

Engineering Least-Privilege Agentic AI Workflows: Patterns, Anti-Patterns, and Practitioner Guidance

Rajat Khanna
0009-0000-6057-0118
Amazon
Bellevue, WA, USA
rajatkhanha1999@gmail.com

Jatin Nandal
0009-0007-0432-6891
Amazon
Bellevue, WA, USA
jatinnandal77@gmail.com

Abstract—Enterprise deployment of agentic AI systems is accelerating sharply — Gartner projects that 40% of enterprise applications will embed task-specific AI agents by the end of 2026, up from fewer than 5% in 2025 [1]. Unlike deterministic software, AI agents execute autonomous tool-call chains, each carrying implicit privilege that classical IAM and RBAC frameworks were not designed to govern. We contribute six engineering patterns and five anti-patterns for applying least-privilege principles to production agentic workflows built on Model Context Protocol (MCP)-based tool-call architectures, grounded in enterprise deployment experience. The patterns address gateway-mediated tool access, ephemeral credential minting, scope declaration manifests, sub-agent permission ceilings, human-in-the-loop circuit breakers, and immutable audit trails, with each pattern explicitly mapped to the corresponding MCP primitives. We characterise the threat model motivating each pattern, illustrate the architecture with two technical diagrams, and catalog the anti-patterns most commonly encountered in enterprise deployments. This contribution fills a documented gap: no prior peer-reviewed work provides a practitioner-oriented engineering pattern catalog for least-privilege agentic systems.

Index Terms—agentic AI, least privilege, tool-call security, MCP, policy-as-code, multi-agent systems, ephemeral credentials, OPA

I. INTRODUCTION

AGEN TIC AI systems — autonomous workflows that perceive inputs, reason, and take actions through tool calls — are entering enterprise production at unprecedented scale [1]. These agents issue real side effects: querying databases, sending emails, executing shell commands, provisioning cloud resources, and spawning sub-agents. Each action is a privilege exercise; each tool call can mutate state, expose data, or trigger downstream cascades.

Classical access-control models (RBAC, ABAC, IAM) assume *deterministic actors*: a human user or static service account requests a specific resource. Agentic systems violate this assumption in three ways. They are *non-deterministic*: the same input may produce different tool-call sequences across inference runs. They are *compositional*: a parent agent may spawn child agents, making it non-trivial to bound the privilege envelope of any subtask. And they are *injection-susceptible*: the natural language surface that makes agents flexible also makes

them attackable via prompt injection [10], enabling adversaries to redirect agents toward unauthorised tool calls.

The Model Context Protocol (MCP), introduced by Anthropic in November 2024 and now under Linux Foundation governance [9], standardises the tool-invocation surface that least-privilege controls must govern. Despite growing recognition of agentic security risks [6], [8], existing work is either theoretical [2]–[5], survey-oriented [6], or non-peer-reviewed industry guidance [17]. No prior peer-reviewed paper provides a practitioner pattern catalog for production least-privilege agentic systems.

The NIST AI Risk Management Framework [12] and the EU AI Act (transparency obligations in force since August 2025 [11]) both identify access control and auditability as core requirements for responsible AI deployment. This paper operationalises those requirements at the engineering level.

Contributions: (1) A production-grounded threat model with five concrete attack scenarios. (2) Six engineering patterns for least-privilege agentic systems, each with architectural guidance. (3) A catalog of five anti-patterns and the security consequences they create. (4) Four key observations distilled from enterprise deployment practice. (5) Explicit mappings of all six patterns to Model Context Protocol (MCP) primitives, enabling direct implementation on MCP-based agentic architectures. The patterns and observations are grounded in the authors’ experience deploying and securing agentic AI systems in enterprise production environments.

II. THREAT MODEL

We consider a production enterprise agentic deployment with these trust boundaries: a human user, one or more orchestrator agents, specialist sub-agents, MCP tool servers, and external data sources ingested via Retrieval-Augmented Generation (RAG). We assume the LLM behaves as trained but is susceptible to prompt injection. We assume persistent adversaries with the ability to inject content into at least one data source processed by the agent.

T1 — Prompt Injection to Privilege Escalation. Adversarial instructions embedded in a processed document redirect the agent to invoke tools (e.g., `send_email`, `exec_shell`) outside the scope of the original task. If the agent holds a broad credential, the injected action executes. This threat is

This work is the independent personal research of the authors and does not represent the views or positions of Amazon or its affiliates.

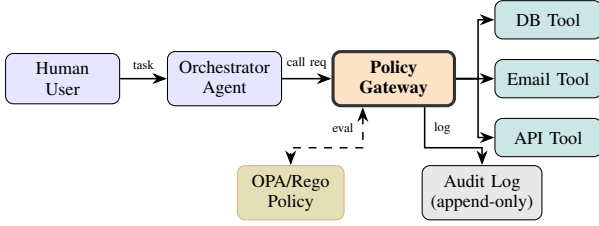


Fig. 1. Least-privilege agent gateway architecture. Every tool-call request from any agent is intercepted by the Policy Gateway, evaluated against the OPA/Rego policy, logged to the append-only audit store, and only then forwarded with a short-lived ephemeral credential.

documented extensively in 2024–2026 literature [6], [7], [10], [16].

T2 — Sub-Agent Permission Inheritance. An orchestrator spawns a sub-agent that, by framework default, inherits the orchestrator’s full credentials. A poisoned RAG retrieval directs the sub-agent to invoke a data-deletion tool it would never have been granted independently. Tallam [5] formalises this as a *transitive delegation* failure.

T3 — Long-Lived Credential Exfiltration. An API key stored in an agent’s context window or environment is exfiltrated by an injected instruction. Credentials valid for days or weeks provide a large exposure window compared to per-invocation tokens.

T4 — Over-Scoped Tool Abuse. A database tool provisioned with read-write access is available to an agent tasked only with querying. An injected or erroneous instruction issues DELETE statements the credential permits but no task requires.

T5 — Email Tool Weaponisation. An email tool without recipient restrictions is available to a customer-service agent. A prompt injection causes the agent to send phishing emails or exfiltrate sensitive data to attacker-controlled addresses.

Defence-in-depth posture. The gateway adopts a zero-trust posture toward the orchestrator agent itself: a tool-call request carries no implicit legitimacy beyond its declared identity and scope, regardless of which internal component issues it. This posture provides blast-radius containment independent of the model’s internal state. Even when the base LLM is adversarially manipulated — via prompt injection (T1) or other means — the scope declaration manifest (P3) statically bounds every call to the declared allowlist, and the per-invocation credential (P2) limits any exploitation to a single 30-second window. The gateway does not prevent model-level compromise; it constrains what a manipulated model *can do*, regardless of how it came to attempt it. Model-level attacks (fine-tuning-based jailbreaks, adversarial prompt suffixes) require complementary defences at the inference layer and are outside the scope of this work.

III. ENGINEERING PATTERNS

Figure 1 illustrates the overall architecture. Six patterns implement the least-privilege discipline within it.

A. P1: Gateway-Mediated Tool Access

Intent. Interpose a policy enforcement gateway between all agents and all tool servers (Fig. 1); no agent calls infrastructure

directly. **Implementation.** The gateway evaluates every tool-call request against a declarative policy (OPA/Rego [14]), checking agent identity, declared scope, tool name, parameters, and task context. Denied requests receive an error and an audit record; allowed requests are proxied with a freshly minted ephemeral credential (P2). Critically, the gateway is the *only* path to any tool — eliminating side-channel direct calls is a prerequisite for the policy to be meaningful. The gateway must be deployed with high availability; its failure policy must be *fail-closed* (all tool calls blocked when the gateway is unavailable) rather than fail-open, since a fail-open fallback negates the enforcement model entirely. **Addressed threats.** T1, T2, T4, T5. **Performance budget.** OPA documentation characterises microservice authorization policy evaluation as targeting a latency budget in the order of 1 ms [14], with benchmarked per-query evaluation at approximately 36–40 μ s on representative RBAC-style policies. For the tool categories most common in enterprise agentic deployments — REST API calls, database queries, and messaging service invocations — network round-trip latency is measured in tens to hundreds of milliseconds; sub-millisecond policy evaluation therefore adds negligible overhead relative to the tool call itself. End-to-end production benchmarking of the full gateway pipeline (policy evaluation plus ephemeral credential minting) is a planned future measurement.

B. P2: Ephemeral Credential Minting

Intent. Issue credentials per tool invocation, not per agent session. A credential valid for minutes or hours is a liability: an injected instruction or exfiltrated token provides an attacker a proportionally large exploitation window. Per-invocation minting bounds that window to a single call. **Implementation.** In practice, the gateway mints a signed JWT or short-lived STS token encoding agent-id, task-id, permitted tool, permitted parameter scope, and a short TTL calibrated to the expected call latency. The token is passed with the proxied request and expires on completion, invalidating it for replay or reuse across tasks. A representative JWT payload — with fields chosen to enable gateway-side revocation via the `jti` claim — is shown below.

```

{
  "agent_id": "svc-agent-v2",
  "task_id": "t-7f3a91bc",
  "tool": "db.query_orders",
  "scope": {"ops": ["SELECT"]},
  "exp": 1748200030, // 30 s TTL
  "jti": "cred-a1b2" // single-use
}

```

Workload identity infrastructure such as SPIFFE/SPIRE [15] provides the issuance substrate on which the gateway’s ephemeral minting can be built, replacing long-lived API keys with cryptographically attested, short-lived SVIDs (SPIFFE Verifiable Identity Documents) that carry the agent’s identity claim. **Addressed threats.** T1, T3 (stolen credentials expire before re-use).

C. P3: Scope Declaration Manifest

Intent. Require agents to declare their minimum required tool scope at registration; the gateway enforces that runtime

calls never exceed the declared scope. **Implementation.** Each agent version ships with a machine-readable manifest (a JSON Schema document or OpenAPI 3.1 components definition) enumerating permitted tools, parameter ranges, and data classifications. Any call not in the manifest is denied without further evaluation, creating a static upper bound on privilege independent of runtime state. Machine-checkable manifests enable automated scope diff and approval gating on every deployment pipeline update. **Addressed threats.** T1, T4, T5.

D. P4: Sub-Agent Permission Ceiling

Intent. Child agents receive the *intersection* of the parent’s granted permissions and the permissions requested for the child task; no child can hold more permissions than its parent. **Implementation.** At spawn time the gateway computes $\text{child_scope} = \text{parent_granted} \cap \text{child_requested}$. Any excess is dropped silently. The ceiling is enforced structurally by the gateway, not delegated to the parent agent’s logic. Figure 2 illustrates the ceiling enforcement. **Addressed threats.** T2. **Correctness sketch:** By induction on delegation depth, each ceiling application yields a scope that is a subset of its parent’s granted permissions. Set intersection is monotonically non-expansive, guaranteeing that privilege cannot escalate across arbitrarily deep spawning chains.

E. P5: Human-in-the-Loop Circuit Breaker

Intent. Irreversible or high-blast-radius tool calls pause and require explicit human approval before the gateway mints the credential. Human judgment is the last line of defence when the blast radius of an erroneous or injected action exceeds any automated policy’s ability to bound the damage. **Implementation.** Each tool in the scope manifest carries a risk classification (LOW / MEDIUM / HIGH / IRREVERSIBLE). Calls at or above a configured threshold suspend the task, hold the credential unminted, and route an approval request to the responsible approver; execution resumes only on explicit authorisation. A recurring pattern in deployment is that tools classified MEDIUM during rapid prototyping — unrestricted outbound email, bulk record deletion, and external API calls with write access — are reclassified HIGH or IRREVERSIBLE upon first security review. **Addressed threats.** T4, T5.

F. P6: Immutable Audit Trail

Intent. Every tool-call decision — allow or deny — is logged with sufficient detail for forensic reconstruction and compliance reporting. **Implementation.** Each log record includes: timestamp, agent-id, task-id, tool-name, redacted parameters, scope-requested, scope-granted, decision (ALLOW/DENY), policy rule triggered on deny, credential jti, and call outcome. Records are written to an append-only store; retention is determined by enterprise policy and applicable regulatory requirements. **Note:** Read operations must be logged; data exfiltration (T3) is realised through reads, not writes. **Addressed threats.** Detection and forensic reconstruction for T1–T5.

MCP integration. In deployments built on the Model Context Protocol [9], the six patterns map directly onto MCP

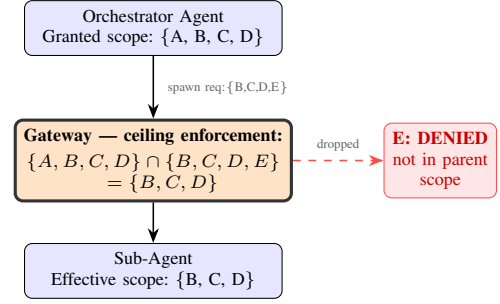


Fig. 2. Sub-agent permission ceiling (P4). The gateway computes the intersection of the parent’s granted scope and the child’s requested scope. Permission E is silently dropped because the parent does not hold it.

primitives: P1 interposes the gateway on all `tools/call` requests before they reach any MCP server; P2 attaches scoped, short-lived JWTs as bearer tokens to each proxied MCP call; P3 scope manifests correspond to the `tools` array in each MCP server’s capability declaration, constraining the runtime call set to the declared allowlist; P4 ceiling enforcement applies at the orchestrator’s MCP client layer before issuing tool calls on behalf of a spawned sub-agent; P5 circuit breakers suspend `tools/call` dispatch pending explicit human approval; and P6 audit records are generated from gateway intercept events, capturing both the MCP request and the server’s response.

IV. ANTI-PATTERNS CATALOG

Table I catalogs five anti-patterns documented in enterprise agentic deployments, their typical root causes, security consequences, and remedies.

TABLE I
ANTI-PATTERNS IN PRODUCTION AGENTIC DEPLOYMENTS

Anti-Pattern	Root Cause	Security Risk	Remedy
Persistent API keys in agent context	“Configure once, rotate later”	Exfiltration via injection; single key compromises all tasks	P2: Ephemeral minting
Sub-agent inherits parent full scope	Framework default behaviour [5]	Transitive delegation privilege escalation [5]	P4: Permission ceiling
Blocklist-based tool filtering	Fear of blocking legitimate paths	Novel vectors bypass known blocks	P3: Allowlist manifest
Auth logic in agent system prompt	Rapid prototyping left in production	Injection overwrites in-context policy; untestable	P1: External gateway
No audit log for read operations	“Reads are harmless”	Silent data exfiltration; no forensic reconstruction	P6: Log all calls

Among the five anti-patterns, blocklist-based tool filtering deserves particular attention. Enumerating disallowed tool calls is intractable: the combinatorial space of sequences, parameters,

and prompt contexts is unbounded. An allowlist — declaring exactly what is permitted and denying everything else — bounds the attack surface to the declared scope regardless of attacker creativity. Similarly, embedding authorisation logic in a system prompt creates a soft trust boundary that prompt injection can overwrite; only a gateway-enforced policy provides the hard boundary required in production.

V. KEY OBSERVATIONS FROM DEPLOYMENT PRACTICE

The observations below are distilled from applying the patterns in Section III to enterprise production agentic deployments. To illustrate the discovery process: applying the scope declaration manifest (P3) to an existing order-management agent revealed that its permission set substantially exceeded what its declared task required; the excess permissions were remnants of iterative feature development cycles that had never been formally revoked. A subsequent gateway audit log review (P6) showed that several of those excess permissions had in fact been exercised by the agent during normal operation — not for the declared task, but as side effects of prompt variations that steered the agent toward adjacent capabilities. These findings directly motivated O1 and O4 below, and illustrate why pattern adoption must be accompanied by active scope monitoring rather than one-time manifest configuration.

O1: Permission scope expands by default. In our experience, agent permission scopes tend to grow with each new feature iteration and seldom contract without deliberate process control. Each new capability request arrives with a tool requirement; no symmetric trigger exists for removing permissions once granted. The asymmetry is structural: granting a permission is a single-team decision taken at feature-request time, whereas revoking one risks breaking undocumented callers across the integration graph — a coordination cost that consistently defers removal. The excess permissions most consistently discovered in scope manifest audits fall into three categories in our experience: (a) *cross-domain write access* accumulated incrementally across feature iterations — for example, an order-management agent retaining write access to an inventory table from a prior integration that has since been superseded; (b) *decommissioned API endpoints* registered during development and not removed from the manifest after the scope was narrowed at launch; and (c) *inherited broad-scope credentials* originally provisioned for a different agent and reused by convention rather than by design. Each category has a distinct remediation path: write access is narrowed at the tool server’s parameter constraints; development endpoints are removed from the manifest as part of launch gating; inherited credentials require agent-level identity separation at registration. Automated scope review on every manifest change — with diff-based approval gating any expansion and explicit ownership attestation required for each retained permission — is the most effective countermeasure we have found.

O2: Proactive gateway deployment is substantially cheaper than reactive retrofit. In our experience, teams that deploy the policy gateway as part of initial system design — before agents reach production — face substantially lower

remediation costs than those who retrofit controls after an incident. Retrofitting requires dismantling direct agent-to-API integrations that have accumulated business logic, disrupts in-flight features, and is performed under the pressure of an active security finding.

O3: Sub-agent permission enforcement is the hardest pattern to deploy. Open-source orchestration frameworks such as LangGraph, AutoGen, and CrewAI do not natively model sub-agent identity as an independent credential-bearing principal; child agent calls inherit the orchestrator’s execution context and whatever credentials are in scope at the parent level. Pattern P4 therefore requires wrapping framework-provided spawning APIs with gateway interception rather than relying on native framework support — an integration overhead that teams consistently underestimate during initial deployment planning.

O4: Audit gaps arise from individually reasonable scoping decisions. Common patterns that produce detectability gaps include logging only denied calls, logging only write operations, and applying short retention windows on the assumption that read operations are harmless. Each individually seems reasonable; together they leave exfiltration paths unobservable.

VI. RELATED WORK

Uchibeke [2] frames the agentic AI security gap as an authorisation problem and proposes the Open Agent Passport (OAP) specification for pre-action authorisation; our patterns provide the engineering layer through which OAP would be operationalised. Ji et al. [3] propose SEAgent, a mandatory access control (MAC/ABAC) framework achieving 0% attack success rate on benchmarked privilege-escalation attacks; SEAgent is an academic framework evaluated in controlled settings, while our contribution focuses on engineering patterns for production deployment constraints. Agostino and D’Souza [4] apply the ALARA principle to context engineering in multi-agent systems, addressing information exposure in context windows rather than tool-call credential governance. Tallam [5] formalises authorisation propagation as a workflow-level property, identifying transitive delegation as a sub-problem our P4 directly addresses. Chhabra et al. [6] and Dehghantanha and Homayoun [8] provide the threat taxonomies that motivate our threat model. AWS Well-Architected Generative AI Lens GENSEC05-BP01 [17] recommends least-privilege for agentic workflows as non-peer-reviewed industry guidance; this paper provides the peer-reviewed technical complement.

VII. CONCLUSION

The central architectural principle this paper establishes is that the agent’s reasoning process is not a trust boundary — the gateway is. Classical IAM and RBAC, designed for deterministic principals, cannot govern non-deterministic tool-call chains, prompt-injection attack surfaces, or dynamically spawned sub-agents; a policy-as-code gateway must stand between every agent and every effector. This paper presented six engineering patterns — gateway-mediated tool access, ephemeral credential minting, scope declaration manifests, sub-agent permission ceilings, human-in-the-loop circuit breakers,

and immutable audit trails — together with five anti-patterns and four key observations, grounded in enterprise deployment practice.

Operationally, authorisation must be externalised to a policy-as-code engine, enforced before ephemeral credentials are minted, and logged for every call regardless of direction. Sub-agent scope must be ceilinged at spawn time via monotonically non-expansive intersection. These are applications of established least-privilege discipline [13] to a new class of principal; the engineering contribution of this paper lies in the concrete patterns required to make that discipline work at production scale with non-deterministic actors.

Open problems for future work include: formalising the scope manifest as a machine-checkable schema standard shared across orchestration frameworks; adding native P4 ceiling enforcement to mainstream agentic runtimes that currently model child agents as transparent extensions of the parent; and empirical measurement of gateway overhead and scope-drift rates across real deployments at scale.

Limitations. This pattern catalog is grounded in enterprise environments using MCP-compatible agentic architectures. The six patterns are protocol-agnostic at the architectural level: P1 may be deployed by intercepting the tool-dispatch layer in any framework — wrapping LangChain’s tool invocation call or intercepting an OpenAI-based deployment’s tool-call response handler before execution; P2 replaces session-scoped credentials with per-invocation tokens independent of the calling convention; P3 is an allowlist manifest enforced at tool registration regardless of the underlying library; and P4 is enforced at the agent-spawning call site independent of the orchestration framework. The MCP primitive mapping in Section III demonstrates concrete protocol-level implementability; production validation in non-MCP environments is deferred. End-to-end gateway overhead beyond the OPA policy evaluation component characterised in P1 has not been benchmarked in production and is deferred to a future quantitative study. Model-level attacks are addressed by the defence-in-depth posture described in Section II.

Disclosure: The authors used AI-assisted tools for grammar and copyediting only. All technical content, patterns, threat model, architectural decisions, and observations are the authors’ original work.

REFERENCES

- [1] Gartner, Inc., “Gartner Predicts 40 Percent of Enterprise Apps Will Feature Task-Specific AI Agents by 2026, Up from Less Than 5 Percent in 2025,” Press release, Aug. 26, 2025. [Online]. Available: <https://www.gartner.com/en/newsroom/press-releases/2025-08-26-gartner-predicts-40-percent-of-enterprise-apps-will-feature-task-specific-ai-agents-by-2026-up-from-less-than-5-percent-in-2025>
- [2] U. Uchibeke, “Before the Tool Call: Deterministic Pre-Action Authorization for Autonomous AI Agents,” arXiv preprint arXiv:2603.20953, Mar. 2026. [Online]. Available: <https://arxiv.org/abs/2603.20953>
- [3] Z. Ji, D. Wu, W. Jiang, P. Ma, Z. Li, Y. Gao, S. Wang, and Y. Li, “Taming Various Privilege Escalation in LLM-Based Agent Systems: A Mandatory Access Control Framework,” arXiv preprint arXiv:2601.11893, Jan. 2026. [Online]. Available: <https://arxiv.org/abs/2601.11893>
- [4] C. J. Agostino and N. D’Souza, “ALARA for Agents: Least-Privilege Context Engineering Through Portable Composable Multi-Agent Teams,” arXiv preprint arXiv:2603.20380, Mar. 2026. [Online]. Available: <https://arxiv.org/abs/2603.20380>
- [5] K. Tallam, “Authorization Propagation in Multi-Agent AI Systems: Identity Governance as Infrastructure,” arXiv preprint arXiv:2605.05440, May 2026. [Online]. Available: <https://arxiv.org/abs/2605.05440>
- [6] A. Chhabra, S. Datta, S. K. Nahin, and P. Mohapatra, “Agentic AI Security: Threats, Defenses, Evaluation, and Open Challenges,” arXiv preprint arXiv:2510.23883, Oct. 2025 (v3 Apr. 2026). [Online]. Available: <https://arxiv.org/abs/2510.23883>
- [7] E. DeBenedetti, J. Zhang, M. Balunović, L. Beurer-Kellner, M. Fischer, and F. Tramèr, “AgentDojo: A Dynamic Environment to Evaluate Prompt Injection Attacks and Defenses for LLM Agents,” in *Proc. 38th Conf. Neural Information Processing Systems (NeurIPS)*, 2024. arXiv:2406.13352. [Online]. Available: <https://arxiv.org/abs/2406.13352>
- [8] A. Dehghantanha and S. Homayoun, “SoK: The Attack Surface of Agentic AI – Tools, and Autonomy,” arXiv preprint arXiv:2603.22928, Mar. 2026. [Online]. Available: <https://arxiv.org/abs/2603.22928>
- [9] Anthropic, “Introducing the Model Context Protocol,” Nov. 2024. [Online]. Available: <https://www.anthropic.com/news/model-context-protocol>
- [10] OWASP Foundation, “OWASP Top 10 for LLM Applications and Generative AI v1.1,” 2025. [Online]. Available: <https://genai.owasp.org/llmrisk/llm01-prompt-injection/>
- [11] European Parliament and Council of the European Union, “Regulation (EU) 2024/1689 (Artificial Intelligence Act),” *Official Journal of the European Union*, Jul. 2024. [Online]. Available: <https://digital-strategy.ec.europa.eu/en/policies/regulatory-framework-ai>
- [12] National Institute of Standards and Technology, “Artificial Intelligence Risk Management Framework (AI RMF 1.0),” NIST AI 100-1, Jan. 2023. [Online]. Available: <https://doi.org/10.6028/NIST.AI.100-1>
- [13] J. H. Saltzer and M. D. Schroeder, “The Protection of Information in Computer Systems,” *Proc. IEEE*, vol. 63, no. 9, pp. 1278–1308, Sep. 1975. doi: 10.1109/PROC.1975.9939
- [14] Open Policy Agent Project (CNCF), “Open Policy Agent,” 2023. [Online]. Available: <https://www.openpolicyagent.org/>
- [15] SPIFFE Project (CNCF), “SPIFFE: Secure Production Identity Framework for Everyone,” CNCF Graduated Project, Sep. 2022. [Online]. Available: <https://spiffe.io/docs/latest/spiffe-about/overview/>
- [16] MITRE Corporation, “MITRE ATLAS: Adversarial Threat Landscape for Artificial-Intelligence Systems,” 2025. [Online]. Available: <https://atlas.mitre.org/>
- [17] Amazon Web Services, “AWS Well-Architected Generative AI Lens — GENSEC05-BP01: Implement Least Privilege Access and Permissions Boundaries for Agentic Workflows,” 2026. [Online]. Available: <https://docs.aws.amazon.com/wellarchitected/latest/generative-ai-lens/gensec05-bp01.html>

- [1] Gartner, Inc., “Gartner Predicts 40 Percent of Enterprise Apps Will Feature Task-Specific AI Agents by 2026, Up from Less Than 5