PINAKA RESEARCH

MCP SECURITY REPORT · 2026

The State of Python MCP Server Security

A static taint-analysis survey of 39 public Python Model Context Protocol servers, the most widely used ones included, and what 1,939 agent tools tell us about where agent risk actually lives.

39	1,939	0	34/39
public Python MCP servers scanned	agent tools analyzed	command or SQL injection in real servers	clean of any injection finding

Pinaka Security · Agent Attack Surface research
pinaka.sh/state-of-mcp-security

First edition
June 2026

SUMMARY OF FINDINGS

The popular ecosystem is more careful than the headlines

We scanned 39 public Python MCP servers, from the most widely used down to single-purpose ones, with static taint analysis, tracing every tool input to the dangerous sinks it could reach. We were prepared to write the alarming report. The data did not cooperate, and we are publishing the less sensational result because it is the true one.

0

command injection (RCE) in real servers. The only hit was the intentionally vulnerable benchmark.

0

SQL injection. No tool input reached a query through unsafe string building.

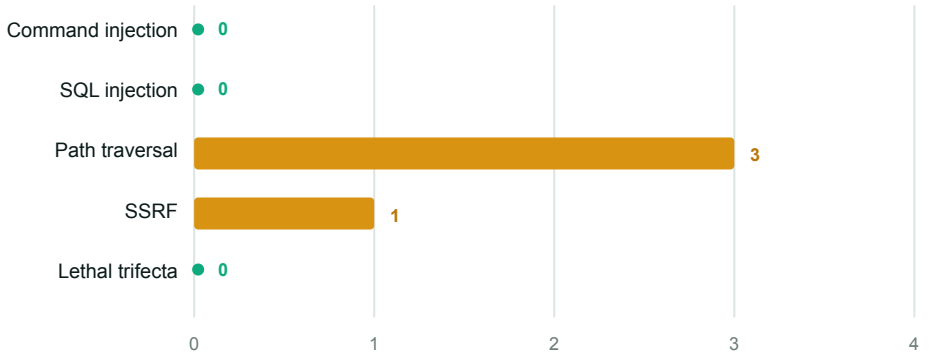
4

real servers carried a single lower-severity finding each. None a clean exploit.

34/39

servers were clean of any injection finding at all.

Figure 1. Injection findings by class, across the 38 real servers



Command and SQL injection: zero. The three path-traversal findings were read-only, by-design for a file tool, or off by default. The one SSRF was a server-side URL fetch gated behind an explicit flag. The intentionally vulnerable benchmark server is excluded here and reported separately in Figure 3.

> **The injection panic is overblown. The real agent attack surface is the part a vulnerability scanner cannot count.**

01 · INTRODUCTION

Why we measured this

The Model Context Protocol gave AI agents a universal plug, and the security commentary that followed has been loud and largely negative. Headlines describe the MCP ecosystem as a security disaster waiting to happen. Some of that is fair. A lot of it is sampling the long tail, counting configuration and behavioral weaknesses together, and rounding up.

So we did the boring thing and measured one specific, falsifiable property: in the Python MCP servers people actually depend on, does a tool input reach a dangerous sink without being sanitized first. That is the class of bug a static analyzer can find with high confidence: command injection, SQL injection, path traversal, server-side request forgery. It is not the whole of agent security, and we are explicit about that in the discussion. But it is measurable, and nobody had measured it cleanly for this slice.

02 · METHODOLOGY

How the survey was run

01 We cloned 46 public Python MCP server repositories as of June 2026, spanning the most widely used servers (AWS, Kubernetes, Telegram, Google Workspace, Atlassian, ElevenLabs, Home Assistant, Jupyter) and a long tail of single-purpose ones. A representative sample, not a ranked top 46.

02 Each repository was scanned with Pinaka's static taint engine: an AST-based, intra-procedural analysis that traces every tool parameter to shell, eval, SQL, file, and network sinks, with a sanitizer model so validated inputs do not fire. It is stdlib-only and runs locally.

03 The engine reads three tool-registration styles (FastMCP decorators, the low-level dispatch API, and custom wrapper decorators) and detected tools in 39 of the 46, for 1,939 tools in total. The 39 include shell, database, and the official reference servers, the ones most likely to do something dangerous.

04 The engine is precision-first by design. A clean codebase produces zero findings, which is what makes a non-zero finding worth acting on.

The honest limitation

7 of the 46 repositories still returned zero tools. That is a property of the scanner, not of the servers: a few are not Python MCP servers or are frameworks, the rest expose tools through a naming indirection the scanner does not yet resolve. They are undercounted, not proven clean. Static analysis also does not observe runtime, so behavioral risks (prompt injection, tool poisoning, rug pulls) are out of scope by construction. Unmeasured is not the same as safe, and we would rather say so than imply coverage we do not have.

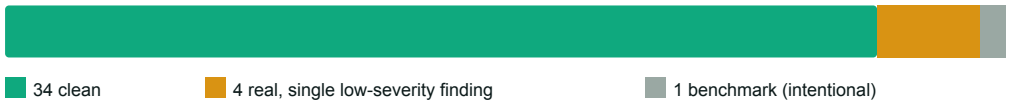
What 1,939 tools looked like

Across the 39 servers, five tripped an injection rule. One is the deliberately vulnerable benchmark. The other four each carried a single lower-severity finding. The table below counts the injection classes across the 38 real servers; the benchmark is analyzed separately.

INJECTION CLASS	RULE	REAL SERVERS	SEVERITY ON REVIEW
Command injection (RCE)	`AS-TS-004`	0	none
SQL injection	`AS-TS-005`	0	none
Path traversal	`AS-TS-006`	3	low: read-only / by-design / off by default
Server-side request (SSRF)	`AS-TS-003`	1	low: gated behind an explicit flag
Lethal trifecta	`AS-DL-001`	0	none

We report these findings in aggregate rather than naming the projects. The right next step for a real finding is responsible disclosure, not a report callout.

Figure 2. Server population by injection status (n = 39)



34 of 39 servers were completely clean of injection. Each segment is one server.

Figure 3. Benchmark validation: the engine on damn-vulnerable-mcp-server

A scanner that returns zero is only worth trusting if it returns non-zero when it should. Against the open-source damn-vulnerable-mcp-server, the engine flagged 15 injection sinks. The benchmark also includes behavioral attacks such as rug pulls and tool shadowing that no static scanner can see by reading code; we do not claim to catch those.

04 · DISCUSSION

Does this mean MCP is safe? No.

A low injection rate is a real, useful finding. It is not a clean bill of health, for three reasons.

01 **Popular is not the same as yours.** The servers in this survey are the well-maintained, heavily reviewed ones. The MCP server a team wrote last sprint to wrap an internal API has had none of that scrutiny, and that is the one running next to your production data.

02 **Injection is the part a scanner can see.** Prompt injection, tool poisoning, rug pulls, excessive agency, and cross-surface exposure do not show up as a tainted parameter reaching a sink. A server with zero injection bugs can still be walked out the door by a poisoned tool description or a single over-broad credential.

03 **A clean scan is not a clean bill of health.** 7 of the 46 repositories still returned zero tools (non-MCP repos, frameworks, or a tool-name indirection the scanner does not yet resolve). Unmeasured is not the same as safe.

> **An agent with zero injection bugs can still leak everything it can reach. The risk moved; the metrics did not follow it.**

05 · CONCLUSION

Where to look next

The headline story of MCP insecurity, that the servers are riddled with injection bugs, does not hold for the popular Python ecosystem. The maintainers of these servers are doing the unglamorous work: argument lists instead of shell strings, parameterized queries, authenticated SDK clients between the model and the dangerous calls.

The real agent attack surface is the part the headline metrics cannot count: every tool your agents expose, what each one can reach, which tool descriptions can be poisoned, and where an agent host on your perimeter becomes a way in. That is the layer Pinaka maps. Injection scanning is table stakes; the differentiator is correlating the agents you ship with the perimeter they run on, and catching what only shows up where the two meet.

Run this scan on your own code

The engine behind this survey is the same one in Pinaka's Agent Surface. Point it at your repository and it maps the MCP servers and agent tools in your code, traces each tool input to its sinks, and flags the issues against the OWASP MCP Top 10. It runs locally, and your source never leaves your machine. **pinaka.sh/state-of-mcp-security**

Methodology notes and limits

- Scope: 46 cloned repositories, 39 with at least one registered tool (across FastMCP decorators, the low-level dispatch API, and wrapper decorators), 1,939 tools total, as of June 2026.
- Technique: AST-based, intra-procedural taint analysis. No cross-function taint, no runtime observation. Precision-first: validated inputs and authenticated-client sinks do not fire.
- Out of scope by construction: prompt injection, tool poisoning, rug pulls, excessive agency, supply chain, and cross-function taint flows.
- The "lethal trifecta" is a term coined by security researcher Simon Willison.
- Findings in real servers are reported in aggregate; only the intentionally vulnerable benchmark is named.