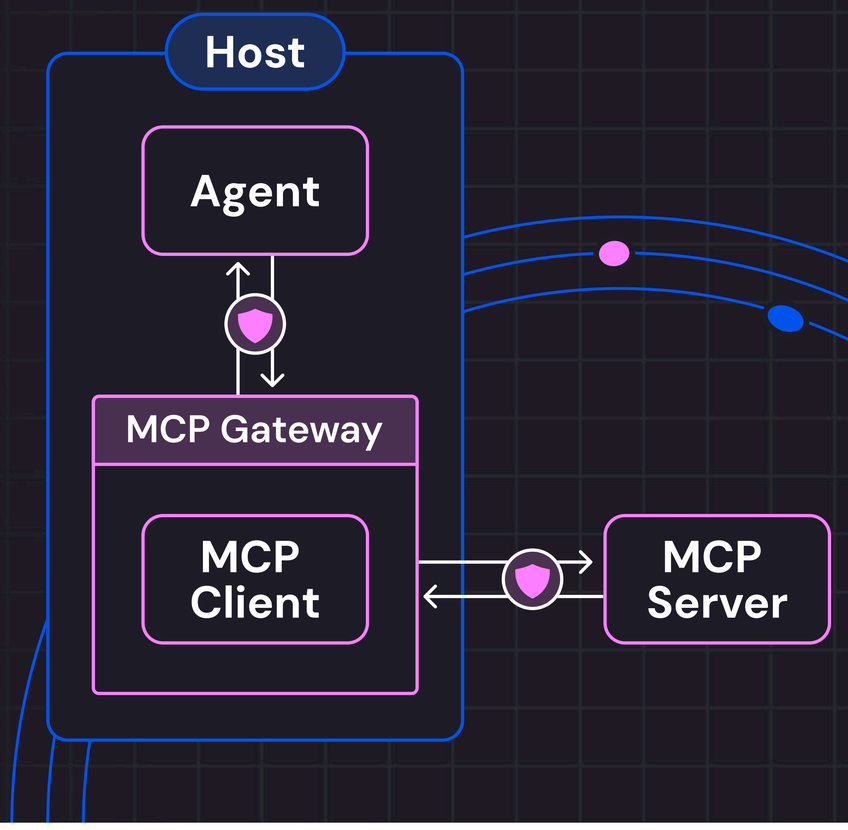
Cheat Sheet

Model Context Protocol (MCP) Security Best Practices



Model Context Protocol (MCP) is a fundamental connection method in the AI landscape. It enables large language model (LLM) agents to access live data and tools, much like how a USB-C port connects external devices. While this capability provides significant advantages, it also opens new opportunities for attackers to gain control of the system. Attackers can execute supply-chain attacks, use prompt triggers for remote code execution, and silently steal data by bypassing MCP system defenses.

Created by Anthropic in 2024, MCP has gained universal acceptance as a standard connection protocol after endorsements by OpenAI and Hugging Face. Its adoption has rapidly expanded, challenging security teams to keep pace.

In this guide, you'll find actionable tips for how to keep MCP services safe. Let's get started.

1 Enforce strict server and supply-chain security

The MCP server is more than an API endpoint; it executes code with user-level permissions, akin to a browser extension with session access. If a malicious or hijacked package performs a rug-pull update that inserts credential-stealing code, any agent loading that package will be compromised. That means your MCP supply chain requires the same level of security applied to your container images.

- **Source servers responsibly:** Never retrieve servers from unknown internet sources. Prioritize vendors and open-source projects that use Sigstore signatures for their images. (Sigstore code signing validates authorship by linking code to its creator.) Also, your CI/CD pipeline must automatically reject unsigned build artifacts.
- **Maintain an internal allowlist:** Control agent connections to the public internet using an internal allowlist. The MCP gateway and client policy should block server connections that don't match your allowed server list, protecting against compromised servers regardless of local configuration changes.
- **Pin and verify versions:** Avoid the "latest" tag as it's mutable and can lead to unexpected vulnerabilities if the underlying code changes without your knowledge. Always pin to an immutable digest, and ensure the operating server's hash matches your accepted value, preventing stealthy unauthorized modifications.

You'll also want to evaluate and harden MCP clients used by your organization:

- **Prioritize security maturity:** When selecting clients, evaluate explicit approval flows, the ability to export audit logs, role-based access control, and verification of signed plugin manifests.
- **Prevent tool-name collisions:** If two active servers expose a tool named `run_analysis`, the client may call the wrong one. Standardize tool names and prefer clients that namespace tools by server to prevent workflow hijacking.

- **Use browser isolation:** If your client is a browser extension, scope site permissions tightly and run in isolated worlds so page JavaScript cannot tamper with the extension.

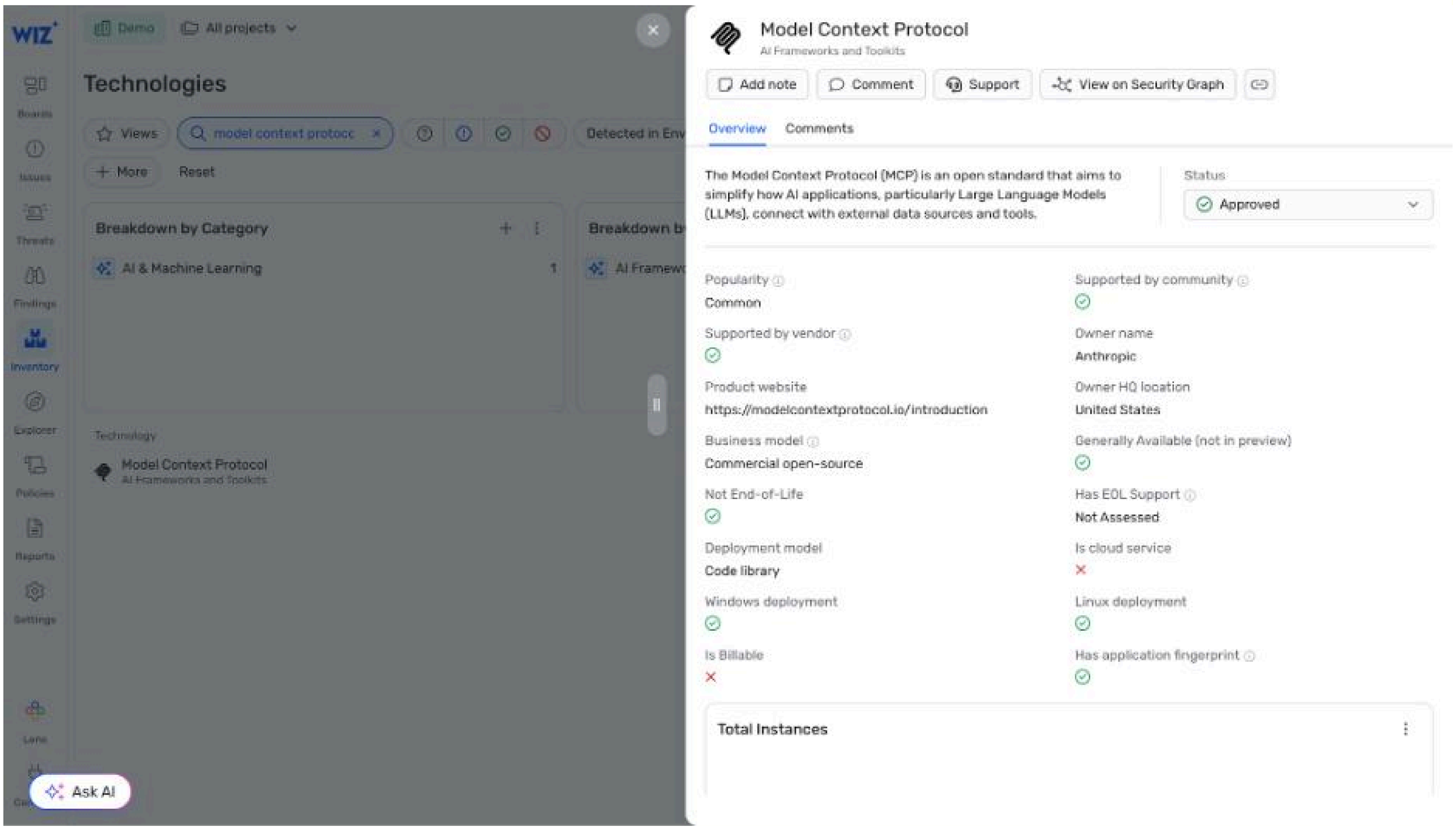


Figure 1: Technology inventory and MCP server instances found by Wiz

2 Adhere to least privilege for context and tooling

Attackers commonly elevate privileges in MCP environments by exploiting over-scoped tokens and permissive tools. Successful prompt injection attacks can allow attackers to move between your cloud management interface, Git repositories, and production databases. OWASP classifies prompt injection as LLM01 due to its proven real-world impact.

- **Scope credentials tightly:** MCP servers should never have long-lived, "god-mode" tokens. Instead, use short-lived tokens that are automatically rotated and include an audience restriction. For example, a tool that only reads from a specific S3 bucket only needs `s3:GetObject` permissions on `arn:aws:s3:::that-specific-bucket/*`.
- **Enforce tool annotations in policy:** Many MCP tools include annotations like readOnlyHint and destructiveHint, but the central gateway needs its own policy to stop unauthorized actions, regardless of the tool's declared intent. For instance, the gateway should prevent POST, PUT, and DELETE requests from tools.
- **Partition human vs. agent roles:** Create specific IAM or service account roles for your MCP agents. An automated server should never use the same IAM role as a human administrator. This ensures a clear audit trail and prevents compromised agents from accessing full human-level permissions.
- **Monitor and alert:** Track all authentication and authorization events. Trigger alerts for unusual activity, such as service account tokens with read-only permissions requesting write permission tokens. This indicates a significant security threat.

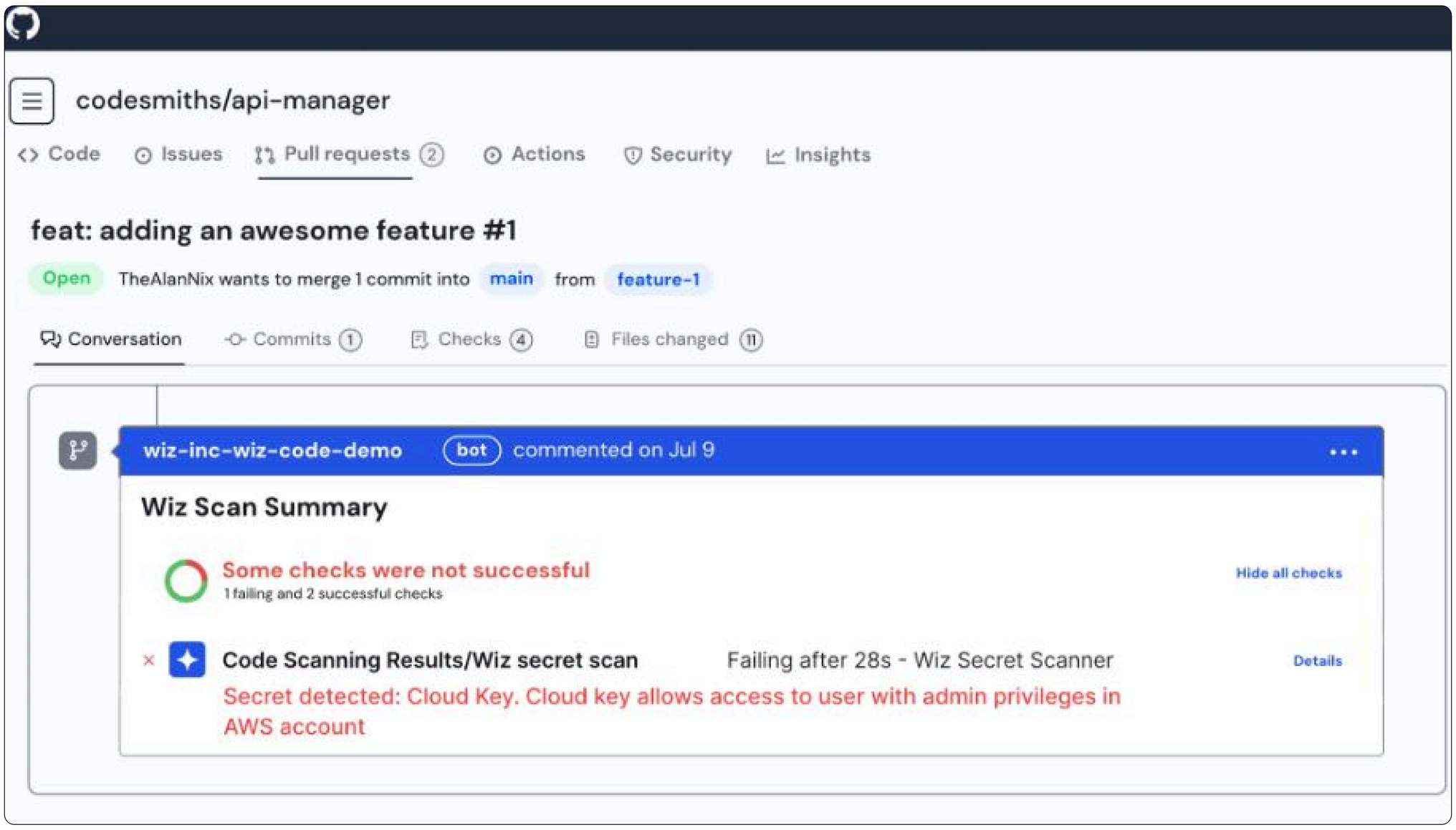


Figure 2: A Wiz Scan Summary highlighting an over-privileged token

3 Implement human-in-the-loop and output validation

Unmonitored, auto-run LLM agents can quickly escalate a harmless prompt-injection glitch into a full-scale production outage. This rapid automation creates significant problems; for example, the DeepSeek R1 model was successfully jailbroken in 50 different prompt tests. Since models cannot self-monitor, human intervention is crucial for sensitive operations.

- **Require user confirmation:** MCP client configurations should include a confirmation dialog that prompts users to authorize write and execute tool calls. This dialog should offer more than basic "Allow/Deny" options, presenting a preview that shows the upcoming action: “This action will execute g`cloud compute instances delete prod-db-1`"
- **Sanitize untrusted content:** Any text passed to the model or tool must be stripped or escaped of dangerous content before processing. Your system needs to eliminate code blocks, neutralize command injection characters, and remove markdown that could perform IP address exfiltration.
- **Default to manual review:** New or untrusted MCP servers should default to manual review. Additionally, each tool execution must undergo manual review before users or their teams can allow auto-run on a trusted server, following a model of explicit trust.

4 Isolate and sandbox MCP environments

When an MCP server is infiltrated, whether on a local machine or in a Kubernetes cluster, it creates immediate security risks. An attacker could use this entry point to access local files, move between systems, or launch RCE attacks. Your server security design should assume breaches and minimize the damage area.

- **Run in non-root containers:** This is a fundamental principle of container security. Build server images with a non-root user and group (USER 1001). In your pod security context, disable all privilege escalation features to prevent a compromised process from escaping to the host node.
- **Apply an egress allowlist:** Prevent your MCP pods from communicating with the entire internet. The pod should only be able to initiate connections to a specific, allowlisted set of APIs, while all other outbound traffic should be dropped.
- **Filter system calls:** For highly sensitive workloads, attach a custom seccomp (secure computing mode) profile to your container. This allows you to block dangerous system calls rarely needed by a typical web service, such as ptrace, mount, and clone, dramatically reducing the kernel attack surface.
- **Enforce resource quotas:** A typical abuse pattern for compromised compute is crypto-mining or denial-of-wallet attacks. Enforcing CPU and RAM quotas for your MCP deployments ensures that even if a server is compromised, it cannot consume excessive resources (which drives up cloud bills and/or starves other applications).

5 Centralize control via an MCP gateway and proxy

As your AI ecosystem grows, you will quickly have dozens of direct client-to-server tunnels, a sprawl that is impossible to secure. An MCP gateway provides a single pane of glass for observability, policy enforcement, and anomaly detection, much like an API gateway for microservices.

- **Enforce mTLS through the gateway:** All MCP traffic must tunnel through the proxy. The gateway should enforce mutual TLS (mTLS), where both the client and server present valid, signed certificates for authentication. This prevents unauthenticated components from accessing the ecosystem.
- **Full-fidelity audit logging:** The gateway is the ideal place for high-quality audit logs. For every transaction, it should record the prompt, the full tool call with arguments, response latency, and response size. These structured logs should be streamed to your SIEM with an explicit schema including user IDs, source IPs, and tool IDs.
- **Implement inline guardrails:** Since all traffic flows through the gateway, it can act as an inline policy enforcement point. You can deploy data leak prevention (DLP) sensors to redact sensitive data, use regex to block known prompt-injection patterns, or integrate with ML-based sensors to identify and block toxic or malicious content.
- **Rate limit and maintain a binary allowlist:** The gateway should enforce your server allowlist, refusing to connect to any server whose digest isn't approved. It's also the ideal place to implement rate limiting to throttle anomalous request bursts from a single client, which could indicate a runaway script or an attack.

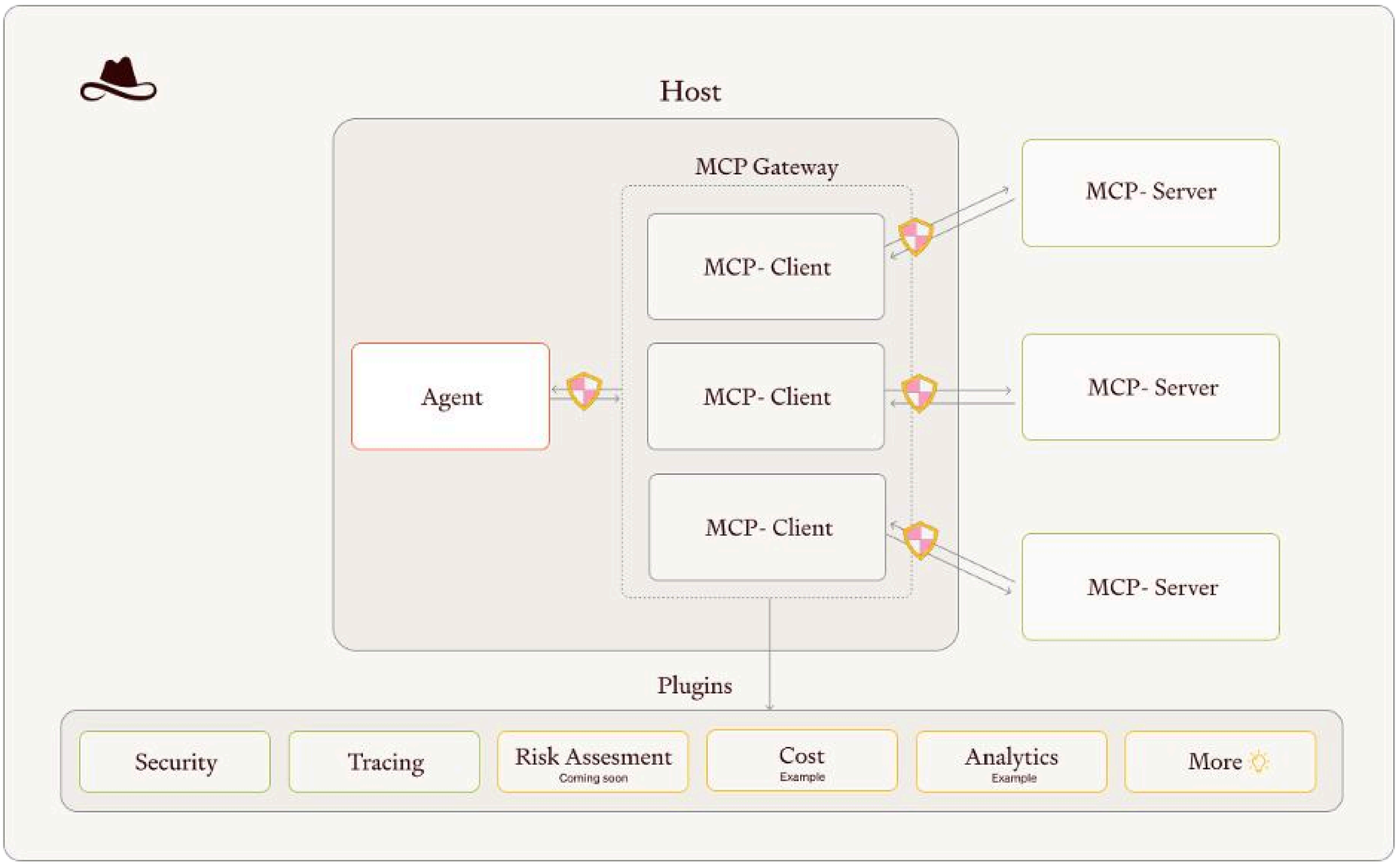


Figure 3: MCP gateway overview (Source: GitHub)

6 Evaluate and select secure MCP servers

Evaluate providers against the criteria below and require contractual commitments:

- **Prioritize vendors with secure development practices:** Ask for documented threat models, SBOMs, SLSA or Sigstore attestations, and CI pipelines that run SAST/DAST.
- **Demand granular access controls from providers:** Expect first-class RBAC, short-lived scoped tokens, default-deny tool policies, per-tool allow/deny lists, and mTLS.
- **Require immutable and exportable logging:** Logs should be structured JSON with request IDs, streamed to your SIEM, and optionally written to append-only storage.
- **Verify patching and vulnerability management:** Vendors should commit to SLAs for critical CVEs, provide change notes, and support safe rollout/rollback options.
- **Constrain network and runtime behavior:** Providers should enforce egress allowlists, non-root containers, seccomp/AppArmor, and resource quotas—and expose knobs so your team can configure these controls.

7 Measure your success with outcomes and metrics

Demonstrating the worth of your program and securing funding requires measuring specific data-based indicators:

- **Reduced incident frequency:** The ultimate goal. Track the number of security incidents (Sev 2 or higher) attributed to the MCP ecosystem per quarter. A downward trend indicates effective security controls.
- **Enhanced visibility:** What percentage of your total MCP traffic is logged with a clear user ID and tool ID? Aim for 100% coverage, ensuring every action is attributable and auditable.
- **Mean time to remediate (MTTR): T**rack MTTR for MCP-related vulnerabilities. Your company should aim for critical issue remediation in less than 4 hours, reflecting agile response capabilities.
- **Supply-chain integrity:** What percentage of your running MCP servers are verified by Sigstore and pinned to a specific digest? This measures your defense against supply-chain attacks, indicating robust build and deployment processes.
- **Compliance adherence:** Link your security controls and audit logs to relevant compliance frameworks—such as ISO 42001, SOC 2, and GDPR Article 32—to demonstrate due diligence and meet regulatory requirements.
- **Human-in-the-loop engagement:** Monitor the frequency of user confirmations for sensitive operations. Increased engagement suggests successful implementation of manual review processes for critical actions.

Wiz: Comprehensive security for your MCP environment

Wiz provides an integrated system that protects MCP environments across your entire code-to-cloud infrastructure. (This includes Wiz's own MCP server!) You benefit from a complete risk assessment through a single platform, eliminating the need for multiple point solutions:

- **Visibility into MCP Technology –** Discover MCP servers and clients across your environment without agents, and provide visibility into overall AI usage—including unmanaged or shadow deployments that bypass governance.
- **Risk in Context –** Connect MCP misconfigurations, identity exposures, and sensitive data flows into Wiz's Security Graph to show which issues are truly exploitable. By grounding MCP posture in full cloud context, Wiz helps teams focus on the risks that matter most.
- **Governance & Response –** Enforce guardrails across MCP services, continuously generate audit-ready compliance evidence, and remediate risks with guided or automated actions integrated into enterprise workflows.

Example: A Wiz customer identified an MCP server running on a GCP VM with privileged role access and sensitive data exposure—flagged, contextualized, and remediated within the platform.

Wiz continues to expand its ability to secure MCP usage as part of a broader commitment to AI Security Posture Management (AI-SPM). As MCP adoption accelerates, Wiz ensures organizations can manage this new layer of AI and cloud risk with the same clarity, context, and confidence delivered across the rest of the platform.

Ready to see how Wiz can secure your MCP environment? Request a demo today!