What is the Model Context Protocol (MCP)?

The Model Context Protocol (MCP) is an open standard that connects AI applications to external tools and data. An MCP host (the AI app) runs clients that connect to MCP servers over stdio for local servers or HTTP for remote ones, using JSON-RPC 2.0. Servers expose tools the model can call, resources it can read and prompt templates. Remote servers are protected with OAuth 2.1. Because tool descriptions and outputs flow straight into the model's context, every MCP server is a trust decision: attacks include tool poisoning, tool shadowing, token passthrough and confused deputy flaws.

HOW IT WORKS

01 Tools, resources and prompts

A server offers up to three kinds of capability, each with a name, a description and a typed schema the client discovers when it connects:

- Tools are functions the model can call: create an issue, run a query, send a message. The specification is explicit that tools represent arbitrary code execution and must be treated with caution.
- Resources are data the model or user can read, such as files, records or documents.
- Prompts are reusable message templates the user can invoke.

Clients can also offer features back to servers, such as elicitation, where a server asks the user for more information mid-task. The key security fact is simple: tool names, descriptions and outputs all become text in the model's context, and the model cannot reliably tell instruction from data.

02 Where MCP's trust boundaries sit

MCP deliberately leaves enforcement to implementations. The specification sets principles (user consent before data access, explicit approval before any tool runs, tool descriptions treated as untrusted unless the server is trusted) and states that the protocol itself cannot enforce them. That puts three boundaries in your hands:

SOURCES

[1] Model Context Protocol specification

[2] MCP specification: Authorization

[3] MCP Security Best Practices

[4] OWASP Top 10 for Agentic Applications (2026)

- Which servers the host connects to. Installing an MCP server is closer to installing software than to adding a bookmark. A local server runs with the user's privileges.
- What the server is allowed to do downstream. For remote servers, the specification uses OAuth 2.1: the MCP server is an OAuth resource server, clients must request tokens bound to that specific server with RFC 8707 resource indicators, and servers must reject tokens issued for anyone else.
- What the model is allowed to do with the tools. Least privilege, scoped tools and human approval for high-impact actions are host-side decisions. See excessive agency for what happens when that boundary is too wide.

03 Common attacks on MCP

- [MCP tool poisoning](/learn/ai-security/mcp-tool-poisoning): a server hides instructions in a tool description, and the model obeys them.
- [MCP tool shadowing](/learn/ai-security/mcp-tool-shadowing): one server's tool description changes how the agent uses a different, trusted server's tools.
- Token passthrough: a server accepts tokens that were not issued for it and forwards them downstream. The specification forbids it because it breaks audit trails and lets a stolen token ride through the server.
- Confused deputy: an MCP proxy using one static OAuth client ID can be tricked into handing an attacker an authorization code without fresh user consent.
- SSRF during OAuth discovery: a malicious server points the client's metadata fetches at internal addresses or cloud metadata endpoints.
- Local server compromise: a malicious install command or package runs with the user's full privileges.
- [indirect prompt injection](/learn/ai-security/indirect-prompt-injection): data a tool returns (a web page, an email, a ticket) carries instructions the model follows.

04 How to use MCP safely

- Allowlist servers. Keep a reviewed catalog of approved servers and versions; block ad hoc installs in production. Route remote servers through an an MCP gateway so authentication, policy and audit live in one place.
- Pin and diff tool definitions. Re-approve when a server's tool descriptions change.
- Scope everything. Request the smallest OAuth scopes, give each agent its own AI agent identity, and require approval for writes and destructive actions.
- Isolate execution. Run local servers and code-executing tools in a sandbox. See AI agent sandboxing.
- Log every tool call with its arguments and result, tied to the user and agent that made it.
- Test it adversarially. MCP integrations need the same pentest as any API, plus prompt-level attacks against the agent that uses them.

Test your AI application before an attacker does.

securelayer7.net/learn/ai-security/model-context-protocol

Open online