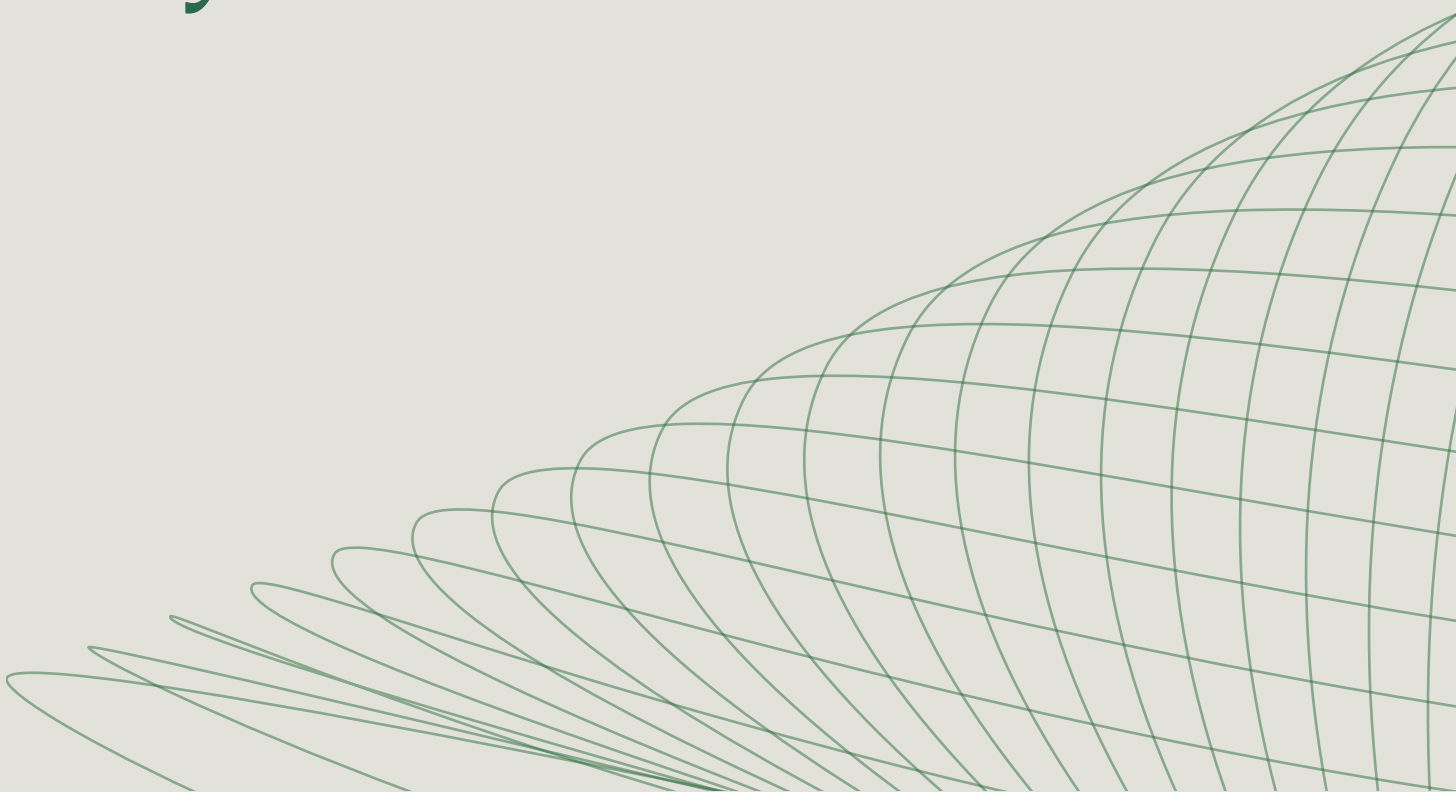
STACKLOK

Shadow MCP:

How to Find Rogue MCP Servers Before They Find You



Contents

Executive Summary

Developers across your organization are running MCP servers you don't know about. Those servers are calling APIs, reading files, making network requests, and acting on behalf of AI agents, all outside any security review, procurement process, or governance framework. This is Shadow MCP: the enterprise's newest and least-understood attack surface.

This paper explains why Shadow MCP is as dangerous as traditional Shadow IT, what security researchers have already documented about malicious and vulnerable MCP servers in the wild, and how security and platform engineering teams can deploy continuous telemetry to detect rogue servers using tools they already own. It then describes the path from detection to enforcement for organizations ready to govern MCP at scale.

If your organization is deploying AI coding agents, including Claude Code and Cursor, and you do not have visibility into which MCP servers are running, this paper is for you.

The Invisible Attack Surface Growing Inside Your Organization

Ask your security team how many MCP servers are running across your developer fleet right now. In most enterprises, the answer is: we don't know.

That is not a gap in policy. It is a gap in instrumentation. MCP servers are configured in local JSON files on developer workstations. They are invoked as child processes of trusted AI tools. They communicate over localhost or standard HTTPS to destinations that are indistinguishable from legitimate API traffic. No procurement record exists. No log is generated by default.

A developer discovers a useful MCP server, drops three lines into their Claude Code configuration, and within minutes it is calling GitHub, querying a database, or reaching an internal API. By the end of the day, they have shared the configuration in a team Slack channel.

A week later, a dozen developers are running the same server. None of this appears in any IT system. This is Shadow MCP. And it is happening at speed.

> **The pattern is familiar. The stakes are not.** Traditionally, Shadow IT spread over years as employees adopted cloud tools. Shadow MCP spreads in hours, and unlike a shadow SaaS tool that reads and displays data, a rogue MCP server acts: it executes code, exfiltrates data, and takes actions inside the trusted context of an AI agent that is designed to follow instructions.

What Rogue MCP Servers Actually Do: The Research

Shadow MCP is not a theoretical risk. Security researchers have spent the past twelve months documenting a cascade of real incidents and structural vulnerabilities that should reframe how security teams think about ungoverned MCP adoption.

The supply chain attack that no one saw coming

In September 2025, a threat actor published a counterfeit npm package called **postmark-mcp**, mimicking the official Postmark Labs email service. The package was convincing: plausible name, accurate description, and correct functions. Version 1.0.16 introduced a single line of code that blind-copied every outgoing email to an attacker-controlled address. Internal project memos, password resets, and invoices were silently exfiltrated.

The attack succeeded because there was no governance process to verify which MCP servers developers were running, or whether the version running matched a version that had been reviewed. A server that passed a one-time security check is not the same as a server that is continuously verified.

The scale of vulnerable servers in public registries

76 confirmed malicious payloads, including credential theft, reverse shells, and data exfiltration, were found among 3,984 AI agent skills scanned by Snyk in February 2026. Five of the top seven most-downloaded skills were malware. The most popular malicious skill had been downloaded over 7,700 times.

Astrix Security's research into over 5,200 open-source MCP server implementations found that 88% of servers require credentials, and over half rely on long-lived static API keys or Personal Access Tokens that are rarely rotated. Those credentials are typically stored in plaintext configuration files on developer workstations. When a developer's machine is compromised, every MCP server credential on that machine is compromised with it.

The RCE vulnerability with 437,000 downloads

CVE-2025-6514, a CVSS 9.6 command injection flaw in **mcp-remote** (a widely-used package for connecting to remote MCP servers) enabled full remote code execution when clients connected to untrusted MCP servers. With over 437,000 downloads, it became the first documented MCP vulnerability with mass-scale impact. Connecting to a compromised remote MCP server triggered arbitrary code execution on the client machine. No user interaction was required beyond the initial connection.

The exposed servers with no authorization

By early 2026, security researchers had catalogued nearly 7,000 internet-exposed MCP servers, and many were operating with no authorization controls whatsoever. Bitsight's research team demonstrated that many of these servers revealed their full tool definitions and data connections to any anonymous requester, and responded to arbitrary commands. An attacker who finds one of these servers finds a ready-made pivot point into whatever internal resources the server is wired into.

The root cause across all of these incidents is not a sophisticated zero-day. It is the absence of governance. Nobody knew the server was running. Nobody verified it matched a reviewed version. Nobody was watching what it was doing.

Why Your Existing Security Tools Don't See This

Before describing a solution, it is worth being precise about what your existing infrastructure does and does not detect, because the temptation to bolt MCP governance onto existing tooling is real, and it does not work.

Network inspection and CASB platforms observe outbound HTTP/S traffic. In principle, a CASB could detect connections to known malicious MCP endpoints. In practice, MCP servers communicate over localhost or through standard HTTPS to destinations like GitHub, Slack, Jira, and database hosts that are indistinguishable from sanctioned developer traffic. The data plane carries no signal that tells you an MCP server is involved.

Endpoint agents and EDR tools can detect process execution. An endpoint agent could alert on a new process spawning when an MCP server starts. In practice, MCP servers are invoked as child processes of Claude Code, Cursor, or VS Code, tools that are already whitelisted on developer workstations. The child process often inherits the parent's trust.

Developer questionnaires and periodic audits are the current default. The accuracy problem is structural: developers configure new MCP servers constantly, and no retrospective survey captures the current state. By the time the quarterly audit spreadsheet is analyzed, the threat surface has changed completely.

What is needed is instrumentation at the point where AI tools invoke MCP servers. We need a solution that captures structured metadata about every tool call and forwards it to the observability infrastructure you already operate. That approach is described below.

How to Detect Shadow MCP: A Step-by-Step Approach

Stacklok's recommended approach instruments AI coding tools directly using their native hook mechanisms. Claude Code exposes a **PreToolUse** hook as a lifecycle event that fires every time an MCP tool is about to be invoked. A lightweight script responding to this hook captures structured telemetry about every MCP tool call across the organization: which server was called, which tool was invoked, who made the call, and when.

This approach has three properties that network- and endpoint-based alternatives lack. First, it captures **application-layer context**: the instrumentation knows the MCP server name, the tool name, and the invoking user identity. Second, it is **zero friction for developers**: the hook is a configuration change to the AI tool, not to any MCP server, and developers continue working without interruption. Third, it provides **continuous telemetry**: unlike a quarterly audit, hook-based instrumentation captures every tool invocation in real time. If a developer fires up a new MCP server, the first tool call generates telemetry immediately.

Step 1: Understand the Hook Mechanism

Claude Code's **PreToolUse** hook fires before every MCP tool call, invoking a script of your choosing and passing structured information about the pending tool call as input. For Shadow MCP detection, the goal is passive observation: capture the telemetry, forward it, and make approval decisions based on the data. Policy enforcement can be added later once the inventory of in-use servers is established.

Step 2: Configure the Hook

Add the hook configuration to Claude Code's or Cursor's settings file to invoke your script on every **PreToolUse** event. For example in Claude Code, this is as simple as entering **/hooks** and then selecting **[User] mcp_.*** to send structured information about every tool call to an OpenTelemetry endpoint.

Step 3: Distribute Across Your Organization

The hook configuration is a file on disk. Distribute it using the management tooling you already use for developer workstation configuration:

- **macOS / Linux:** Push via Ansible, Chef, or Puppet to the standard Claude Code settings path on each managed workstation.
- **Windows:** Distribute via Group Policy or Intune as a managed preference.
- **Containerized dev environments:** Bake the configuration into your base developer image and distribute via your container registry.

This is a one-time distribution. Once deployed, it captures telemetry for every Claude Code or Cursor session on that machine without further intervention.

Step 4: Route Telemetry to Your Observability Stack

The hook script emits OpenTelemetry-formatted data, aligned with the emerging OTel MCP semantic conventions. Route it to whichever platform your organization already operates:

- **Grafana / Prometheus** for dashboard visualization of top servers, call volumes, and user activity
- **Datadog, New Relic, Honeycomb, or Splunk** via the OTel collector with the appropriate exporter
- **SIEM (Sentinel, Splunk ES, Chronicle)** for alerting and correlation with other security events
- **Local log file + existing forwarder** (Filebeat, Fluentd, Vector) if you prefer not to deploy new infrastructure

No new observability infrastructure is required if your organization already ships logs from developer workstations.

Step 5: Build Your MCP Server Inventory

Once telemetry is flowing, build a dashboard or query that answers:

- **Top MCP servers by call volume** to visualize every server name observed, ranked by invocation count
- **Top MCP servers by unique users** to show team-wide adoption versus individual experimentation
- **Tool-level breakdown** to highlight which specific tools each server is calling and at what frequency
- **New server introduction alerts** to trigger an alert when a previously unseen MCP server name appears in the telemetry

This last alert is your real-time rogue server detector. Every new MCP server in your environment surfaces the first time it receives a tool call.

Step 6: Establish an Approval Workflow

Telemetry without a decision process produces an alert backlog, not governance. Define a lightweight approval workflow before you turn on alerting:

1. **New server detected —** automated alert to platform engineering channel with server name, first-seen timestamp, and invoking user identity
2. **Platform engineer reviews —** checks the server against a known-good registry, reviews the tool list, and makes an approval decision
3. **Decision recorded —** server added to the approved list, or flagged for removal with the user notified
4. **Reporting —** weekly summary of approved, pending, and rejected servers to security stakeholders

The goal is not to block all unapproved MCP usage immediately. The goal is to build the inventory first — to govern from knowledge rather than from guesswork.

What This Looks Like in Practice

Stacklok uses this approach internally to monitor MCP server usage across our own engineering organization. The telemetry shows top servers by call volume, which tools are being invoked, and which engineers are calling them. When a developer experiments with a new server, it surfaces in the dashboard on the first tool call.

The data changes the governance conversation. Instead of asking developers to complete a quarterly audit form, platform teams proactively approach a developer with specific information: "We noticed you're running the Jira MCP server. We've reviewed it and it's approved. We also saw a server called X that we haven't seen before. Can you tell us about it?" That conversation is specific, timely, and grounded in real data.

> **For a Fortune 500 financial services customer running Stacklok on Kubernetes**, this telemetry fed directly into an existing New Relic deployment. The platform team configured an alert that fired whenever a new MCP server name appeared. In the first week, the alert surfaced three previously unknown servers. Two were legitimate tools the team approved. One was a third-party server connecting to an external API that violated the organization's data residency policy. It was removed before any sensitive data left the environment.

From Detection to Governance: Where Stacklok Fits

Hook-based telemetry is a detection capability. It answers: "What MCP servers are running?" It does not prevent a rogue server from running in the first place, enforce policy on what tools can be called, or provide centralized credential management for MCP server authentication.

For organizations ready to move from detection to enforcement, Stacklok's platform, a production-hardened enterprise distribution of the open-source ToolHive project (Apache 2.0 licensed, fully auditable on GitHub), provides the complete governance stack.

Centralized registry with curated catalog
Administrators define a set of approved MCP servers. Developers discover and deploy from a portal. Unapproved servers are blocked at the platform layer.

Container-based isolation by default
Every MCP server runs in an isolated container with minimal permissions. Network access and filesystem permissions are configurable per server via JSON profiles. A server configured to call the GitHub API cannot, by design, reach internal database hosts.

Per-request identity enforcement
Stacklok's embedded authorization server supports OIDC/OAuth SSO, Okta, Entra ID, and Google. Every MCP tool call carries an authenticated identity. There are no locally stored credentials for MCP servers to harvest.

Supply chain attestation
MCP servers deployed through Stacklok are signed and include provenance attestation. The **postmark-mcp** supply chain attack described above (where a malicious version silently replaced a legitimate one) is structurally prevented; the server running in production is verifiably the server that was reviewed.

Native OpenTelemetry observability
As of March 2026, Stacklok's Kubernetes-native deployment includes a Prometheus metrics endpoint, pre-built Grafana dashboards, and OTel-aligned telemetry that integrates natively with Datadog, New Relic, Honeycomb, and Splunk. The dashboards you build during the detection phase described above work without modification after you deploy Stacklok.

Token optimization at scale
Stacklok's MCP Optimizer, embedded in the platform, reduces token usage by 60–85% per request via on-demand tool discovery. This reduces both LLM cost and the surface area of tool definitions exposed to the model per request, limiting the blast radius of tool poisoning attacks.

Hook-based detection and Stacklok's governance platform are complementary, not alternatives. Most organizations begin with hook-based telemetry to establish their inventory, and graduate to Stacklok when they are ready to enforce policy at the platform layer. Because Stacklok is built on open-source ToolHive under the Apache 2.0 license, the platform can be evaluated and audited before any procurement decision.

Questions We Hear From Security Teams

"We have an EDR and a CASB. Isn't that sufficient for MCP visibility?"
EDR and CASB tools were not designed to understand the MCP protocol, and they do not. An EDR can tell you that Claude Code spawned a child process. It cannot tell you that the child process was a specific MCP server, which tools it called, or what data it accessed. A CASB can observe HTTPS traffic to github.com. It cannot distinguish between a developer pushing code and an MCP server exfiltrating repository contents on behalf of an AI agent. Hook-based telemetry provides application-layer context that neither tool surfaces.

"Can't we just scan for MCP configuration files on workstations?"
Configuration file scanning gives you a static inventory of what is configured, but it does not tell you what is actively running, how frequently it is being called, which tools are being invoked, or by whom. A server present in a config file might be dormant and low risk. A server configured last Tuesday and called ten thousand times since then is a different story. Hook-based telemetry provides the dynamic layer that static scanning cannot. The two approaches are complementary: use config file scanning for breadth, hook-based telemetry for depth.

"How do we handle MCP servers that developers run in their own dev environments or personal machines?"
For managed workstations, the hook distribution approach described in this paper covers the fleet. For personal devices, the organizational answer is policy: MCP servers used for work purposes should be running on managed machines.

The detection approach described here gives you the data to identify which servers are in use on managed devices and have an informed conversation with developers about which of those should be brought under formal governance. For organizations that need enforcement rather than detection, Stacklok's centralized gateway means that MCP traffic flows through an audited, policy-enforced control plane regardless of the client device.

"We're not ready to deploy a full MCP governance platform. Is the detection approach worth doing on its own?"
Yes. Detection is the prerequisite for every governance decision that follows. You cannot approve servers you don't know about. You cannot block servers you can't see. You cannot report on MCP risk to your CISO or board without data. The hook-based approach described in this paper requires no new infrastructure for most organizations, takes one afternoon to deploy, and produces actionable intelligence immediately. Start there. Governance follows from inventory.

"How does Stacklok's open-source foundation affect our ability to evaluate the platform?"
ToolHive is Apache 2.0 licensed and fully auditable on GitHub. Your security team can review the source code, run the platform in a test environment, and validate its behavior before any procurement conversation. Stacklok is the production-hardened enterprise distribution of ToolHive. It uses the same codebase, with the Kubernetes Operator, enterprise identity integration, and commercial support that production deployments require.

The Bottom Line

Shadow MCP is not a future risk. Developers are running ungoverned MCP servers in your environment today. Security researchers have documented supply chain attacks, remote code execution vulnerabilities, and credential theft targeting the exact servers your developers are most likely to adopt.

The detection approach in this paper is a lightweight forwarding script to your existing observability stack that can be deployed in an afternoon. It requires no new infrastructure, no changes to developer workflows, and no modifications to MCP servers. It produces real-time telemetry that answers the question every security team needs to answer before anything else: what is actually running?

It's not a final solution, it's an important start point. As enterprises progress along their AI native journey and consider MCP use at scale, Stacklok is an expert partner.

To see what MCP usage looks like in your environment and to have Stacklok help you set up the hooks-based solution described above, reach out at stacklok.com/demo.

Technical documentation for ToolHive's Kubernetes Operator, observability integration, and curated registry is available at docs.stacklok.com.

ToolHive is open source under the Apache 2.0 license. github.com/stacklok/toolhive