Journal of Recent Trends in Computer Science and Engineering (JRTCSE)
Volume 14, No. 4, for the July - August 2026, PP. 10-26.
ISSN: 2322-0872
https://jrtcse.com
DOI: https://doi.org/10.70589/JRTCSE.2026.14.4.2

Runtime Authority-Aware Threat Modeling for Agentic AI Systems: A Framework for Intent, Tool, Memory, and Delegation Risk

Ravindra Annam,
Product & AI Security Architect, United States.

Abstract

Enterprise AI agents increasingly retrieve external content, select tools, invoke APIs, retain memory, delegate work, and compose multi-step actions. These capabilities complicate conventional threat modeling because security consequences depend not only on where data flows, but also on what can influence agent decisions and what authority those decisions can exercise. This paper proposes the Authority-Aware Agentic Threat Model (AATM), a structured review overlay organized around seven dimensions: intent, context, authority, tool capability, memory, delegation, and action composition. AATM adds authority surface, influence flow, and authority graph artifacts to established architecture analysis. Six representative scenarios and a worked enterprise case study are examined using a clearly defined Explicit/Derived/Absent rubric. The analysis is scenario-based and single-analyst; it does not claim empirical superiority over STRIDE or other established methods. Instead, it shows how AATM makes task-authority mismatch, transitive privilege, persistent influence, and sequence-dependent risk explicit in review artifacts. The resulting design guidance emphasizes task-scoped credentials, narrow semantic tools, provenance-aware memory, independent runtime authorization, delegation constraints, and sequence-aware policy. AATM is intended as a practical architectural lens for agent systems whose execution paths are partly determined at runtime.

Keywords: Agentic AI Security; AI Agents; Threat Modeling; Runtime Security; Prompt Injection; Authorization; Tool Security; Memory Poisoning; Multi-Agent Systems; Least Privilege.

Citation: Ravindra Annam. (2026). Runtime Authority-Aware Threat Modeling for Agentic AI Systems: A Framework for Intent, Tool, Memory, and Delegation Risk. Journal of Recent Trends in Computer Science and Engineering, 14(4), 10-26.
DOI: https://doi.org/10.70589/JRTCSE.2026.14.4.2

1. Introduction

Large language models are increasingly embedded in software that can act on external systems. An enterprise agent may search repositories, retrieve documents, open or update tickets, call internal APIs, send messages, modify cloud resources, retain state, and ask other agents to complete portions of a task. In such systems, the model is no longer only a content generator; it participates in selecting an execution path.

That distinction matters to threat modeling. A conventional application generally exposes a set of developer-defined paths whose components, identities, trust boundaries, and

data stores can be enumerated during design review. An agentic application still has an engineered architecture, but the particular sequence of tools and parameters may be selected at runtime in response to natural-language objectives and retrieved context.

For a security reviewer, this introduces a second question alongside the familiar question of where data moves: what information can influence a decision, and what authority can that decision exercise? An attacker-controlled document, for example, may have no direct permission to query a customer database. If the document can influence an agent that has such permission, however, the relevant security path crosses an influence boundary before it crosses an authorization boundary.

Recent agent-security benchmarks make this problem concrete. InjecAgent evaluates indirect prompt injection across 1,054 test cases, 17 user tools, and 62 attacker tools [5]. AgentDojo provides 97 realistic tasks and 629 security test cases for agents operating over untrusted data [6]. These results motivate architectural controls that do not assume the model will always distinguish legitimate task context from adversarial instructions.

This paper introduces AATM to make those relationships explicit during architecture review. Its contribution is not a replacement for STRIDE, attack trees, data-flow diagrams, zero trust, or least privilege. Instead, AATM adds a set of agent-specific questions about intent, context provenance, effective authority, tool granularity, persistent memory, delegation, and multi-step action composition.

The paper makes five contributions: (1) a definition of agent authority surface; (2) a seven-dimensional AATM model; (3) data, influence, and authority flow overlays; (4) a structured scenario-based comparison using a clearly defined rubric; and (5) a worked enterprise case study showing how the method changes concrete design decisions.

Contribution positioning. AATM is primarily a structured threat-modeling overlay and review framework, not a new general-purpose security primitive or fully formal modeling language. Its originality lies in combining agent-specific concerns into reusable artifacts that explicitly connect untrusted influence to direct and reachable authority. Individual mechanisms such as least privilege, scoped credentials, workflow controls, provenance, and quotas predate AATM; the contribution here is to require them to be examined together as part of agentic threat modeling and to provide a common rubric for doing so.

2. Background and Related Work

AATM draws on long-standing security principles. Saltzer and Schroeder's principles of least privilege and complete mediation remain directly applicable: an agent should receive only the authority needed for the current task, and consequential operations should be independently mediated [1]. NIST Zero Trust Architecture similarly emphasizes explicit, resource-centric authorization rather than implicit trust [2]. NIST's AI Risk Management Framework and Generative AI Profile provide broader governance and risk-management context for AI-enabled systems [3][4].

Agent-specific research has focused heavily on prompt injection and tool use. InjecAgent demonstrates how malicious instructions embedded in external content can redirect tool-integrated agents [5]. AgentDojo provides an environment in which utility and security can be evaluated together [6]. CaMeL subsequently separates control and data flows and applies capability concepts to constrain unauthorized information flow; the authors report 67% task completion on AgentDojo under their provable-security

constraints [7]. These works support an architectural conclusion that is central to AATM: security should constrain what manipulated model behavior can accomplish, not rely only on detecting manipulation.

Operational taxonomies provide complementary perspectives. The OWASP Top 10 for Agentic Applications 2026 catalogs risks specific to systems that plan and act [8]. MITRE ATLAS provides an adversary-oriented knowledge base for AI systems, including agentic-AI techniques [9]. The Model Context Protocol (MCP) authorization and security guidance addresses resource-bound tokens, confused-deputy risks, least-privilege scopes, and token-passthrough concerns [10][11][12]. AATM uses these sources as inputs but focuses on a narrower architecture-review question: how does influence become effective authority in a specific agent system?

Scope of literature review. This section is not intended as a systematic literature review. The cited sources are representative of the categories most directly relevant to architectural authority and runtime control: foundational protection principles, AI risk guidance, agent-security benchmarks, control/data-flow defenses, agentic taxonomies, and tool authorization. Other emerging work on agent guards, tool safety, and authorization middleware remains important but is outside the review's exhaustive scope.

3. Research Questions

The primary research question is: How can conventional threat modeling be extended so that security reviewers systematically identify risks created when AI agents dynamically exercise, delegate, persist, and compose authority at runtime?

RQ1. Which agent-specific properties are easy to leave implicit when analysis focuses on components, identities, interfaces, and data flows?

RQ2. Does explicit modeling of influence, authority, memory, delegation, and action composition expose additional security paths or clearer control points?

RQ3. Which architectural controls constrain runtime authority while preserving useful autonomous execution?

4. Authority-Aware Agentic Threat Model (AATM)

AATM = { I, C, A, T, M, D, S }

I denotes intent, C context, A authority, T tool capability, M memory, D delegation, and S action sequence/composition. The dimensions are deliberately operational: each should result in questions that can be answered from architecture, policy, and runtime design.

The notation used throughout this paper is engineering shorthand intended to make review artifacts precise and comparable. It is not presented as a complete mathematical semantics for agent behavior.

4.1 Intent

Intent is the outcome authorized by the originating principal. Natural-language goals are often underspecified. “Resolve the customer's billing issue” does not automatically authorize the largest possible refund. Reviewers should therefore distinguish the authorized outcome from the plan inferred by the model.

4.2 Context

Context is information capable of influencing a future decision. It includes system instructions, user prompts, retrieved documents, webpages, email, tool responses, conversation state, persistent memory, and messages from other agents. AATM records provenance, trust, persistence, and sensitivity rather than treating all context as equivalent.

4.3 Authority

Authority is the set of consequential operations available to an execution. A useful representation is Auth(a) = {(Resource, Operation, Scope)}. This makes the permission behind a tool visible and supports comparison between task requirements and granted capability.

4.4 Tool Capability

Tool design determines the language of effects an agent can express. execute_shell(command) exposes substantially more authority than restart_service(service_id). Typed arguments, narrow semantic operations, allow lists, server-side validation, and business invariants therefore become threat-model properties rather than implementation details.

4.5 Memory

Memory is state that can influence later execution. A security-relevant memory record can be represented as M = (Value, Source, Writer, Timestamp, Trust, TTL). Provenance matters because a future task should be able to distinguish a user-confirmed preference from content copied from an untrusted webpage or inferred by the model.

4.6 Delegation

Delegation transfers work across agents or services. The receiving component should retain the originating principal, original intent, delegated scope, and approval state. A low-privilege coordinator should not obtain high effective privilege merely because it can ask a privileged agent to act.

4.7 Action Composition

Authorization of individual operations does not guarantee authorization of their sequence. Reading a confidential document, creating a copy, enabling public sharing, and sending an external email may each be permitted in isolation while their composition creates exfiltration.

$$DelegatedAuthority \leq AuthorizedTaskAuthority$$

$$EffectiveAuthority(a) = DirectAuthority(a) \cup ReachableDelegatedAuthority(a)$$

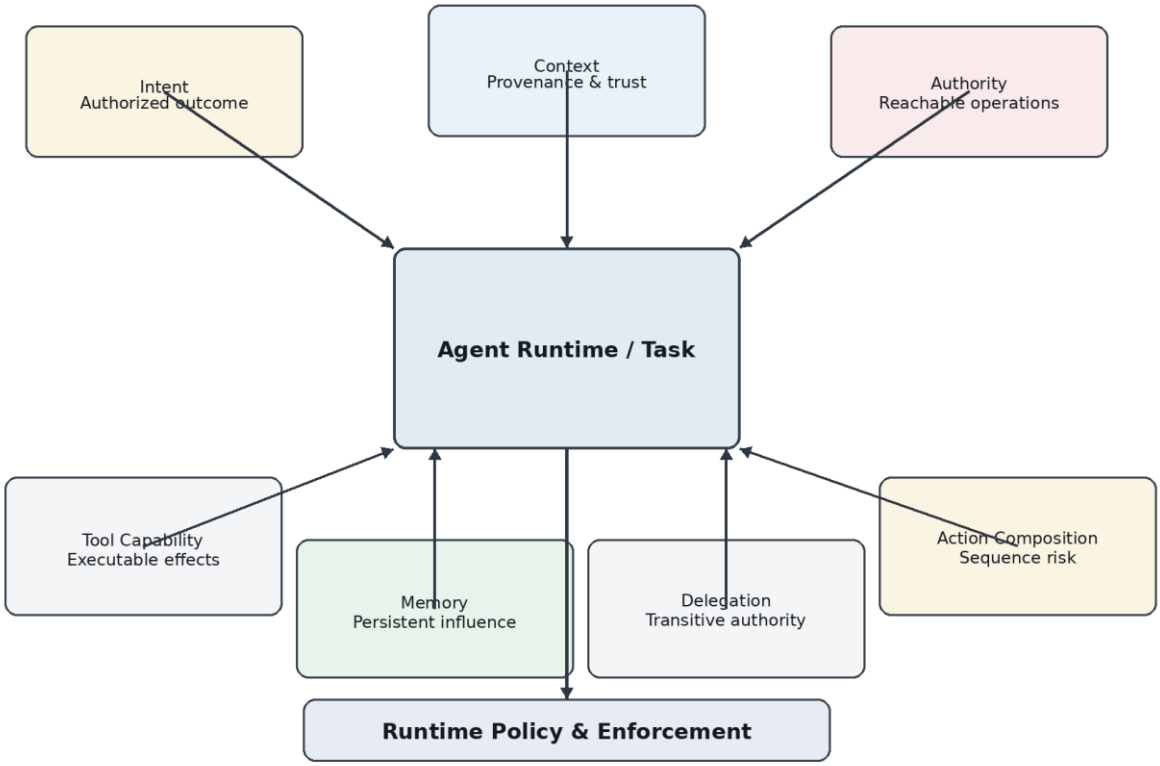

Figure 1. AATM conceptual model. Seven agent-specific review dimensions surround runtime execution, with policy and enforcement constraining consequential actions.

5. Authority Surface and Flow Model

AATM defines an agent's authority surface as the complete set of consequential operations the agent can cause directly or through reachable delegation. This differs from attack surface. Attack surface asks where an adversary can interact with the system; authority surface asks what consequences an influenced execution can produce.

Two agents using the same underlying model may therefore have very different risk profiles. A public-document summarizer has limited authority. An agent connected to corporate email, customer records, source repositories, cloud administration, and financial systems has a much larger authority surface. Reducing unnecessary authority limits consequences even when adversarial influence cannot be eliminated.

$$\textit{Potential Impact}(a) \propto \textit{Authority Surface}(a)$$

AATM overlays three flows on the architecture: data flow (where information moves), influence flow (what information can alter decisions), and authority flow (what component can cause a consequential operation). The security path of interest is often:

Untrusted Source → influence → Agent Decision → authority → Protected Resource

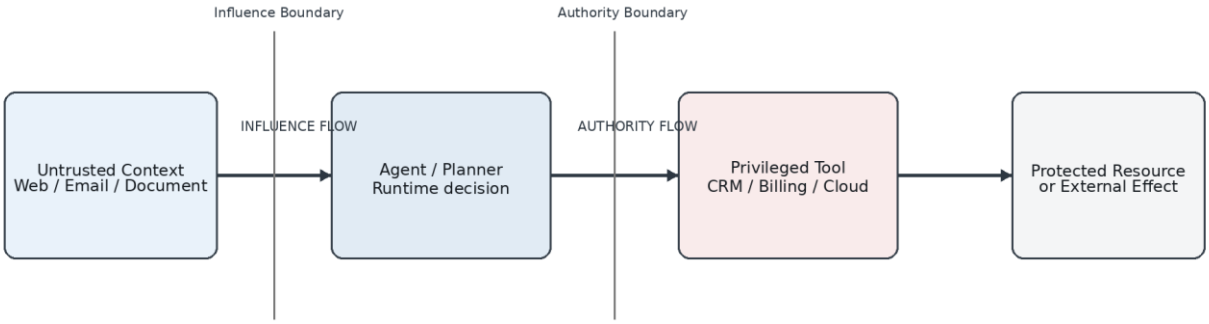

Figure 2. Influence-to-authority flow. Untrusted context can shape an agent decision before the decision crosses an authority boundary into a privileged tool or protected resource.

5.1 Authority Graphs and Influence Paths

An authority graph is a directed, labeled graph G = (V, E_a, E_d), where V contains principals, agents, tools, and protected resources; E_a represents direct authority or tool-invocation relationships; and E_d represents delegation relationships. Context sources are connected to agent decision points through a separate influence relation E_i. This separation allows a reviewer to ask both "what can this component directly do?" and "what additional authority can it reach through another agent or tool?"

For agent a, the authority surface can be written as AS(a) = {(r, op, scope) in DirectAuthority(a) union ReachableDelegatedAuthority(a) | op has a consequential external effect}. EffectiveAuthority(a) is the reachable set used to construct AS(a). In practice, the graph can be drawn as an overlay on an existing DFD: influence edges are labeled with provenance/trust, while authority and delegation edges are labeled with operations or scopes.

For the customer-support case study, a simplified path is Public Web --influence--> Support Agent --authority--> CRM; Support Agent --authority--> Billing; Support Agent --authority--> Email; and Support Agent --write/read--> Persistent Memory. The graph exposes a path from an untrusted source to billing and email authority even though the web source has no direct enterprise credential. Standard DFDs can contain the same nodes, but the AATM overlay distinguishes information movement from decision influence and reachable authority.

6. Risk Characterization

AATM can be used as a qualitative assessment aid. Each dimension may be scored from 0 to 5 to force explicit discussion of exposure. The score is not a calibrated probability of breach and should not be presented as one.

Table 1

Dimension	**0 – Lower exposure**	**5 – Higher exposure**
Intent ambiguity	Fixed operation	Open autonomous objective
Context exposure	Trusted sources only	Arbitrary external content

Authority	No consequential action	Administrative/destructive
Tool capability	Narrow/read-only	Generic privileged execution
Memory	None	Persistent security-sensitive memory
Delegation	None	Recursive privileged delegation
Composition	Single operation	Arbitrary multi-step workflow

Rbase = (I + C + A + T + M + D + S) / 35

Organizations may weight dimensions differently, but any weighting should be documented and calibrated to the system being reviewed. The value of the score is consistency of review, not mathematical precision.

7. Evaluation Methodology

The evaluation is a structured analytical comparison rather than an empirical benchmark. The same six scenarios are examined first with a conventional architecture view—components, data flows, trust boundaries, identities, authorization, and STRIDE categories—and then with AATM overlays. The purpose is to determine whether agent-specific questions are explicit, require analyst inference, or are not naturally represented.

7.1 Step-by-Step Comparison Protocol

Step 1 - Inputs. For each scenario, define the system objective, actors, components, data stores, tools, trust boundaries, identities/credentials, and relevant external context sources. No hidden implementation assumptions are added after the analysis begins.

Step 2 - Conventional analysis. Produce a baseline DFD and apply conventional trust-boundary, identity/authorization, and STRIDE-oriented reasoning. Record identified threats and control points without using AATM terminology.

Step 3 - AATM overlay. Add intent, context provenance, influence edges, direct authority, reachable delegated authority, tool granularity, memory persistence, and action-sequence relationships. Construct or update the authority graph.

Step 4 - Rubric coding. For each of C1-C7, classify the property as Explicit, Derived, or Absent in the baseline and AATM artifacts. A short rationale is recorded for each classification so that another reviewer can challenge or reproduce the reasoning.

Step 5 - Delta analysis. Compare the two sets of artifacts and record newly explicit paths, task-authority mismatches, persistence/delegation relationships, sequence risks, and concrete enforcement points. The output is a comparison table plus updated architecture/authority diagrams.

Step 6 - Interpretation. Treat differences as analytical visibility, not proof that one method is empirically superior. This study was performed by a single analyst; inter-rater reliability is not measured.

Table 2

Criterion	Question
C1 – Entry	Is the adversarial entry point identified?
C2 – Influence	Is the path from hostile information to agent decision explicit?
C3 – Authority	Is the consequential authority exercised by the agent identified?
C4 – Persistence	Is cross-session or persistent influence represented?
C5 – Delegation	Is transitive authority through other agents represented?
C6 – Composition	Is sequence-dependent risk represented?
C7 – Enforcement	Does the model expose a concrete control point?

Coverage is classified as Explicit, Derived, or Absent. Explicit means the modeling construct directly requires the reviewer to represent the property. Derived means a skilled analyst can identify it using existing artifacts but must add reasoning not directly encoded by the artifact. Absent means the artifact, as drawn, does not naturally express the property. This rubric makes the comparison reproducible while avoiding claims based on unperformed experiments.

8. Scenario Analysis

8.1 Indirect Prompt Injection

A research agent retrieves an attacker-controlled webpage while answering a legitimate business question. The page attempts to redirect the agent toward confidential data and an outbound tool. A conventional data-flow diagram can show Internet-to-agent and agent-to-enterprise-API connections; AATM additionally marks the external content as an influence source and traces the path to the authority-bearing tool. The resulting controls are provenance labeling, restricted outbound destinations, independent authorization of sensitive actions, and data-flow constraints. This attack class is represented directly in InjecAgent and AgentDojo [5][6].

8.1.1 Worked Comparison Using C1-C7

For the indirect prompt-injection scenario, the baseline DFD contains the external webpage, research agent, enterprise API, and outbound channel. STRIDE-oriented review identifies the external trust boundary and can derive tampering/information-disclosure concerns. The AATM overlay additionally labels the webpage-to-planner edge as untrusted influence and the agent-to-enterprise/API and agent-to-outbound-tool edges as authority. The coding below shows the judgment used in this manuscript.

Table 3. Worked C1-C7 comparison for indirect prompt injection. Explicit = directly represented; Derived = requires analyst inference; Absent = not represented in the scenario/artifact

Criterion	Conventional	AATM	Rationale
C1 Entry	Explicit	Explicit	External webpage/trust boundary is visible in both.
C2 Influence	Derived	Explicit	Baseline shows data movement; AATM labels decision influence.
C3 Authority	Derived	Explicit	Baseline shows API calls; AATM labels consequential scope.
C4 Persistence	Absent	Absent	This scenario contains no persistent state.
C5 Delegation	Absent	Absent	No delegated agent is present.
C6 Composition	Derived	Explicit	Exfiltration requires a sequence across retrieval and outbound action.
C7 Enforcement	Derived	Explicit	AATM identifies policy points at authority boundaries.

The delta is therefore not a count of newly discovered vulnerabilities. It is a change in how directly the architecture artifacts represent the influence path, reachable authority, sequence risk, and enforcement location. This distinction is important because a skilled conventional reviewer may still discover the same threat.

8.2 Excessive Agency

A cloud administrator asks an agent to diagnose a production problem. The task requires read access to metrics, logs, and deployment state, but the agent inherits broad administrative credentials. AATM compares RequiredAuthority with AvailableAuthority

and identifies the mismatch as part of the threat model. The preferred design is a short-lived task credential containing only the operations and resources required for diagnosis.

8.3 Generic Tool Misuse

A tool such as execute_shell(command) exposes a broad execution language. A semantic tool such as restart_service(service_id) exposes a constrained business capability. AATM records this difference as tool authority surface. The control is not merely prompt filtering; it is a narrower interface backed by typed parameters, server-side checks, and deterministic invariants.

8.4 Persistent Memory Poisoning

An attacker-controlled document causes an agent to store a malicious preference or operational fact. The initial interaction ends, but the stored value influences a privileged decision in a later session. AATM requires the reviewer to model writer, source, trust level, retention, and future consumers of the memory. Controls include provenance, confirmation for sensitive writes, expiry, audit history, integrity checks, and deletion.

8.5 Delegation-Based Privilege Amplification

A low-privilege coordinator can request arbitrary actions from a high-privilege operations agent. The coordinator's effective authority therefore exceeds its direct credential set. The receiving agent should independently evaluate originating principal, original intent, delegated scope, requested action, resource sensitivity, and approval state. MCP's authorization guidance addresses related confused-deputy and token-audience concerns [10][11].

8.6 Unsafe Action Composition

An enterprise assistant may legitimately read internal documents, create files, change sharing settings, and send external email. Each action may pass an isolated authorization check. The sequence read confidential data → create copy → public share → external email creates an exfiltration path. AATM therefore requires selected high-risk combinations to be evaluated at task or sequence level.

9. Worked Enterprise Case Study

To make the framework concrete, consider a customer-support agent used by a telecommunications enterprise. The agent can retrieve product and policy documents, read customer records from a CRM, issue limited billing credits, send customer email, and retain selected preferences in persistent memory. It also receives knowledge from public vendor documentation when troubleshooting device issues.

Table 4

Component	Direct capability	Trust / concern
Public retrieval	Read external web content	Untrusted influence source
Knowledge base	Read internal policy	Trusted but may be stale
CRM tool	Read assigned customer records	Sensitive data

Billing tool	Issue credit up to task limit	Financial authority
Email tool	Send approved customer messages	External communication
Persistent memory	Store selected preferences	Cross-session influence
Support agent	Plan and compose tool calls	Runtime decision point

A conventional review would identify the external retrieval boundary, customer-data sensitivity, billing authorization, and email egress. AATM adds four questions that materially affect the design. First, can public retrieval influence a later billing or email decision? Second, does the billing tool expose a fixed task-scoped limit or merely inherit the employee's broader authority? Third, can untrusted content be written into persistent memory and later be treated as trusted? Fourth, can CRM read + email send create an unintended disclosure path even when both operations are independently authorized?

Applying the AATM dimensions produces the following design changes. External retrieval is labeled as low-trust context and cannot directly justify a financial or disclosure action. Billing authority is minted per task and capped by a policy-defined amount. Memory writes that affect future transactions require source provenance and confirmation. Email policy considers the sensitivity of data collected earlier in the same task. Delegated operations carry the original customer-service objective and do not inherit broader permissions from downstream services.

Illustrative use of the qualitative score. To demonstrate rather than merely define Rbase, the case study was scored before and after the proposed controls using the 0-5 scale in Section 6. The values are analyst judgments for prioritization, not measured probabilities: baseline I=3, C=4, A=4, T=3, M=4, D=2, S=4, giving Rbase=24/35=0.69. After task-scoped billing credentials, provenance-aware memory, narrower tools, and sequence-aware egress policy, the illustrative values become I=2, C=3, A=2, T=2, M=2, D=1, S=2, giving Rbase=14/35=0.40. The reduction indicates which dimensions the design changes address; it should not be interpreted as a 29-percentage-point reduction in breach probability.

Table 5

AATM dimension	**Case-study finding**	**Resulting control**
Intent	“Resolve billing issue” is broader than a specific credit	Bind financial action to explicit task policy
Context	Public troubleshooting content can influence planning	Preserve provenance; block low-trust context from authorizing high-impact actions

Authority	Employee identity may exceed task need	Mint task-scoped credential
Tool	Generic billing API could expose unnecessary operations	Expose narrow credit operation with server-side cap
Memory	Stored preference can persist beyond original interaction	Provenance, TTL, confirmation for sensitive memory
Delegation	Downstream service may hold stronger privilege	Carry original principal and delegated scope
Composition	CRM read plus external email can disclose data	Sequence-aware DLP / egress policy

This case study shows how applying AATM leads to specific changes in credential design, memory handling, tool interfaces, delegation scope, and email policy. The individual controls are established security practices; the AATM contribution is to surface their relevance together when an agent can convert information into actions across several trusted systems.

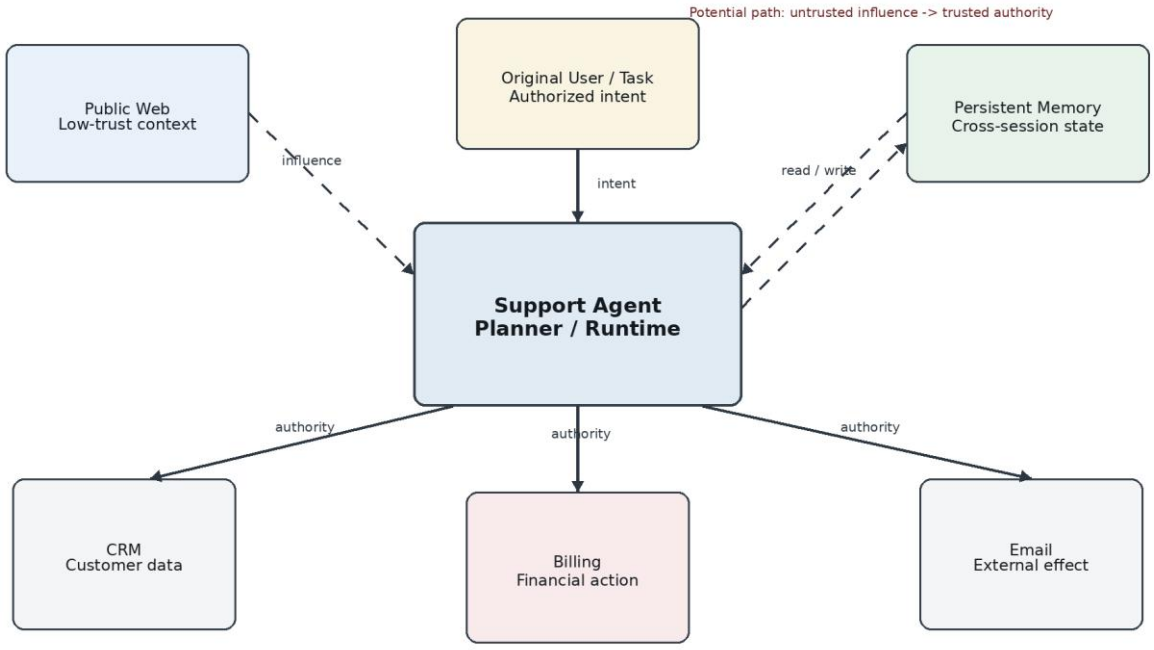

Figure 3. Simplified authority graph for the enterprise support-agent case study. Dashed relationships denote influence/state relationships; solid paths denote consequential authority.

10. Comparative Threat Visibility

Table 6

Threat scenario	Conventional analysis	AATM addition
Indirect prompt injection	Entry and data path visible	Explicit influence-to-authority path
Excessive agency	Credential visible	Task-authority mismatch
Generic tool misuse	Interface visible	Tool authority surface
Memory poisoning	Data store visible	Provenance and temporal influence
Delegation amplification	Service calls visible	Transitive/effective authority
Unsafe composition	Individual calls visible	Sequence-level authorization

This comparison should not be interpreted as evidence that STRIDE or conventional threat modeling cannot discover these risks. A skilled analyst can identify them. The narrower claim is that AATM makes the relevant questions first-class artifacts of an agent review, reducing dependence on unstated analyst intuition.

11. Runtime Security Architecture

Because the complete action sequence may not be known before execution, selected decisions should be mediated immediately before consequential operations. A runtime policy decision can consider the principal, task intent, proposed action, target resource, context provenance, execution history, approval state, and remaining authority budget.

Decision = f(Principal, Intent, Action, Resource, ContextProvenance, ExecutionHistory, Budget)

The architectural separation is deliberate: the model proposes an operation; an independent policy enforcement point determines whether that operation is allowed. This design direction is consistent with the broader lesson from capability-based and control/data-flow defenses such as CaMeL [7].

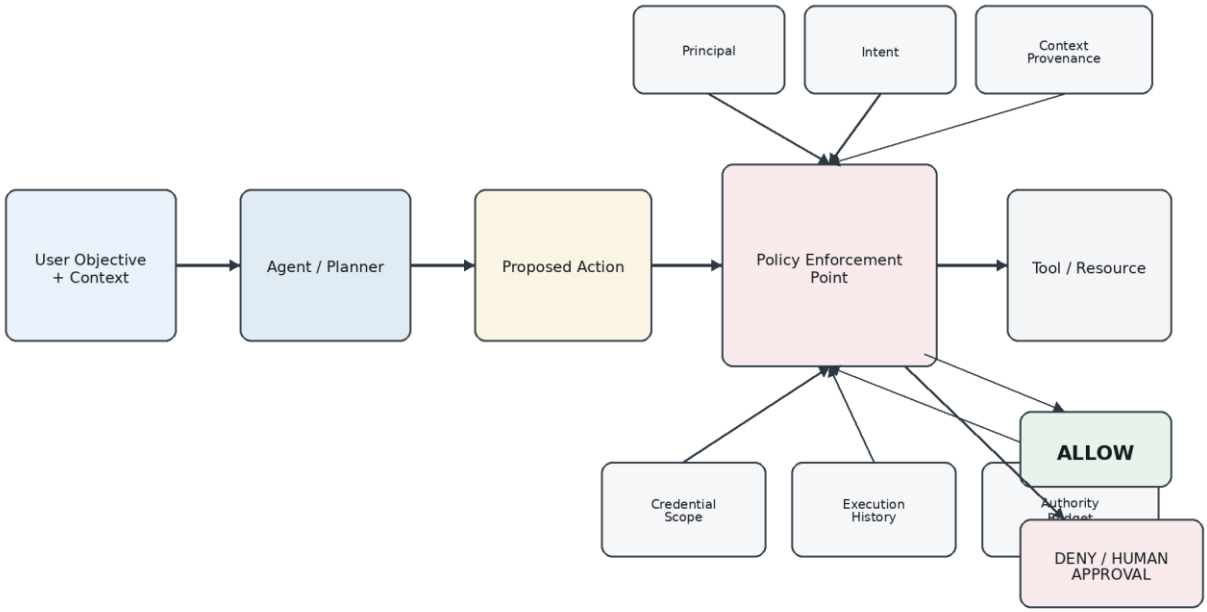

Figure 4. Runtime policy-enforcement architecture. The agent proposes an action; an independent policy enforcement point evaluates task and runtime context before execution.

12. Task-Scoped Authority and Authority Budgets

Agent authority should not automatically equal user authority. Where feasible, TaskAuthority should be a subset of AgentAuthority, which in turn should not exceed UserAuthority. Credentials can be short-lived, resource-specific, operation-specific, revocable, and auditable. This applies least privilege at the execution level [1][2].

$TaskAuthority \subseteq AgentAuthority \subseteq UserAuthority$

Authority can also be quantitative. A task may have limits on refund value, external messages, records retrieved, privileged tool calls, delegation depth, or execution duration. The security distinction is simple: CanPerform(X) does not imply CanPerformUnlimited(X). Budgets limit blast radius when individual actions are otherwise legitimate.

13. Security Observability and Human Oversight

Autonomous action requires enough audit context to reconstruct consequential execution. Useful fields include task_id, agent_id, originating_principal, declared_objective, tool, operation, resource, context_provenance, credential_scope, policy_decision, approval_reference, execution_result, and timestamp. This does not require storage of unrestricted private chain-of-thought; the objective is operational accountability.

Human approval is most useful at meaningful risk boundaries. Read-only status checks may remain autonomous, while external disclosure, financial transactions, production modification, permission changes, destructive operations, and security-sensitive memory writes may require explicit confirmation. Approval prompts should expose the actual action and data involved rather than asking users to approve a generic tool.

14. Discussion

The central observation from the scenario analysis is that agent risk is often created by the interaction of autonomy, authority, and untrusted influence. Autonomy without consequential authority has limited operational impact. Privileged deterministic software can be secured with established mechanisms because its control paths are more constrained. Untrusted information becomes especially consequential when it can alter a component that possesses real authority.

AATM therefore shifts part of the review from "Can this identity call this API?" to "Why is this action authorized for this task, given the context that produced it?" Authentication remains necessary, but it is insufficient as a statement about the legitimacy of a particular runtime action. A correctly authenticated agent can still be operating on poisoned context or a compromised memory record.

The framework also clarifies where prevention should occur. Prompt-injection detection is useful, but architecture should assume that some manipulations will bypass detection. Narrow tools, task-scoped credentials, policy enforcement, memory provenance, and sequence-aware controls reduce the consequence of that failure. This defense-in-depth interpretation is consistent with the direction of AgentDojo, CaMeL, OWASP agentic guidance, and MCP security guidance [6][7][8][11].

AATM is intentionally compatible with existing methods. STRIDE can still classify spoofing, tampering, repudiation, information disclosure, denial of service, and elevation of privilege. Attack trees can still model adversary goals. Data-flow diagrams still expose trust boundaries. AATM contributes an agent-specific overlay that asks what can influence decisions and what effective authority can be reached from those decisions.

15. Limitations

This study has six important limitations. First, the proposed risk score is a structured assessment aid, not a statistically validated predictor of incident probability. Second, the evaluation is scenario-based rather than a controlled benchmark comparison. Third, all scenario analysis and rubric coding in this paper were performed by a single analyst; inter-reviewer agreement has not yet been measured. Fourth, user intent is semantically ambiguous, so deterministic enforcement cannot always determine whether a proposed action faithfully reflects an open-ended objective. Fifth, AATM focuses on how model or context failures propagate into authority; it does not directly model every model-specific vulnerability such as adversarial examples, training-data poisoning, or fine-tuning attacks unless they affect an influence or authority path. Sixth, several recommended controls—task-scoped credentials, provenance infrastructure, and sequence-aware policy—may be difficult to retrofit into large legacy environments. Modeling and control depth should therefore be proportional to consequence.

16. Future Work

Future work should evaluate AATM against benchmark environments such as AgentDojo, measure inter-reviewer agreement for the Explicit/Derived/Absent rubric, and compare threat discovery with and without the overlay. A concrete next step is a multi-reviewer pilot in which the same implemented agent is analyzed using baseline DFD/STRIDE artifacts and AATM artifacts, measuring agreement, analysis time, and the number of security paths made explicit. Additional engineering directions include automated authority-graph generation from tool schemas and IAM policy, sequence-aware policy

engines, cryptographically verifiable task capabilities, provenance mechanisms for persistent memory, and runtime telemetry comparing granted authority with authority actually exercised.

17. Conclusion

Agentic AI changes the relationship between information and control. An external document, tool response, memory record, or delegated message may influence a runtime decision that ultimately exercises trusted enterprise authority. Threat modeling for these systems should therefore examine not only where an attacker can enter, but what an influenced execution can cause afterward.

AATM organizes this analysis around intent, context, authority, tool capability, memory, delegation, and action composition. It adds authority surface, influence flow, effective authority, and sequence-aware analysis to established security practice. The core principle is that possession of valid authority is not equivalent to permission to exercise that authority under every runtime circumstance. As enterprise agents move from recommendation to execution, preserving the boundary between task intent and effective authority becomes a central security requirement.

Declarations

Funding: No external funding was received for this study.

Conflict of Interest: The author declares no conflict of interest.

Data Availability: No proprietary dataset was used. This paper presents a security framework and reproducible scenario-based analytical methodology based on publicly documented threat classes and published research.

Ethics Statement: This research did not involve human participants, personal data, or animal subjects.

Author Contributions: The author developed the framework, threat-modeling methodology, scenario analysis, security architecture, and manuscript.

References

1. J. H. Saltzer and M. D. Schroeder, “The Protection of Information in Computer Systems,” Proceedings of the IEEE, vol. 63, no. 9, pp. 1278–1308, 1975. https://doi.org/10.1109/PROC.1975.9939
2. S. Rose, O. Borchert, S. Mitchell, and S. Connelly, “Zero Trust Architecture,” NIST Special Publication 800-207, 2020. https://doi.org/10.6028/NIST.SP.800-207
3. E. Tabassi, “Artificial Intelligence Risk Management Framework (AI RMF 1.0),” NIST AI 100-1, 2023. https://doi.org/10.6028/NIST.AI.100-1
4. C. Autio et al., “Artificial Intelligence Risk Management Framework: Generative Artificial Intelligence Profile,” NIST AI 600-1, 2024. https://doi.org/10.6028/NIST.AI.600-1

5. Q. Zhan, Z. Liang, Z. Ying, and D. Kang, “InjecAgent: Benchmarking Indirect Prompt Injections in Tool-Integrated Large Language Model Agents,” Findings of ACL 2024, pp. 10471–10506. https://aclanthology.org/2024.findings-acl.624/
6. E. Debenedetti, J. Zhang, M. Balunovic, L. Beurer-Kellner, M. Fischer, and F. Tramer, “AgentDojo: A Dynamic Environment to Evaluate Prompt Injection Attacks and Defenses for LLM Agents,” 2024. https://arxiv.org/abs/2406.13352
7. E. Debenedetti et al., “Defeating Prompt Injections by Design,” 2025. https://arxiv.org/abs/2503.18813
8. OWASP GenAI Security Project, “OWASP Top 10 for Agentic Applications for 2026,” 2025. https://genai.owasp.org/resource/owasp-top-10-for-agentic-applications-for-2026/
9. MITRE, “ATLAS: Adversarial Threat Landscape for Artificial-Intelligence Systems.” https://atlas.mitre.org/
10. Model Context Protocol, “Authorization Specification.” https://modelcontextprotocol.io/specification/2025-06-18/basic/authorization
11. Model Context Protocol, “Security Best Practices.” https://modelcontextprotocol.io/docs/tutorials/security/security_best_practices
12. Model Context Protocol, “Understanding Authorization in MCP.” https://modelcontextprotocol.io/docs/tutorials/security/authorization.

Author Biography

Ravindra Annam is a Product and AI Security Architect with more than 20 years of cybersecurity experience, specializing in enterprise AI security, agentic AI security, threat modeling, application security, cloud security, and runtime controls. His professional work includes technical authorship, speaking, judging, and peer review across cybersecurity and AI-security communities.

Website: https://www.ravindraannam.com/

LinkedIn: https://www.linkedin.com/in/ravindra-annam/

ORCID: https://orcid.org/0009-0002-0986-1184