Journal of Computer Science and Technology Studies
ISSN: 2709-104X
DOI: 10.32996/jcsts
Journal Homepage: www.al-kindipublisher.com/index.php/jcsts

Evaluating Security and Reliability of Stateless Model Context Protocol Gateways

Lohith Reddy Kalluru

Cloud Developer III, Hewlett Packard Enterprise, Plainfield, Illinois, USA

Corresponding Author: Lohith Reddy Kalluru**, E-mail:** kallurulohithreddy@gmail.com

ABSTRACT

The July 2026 revision of the Model Context Protocol (MCP) introduces a stateless request model and routing headers that enable conventional HTTP gateway controls for language-model tools. This study examines whether three deterministic controls—tool-definition pinning, least-privilege tool exposure, and argument-level policy enforcement—prevent unauthorized requests from reaching a backend without rejecting authorized operations. We implemented a minimal HTTP/JSON-RPC gateway prototype and evaluated eight combinations of the controls in a 2 × 2 × 2 factorial design. Each configuration received 120 synthetic adversarial requests (40 changed tool descriptors, 40 out-of-scope calls, and 40 disallowed argument sets), 60 valid requests, and 400 timed valid requests over loopback. In this limited implementation, the unprotected baseline rejected none of the 120 adversarial cases, whereas the configuration applying all three controls rejected all 120; each single control rejected precisely the 40 cases constructed to violate its rule. All configurations completed 60 of 60 valid requests. Median observed HTTP latency was 0.533 ms for the baseline and 0.530 ms for the full-control configuration, but these loopback timings do not establish production-scale performance equivalence. The findings demonstrate coverage of the specified access-control invariants, not resistance to model-driven prompt injection. We offer a reproducible pilot methodology and identify the model-in-the-loop, distributed-systems, and identity-validation experiments necessary for stronger external validity.

KEYWORDS

Model Context Protocol; stateless gateway; tool-definition integrity; least privilege; policy enforcement; AI security

ARTICLE INFORMATION

ACCEPTED: 30 September 2026 **PUBLISHED:** 31 October 2026 **DOI:** 10.32996/jcsts.2026.8.7

1. Introduction

Tool-augmented large language models can translate natural-language requests into calls against inventory, finance, ticketing, and document services. Standardization through the Model Context Protocol (MCP) reduces integration duplication but also means that an assistant may encounter untrusted tool metadata, execute tool calls with meaningful side effects, and handle data drawn from systems with different authorization rules. These risks cannot be addressed only by asking a model to ignore malicious text: backend access must be governed independently of the model (Greshake et al., 2023; Huang et al., 2026).

The MCP specification released on July 28, 2026 replaces transport-level sessions with independently routable requests, exposes method and resource names through HTTP routing headers, and introduces self-describing request metadata (Soria Parra & Delimarsky, 2026). This presents a concrete engineering opportunity: validate each request at a gateway before routing it to the server. Yet a stateless transport alone supplies neither approved tool definitions nor policy decisions for individual actions.

This paper evaluates three narrowly scoped gateway controls. Definition pinning verifies that an invoked tool still matches a reviewed description and schema; least-privilege exposure restricts which tools a principal can list and invoke; and argument-level policy blocks calls whose payload exceeds approved limits. Unlike semantic detectors, these mechanisms enforce deterministic invariants. The key contribution is a transparent, reproducible pilot experiment that reports what this design does—and does not—measure. The study is intentionally distinguished from existing work that already tests real language models against poisoned descriptions and tool outputs (Jamshidi et al., 2025; Yergattikar, 2026).

Copyright: © 2026 the Author(s). This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC-BY) 4.0 license (https://creativecommons.org/licenses/by/4.0/). Published by Al-Kindi Centre for Research and Development, London, United Kingdom.

2. Related Work

Early tool-use research established that language models can select and invoke external APIs; Toolformer is an influential example (Schick et al., 2023). MCP subsequently standardized discovery and invocation across tools and hosts (Anthropic, 2024). Its 2026-07-28 update explicitly enables request-by-request load balancing and header-oriented gateway enforcement (Soria Parra & Delimarsky, 2026).

Indirect prompt injection occurs when lower-trust data presented to a model influences subsequent decisions (Greshake et al., 2023). For MCP, attack surfaces include mutable tool descriptions, shadowing of tool identities, and malicious output content. Huang et al. (2026) examine client vulnerability to tool poisoning. Jamshidi et al. (2025) discuss descriptor integrity and layered defenses, while Narajala and Habler (2026) present a broader enterprise security framework. The ShieldMCP study evaluates a runtime interception framework against multiple model backends and reports empirical attack-success measures (Yergattikar, 2026).

The present study does not claim to supersede those semantic-attack evaluations. Its narrower research gap is reproducible validation of three independent, mechanically enforceable gateway rules within the 2026 stateless request model, paired with benign-request and loopback latency measurements. The distinction is important: blocking unauthorized calls at a gateway is not the same as preventing a model from forming malicious intent or detecting malicious content prior to approval.

3. Research Questions and Threat Model

The experiment addresses four questions: RQ1—Does descriptor pinning reject changes to an approved tool definition? RQ2—Does per-principal tool scoping reject a call to an otherwise valid but unauthorized tool? RQ3—Does argument-level policy reject out-of-range or disallowed destinations for an authorized tool? RQ4—Do these controls admit valid calls, and what latency is observed under a local, low-load test?

The attacker is assumed capable of changing the descriptor returned for one already registered tool, proposing a tool call outside the authenticated principal's approval set, or selecting arguments that violate established limits. The adversary does not possess the gateway configuration signing key, compromise the gateway process, or rewrite the policy registry. These are explicit assumptions rather than claims about a fully compromised enterprise. No live end-user content, real secrets, or external backend systems were used.

Table 1. Threat scenarios, observable outcomes, and enforcement rules

Scenario	Study fixture	Expected control
Definition drift	Modified search_catalog description after approval	Pin: compare canonical descriptor hash
Excess privilege	Analyst calls issue_refund	Scope: deny tool not granted to analyst
Argument violation	Analyst exports >100 rows to external destination	Policy: validate row count and destination
Routing discrepancy	Mcp-Method header disagrees with JSON-RPC body	Common baseline: reject malformed routing

4. Gateway Architecture and Controls

The prototype uses HTTP POST /mcp requests carrying JSON-RPC 2.0 data, 2026-07-28 protocol metadata, and routing headers. A host presents a test-only authenticated principal. Before calling a mock backend, the gateway checks header/body agreement, a registered tool identifier, and any enabled controls. It does not persist client transport sessions. Stateful policy fixtures and the reviewed tool catalog are held locally by the test process, not inferred by the language model.

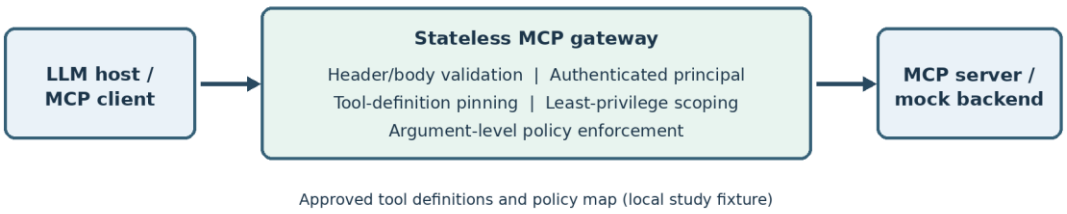


Figure 1. The evaluated request path (simplified, single-process prototype).

4.1 Tool-definition pinning

At approval, the prototype computes a SHA-256 digest over a deterministically serialized tool name, description, and input schema. Serialization sorts object keys, normalizes Unicode to NFC, and removes inconsequential whitespace. Before exposing a catalog entry or executing a tool, the gateway compares the current descriptor with this digest and refuses changed definitions. The trusted registry and the descriptor store are separate objects in the prototype. A hash detects alteration only relative to an already trusted

reference; it cannot identify harmful instructions that were approved initially, and it does not provide publisher authenticity or protect a compromised registry.

4.2 Least-privilege tool scope

A policy map assigns search_catalog and export_records to an analyst, and search_catalog and issue_refund to a finance role. The gateway filters tool listings by authenticated role and repeats the check for every invocation. Performing only catalog filtering would be insufficient because a client could call an unlisted tool directly. The test uses a fixed HMAC bearer token solely to simulate role authentication; this is not an implementation of OAuth or an enterprise identity provider. In production, token audience validation and action-scoped permissions should follow OAuth security guidance (Campbell et al., 2020; Lodderstedt et al., 2025).

4.3 Argument-level policy

The policy permits an analyst to export at most 100 records per request only to approved.internal. It allows a finance user to issue refunds between 1 and 500 units. Values with missing or wrong types are denied when argument policy is enabled. This control is distinct from tool scoping: permission to call a tool does not imply permission to supply arbitrary quantities or destinations. The example rule set is intentionally small and does not demonstrate semantic analysis of free-text arguments.

4.4 Protocol invariant and placement

For all configurations, the gateway verifies that Mcp-Method, Mcp-Name, and the advertised protocol version agree with the request body. This baseline check prevents a caller from routing under one method header while invoking another body operation. The verified 2026 revision uses these HTTP headers for routing, but requires their consistency with request content (Soria Parra & Delimarsky, 2026). The study tests a subset of tools/list and tools/call; no claim of comprehensive MCP SDK conformance is made.

5. Methodology

5.1 Experimental design

A full factorial experiment enables or disables each of the three rules, producing eight configurations. We name them P–S–A, where P indicates descriptor pinning, S tool scoping, and A argument policy (0 = disabled, 1 = enabled). Each configuration processes three adversarial classes of 40 requests each: 40 uniquely labeled unapproved description revisions, 40 refund calls issued as the analyst principal, and 40 out-of-policy exports with varying record counts. Sixty authorized requests (20 catalog queries, 20 compliant exports, and 20 compliant finance refunds) serve as a correctness control. Each run also checks a mismatched routing-header request, which must be rejected regardless of security-rule selection.

The suite is deterministic with seed 20261025. Repeated request values are variations of three templates, not statistically independent naturalistic attacks. Attack requests originate from the test harness rather than a language model, and no prompt injection success rate is measured. A request is counted as blocked only when its corresponding policy returns the expected denial response. A benign request succeeds only if a mock backend returns executed = true.

5.2 Reliability and timing measurement

The same authorized search request is executed 400 times for each configuration in five blocks of 80 requests, following 15 warmup requests per configuration. Block order is shuffled with a fixed random seed to reduce systematic warmup bias. Timing spans client-side HTTP request submission through parsed response receipt on the local loopback interface, including connection setup and Python HTTP handling; TLS, upstream networking, container orchestration, and storage are absent. We report empirical median (p50) and 95th percentile (p95) latency; these short single-host measurements are descriptive, not an inference about high-concurrency throughput.

5.3 Implementation and reproducibility

The gateway, mock backend, evaluation scenarios, HTTP client, and CSV/JSON exporters are implemented using Python 3 standard-library modules. The evaluation was run under Python 3.13.5 on Linux-6.18.55-x86_64-with-glibc2.41. One HTTP server process listens on a dynamically allocated loopback port and validates signed test roles. Each experiment performs real HTTP requests rather than merely calling the decision function in-process. Supplementary source code and machine-readable output are supplied with this draft. Re-running the script will reproduce the deterministic block/allow counts, while latency numbers may change with CPU load and operating system scheduling.

Table 2. Configurations and measured prototype results

Configuration	Adversarial requests blocked	Valid calls completed	p50 (ms)	p95 (ms)
P0-S0-A0	0/120	60/60	0.533	0.819
P1-S0-A0	40/120	60/60	0.543	0.871
P0-S1-A0	40/120	60/60	0.513	0.753
P0-S0-A1	40/120	60/60	0.506	0.822
P1-S1-A0	80/120	60/60	0.527	0.715
P1-S0-A1	80/120	60/60	0.544	0.814

Configuration	Adversarial requests blocked	Valid calls completed	p50 (ms)	p95 (ms)
P0-S1-A1	80/120	60/60	0.515	0.693
P1-S1-A1	120/120	60/60	0.530	0.776

6. Results

6.1 Denial coverage

The baseline without the three optional controls allowed all 120 adversarial requests to reach the mock backend. Enabling one rule rejected 40 of 120 cases (33.3%); enabling any two rules rejected 80 (66.7%); and enabling all three rejected 120 (100%). As shown in Table 3, each control blocked the cases specifically constructed to violate its corresponding invariant. These figures measure enforcement completeness against the predefined fixture set, not attack success probabilities in the field. The observed additivity is a direct consequence of disjoint test classes and should not be interpreted as statistical synergy.

Table 3. Adversarial-request rejection by rule (40 requests per class)

Enabled control(s)	Changed descriptor	Unauthorized tool	Bad arguments
No optional control	0/40	0/40	0/40
Pin only	40/40	0/40	0/40
Scope only	0/40	40/40	0/40
Argument policy only	0/40	0/40	40/40
All three	40/40	40/40	40/40

6.2 Authorized task completion

All eight configurations executed 60 of 60 authorized requests. Accordingly, no false rejection was observed in the valid examples chosen for this experiment. This does not imply a zero false-positive rate for arbitrary workloads: the valid suite deliberately avoids ambiguous roles, free-text policy requests, and dynamic tool definitions. The common routing consistency invariant also rejected the deliberate header/body mismatch in every configuration.

6.3 Observed local latency

The baseline recorded p50 = 0.533 ms and p95 = 0.819 ms. The full-control configuration recorded p50 = 0.530 ms and p95 = 0.776 ms. These values fall within the variability of the loopback microbenchmark; the experiment neither demonstrates a performance improvement from controls nor establishes an operational latency budget. More meaningful performance inference would require repeated runs across machines and load levels, controlled CPU utilization, real upstream services, and confidence intervals.

7. Discussion

The most defensible observation is that deterministic rules can prevent specifically unauthorized tool actions from reaching a backend even if the caller submits the request. Pinning addresses post-approval definition changes, not originally malicious approved definitions; scoping limits which tools a principal may invoke, not how a model chooses among allowed tools; argument policy constrains explicit structured values, not the semantics of unconstrained natural language. Taken together, the controls narrow the consequences of some manipulated model outputs but do not remove the underlying prompt-injection attack surface.

Relative to prior research, this paper contributes a small, runnable test matrix for the stateless gateway enforcement boundary. ShieldMCP investigates semantic detection and model-level attack success (Yergattikar, 2026), whereas the present pilot evaluates deterministic access decisions. Both are potentially complementary; the prototype is not a comparative alternative to a model-aware defense. Deployment would also require durable identity propagation, scoped OAuth credentials, centrally governed registry publication, audit trails, policy versioning, and independently verified authorization and confirmation paths (Lodderstedt et al., 2025; Narajala & Habler, 2026).

8. Threats to Validity and Future Evaluation

Construct validity: "adversarial request blocked" reflects known rule violations generated by the test harness, not attacker objective failure after an adaptive interaction with a language model. Forty variants per class do not represent 40 independent exploit strategies. The manipulated descriptor tests integrity comparison but no model ever reads it. Internal validity: the rules were designed for the fixtures, no competing gateway implementation was benchmarked, and the Python server is not a deployed MCP SDK. External validity: tests used one loopback process without TLS, load, policy-service outages, or real backends. HMAC role tokens are test fixtures and are not adequate enterprise authentication or an OAuth implementation.

Future work should execute model-in-the-loop agent tasks across multiple models and clients; draw attack and benign suites from independent sources; measure detection, unauthorized backend reach, benign completion, and false-positive rates with confidence intervals; vary descriptor similarity and tool-catalog sizes; validate token audience and issuer restrictions; compare policy placement at host, gateway, and server; and test horizontal scaling and fail-closed behavior on Kubernetes under concurrent load. Such a study would support stronger conclusions about enterprise reliability than the pilot can presently justify.

9. Conclusion

A stateless MCP gateway is a practical enforcement point, but its value depends on explicit policies rather than on the stateless protocol alone. In a limited, reproducible HTTP prototype, the combined tool pinning, least-privilege scoping, and argument constraints rejected every predefined violating request while admitting the predefined authorized calls. The result establishes only that these particular rules were correctly enforced in a controlled environment. Demonstrating security against prompt injection, compatibility across implementations, and acceptable production latency remains necessary before making enterprise-wide or generalizable claims.

Data and Code Availability: A reproducible Python script and its JSON and CSV results are part of this paper. The script includes a test-only identity stub and must not be treated as a production gateway.

Disclaimer: The views expressed are the author's own and do not represent the position of Hewlett Packard Enterprise.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

Publisher's Note: All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers.

References

[1] Anthropic. (2024, November 25). *Introducing the Model Context Protocol*. https://www.anthropic.com/news/model-context-protocol

[2] Campbell, B., Bradley, J., & Tschofenig, H. (2020). *Resource indicators for OAuth 2.0 (RFC 8707)*. RFC Editor. https://doi.org/10.17487/RFC8707

[3] Greshake, K., Abdelnabi, S., Mishra, S., Endres, C., Holz, T., & Fritz, M. (2023). *Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection. In Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security (pp*. 79–90). https://doi.org/10.1145/3605764.3623985

[4] Huang, C., Huang, X., Tran, N. P., & Fard, A. M. (2026). Model Context Protocol threat modeling and analysis of vulnerabilities to prompt injection with tool poisoning. *Journal of Cybersecurity and Privacy*, *6*(3), Article 84. https://doi.org/10.3390/jcp6030084

[5] Jamshidi, S., Nafi, K. W., Dakhel, A. M., Shahabi, N., Khomh, F., & Ezzati-Jivan, N. (2025). *Securing the Model Context Protocol: Defending LLMs against tool poisoning and adversarial attacks [Preprint]*. arXiv. https://arxiv.org/abs/2512.06556

[6] Lodderstedt, T., Bradley, J., Labunets, A., & Fett, D. (2025). *Best current practice for OAuth 2.0 security (RFC 9700)*. RFC Editor. https://doi.org/10.17487/RFC9700

[7] Narajala, V. S., & Habler, I. (2026). *Enterprise-grade security for the Model Context Protocol (MCP): Frameworks and mitigation strategies*. In 2026 IEEE 5th International Conference on AI in Cybersecurity (ICAIC). https://doi.org/10.1109/ICAIC67076.2026.11395723

[8] Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Hambro, E., Zettlemoyer, L., Cancedda, N., & Scialom, T. (2023). Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, *36*. https://proceedings.neurips.cc/paper/2023/hash/d842425e4bf79ba039352da0f658a906-Abstract-Conference.html

[9] Soria Parra, D., & Delimarsky, D. (2026, July 28). *The 2026-07-28 specification*. Model Context Protocol Blog. https://blog.modelcontextprotocol.io/posts/2026-07-28/

[10] Yergattikar, S. (2026). *Securing the tool layer: A threat taxonomy and runtime defense framework for Model Context Protocol deployments*. In Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 6: Industry Track, pp. 865–871). https://doi.org/10.18653/v1/2026.acl-industry.58