

But 91.8% without OAuth tells me the friction curve is wrong. Setting up OAuth for an MCP server is meaningfully harder than pasting a static token into a config file, and the protocol accepts both. When a spec permits an insecure-but-easy path, that is not a developer-discipline story; it is a defaults story.

**What I expect over the next few quarters** (prediction, not established fact):

- **The CVE rate keeps climbing, and the severities stay moderate.** Ten in a day, mostly SSRF and path traversal, mostly 6.x. That is the signature of a broad, shallow attack surface — many small servers written fast. Expect volume, with exceptions like CVE-2026-75149 where an MCP path lands inside an already-privileged tool.
- **Registries become the enforcement point.** The 41%-unauthenticated finding is the most actionable number in this dataset, because it is centralized. A registry that refuses to list a server responding to unauthenticated `tools/list` would move the aggregate numbers faster than any spec revision.
- **Client-side approval UX gets rebuilt.** The arXiv finding on parameter visibility across seven clients points at the real chokepoint. Tool poisoning only works because the human sees a summary and the model sees the payload.

**If you run MCP in production, the short list:**

- Pin the MCP Python SDK to v1.27.2 or later for CVE-2026-52869.
- Treat every tool description as untrusted input. It reaches your model’s decision loop.

- Assume `tools/list` is public unless you have explicitly authenticated it. 41% of registry servers assumed otherwise.
- Audit shell-capable tools separately. 687 public instances currently offer shell execution with no access controls — do not become 688.
- Never treat localhost binding as an access control. DNS rebinding (CVE-2026-34742) exists precisely to break that assumption.

The interesting thing about MCP's security record is how mundane it is. SSRF, path traversal, session hijacking, DNS rebinding — a 2010 web-application vulnerability list, applied to a new class of client that happens to be a language model with credentials and no skepticism.

**MCP's spec has been patched twice for structural security gaps; the 91.8% of servers shipping without authentication were never blocked by the spec in the first place, and that is where the actual work lies.**