# OWASP

## GenAI SECURITY PROJECT
### TOP 10 FOR LLM AND GENERATIVE AI

# A Practical Guide for Securely Using Third-Party MCP Servers

# LICENSE AND USAGE

# Table of Content

# Introduction & Background

This cheat sheet is intended for companies and developers who plan to use a third-party MCP, meaning one that was not developed within their own organization. It isn't a cheat sheet for MCP developers.

The **Model Context Protocol (MCP)** is an open protocol that standardizes how LLM/agent hosts connect to external tools, data, and prompt templates via a server. By connecting models to systems such as databases, APIs, and internal applications, MCP unlocks powerful automation and allows AI to perform actions beyond text generation. However, it also presents a new attack surface that could open organizations up to malicious attacks. Since tools can execute code, access files, and make network calls, a compromised MCP stack can lead to data theft, malicious code execution, and system sabotage.
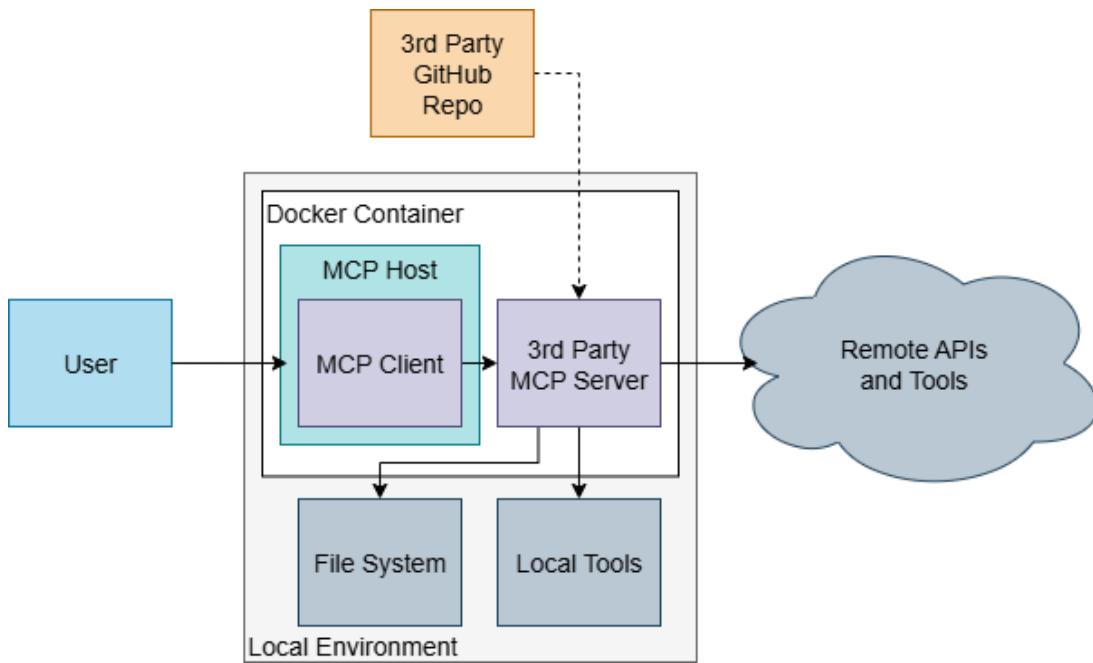
MCP uses a simple client-server model over two primary layers: a **data layer** for API semantics (JSON-RPC) and a **transport layer** for communication (e.g., STDIO, Streamable HTTP).

- **MCP Host**: The application (e.g., AI agent runtime) that manages clients and routes requests.
- **MCP Client**: The connector that communicates with a single MCP server.
- **MCP Server**: A program that exposes tools and context (e.g., calendars, SaaS APIs) to clients. MCP defines three core primitives that servers can expose:
  - Tools: Executable functions that AI applications can invoke to perform actions (e.g., file operations, API calls, database queries)
  - Resources: Data sources that provide contextual information to AI applications (e.g., file contents, database records, API responses) to the model.
  - Prompts: Reusable templates that help structure interactions with language models (e.g., system prompts, few-shot examples)
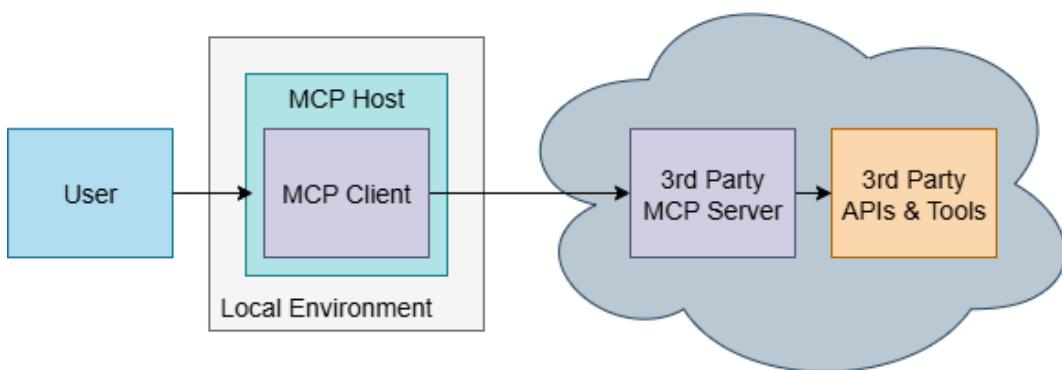
This cheat sheet outlines key security controls for developers using a third-party MCP server (using MCP securely).

# Use Cases



A downloadable MCP server that runs locally, using native tools and system-level permissions to perform tasks independently of external infrastructure. These systems will often use STDIO for communication between the MCP client and server. While core operations are executed locally, some functionality, such as LLM execution or remote API calls, may interface with remote services depending on configuration.



This MCP client connects to the server via MCP's HTTP Streamable communication, with the server running remotely. The server is capable of interacting with external tools and APIs, often hosted in cloud environments. Core considerations for this deployment include authentication/authorization, server discovery, and unreviewed MCP tool changes.

# Current Vulnerability Landscape

Common attack patterns have begun to emerge around MCP, many focused on malicious inputs designed to influence an LLM, or attacks that impact the tools an LLM calls.

## Tool Poisoning & Rug Pull Attacks

Tool poisoning is a form of indirect prompt injection where adversaries embed malicious commands within a tool's description or parameters. The LLM executes these hidden instructions, potentially leading to unauthorized data access or unauthorized file system access. A rug pull is a specific type of tool poisoning where a legitimate tool is secretly replaced with a malicious version, leveraging the trust established by the original tool to execute the attack.

**Mitigations:**

- **Enforcing Full Tool Transparency:** MCP clients should display the full tool manifest, including descriptions, parameters, and capabilities, before user activation. Avoid rendering shortened summaries that can hide injected instructions.
- **Sanitize Descriptions**: Review descriptions for suspicious keywords or markup. Refer to the OWASP LLM Prompt Injection Prevention Cheatsheet. Review the full MCP code if available, and compare tool names to descriptions and code to identify any suspicious mismatches.
- **Monitor Tool Integrity and Version**: Clients should pin the version of the MCP server and its tools at the time of initial user approval. Use a hash or checksum to verify that tool descriptions and functionality haven't been maliciously altered. Maintain a version history and alert administrators on any unauthorized changes.
- **Runtime Policy Enforcement:** Enforcing least-privilege policies at the MCP server boundary (host/container/proxy) to restrict tools from reading local files, accessing sensitive APIs, or exfiltrating data.

## Prompt Injection

Attackers craft malicious user inputs, content retrieved by tools or tool arguments to hijack the model's context and force it to perform unintended actions, such as calling a dangerous tool, leaking conversation history, or overriding safety policies.

**Mitigations:**

- **Validate Untrusted Data**: Sanitize and validate all external data, such as tool descriptions and server responses, and user inputs before passing it to the model. Refer to the [Input Validation and Sanitization](#) section in the [OWASP LLM Prompt Injection Prevention Cheat Sheet](#).
- **Use Strong Schemas**: Define and enforce strict JSON/YAML schemas for all tool inputs. Use libraries like [Pydantic](#) (Python) or [JSON Schema](#) for validation.
- **Segment Contexts**: Segment client interactions with different users, and establish new connections/sessions with third-party MCP servers for new or distinct operations.

# Memory Poisoning

This attack corrupts an agent's short-term or long-term memory (e.g., a vector database), causing it to store false information, bypass security checks, or make flawed decisions. This can lead to systemic misinformation or privilege escalation.

**Mitigations:**

- **Secure and Validate Memory**: Enforce validation on every memory update. Scan for anomalies, require source attribution, and use cryptographic hashes.
- **Limit Memory Retention**: Implement a Time-To-Live (TTL) on stored data to prevent outdated or malicious information from persisting.
- **Segment Memory**: Prevent agents from writing to a shared memory store to avoid cascading failures. Isolate memory by session or user identity.

# Tool Interference

Using multiple MCP servers can lead to unintended tool execution chains. An LLM's output from one tool might accidentally trigger a tool from another server, causing data leaks, accidental operations, or denial-of-service loops.

**Mitigations:**

- **Require Human-in-the-Loop**: For user-configurable servers, implement an acceptance flow before executing a tool to ensure the user is aware of the action. For agentic systems, opt for a tiered HITL strategy to enable [robust oversight](#).
- **Separate Context**: Isolate the context for each tool execution, passing only the necessary information between them. Reset LLM context between distinct executions.
- **Set Timeouts**: Implement execution timeouts to prevent poorly implemented or looping tools from impacting the host.

# Client Security and Server Discovery

Understanding the distinct security responsibilities and attack surfaces of MCP clients and servers is crucial for implementing comprehensive protection. Each side faces unique threats and requires tailored security measures.

## MCP Client Security Considerations

Clients are often embedded in applications. They act as consumers of tools and are the first line of defense against malicious servers.

- **Trust Minimization:** Do not assume servers are trustworthy; always validate manifests, enforce schemas, and apply allowlists.
- **Sandbox Execution:** Run clients in restricted environments (e.g., containers with limited filesystem/network access) to reduce impact if a server is compromised.
- **Use Containers (e.g. Docker) to run third-party MCP servers.** Docker containers provide a standardized, isolated environment, preventing a compromised server from accessing the host system's resources, files, or network.
- **Just-in-Time (JIT) Access:** Grant tools temporary, narrowly scoped permissions only for the duration of a specific task, with automatic expiration and instant revocation.
- **UI Transparency:** Expose full tool descriptions, permissions, and data access to users before execution. Avoid hidden or "summarized" views that mask risk.
- **Local Data Protection:** Prevent servers from exfiltrating client-side secrets, history, or cached memory. Sanitize any data passed into the LLM.
- **Incident Detection:** Monitor for unusual tool invocation patterns (e.g., mass file reads, excessive API calls) and trigger alerts.

## Server discovery and verification

An important phase in using MCP servers is "discovery", where servers are identified, verified, and connected.

### Discovery & Connection

- **Verify Origin Before Connecting**: Only connect to servers from a trusted registry. Use IP allowlists and network isolation to prevent connections to unknown servers. For open-source tools or locally hosted servers, always verify the source, check signatures, and scan for known vulnerabilities or malicious modifications.
- **Use Registry-Only Discovery**: Maintain a central registry for all approved servers.
- **Choose the Hosting and Connection Paradigm**:
  - **STDIO (Local Connections)**: For locally hosted MCP servers (downloaded from a third-party repository or vendor), utilize STDIO as a connection to an MCP server sub-process. This approach offers the lowest latency and is easier to harden via process sandboxing, seccomp, or AppArmor.
  - **Streamable HTTP (Remote Connections)**: For remote, multi-tenant, or auto-scaling servers, utilize a remote connection via Streamable HTTP. Use TLS/mTLS or OAuth 2.1 for robust authentication, and protect with a WAF and rate limiters. Rotate credentials (API keys, certs) regularly.

## Verification

- **Pin Versions**: Maintain a manifest of approved server and tool versions. Use checksums to verify integrity and prevent silent, malicious upgrades.
- **Staged Rollout**: First, enable new servers in a staging environment with full telemetry. Promote to production only after a probation period with no incidents. Automate rollback procedures in case of regression or compromise.
- **Record Human Approvals**: Log all security and domain-owner approvals in the registry. Monitor and alert on drift from the approved server inventory.

# Authentication and Authorization

Authentication and authorization are both necessary for secure connections to MCP servers, locally or remotely. MCP implements OAuth 2.1 as part of its specification, and can be used as a guide for implementation, noting that different MCP architectures will require distinct OAuth setups.

## Authentication

- **Utilize Client Credentials for system operations.** In cases where MCP clients will interact as systems with no user identity, authenticate to remote MCP servers using identity provider client credentials.
- **Utilize OIDC/PKCE for user operations**. For use cases with a user tightly connected to the operation, like for chatbots or AI assistants, use OIDC and PKCE for user login, and utilize user credentials for any authentication to downstream MCP servers.

- - Where OAuth cannot be implemented, use narrowly scoped and short-lived Personal Access Tokens to identify and restrict user access.
- **Protected Registration.** [Dynamic Client Registration](#) is not advised for most use cases. However, if it is necessary, authorization servers that do not wish to provide unauthorized registration of clients have several options to protect their registration endpoint.
  - **Access Token**: The server may require authorization (e.g., via OAuth access tokens) with registration requests. Most often, this token can be requested manually outside of the client registration flow.
  - **Software Statements**: Clients may provide a [Software Statement](#), which is a signed JWT asserting client metadata. Authorization servers may validate it to establish trust.
  - **Signed Request Bodies**: Servers may require registration requests to be signed and verify signatures based on preconfigured keys.

## Authorization

- **Define least-permission OAuth scopes:** Restrict access to resources using OAuth scopes to ensure MCP clients or users do not have excessive access.
- **Use granular permissions:** Use action-level permissions on a per-identity basis to further limit permissions to specific actions, or individual data elements.
- **Human-in-the-Loop controls:** Require human-in-the-loop approval for actions that have not previously been performed, like local MCP server access to the file system, or remote MCP execution against new resources, like accessing emails.

# Tools & Utilities

No tool will provide complete coverage, and the tools listed will vary in maturity and effectiveness. Use these lists as a jumping-off point to research and match tools for your use case and technical requirements.

## Automated Scanners

- Invariant Labs MCP-Scan: Scans for malicious descriptions in MCP tools and monitoring connections to catch issues like prompt injections, tool poisoning, and unsafe data flows.
- Semgrep MCP Scanner: A static analysis tool with rules for MCP(beta), which scans Python and Node.js dependencies, and integrates with MCP-Get to scan all listed servers.
- mcp-watch: Scan vulnerabilities such as insecure credentials storage and tool poisoning attempts.
- Trail Of Bits mcp-context-protector is a security wrapper for MCP servers that addresses risks associated with running untrusted MCP servers.
- Vijil Evaluate is a platform that evaluates your AI agents for reliability, safety, and security.

## Content Monitoring/Moderation

- LangKit: A toolkit for monitoring LLM outputs.
- OpenAI Moderation API: An API for detecting inappropriate content
- Invariant Labs Invariant: Contextual guardrails for securing agent systems.
- LlamaFirewall: A framework with scanners for risks affecting agenting LLM use.

## Open Source Management

- OpenSSF Scorecard: verify repository maturity.
- Snyk package health: verify repository maintenance.

## Sandboxed Execution

- Docker: Run MCP servers inside Docker Containers for both local and remote compute, to limit access to local files and prevent malicious tools from escaping the defined operating environment.

# Governance

Establish a formal governance strategy to ensure only vetted and monitored MCP servers are used. The core of this strategy is a **trusted MCP registry**.

The registry should hold metadata about each approved server and enforce policy at runtime. In this way, you will be able to ensure only **vetted, least-privileged, continuously monitored** MCP servers are available to hosts/agents, with **policy-as-code** gates and evidence for audit.

## Governance Workflow

1. **Submission**: A developer submits a new third-party MCP server for review with its documentation and a hash of its tool descriptions.
2. **Scanning**: Automated security tools analyze the server (or code repository if available) for malware, hidden instructions, and other risks.
3. **Review & Sign-off**: Security and domain experts review the scan results. Approved servers are version-pinned and added to the registry.
4. **Deployment & Monitoring**: The server is deployed to staging and monitored. After a probation period, it is promoted to production.
5. **Periodic Re-validation**: Servers are automatically re-scanned at regular intervals or when versions change.

## Recommended Roles & Responsibilities

- **Submitter (Developer)**: Proposes a new server integration and provides documentation.
- **Security Reviewer**: Validates security controls, scans, and supply chain integrity.
- **Domain Owner**: Confirms the server's functional necessity and approves scopes.
- **Approver**: Security and domain owners must both sign off before a server is enabled.
- **Operator (SRE)**: Manages connection rollout, monitoring, and the kill-switch.

# Acknowledgements

## Contributors

# OWASP GenAI Security Project Sponsors

We appreciate our Project Sponsors, funding contributions to help support the objectives of the project and help to cover operational and outreach costs augmenting the resources provided by the OWASP.org foundation. The OWASP GenAI Security Project continues to maintain a vendor neutral and unbiased approach. Sponsors do not receive special governance considerations as part of their support.

Sponsors do receive recognition for their contributions in our materials and web properties. All materials the project generates are community developed, driven and released under open source and creative commons licenses. For more information on becoming a sponsor, visit the Sponsorship Section on our Website to learn more about helping to sustain the project through sponsorship.

## Project Sponsors:

# Project Supporters

Project supporters lend their resources and expertise to support the goals of the project.

| | | | |
|---|---|---|---|
| Accenture | Cobalt | Kainos | PromptArmor |
| AddValueMachine Inc | Cohere | KLAVAN | Pynt |
| Aeye Security Lab Inc. | Comcast | Klavan Security Group | Quiq |
| AI informatics GmbH | Complex Technologies | KPMG Germany FS | Red Hat |
| AI Village | Credal.ai | Kudelski Security | RHITE |
| aigos | Databook | Lakera | SAFE Security |
| Aon | DistributedApps.ai | Lasso Security | Salesforce |
| Aqua Security | DreadNode | Layerup | SAP |
| Astra Security | DSI | Legato | Securiti |
| AVID | EPAM | Linkfire | See-Docs & Thenavigo |
| AWARE7 GmbH | Exabeam | LLM Guard | ServiceTitan |
| AWS | EY Italy | LOGIC PLUS | SHI |
| BBVA | F5 | MaibornWolff | Smiling Prophet |
| Bearer | FedEx | Mend.io | Snyk |
| BeDisruptive | Forescout | Microsoft | Sourcetoad |
| Bit79 | GE HealthCare | Modus Create | Sprinklr |
| Blue Yonder | Giskard | Nexus | stackArmor |
| BroadBand Security, Inc. | GitHub | Nightfall AI | Tietoevry |
| BuddoBot | Google | Nordic Venture Family | Trellix |
| Bugcrowd | GuidePoint Security | Normalyze | Trustwave SpiderLabs |
| Cadea | HackerOne | NuBinary | U Washington |
| Check Point | HADESS | Palo Alto Networks | University of Illinois |
| Cisco | IBM | Palosade | VE3 |
| Cloud Security Podcast | iFood | Praetorian | WhyLabs |
| Cloudflare | IriusRisk | Preamble | Yahoo |
| Cloudsec.ai | IronCore Labs | Precize | Zenity |
| Coalfire | IT University Copenhagen | Prompt Security | |

**Supporters list, as of publication date. Find the full supporter [list here](#).**