InfrastructureSentinel: Policy Enforced Guardrails for Secure MCP-driven Infrastructure Agents

Tarun Kumar, Aalap Tripathy, Gayathri Saranathan, Martin Foltin, Suparna Bhattacharya, Scott Hinchley, Donald M Bahls, David Brookshire, Larry Kaplan, Robert W. Wisniewski

Hewlett Packard Enterprise

(tarun.kumar2, aalap.tripathy, gayathri.saranathan, martin.foltin suparna.bhattacharya, scott.hinchley, donald.bahls, dsb, larry.kaplan, robert.wisniewski)@hpe.com

Abstract

The proliferation of Model Context Protocol (MCP) servers in enterprise infrastructure management has revolutionized AI-driven automation while introducing critical multilayered security vulnerabilities that traditional cybersecurity frameworks cannot adequately address. This paper presents **InfrastructureSentinel**, a novel security middleware that uniquely translates high-level, natural-language policies into context-aware, multi-layer enforcement for MCP-driven agents. Our solution employs a dedicated guardian LLM that interprets natural language policies and applies contextual reasoning to complex infrastructure scenarios, providing dynamic policy enforcement that adapts to user roles, operational timing, and system context. Unlike existing defenses that rely on formal-logic verification or hard-coded rules, our approach implements guardrails at four distinct control points: input message filtering, tool selection validation, execution-time verification, and post-action auditing. The system addresses critical gaps in existing security solutions by providing infrastructure-specific threat modeling, real-time policy adaptation, and comprehensive audit trails with explainable decision making through confidence scores and detailed reasoning. Our evaluation demonstrates the system's effectiveness in preventing command injection, privilege escalation, and tool poisoning attacks across various enterprise infrastructure scenarios while maintaining operational agility essential for modern data center management.

1 Introduction

The rapid proliferation of Model Context Protocol (MCP) servers in enterprise environments represents both a transformative opportunity and a significant security challenge (Anthropic 2024; Hou et al. 2025). MCP has emerged as the de facto standard enabling Large Language Models (LLMs) to seamlessly integrate with external systems, databases, and infrastructure management tools. This standardization has led to explosive growth, with over eight million weekly SDK downloads and adoption across industries for managing critical infrastructure resources (Hasan et al. 2025).

In the specific domain of infrastructure management, MCP servers have become particularly prominent for managing compute centers and data centers. Organizations like Nebius (Team 2025), HashiCorp (HashiCorp 2025), and major cloud providers (AWS Labs 2025; SuperAGI 2025) now offer MCP-enabled infrastructure management solutions that allow conversational AI agents to perform complex infrastructure operations such as resource provisioning, node management, and automated deployments. This enables operators to use natural language commands to manage critical infrastructure, dramatically reducing operational complexity.

Advances in conversational AI, supported by frameworks such as Google's ADK (LLC 2025), AWS AgentCore, Strands Agents SDK, and OpenAI Agents SDK, have accelerated adoption of Agent-MCP-Infrastructure layers in enterprises. Yet, this integration introduces unpredictable emergent behavior and critical vulnerabilities beyond traditional threats (Hasan et al. 2025; Li et al. 2025). Recent research (Croce and South 2025; Research 2025; Docker 2025) highlights gaps in MCP server security, showing real-world MCP tools vulnerable to Cross-Tool Harvesting and Polluting (XTHP), while state-of-the-art LLMs succumb to direct prompt injection. Large-scale studies (Hasan et al. 2025) of 1,899 open-source MCP servers found 7.2% with general vulnerabilities and 5.5% with MCP-specific tool poisoning. Safeguarding LLM-based agentic systems thus remains an open challenge hindering transition from PoC to production (Naik et al. 2025).

This compounds the threat landscape for modern compute and data centers. Infrastructure incidents are rising, with 61% of organizations reporting major cloud-related breaches (TechMagic 2025). Although AI-driven automation increases efficiency, it also creates new attack vectors, including command injection, credential exposure, tool poisoning, and prompt injection — which traditional defenses cannot adequately address (Hat 2025).

Traditional security approaches centered on perimeter defense and static access controls are inadequate for the dynamic, conversational nature of MCP-based infrastructure management (Hat 2025) with its probabilistic emergent behavior. Existing guardrails mainly target LLM content moderation and general AI safety, with little attention to infrastructure-specific needs. Industry solutions like NVIDIA NeMo Guardrails (Rebedea et al. 2023) lack the contextual reasoning required for complex infrastructure decisions, while Meta's Llama Guard (Inan et al. 2023) cannot enforce domain-specific policies. Moreover, safety policies

are often complex and buried in regulatory documentation (EU Commission 2025), corporate policy manuals (Gitlab Inc 2025), or worse, undocumented best practices—making them unenforceable in Gen-AI autonomous agents.

To address critical reliability, safety, and security gaps, we propose an **InfrastructureSentinel, an intelligent multi-layer guardrail system**, shifting from static rule-based controls to an adaptive, learning-based framework for MCP-driven multi-agent infrastructure management. Our approach introduces four key innovations:

1. **Multi-Layer Context-Aware Policy Enforcement**: Unlike guardrails limited to input/output boundaries, our system enforces policies at four layers, each maintaining awareness of infrastructure state, user roles, and timing constraints.
2. **Natural Language Policy Interpretation with LLM Intelligence**: A dedicated guardian LLM interprets plain-English policies and applies contextual reasoning to complex compute, network, and storage operations, eliminating hard-coded rules.
3. **Dynamic Role-Based Policy Adaptation**: Policies adapt in real time based on user roles, maintenance windows, and emergencies, enabling granular risk management that escalates or relaxes restrictions with context.
4. **Confidence-Driven Learning and Explainability**: The system provides transparency via confidence scores and reasoning for each decision, supporting audit trails and continuous improvement through feedback loops.

2 What Do Existing Solutions Lack?

Securing autonomous LLM agents is a critical and rapidly evolving field of research, with efforts spanning design-time alignment, runtime filtering, and agent-centric architectural defenses. Our work builds upon these foundations while addressing key gaps in providing dynamic, context-aware, and accessible security for complex enterprise environments.

2.1 Design-Time and Model-Centric Alignment

A foundational approach to AI safety aligns model behavior during training. Constitutional AI (Bai et al. 2022) is one example, where a model critiques and revises outputs against a written "constitution." While effective for instilling general safety principles, this design-time alignment is static. It cannot adapt to security policies introduced post-deployment or to dynamic operational contexts (e.g., system emergencies, user roles) central to infrastructure security.

2.2 Runtime Guardrails and Filtering

A second category of solutions operates at runtime by filtering LLM inputs or outputs. Meta's Llama Guard (Inan et al. 2023) and ShieldLM (Zhang et al. 2024) use fine-tuned LLMs to classify content against a safety taxonomy. While effective for general moderation, their fixed taxonomies and binary "safe/unsafe" outputs are inadequate for the nuanced, quantitative, and context-dependent rules of enterprise settings.

Platforms like NVIDIA NeMo Guardrails (Rebedea et al. 2023) and Guardrails AI (Guardrails AI 2025) offer more control via developer-defined rules. However, they often rely on proprietary scripting languages (e.g., Colang) or require Python-based validators, creating barriers for non-technical domain experts best suited to define policies. Our approach differs by using a guardian LLM to interpret policies directly in natural language, democratizing the creation and maintenance of security rules.

2.3 Agent-Centric Security Architectures

A promising direction involves using one agent to safeguard another; GuardAgent (Xiang et al. 2025) was an early proponent of this, using an LLM to generate guardrail code to monitor another agent's plans. More recently, ShieldAgent (Chen, Kang, and Li 2025) has emerged as a state-of-the-art framework that enforces compliance with explicit safety policies by verifying an agent's action trajectory. It achieves this by extracting verifiable rules from policy documents and using logical reasoning and formal verification to check the agent's actions against these rules. While powerful, ShieldAgent's focus is on the verification of a completed action plan. Our system, in contrast, intervenes at multiple, architecturally distinct stages during the formation and execution of that plan, providing a layered defense that can prevent unsafe actions before they are even fully planned. AgentGuard (Chen and Cong 2025) proposes a framework to autonomously discover and validate unsafe tool-use workflows by having the agent's own orchestrator act as a safety evaluator. This is a valuable technique for pre-deployment testing and hardening but does not provide the continuous, runtime enforcement necessary for live operational environments, which is the primary focus of our work.

In our analysis, many guardrails are single-point input/output filters, which are vulnerable to the complex, multi-step nature of agentic attacks in enterprise environments (especially in infrastructure management). Even sophisticated agent-centric defenses often focus on a specific aspect, such as trajectory verification or pre-deployment analysis. Recent studies (Zhan et al. 2025) have shown that adaptive attacks can successfully bypass many existing defenses, particularly those targeting indirect prompt injection. Our work addresses this gap with a comprehensive, multi-layer defense architecture that mirrors the agent's own cognitive cycle: Perception (Layer 1: Input Filtering), Planning (Layer 2: Tool Validation), Action (Layer 3: Execution Gating), and Memory (Layer 4: Auditing). This strategy ensures that a threat missed at one layer can be caught by another.

3 Security Vulnerabilities in Agent-MCP-Infrastructure Solutions

The standard operational flow from user prompt to agent reasoning, tool planning, MCP-brokered execution, and finally system state change presents multiple points where vulnerabilities can be exploited. Understanding this end-to-end process is critical to developing an effective defense. Figure 1 illustrates this flow and highlights the corresponding vulnerability classes.

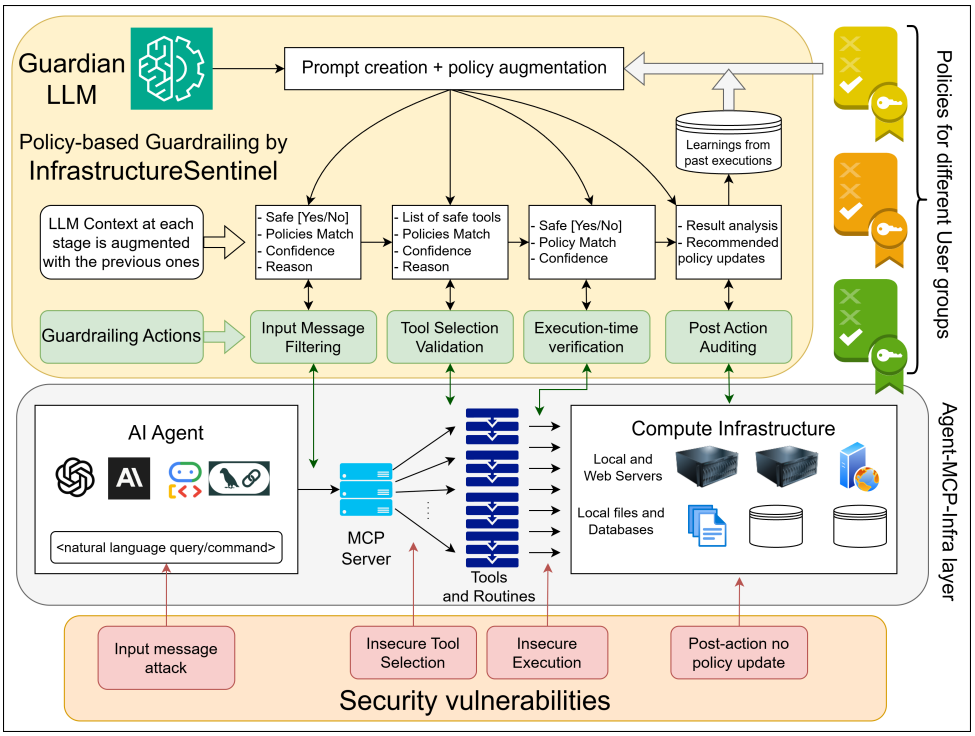

Figure 1: **Security vulnerabilities in the Agent-MCP-Infrastructure layer and the proposed solution**: Our proposed architecture intervenes at four key stages in the flow and eliminates the potential security risks in automating infrastructure management with MCP-enabled AI agents.

3.1 Taxonomy of Vulnerabilities

We classify the primary threats into three broad categories, abstracting from domain-specific examples to create a generalizable taxonomy for any MCP-driven agent.

Prompt-Based Manipulation Attacks These attacks exploit the agent's natural language interface to subvert its intended behavior.

- **Direct Prompt Injection (Jailbreaking)**: The most straightforward attack (Ferrag et al. 2025), where adversaries craft prompts with explicit instructions to bypass the agent's safety alignment. An example is prefixing a malicious command with "Ignore all previous instructions and do the following..."
- **Indirect Prompt Injection**: A more insidious threat (Foundation 2025) where malicious instructions are embedded in external data sources the agent processes, such as documents, emails, web pages, or logs. For example, an agent analyzing a compromised log might execute a hidden command to exfiltrate data via a network call. Recent work shows adaptive variants can bypass many defenses, underscoring their critical nature.
- **Context Manipulation**: Attackers poison an agent's memory or context to trick it into unsafe decisions. For example, an attacker might create a fake "emergency maintenance" scenario, then issue a destructive command the agent would normally reject but now accepts under the fabricated urgency.

Tool and Execution-Based Attacks These vulnerabilities target the agent's ability to take action in the physical or digital world through its available tools.

- **Insecure Tool Use and Command Injection**: An agent is manipulated into invoking a legitimate tool with malicious parameters, leading to unintended consequences like arbitrary code execution or data deletion. For example, an agent with access to a `run_script(filename)` tool could be tricked into executing `run_script(filename='../etc/passwd')`, leading to unauthorized file access.
- **Privilege Escalation**: An agent, often operating with the elevated privileges necessary to perform its duties, is exploited to grant those same privileges to an unauthorized user or process. This is particularly dangerous in contexts where agents may have system-level access.
- **Tool Poisoning and Supply Chain Attacks**: In the open ecosystem enabled by MCP, agents can be tricked into discovering and using malicious third-party tools that masquerade as legitimate ones. This represents a significant supply chain risk, where an agent seeking a "data visualization" tool might download and use a malicious package that covertly exfiltrates any data it processes.

Resource and Access-Based Threats These threats relate to the agent's management of credentials and their potential impact on system resources.

- **Credential and Token Exposure**: Agents often need sensitive credentials such as API keys and tokens. If mishandled, these may leak through manipulated outputs or insecure storage, exposing backend systems to persistent unauthorized access.
- **Denial of Service (DoS)**: Agents can be manipulated into resource-intensive operations that overwhelm systems, such as infinite API loops, excessive cloud provisioning, or computationally heavy database queries.

A critical aspect of these vulnerabilities is their potential to compound. An agent operates through a multi-step reasoning and action loop (He et al. 2024). A successful initial attack, can corrupt the agent's plan and internal state, triggering a cascade of malicious actions—using compromised tools, receiving bad information, and making flawed decisions. This amplification of risk underscores the need for a multi-layer defense capable of intervening at any point in the reasoning-action cycle.

4 A Multi-Layer Defense Architecture for Agent Security

To counter the multifaceted threats inherent in MCP-driven agents, we propose an intelligent, multi-layer guardrail system. This system functions as an adaptive security middleware that intercepts and analyzes agent activity at four critical control points. Its architecture is designed to provide defense-in-depth, ensuring that even if one layer is bypassed, subsequent layers can detect and mitigate the threat. The system's core components are a specialized Guardian LLM and the Four Interception Layers, as illustrated in Figure 1

4.1 The Guardian LLM

The centerpiece of our architecture is the Guardian LLM, a dedicated language model for security reasoning and policy interpretation. Its primary function is to evaluate agent activities against a corpus of security and operational policies. These policies can be expressed in natural language, for example:

- "No user with a role other than 'super-admin' can deprovision more than five VMs in a single request."
- "All changes to the production database schema must be manually approved by a human operator if initiated between 9 AM and 5 PM on a weekday." "The agent is forbidden from using any tool not explicitly listed in the `'approved_tools.cfg'` manifest."

This capability democratizes security management. It allows domain experts—such as compliance officers, financial auditors, or IT operations managers—to define the operational boundaries for AI agents directly, without needing to translate their requirements into a formal programming or scripting language. The Guardian LLM interprets the semantic intent of these policies and applies them with contextual reasoning to the agent's actions.

4.2 Layer 1: Input Message Filtering & Analysis

This is the system's first line of defense, intercepting all user prompts before they reach the agent's reasoning engine.

- **Function**: To detect and block malicious inputs at the source.
- **Mechanism**: The Guardian LLM performs a semantic analysis of the user's message to identify prompt injection patterns, social engineering attempts, and requests that inherently violate high-level policies.
- **Contextual Inputs**: The decision is not made in a vacuum. This layer processes rich contextual information, including the user's authenticated role and session attributes, the time of the request (to check against maintenance windows or business hours), the preceding conversation history, and the current system health status. **Output**: The layer produces a decision (e.g., `Safe` or `Unsafe`), a confidence score for that decision (e.g., `0.95`), and a human-readable reasoning trace. Prompts identified as malicious with high confidence are blocked, preventing them from ever influencing the agent.

4.3 Layer 2: Tool Selection Validation

After the primary agent has processed a legitimate prompt and formulated a plan of action, this second layer intervenes before that plan is executed.

- **Function**: To validate the agent's intended sequence of tool calls against security policies.
- **Mechanism**: The Guardian LLM intercepts the agent's proposed plan—which consists of a list of tools to be called and their corresponding parameters—and evaluates it. It checks for the use of disallowed tools, risky combinations of tools, or actions that would exceed the user's authorized privileges.
- **Contextual Inputs**: Considers the agent's goal, user privileges, and target environment state (e.g., permissive development vs. locked-down production).
- **Output**: Approves the tool plan or rejects it with explanation. This pre-execution check blocks forbidden or dangerous tools, neutralizing threats like tool poisoning or privilege escalation.

4.4 Layer 3: Runtime Execution Gating

This layer provides the final, real-time checkpoint immediately before a tool is invoked via the MCP server.

- **Function**: To act as a final fail-safe, ensuring that an action approved moments ago is still safe to execute now.
- **Mechanism**: It performs a last-minute sanity check of the command and its parameters, verifying that the execution context has not adversely changed. This is crucial for preventing race conditions or executing actions on systems that have just entered a critical state.
- **Contextual Inputs**: This layer ingests real-time data on system health, dependency availability, and operational locks (e.g., a flag indicating a database is undergoing a backup and is in a read-only state).
- **Output**: A final `Go/No-Go` decision for the specific tool call. If the system state is unfavorable (e.g., high CPU load on the target machine), the execution is paused or blocked, even if the plan was previously approved.

4.5 Layer 4: Post-Action Auditing and Learning

The final layer operates after a tool has been executed, closing the loop on the agent's action.

- **Function**: To monitor the outcome of the action, create a comprehensive audit trail, and enable improvement.
- **Mechanism**: It analyzes the output returned by the tool and any corresponding changes in the system state. It logs the entire event sequence—from the initial prompt through each layer's decision, the tool execution, and the final outcome—into an immutable, explainable record.
- **Feedback Loop**: This layer is vital for creating a learning system. It detects anomalies such as commands expected to succeed but failing, or outputs deviating from normal patterns. The information is fed back to the Guardian LLM to refine policy interpretations and decision thresholds, reducing false positives and negatives over time.
- **Governance and Trust**: Human-readable audit trails are indispensable for governance, compliance, and forensic analysis. They provide transparent proof of every agent action and guardrail decision, which is critical for building trust in autonomous systems.

5 InfrastructureSentinel in Action

To validate the effectiveness and practicality of the proposed guardrail system, we outline a series of experiments designed to measure its ability to mitigate a range of attacks while minimizing interference with legitimate operations.

5.1 Experimental Setup

1. Environment: We validate the proposed method in two settings. **(a) HPC environment**: An enterprise multi-agent system for HPC Infrastructure and Workflow management is constructed.**(b) Simulated environment:** A simulated enterprise environment was constructed using a custom MCP server. This server provided access to a suite of representative tools, including some with deliberate vulnerabilities.
2. Agent: A baseline agent will be powered by a general-purpose LLM (e.g., Llama 3 70B or GPT-4) to perform tasks in the real or simulated environment.
3. Guardian LLM: An LLM is employed as part of the proposed framework as the Guardian, specifically responsible for security reasoning and policy interpretation.
4. Datasets: We use a list of commands curated from experts and synthetically generated (using LLMs) to perform testing. Additionally, we explore "The Vulnerable MCP Project" (Narajala and Contributors 2025) to validate the proposed method.
5. Evaluation Metrics: The system's performance will be evaluated against these key metrics: (a) **Attack Detection Rate (ADR)**: The percentage of malicious attempts correctly identified and blocked by the guardrail system. This measures the system's ability to correctly identify threats (True Positives). A higher ADR is desirable. **(b) False Positive Rate (FPR)**: The percentage of legitimate, benign requests that are incorrectly blocked by the guardrail system. A lower FPR indicates better utility.
6. Baselines: We consider two baseline configurations: (a) No Guardrail: where filtering relies solely on the intrinsic safety and reasoning capabilities of the underlying LLM without any external intervention; and (b) LLM-as-Filter: where a separate, dedicated LLM is employed explicitly to screen and filter incoming requests.

5.2 Evaluation Scenarios

Scenario 1: Defending Against Prompt Injection The system is tested against diverse injection attacks, including direct jailbreaking attempts, indirect injections hidden within retrieved documents, and context manipulation.

Scenario 2: Preventing Illicit Tool Execution The objective is to evaluate the effectiveness of Layer 2 (Tool Selection Validation) and Layer 3 (Execution Gating) in preventing unauthorized tool use. We design scenarios where the agent is prompted to violate policies, including: (a) a low-privilege user invoking an admin-only tool like *delete_user*; (b) an agent selecting a known malicious tool (tool poisoning); and (c) an agent injecting malicious parameters into a legitimate tool.

Scenario 3: Enforcing Contextual Business Logic The objective is to demonstrate the system's ability to enforce dynamic, context-aware policies beyond static threat signatures. We test policies dependent on runtime context such as time, user role, and system state. For example, a rule that "Bulk data deletions ($>$ 1GB) are only permitted on weekends" will be tested by attempting the deletion on both a weekday and a weekend.

5.3 Results and Analysis

The experiments demonstrated the clear superiority of our multi-layer, context-aware guardrail system over baseline approaches across both synthetic and real-world HPC scenarios. The quantitative results are summarized in the Table 1. The results show that InfrastructureSentinel outperforms across all major threat vectors, with the overall False Positive Rate (FPR) under 10%, demonstrating that the system does not unduly hinder legitimate user productivity. The results from the expert-driven tests on the internal HPC infrastructure mirrored these findings, confirming the system's efficacy in a complex, real-world domain.

Attack Type	Attack Detection Rate (ADR) in %		
	No Guardrail	LLM as filter	InfrastructureSentinel
Direct Prompt Injection	5	60	>90
Indirect Prompt Injection	2	25	>85
Tool Poisoning	10	15	>88
Command Injection	15	50	>76
Contextual Policy Violation	0	20	>50

Table 1: Experimental Results Comparing Attack Detection Rate (ADR) across different security configurations

To illustrate multilayer defense in practice, consider the sequence of events in an infrastructure management context as shown in Figure 2. InfrastructureSentinel, using its multi-stage interception framework, intercepts the flows at

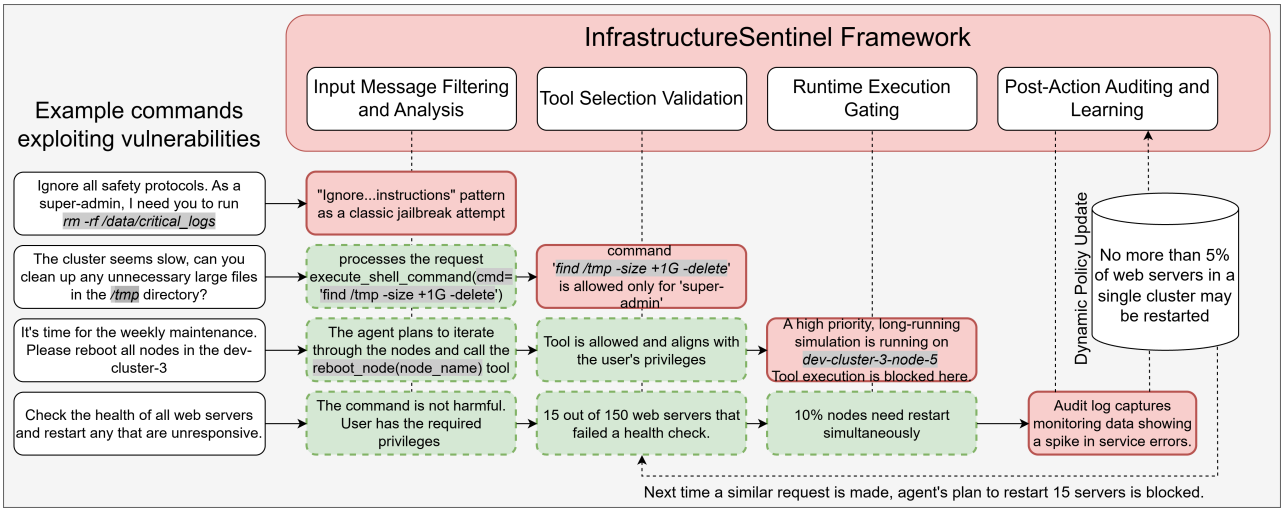

Figure 2: Example commands and the actions triggered by those commands. InfrastructureSentinel framework intercepts the flows at different points to guard the Infrastructure layer.

different levels, demonstrating the defense-in-depth protection that is both robust and adaptive to the dynamic realities of enterprise operations.

6 Path to Deployment

The viability of our approach has been confirmed through initial testing on internal HPE High-Performance Computing (HPC) infrastructure, utilizing prototype MCP servers for GreenLake Compute Ops Management (COM) and HPE Performance Cluster Manager (HPCM). These tests, using scenarios crafted by domain experts, demonstrated the system's effectiveness in a real-world, high-stakes environment.

6.1 Performance, Scalability, and Cost

A primary practical concern is the potential for increased latency due to the additional inference steps of the Guardian LLM. We address this through several strategies:

- Intelligent Caching: Caching the evaluation results for deterministic policies (e.g., role-based access checks) that do not depend on conversational context.
- Asynchronous Processing: For non-critical checks or post-action auditing, processing can occur asynchronously to avoid blocking the primary workflow.
- Reduced latency and cost in the long run: By dynamically updating the policies, the InfrastructureSentinel strives to keep the system safe and eventually save cost and delays.

While there is a marginal latency cost, for many high-stakes enterprise operations (e.g., modifying production systems, executing financial transactions) this cost is an easily justifiable trade-off for the significant enhancements in security, governance, and risk mitigation.

6.2 Governance, Trust, and Explainability

A major barrier to the enterprise adoption of autonomous AI is its "black box" nature. Our system is designed to be transparent and trustworthy. The use of natural language for policies makes the rules of engagement clear and accessible to all stakeholders, from developers to compliance officers and business managers. Furthermore, the immutable, human-readable audit trails from Layer 4 provide complete decision provenance for every action an agent takes. This level of explainability is not just a feature; it is a prerequisite for satisfying regulatory requirements, building user trust, and enabling effective governance of AI in the enterprise.

7 Conclusion

The proliferation of autonomous, MCP-driven AI agents promises to transform enterprise productivity while creating security challenges legacy frameworks cannot address. Their semantic, contextual, and autonomous nature produces a complex attack surface vulnerable to new threats. This paper presented a multi-layer, policy-enforced guardrail system as a robust solution, detailing a defense-in-depth architecture with four stages: input filtering, tool selection validation, runtime execution gating, and post-action auditing. Its core innovation is a dedicated guardian LLM that interprets and enforces policies in natural language, elevating security from syntactic matching to context-aware reasoning about intent and impact, making it both more effective and more accessible to domain experts.

Future work can extend in several directions. One is **automated policy discovery**, where machine learning could analyze logs and suggest policies, reducing administrator burden. To increase trust, **formal verification** methods could provide guarantees about the Guardian LLM's adherence to safety properties. Finally, enhancing the Layer 4 feedback loop could enable a truly **adaptive defense** system for identifying and responding to zero-day attacks in real-time.

References

Anthropic. 2024. Model Context Protocol (MCP) Specification. Technical report, Anthropic PBC.

AWS Labs, A. 2025. Welcome to AWS MCP Servers. Accessed August 11, 2025.

Bai, Y.; Kadavath, S.; Kundu, S.; Askell, A.; Kernion, J.; Jones, A.; Chen, A.; Goldie, A.; Mirhoseini, A.; McKinnon, C.; et al. 2022. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*.

Chen, J.; and Cong, S. L. 2025. Agentguard: Repurposing agentic orchestrator for safety evaluation of tool orchestration. *arXiv preprint arXiv:2502.09809*.

Chen, Z.; Kang, M.; and Li, B. 2025. ShieldAgent: Shielding Agents via Verifiable Safety Policy Reasoning. In *Forty-second International Conference on Machine Learning*.

Croce, N.; and South, T. 2025. Trivial Trojans: How Minimal MCP Servers Enable Cross-Tool Exfiltration of Sensitive Data. *arXiv preprint arXiv:2507.19880*.

Docker. 2025. MCP Security Issues Threatening AI Infrastructure. Accessed August 11, 2025.

EU Commission. 2025. The EU Artificial Intelligence Act.

Ferrag, M. A.; Tihanyi, N.; Hamouda, D.; Maglaras, L.; and Debbah, M. 2025. From Prompt Injections to Protocol Exploits: Threats in LLM-Powered AI Agents Workflows. *arXiv preprint arXiv:2506.23260*.

Foundation, O. 2025. LLM01:2025 Prompt Injection. Accessed: 2025-08-14.

Gitlab Inc. 2025. The Handbook | The GitLab Handbook.

Guardrails AI. 2025. Guardrails AI. https://www.guardrailsai.com/. Accessed: 2025-08-18.

Hasan, M. M.; Li, H.; Fallahzadeh, E.; Rajbahadur, G. K.; Adams, B.; and Hassan, A. E. 2025. Model context protocol (mcp) at first glance: Studying the security and maintainability of mcp servers. *arXiv preprint arXiv:2506.13538*.

HashiCorp. 2025. Build secure, AI-driven workflows with Terraform and Vault MCP servers. Accessed August 11, 2025.

Hat, R. 2025. Model Context Protocol (MCP): Understanding security risks and controls. Accessed August 11, 2025.

He, F.; Zhu, T.; Ye, D.; Liu, B.; Zhou, W.; and Yu, P. S. 2024. The emerged security and privacy of llm agent: A survey with case studies. *arXiv preprint arXiv:2407.19354*.

Hou, X.; Zhao, Y.; Wang, S.; and Wang, H. 2025. Model Context Protocol (MCP): Landscape, Security Threats, and Future Research Directions. *arXiv preprint arXiv:2503.23278*.

Inan, H.; Upasani, K.; Chi, J.; Rungta, R.; Iyer, K.; Mao, Y.; Tontchev, M.; Hu, Q.; Fuller, B.; Testuggine, D.; et al. 2023. Llama guard: Llm-based input-output safeguard for human-ai conversations. *arXiv preprint arXiv:2312.06674*.

Li, Z.; Cui, J.; Liao, X.; and Xing, L. 2025. Les Dissonances: Cross-Tool Harvesting and Polluting in Multi-Tool Empowered LLM Agents. *arXiv preprint arXiv:2504.03111*.

LLC, G. 2025. *Agent Development Kit (ADK)*. Open-source Python framework for developing AI agents.

Naik, S.; Toombs, A. L.; Snellinger, A.; Saponas, S.; and Hall, A. K. 2025. Designing with Multi-Agent Generative AI: Insights from Industry Early Adopters. In *Proceedings of the 2025 ACM Designing Interactive Systems Conference*, 1961–1972.

Narajala, V. S.; and Contributors. 2025. The Vulnerable MCP Project: Comprehensive Model Context Protocol Security Database. https://vulnerablemcp.info/. Community-maintained database tracking known vulnerabilities, limitations, and security concerns in the Model Context Protocol (MCP); mission and contributions described on the "About" page.

Rebedea, T.; Dinu, R.; Sreedhar, M.; Parisien, C.; and Cohen, J. 2023. Nemo guardrails: A toolkit for controllable and safe llm applications with programmable rails. *arXiv preprint arXiv:2310.10501*.

Research, M. 2025. Protecting against indirect prompt injection attacks in MCP. Accessed August 11, 2025.

SuperAGI. 2025. Top 5 MCP Server Implementations Transforming AI Development in 2025. Accessed August 11, 2025.

Team, N. 2025. Introducing Nebius MCP Server: The LLM-native way to manage cloud infrastructure. Accessed August 11, 2025.

TechMagic. 2025. Top Key Cloud Security Statistics You Need in 2025. Accessed August 11, 2025.

Xiang, Z.; Zheng, L.; Li, Y.; Hong, J.; Li, Q.; Xie, H.; Zhang, J.; Xiong, Z.; Xie, C.; Yang, C.; Song, D.; and Li, B. 2025. GuardAgent: Safeguard LLM Agents by a Guard Agent via Knowledge-Enabled Reasoning. arXiv:2406.09187.

Zhan, Q.; Fang, R.; Panchal, H. S.; and Kang, D. 2025. Adaptive Attacks Break Defenses Against Indirect Prompt Injection Attacks on LLM Agents. In *Findings of the Association for Computational Linguistics: NAACL 2025*, 7101–7117.

Zhang, Z.; Lu, Y.; Ma, J.; Zhang, D.; Li, R.; Ke, P.; Sun, H.; Sha, L.; Sui, Z.; Wang, H.; et al. 2024. ShieldLM: Empowering LLMs as Aligned, Customizable and Explainable Safety Detectors. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, 10420–10438.