REVIEW ARTICLE

Beyond Guardrails: Runtime Enforcement for Secure AI Agents-A Systematic Review of Action Control, Isolation, and Policy Enforcement

Jimmy Ming-Tai Wu[1], **Lingping Kong**[2], **Matin Pirouz**[3] and **Mu-En Wu**[4,*]

[1] Department of Information Management, National Kaohsiung University of Science and Technology, Kaohsiung 80778, Taiwan
[2] Department of Computer Science, VSB-Technical University of Ostrava, Ostrava 708 00, Czech Republic
[3] College of Science and Mathematics, California State University, Fresno, CA 93740, United States
[4] Department of Information and Finance Management, National Taipei University of Technology, Taipei 10608, Taiwan

Abstract

Large language model (LLM) agents turn model-generated decisions into tool calls, API requests, code execution, data access, and transactions. When agent reasoning is manipulated or simply wrong, model-centric guardrails cannot by themselves stop the resulting action. We systematically reviewed runtime-enforcement mechanisms for autonomous and tool-using AI agents in literature published or publicly available from January 2023 to September 2026. Searches of Compendex and the Web of Science Core Collection, followed by deduplication, multi-stage screening, source-grounded eligibility assessment, and targeted snowballing, yielded 58 primary studies, which we coded by enforcement placement, security mechanism, and assurance type. Most studies enforce security at the tool/action boundary (33/58, 56.9%) or inside the agent runtime or harness (18/58, 31.0%), whereas only 7/58 (12.1%) work primarily at the OS/process/execution boundary. Authorization and privilege control (41/58, 70.7%) and action validation (34/58, 58.6%) are the most common mechanism families. Across very different mechanisms and assurance models, a common design emerges: the model proposes an action, and a separate enforcement point decides whether that action may take external effect. Durable authorization, multi-agent delegation, cross-layer enforcement, recovery from partially committed effects, and standardized evaluation remain open problems. These findings suggest that secure agent architectures should separate reasoning from execution authority and place consequential actions behind independently enforceable runtime boundaries.

Keywords: AI agents, runtime enforcement, agent security, execution-time authorization, access control, runtime verification.

Submitted: 16 August 2026
Revised: 16 September 2026
Accepted: 05 October 2026
Published: 17 October 2026

Vol. 2, **No.** 4, 2026.
10.62762/JRSC.2026.526903

***Corresponding author:**
✉ Mu-En Wu
mnasia1@gmail.com

Citation
Wu, J. M. T., Kong, L., Pirouz, M., & Wu, M. E. (2026). Beyond Guardrails: Runtime Enforcement for Secure AI Agents-A Systematic Review of Action Control, Isolation, and Policy Enforcement. *Journal of Reliable and Secure Computing*, 2(4), 236–262.

© 2026 by the Authors. Published by Institute of Central Computation and Knowledge. This is an open access article under the CC BY license (https://creativecommons.org/licenses/by/4.0/).

1 Introduction

Large language models (LLMs) are now routinely deployed as autonomous or semi-autonomous agents that plan, call tools, read external data, execute code, keep state, and talk to other agents. This raises the stakes of model failure. In a chat setting an unsafe answer usually stays on the screen; in an agentic system the same failure can become a shell command, a file modification, a database query, a privileged API call, a data leak, or a payment—consequences documented across attacks, red-teaming campaigns, and defense evaluations for LLM-agent deployments [1–3]. Agent security therefore depends on two things: what the model generates, and whether the surrounding system lets a generated action take effect.

Most LLM security research has concentrated on model-centric safeguards: alignment, input and output filtering, prompt-injection detection, learned guardrails, and red teaming [1, 2, 4]. These defenses are useful, yet they do not form a complete security boundary for agents that read untrusted content while holding tools, credentials, and persistent state. Indirect prompt injection, malicious tool responses, poisoned context, excessive privileges, and ordinary reasoning errors can all still end in unsafe actions [2, 3]. The systems question is what happens once the upstream safeguards have failed: *if an AI agent is treated as a potentially compromised principal, what prevents an unsafe proposal from becoming an unauthorized external effect?*

Runtime enforcement answers this question by placing controls on the path between model reasoning and consequential execution. Such controls may authorize or deny a proposed tool call, validate its parameters, isolate execution, track information flow, monitor multi-step behavior, revoke privileges, or support intervention and recovery (representative systems are reviewed in Section 2). The guarantees differ widely. A learned action guard, a deterministic policy engine, a formally specified information-flow mechanism, and a cryptographically protected authorization path do not offer the same kind of assurance.

Recent surveys map the wider agent-security landscape, covering attacks, defenses, red teaming, the use of agents in cybersecurity, and code-agent security [1–4]. They make clear that agent security reaches well beyond prompt-level safety. Mechanisms that directly mediate execution authority, however, are scattered across access control, runtime verification, information-flow security, sandboxing, formal methods, and systems research, and have not been compared systematically.

This review focuses on that execution boundary. We retain studies that describe a concrete execution-time control path able to block, constrain, validate, isolate, or otherwise intervene in an agent's externally effective actions. Runtime monitoring counts only when it feeds an enforcement decision or intervention. Monitoring-only approaches, offline attack evaluation, and model-only safeguards are treated as related work.

We organize the literature along three dimensions: *placement*, *mechanism*, and *assurance*. Placement records whether enforcement happens at the tool/action boundary, inside the agent runtime or harness, or at the OS/process/execution boundary. Mechanism covers authorization and privilege control, action validation, execution isolation, information-flow control (IFC), runtime monitoring, and intervention or recovery. Assurance distinguishes learned/probabilistic, deterministic policy/systems, formal/model-checked, and cryptographic/attested approaches. Keeping the three dimensions apart lets us compare systems that are often lumped together as "guardrails" even though they enforce different properties at different trust boundaries.

We also examine a broader architectural trend: the separation of reasoning from execution authority. In many of the reviewed systems the model proposes an action but does not decide on its own whether that action runs; authorization, validation, information-flow controls, or other mechanisms can step in before an external effect is committed [5–10]. The systematic synthesis lets us test how consistently this pattern appears, instead of assuming that stronger model-level guardrails will serve as the final security boundary.

We address the following research questions:

- **RQ1:** What security threats and failure modes motivate runtime enforcement in AI-agent systems?
- **RQ2:** What runtime mechanisms have been proposed to constrain, validate, isolate, monitor, and control AI-agent actions?
- **RQ3:** Where in the agent execution lifecycle are security controls enforced?
- **RQ4:** What security properties, assurance models, and evaluation methodologies are used to assess runtime enforcement mechanisms?

- **RQ5:** What limitations and open research challenges remain in building reliable runtime security for autonomous AI agents?

The main contributions of this paper are:

- a focused and reproducible scope for runtime enforcement in AI-agent security, centered on concrete execution-time security controls;
- a three-dimensional *Placement–Mechanism–Assurance* taxonomy for comparing heterogeneous runtime-enforcement approaches;
- a systematic synthesis of 58 primary studies, including a quantitative characterization of enforcement placement, mechanism families, and assurance types;
- an architectural synthesis showing a recurring separation between model reasoning and final execution authority; and
- an analysis of open problems in durable authorization, multi-agent delegation, recovery, cross-layer composition, and security–utility evaluation.

The remainder of this paper is organized as follows. Section 2 reviews related surveys and adjacent research. Section 3 describes the review methodology. Section 4 presents the study landscape and taxonomy. Section 5 discusses architectural implications and open challenges. Section 6 summarizes the limitations of the review, and Section 7 concludes the paper.

2 Related Work

Work relevant to runtime enforcement for AI agents spans LLM-agent security, the security of code and tool-using agents, classical runtime enforcement, and recent agent-specific execution control. These areas overlap but differ in emphasis. Agent-security surveys usually organize the field around attacks and defenses; this review instead asks which mechanisms decide whether a model-proposed action may affect the outside world.

2.1 Security of LLM Agents

Recent surveys treat agent security as a broader problem than conventional LLM safety. Ferrag et al. [1] examine attacks, red teaming, evaluation, and defenses ranging from prompt-level safeguards to runtime and multi-agent mechanisms, thereby establishing that the agent-security problem extends to execution time—yet the execution-control dimension is not compared systematically by placement, mechanism, or assurance type, which is the gap the present review addresses. Xu et al. [4] consider both the protection of agents from external threats and the deployment of agents as active participants in the cybersecurity lifecycle, covering offensive and defensive tool use that makes execution-boundary controls doubly important: an agent acting as a security tool may itself become an attack surface if its action authority is not independently mediated. Ling et al. [2] offer a systems-oriented synthesis of 247 studies organized around information flow, delegated authority, and persistent state, and stress trust boundaries, privilege control, provenance, and deployment-sensitive evaluation.

These surveys show how autonomy, persistent state, tool use, and exposure to untrusted environments enlarge the attack surface of LLM systems. Their scope, however, runs from prompt injection and red teaming to privacy, memory, multi-agent interaction, benchmarks, and cybersecurity applications, so runtime enforcement appears as one part of a wider defense landscape and is not compared systematically by enforcement placement, mechanism, and assurance type.

2.2 Security of Code and Tool-Using Agents

Code agents make the consequences of unsafe external action especially visible. Hu et al. [3] survey 82 studies and describe a central risk as the conversion of untrusted semantics into actionable capabilities such as shell execution, file manipulation, and tool invocation. They also identify attack chains that cross tools, memory, and software supply chains.

The same problem extends beyond code generation. Agents may act on databases, enterprise APIs, cloud resources, financial services, communication systems, physical devices, or other agents. The relevant boundary is therefore the general transition from model-generated intent to an externally effective operation, of which code execution is only one case. That broader boundary is the subject of this review.

2.3 Runtime Enforcement and Formal Safety

Runtime enforcement predates LLM agents. Könighofer et al. [11] survey correct-by-construction runtime enforcement for AI systems, including enforcers that constrain or modify actions to preserve formally specified safety properties during execution. That line of work provides the architectural

foundation for the separation of reasoning from execution authority that recurs throughout the present corpus: a learning-enabled component proposes actions, while a separately trusted enforcer—operating at what this review calls the tool/action boundary or OS/process boundary—decides whether those proposals may take effect.

LLM agents add complications. Their action spaces may be exposed dynamically through natural language, structured tool calls, APIs, code, and inter-agent messages, and security decisions may depend on provenance, persistent state, delegated authority, and user intent. Adversarial manipulation of the reasoning context, including indirect prompt injection and malicious tool content, makes enforcement harder still. Recent agent-security systems accordingly combine classical runtime enforcement with access control, information-flow security, sandboxing, cryptographic authorization, semantic validation, and learned risk assessment.

2.4 Emerging Runtime Enforcement for AI Agents

Recent systems place explicit controls between agent reasoning and external execution. AgentSpec provides programmable runtime constraints [6]; Open Agent Passport (OAP) performs deterministic authorization before tool execution [5]; and SEAgent applies mandatory access control to agent–tool interactions [7]. Other approaches enforce information-flow properties [8], mediate semantic flows before privileged sinks [9], or couple learned safety judgments to pre-execution action control [10]. Proactive monitors such as ProbGuard predict the likelihood of reaching an unsafe state and intervene before a violation occurs [12].

These systems expose a distinction that a single "runtime defense" category cannot capture. Controls at similar points in the execution path may implement authorization, validation, information-flow restrictions, or learned risk assessment, and may rest on very different kinds of assurance. Conversely, the same security objective can be enforced at the tool boundary, inside an agent runtime, or through lower-level execution isolation.

Existing surveys give broad accounts of agent threats and defenses [1–4], while earlier runtime-enforcement research lays the foundations for constraining learning-enabled systems during execution [11]. The agent-specific literature sits between the two, and its execution-control mechanisms remain split across different security and systems traditions. A unified comparison of where enforcement occurs, what function it performs, and what basis of assurance supports the decision is still missing.

The present review fills this gap using the execution-time control path as the unit of analysis. We compare agent-specific runtime mechanisms by *placement*, *mechanism*, and *assurance*, covering approaches from tool-level authorization and action validation to information-flow control, isolation, monitoring, intervention, and cryptographically mediated execution. The aim is to complement existing attack-oriented taxonomies by characterizing how agent systems constrain the step from a model-generated proposal to an externally effective action.

3 Methodology

We conducted a systematic review of runtime enforcement for autonomous and tool-using AI agents. The review covers literature published or publicly available from January 2023 to September 9, 2026, and focuses on security mechanisms that operate during execution and can affect whether a model-proposed action is allowed to produce an external effect. Because runtime enforcement for AI agents is a fast-moving field in which relevant systems appear across security, software engineering, and formal methods venues under inconsistent terminology, rigorous and reproducible search and selection procedures are essential to avoid systematic bias toward any single research community. The review design therefore draws on established guidance for systematic literature reviews and snowballing in software engineering [13, 14], and the identification, screening, and inclusion process is reported in the form of a PRISMA 2020 flow diagram so that the scope and coverage of the enforcement-mechanism corpus can be independently assessed [15].

3.1 Research Scope and Eligibility Criteria

The unit of analysis is an *execution-time security control*. We define runtime enforcement as a mechanism on the execution path of an AI agent that can block, constrain, validate, isolate, or otherwise intervene in an externally effective action.

A study was included in the primary corpus when it:

- studied an autonomous, agentic, or tool-using AI/LLM system;
- proposed, implemented, evaluated, or formally analyzed a security mechanism operating during

agent execution;

- connected that mechanism to a concrete control path capable of affecting an externally effective action; and
- provided enough technical detail to determine enforcement placement, security function, and assurance characteristics.

Both peer-reviewed publications and technically substantive preprints were eligible, because much of this fast-moving literature had not completed the publication cycle by the search date. Publication status and evidence maturity were recorded separately. Inclusion depended on the execution-time control mechanism described in each study, not on demonstrated attack-blocking effectiveness; implementation and evaluation evidence were assessed separately.

We excluded work limited to model-level alignment, prompt engineering, jailbreak or prompt-injection detection without execution-time control, generic identity or authentication without action-level enforcement, attack characterization, benchmarks, offline trace analysis, or monitoring unable to affect execution. Surveys, position papers, governance frameworks, and architectural proposals without a sufficiently concrete enforcement path were treated as background or related work. Mechanisms enforcing only non-security properties, such as user preferences, were also excluded. Monitoring was eligible only when it was connected to an execution-time enforcement decision or intervention.

The operational inclusion rule was:

> A study was retained in the primary corpus only if it described a concrete execution-time control path capable of blocking, constraining, validating, isolating, or otherwise intervening in an AI agent's externally effective actions.

3.2 Information Sources and Search Strategy

We searched Compendex through Engineering Village and the Web of Science Core Collection (WoS) on September 9, 2026. The search period was January 2023 through the search date.

For Compendex, three concept groups were combined with the Boolean operator AND. The first identified autonomous and tool-using AI agents:

> "AI agent*" OR "LLM agent*" OR "language model agent*" OR "agentic AI" OR "tool-using agent*" OR "tool-using LLM*" OR "tool-calling agent*"

The second captured runtime-security and enforcement mechanisms:

> "runtime security" OR "runtime protection" OR "runtime enforcement" OR "policy enforcement" OR "policy engine*" OR "access control" OR authoriz* OR "least privilege" OR "privilege control" OR "capability control" OR sandbox* OR containment OR "execution isolation" OR "runtime isolation" OR "information flow control" OR "information-flow control" OR "action validation" OR "action validator*" OR "tool-call validation" OR "tool call validation"

The third restricted results to execution-related contexts:

> runtime OR execution OR "tool call*" OR "tool use" OR "tool invocation*" OR "API call*" OR "agent action*" OR "pre-action" OR "post-action"

The Compendex result page reported 669 records, of which 660 were retrieved in the detailed export used for screening. We report this nine-record gap explicitly instead of assuming that the missing records were screened.

For the WoS Core Collection we used a broader two-group Topic search. The first group used the same agent-related terms; the second combined security and runtime-enforcement concepts:

> secur* OR "runtime security" OR "runtime protection" OR "runtime enforcement" OR "policy enforcement" OR "access control" OR authoriz* OR "least privilege" OR "privilege control" OR "capability control" OR sandbox* OR containment OR "execution isolation" OR "runtime isolation" OR "information flow control" OR "action validation"

The WoS search returned 291 records. Three of them carried a 2027 issue year although they were publicly searchable by September 9, 2026; eligibility was therefore judged on public availability by the search date, not on issue-year metadata.

In total, the database result pages identified 960 records. Because nine Compendex records were unavailable in the detailed export, 951 records were imported for processing.

3.3 Deduplication and Multi-Stage Screening

Records from Compendex and WoS were normalized and deduplicated before screening. Duplicates were detected by normalized DOI matching, exact normalized-title matching, and high-similarity title matching when no reliable DOI correspondence was available. For duplicate records of the same study, we kept the more complete bibliographic record and preferred the peer-reviewed version over a corresponding preprint.

Of the 951 imported records, 51 duplicates were consolidated, leaving 900 unique records for title-and-abstract screening.

Screening proceeded in several stages. An initial high-recall pass identified studies potentially related to runtime agent security; progressively stricter title-and-abstract screening then separated concrete enforcement mechanisms from attack studies, application papers, surveys, generic security architectures, and work without an execution-time control path.

Because relevant enforcement systems did not always use consistent runtime-security terminology in their abstracts, we also ran a high-recall rescue audit of lower-priority candidates. This step recovered additional studies on authorization, isolation, information-flow control, and execution control. It was part of the screening workflow and did not constitute an additional database search.

Automated assistance was used to support record organization, evidence extraction, and consistency checking during screening. Ambiguous cases were advanced to source-grounded assessment, and eligibility decisions and extracted quantitative claims were checked against the primary sources.

After multi-stage screening, 74 database-derived studies advanced to source-grounded eligibility assessment. The remaining 826 records were not advanced because, on the available screening evidence, they did not appear to meet the operational inclusion criterion or belonged to an excluded study category.

3.4 Source-Grounded Eligibility Assessment

Eligibility of the 74 database-derived studies was assessed against the available primary sources, focusing on whether a concrete execution-time control path was present and whether there was enough technical evidence for coding.

The initial assessment retained 64 database-derived studies. A subsequent reconciliation of the execution-time enforcement criterion against the sources excluded nine of these, leaving 55 database-derived studies in the final corpus. In total, 19 of the 74 candidates were not retained. Reasons included monitoring or auditing without in-line control, architecture or governance proposals without a sufficiently concrete enforcement path, theoretical analyses of enforcement limitations, preliminary conceptual or formal precursors, interaction sandboxes not primarily designed for security enforcement, and runtime enforcement of non-security properties.

Targeted backward and forward snowballing [14] from studies that received a primary placement or assurance classification in the initial corpus—that is, studies that clearly met the operational inclusion criterion—identified four additional candidates, which were assessed against the same criteria. After reconciliation, three were retained; one was excluded because the available evidence did not establish a concrete execution-time enforcement path.

The final primary corpus therefore comprised 58 studies: 55 from database screening and three from targeted snowballing. Figure 1 summarizes the identification, screening, eligibility, and inclusion process.

3.5 Data Extraction and Taxonomy Development

For each retained study we extracted bibliographic information, publication status, enforcement placement, security mechanisms, assurance type, threat or failure mode, evaluation methodology, principal security results, reported systems overhead, and evidence maturity.

Coding used three dimensions: *placement*, *mechanism*, and *assurance*. Placement has three categories: (1) tool/action boundary, (2) agent runtime/harness, and (3) OS/process/execution boundary. Each study was assigned one primary placement according to where its principal enforcement decision is implemented.

Mechanism captures the security functions a system performs. Six families were used: (1) authorization and privilege control, (2) action validation, (3) execution isolation, (4) information-flow control, (5) runtime monitoring, and (6) intervention and recovery. Mechanism coding was multi-label because systems frequently combine several forms of enforcement.

Assurance describes the primary basis for trusting an enforcement decision or security

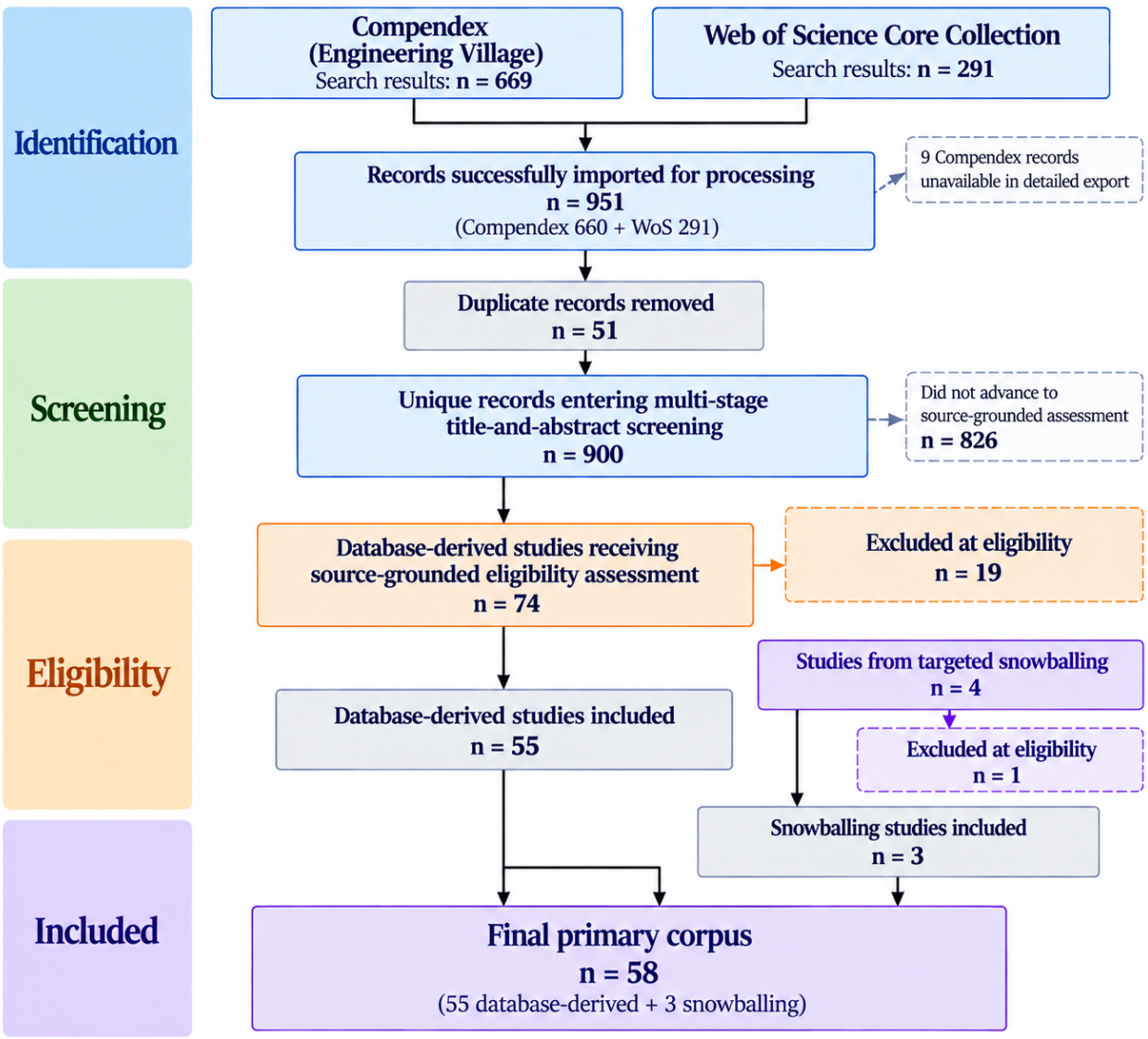


Figure 1. Study identification, screening, and inclusion flow. The database searches identified 960 records (669 from Compendex and 291 from Web of Science). Of these, 951 records were retrieved for processing because nine Compendex records reported by the search interface were unavailable in the detailed export. After consolidation of 51 duplicate records, 900 unique records underwent multi-stage title-and-abstract screening. Seventy-four database-derived studies advanced to source-grounded eligibility assessment, of which 55 were retained and 19 were excluded following eligibility reconciliation. Targeted backward and forward snowballing identified four additional candidates, of which three were retained and one was excluded. The final primary corpus comprised 58 studies.

property. Four categories were used: (1) learned/probabilistic, (2) deterministic policy/systems, (3) formal/model-checked, and (4) cryptographic/attested. Each study received one primary assurance category for aggregate analysis, even when secondary assurance mechanisms were present.

The taxonomy was refined iteratively during source-grounded assessment. A new top-level category was introduced only when the existing categories could not coherently represent several independent studies. Later screening and rescue rounds mainly populated existing categories and did not require new mechanism families.

3.6 Evidence Synthesis

The retained studies were synthesized through descriptive quantitative analysis and structured qualitative comparison. We report frequencies and proportions across enforcement placements, mechanism families, assurance types, publication years, and selected evaluation characteristics. Because mechanism coding is multi-label, these percentages are read independently and do not sum to 100%.

We did not conduct a statistical meta-analysis. The studies differ substantially in threat models, agent architectures, benchmarks, attack definitions, security metrics, utility measures, and systems-performance measurements. Attack success rate, unsafe-action blocking, task-completion rate, authorization latency, information-flow guarantees, replay resistance, and formally verified properties are different outcomes and cannot be pooled into a common effect size.

Quantitative results are used to characterize the structure of the literature, while individual security and performance findings are interpreted within their mechanism and assurance contexts. Cross-study qualitative synthesis identifies recurring architectural patterns, limitations, and open research challenges.

4 Results

4.1 Study Landscape

The final primary corpus comprises 58 studies: 55 identified through database screening and three through targeted backward or forward snowballing. Publication year was coded by the first public availability of each study, not by the year of a later revised or formally published version. On this basis, four studies (6.9%) first appeared in 2025 and 54 (93.1%) in 2026. Runtime enforcement for LLM-based agents has thus emerged as a distinct research topic only recently, even though it builds on reference monitors, access control, information-flow security, sandboxing, and runtime verification.

The studies span security and privacy, software engineering, formal methods, operating systems, natural language processing, and AI safety, and their terminology varies considerably. Related mechanisms are described as *runtime enforcement*, *policy enforcement*, *pre-action authorization*, *guardrails*, *runtime verification*, *mandatory access control*, *information-flow control*, *sandboxing*, and *safety kernels*. Similar labels can hide different security properties. AgentSpec expresses programmable runtime constraints through triggers, predicates, and enforcement actions [6]; OAP performs deterministic policy-based authorization before individual tool calls [5]; and StepGuard also operates before execution but relies on a learned step-level guard model [10]. Such differences are why we separate enforcement placement and mechanism from the assurance basis of the decision.

Figure 2 summarizes the distribution of the corpus across enforcement placement, mechanism families, and assurance types.

4.2 Threat and Failure-Mode Landscape

RQ1 asks which threats and failure modes motivate runtime enforcement in AI-agent systems. Across the corpus, the motivation is a recurring systems condition more than any single attack class: model-generated reasoning can be turned into privileged external effects after being influenced by untrusted information, excessive authority, wrong assumptions, or adversarial manipulation.

We identified ten recurring threat and failure-mode families:

1. **Indirect prompt injection and untrusted tool content.** Adversarial instructions embedded in webpages, documents, tool responses, retrieved data, or inter-agent messages can steer subsequent tool use. The security question is whether compromised reasoning can exercise consequential authority [7, 9, 16].

2. **Confused-deputy behavior and excessive privilege.** Agents may hold credentials or capabilities broader than the current task needs, so legitimate authority can be redirected toward an illegitimate purpose. Pre-action authorization and mandatory access control target this problem directly [5, 7].

3. **Capability or policy drift.** Permissions, tools, environmental state, or task context may change during long-running execution, so previously valid authority becomes stale or overly permissive. Stateful authorization mechanisms bind authority to the evolving task or capability state [17].

4. **Unsafe concrete actions and parameter manipulation.** A permitted tool may still be invoked with unsafe arguments, destinations, commands, amounts, or file paths. This calls for checks on the concrete proposed action, not only on coarse-grained tool permissions [6, 18].

5. **Sensitive-data leakage and cross-context contamination.** Agents combine information across tools, conversations, memories, and persistent stores, so confidential data can propagate into unauthorized outputs or sinks. Information-flow mechanisms track provenance or security labels and constrain later use or disclosure [8, 9].

6. **Malicious or vulnerable code and tool execution.** Generated code, shell commands, scripts, or third-party tools may behave unsafely.

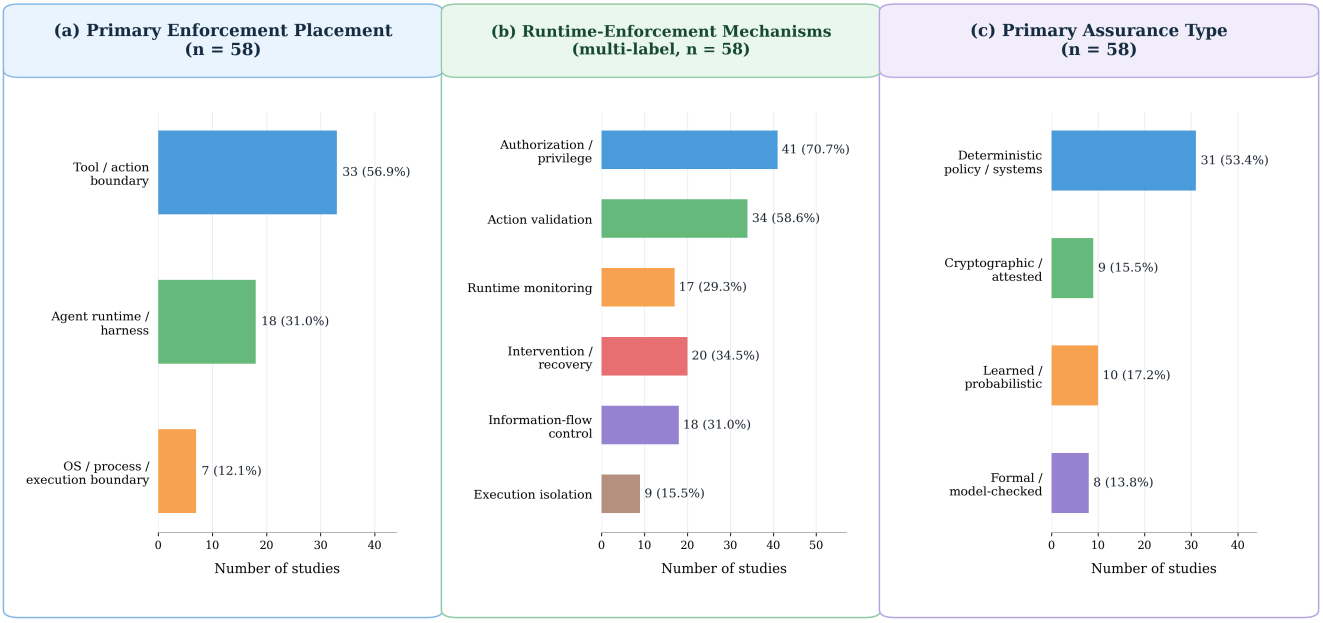


Figure 2. Distribution of the 58 primary studies across the three dimensions of the proposed taxonomy: (a) primary enforcement placement, (b) runtime-enforcement mechanism families, and (c) primary assurance type. Mechanism coding is multi-label; therefore, counts in panel (b) exceed 58. Frequencies indicate research prevalence rather than comparative security effectiveness.

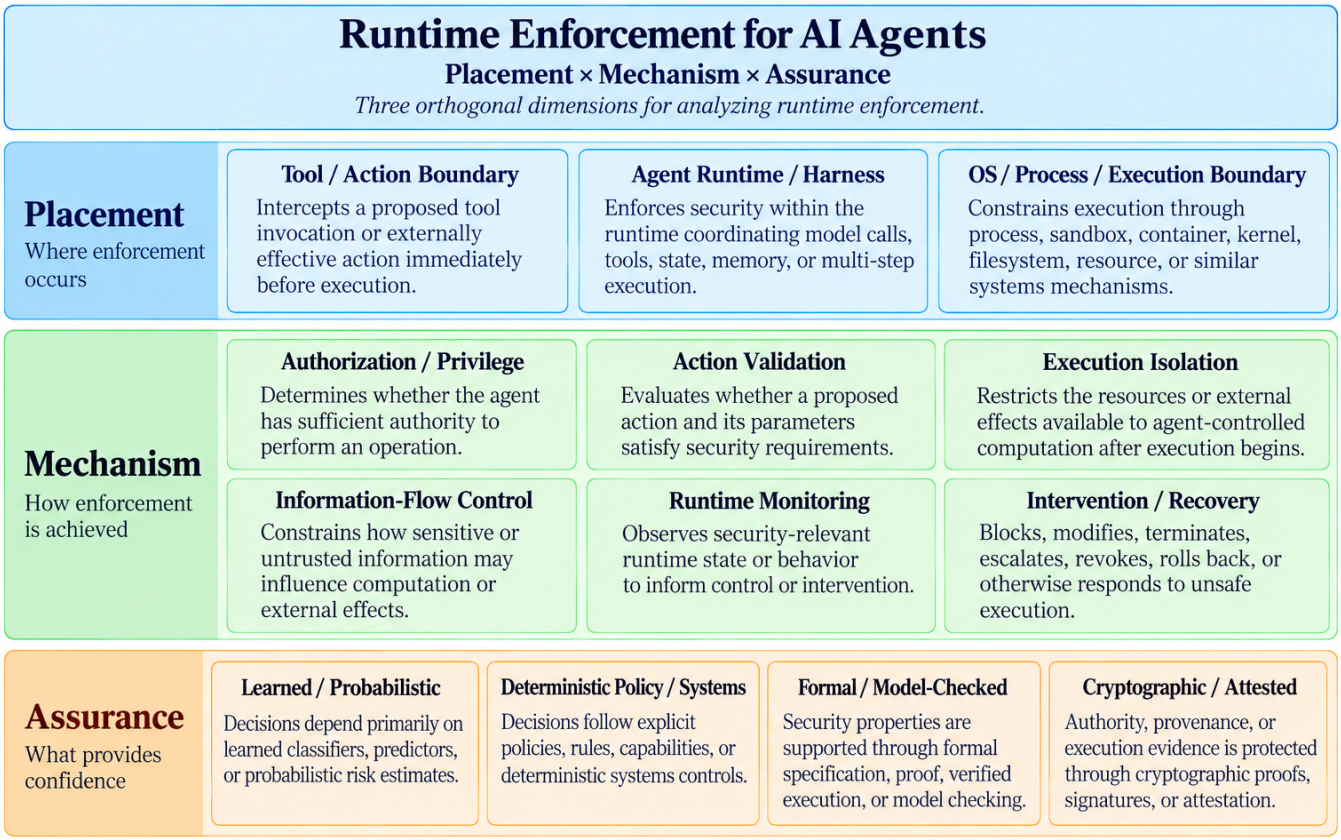


Figure 3. Placement–Mechanism–Assurance taxonomy for runtime enforcement in secure AI-agent systems. Placement identifies where enforcement occurs, mechanism describes the security function performed, and assurance captures the primary basis for trusting the enforcement outcome. The dimensions are complementary: similar mechanisms may operate at different layers and rely on different forms of assurance.

Isolation and sandboxing limit the blast radius when upstream reasoning or validation fails [19, 20].

7. **Long-horizon behavioral drift.** Individually plausible actions may add up to an unsafe trajectory. Step-level approaches such as StepGuard bring trajectory context into pre-execution decisions instead of judging each action in isolation [10].

8. **Replay, retry, concurrency, and crash-recovery hazards.** An authorization valid for one execution may become unsafe if replayed, retried after partial failure, or consumed more than once under concurrency. Recent systems bind authorization to durable execution state and model replay and recovery semantics explicitly [17, 21].

9. **Multi-agent delegation and authorization propagation.** Delegated authority may expand as it passes through an agent graph unless scope attenuation and lineage are enforced. Recent approaches cryptographically bind downstream authority to delegation history and constrained scope [22].

10. **Monitor, evidence, or enforcement-path bypass.** Runtime controls can themselves be attacked through policy tampering, evidence suppression, or execution paths that skip the enforcement point. Some architectures isolate enforcement and evidence generation from model-controlled execution and protect them with authenticated or verifiable records [23–25].

Although these threats differ in origin, they lead to the same architectural requirement: *the component that generates an action should not automatically be the component that authorizes its effects*. Runtime enforcement inserts an independent control point between proposal and execution. OAP does so through deterministic authorization before tool execution [5], while LLMbda constrains how information may influence security-sensitive computation even when adversarial inputs reach the agent [8].

4.3 Taxonomy of Runtime Enforcement

RQ2–RQ4 concern what is enforced, where enforcement occurs, and what assurance supports the resulting security decision. We classify each primary study along three complementary dimensions, *placement*, *mechanism*, and *assurance*, as summarized in Figure 3 and Table 2.

Placement identifies the principal architectural boundary at which enforcement occurs: the *tool/action boundary*, the *agent runtime/harness*, or the *OS/process/execution boundary*. Mechanism describes the security function and is coded with six multi-label families: *authorization and privilege control*, *action validation*, *execution isolation*, *information-flow control*, *runtime monitoring*, and *intervention/recovery*. Assurance identifies the primary basis for trusting the enforcement outcome and is classified as *learned/probabilistic*, *deterministic policy/systems*, *formal/model-checked*, or *cryptographic/attested*.

The dimensions expose distinctions that a single "runtime defense" category would blur. StepGuard and OAP both mediate actions before execution, but StepGuard relies mainly on learned safety classification [10], whereas OAP uses deterministic declarative authorization [5]. AgentSpec and LLMbda both operate largely inside an agent runtime or harness, yet AgentSpec emphasizes programmable runtime rules [6] and LLMbda provides information-flow enforcement backed by formal reasoning [8]. Similar placements can thus support different mechanisms and assurance models, and similar security objectives can be enforced at different layers. Table 1 gives representative examples of how individual systems map onto the dimensions and which security properties they enforce.

4.4 Where Runtime Enforcement Occurs

Across the 58 primary studies, the *tool/action boundary* is the most common primary placement, with 33 studies (56.9%). The *agent runtime/harness* accounts for 18 studies (31.0%), and 7 studies (12.1%) operate primarily at the *OS/process/execution boundary*. In other words, 87.9% of the corpus places its main enforcement logic either around externally effective actions or within the runtime that coordinates agent execution (Figure 2).

4.4.1 Tool/Action Boundary

Tool-level interposition puts a control point between model reasoning and an externally visible effect. OAP synchronously intercepts proposed tool calls and evaluates them against a declarative authorization policy before execution [5]. Under its restrictive policy, the authors report no successful attacker-triggered execution across 879 test attempts under that policy configuration, even though the model itself had been successfully influenced under more permissive settings [5].

Table 1. Representative runtime-enforcement approaches for secure AI agents.

Study	Placement	Mechanism(s)	Assurance	Representative Security Property
AgentSpec [6]	Agent runtime / harness	Validation; intervention	Deterministic	Runtime predicates constrain unsafe actions.
OAP [5]	Tool / action boundary	Authorization; validation	Deterministic	Pre-action authorization mediates model-proposed tool calls.
SEAgent [7]	Tool / action boundary	Authorization; IFC	Deterministic	Mandatory access control constrains agent–tool privileges.
LLMbda [8]	Agent runtime / harness	Information-flow control	Formal	Dynamic IFC constrains information propagation with formal guarantees.
Token-Flow Firewall [9]	Agent runtime / harness	IFC; validation; monitoring	Learned / probabilistic	Semantic flow analysis identifies risky information propagation before privileged operations.
StepGuard [10]	Tool / action boundary	Validation	Learned / probabilistic	Learned step-level guards screen proposed actions before execution.
MCP-Secure [26]	Agent runtime / harness	Authorization; intervention	Deterministic	Runtime access control mediates Model Context Protocol (MCP) tool privileges and restricts unauthorized operations.
Agent libOS [19]	Agent runtime / harness	Authorization; IFC; isolation; intervention	Deterministic	Capability-controlled execution constrains operations and information flows.
Safety Kernel [25]	OS / process / execution boundary	Authorization; isolation; intervention	Formal	A trusted safety kernel constrains agent-controlled execution and intervenes in unsafe operations.
CapLease [17]	Tool / action boundary	Authorization; intervention	Deterministic	Durable authorization state prevents unsafe reuse across retries and failures.
Lineage [22]	Tool / action boundary	Authorization	Cryptographic	Signed delegation ancestry constrains downstream authority propagation.
DTF [23]	Tool / action boundary	Authorization; validation; monitoring	Cryptographic	Proof-derived authorization binds actions to verifiable execution evidence.

The same placement can host quite different decision procedures. StepGuard evaluates proposed tool actions with a learned guard model and reports a 77.3% relative reduction in mean attack success rate at the cost of a 2.8-percentage-point drop in mean utility [10]. OAP and StepGuard intercept at much the same point, yet their assurance bases are very different.

The size of this category reflects the practical value of checking a concrete proposal immediately before it becomes an external effect.

4.4.2 *Agent Runtime/Harness*

Eighteen studies (31.0%) place their principal enforcement logic inside the agent runtime or harness. This placement suits controls that depend on state beyond a single invocation, such as prior actions, provenance, information labels, task state, persistent authorization, and multi-step constraints.

AgentSpec integrates programmable runtime policies through triggers, predicates, and enforcement strategies. The authors report prevention of unsafe execution in more than 90% of the evaluated code-agent cases, elimination of hazardous actions across the embodied-agent tasks, and full compliance in the autonomous-driving scenarios, with millisecond-scale overhead [6].

LLMbda takes a different route, building dynamic information-flow control into an executable framework for agent conversations and tool use. Its Lean-mechanized formal development establishes termination-insensitive probabilistic noninterference, while the harness propagates security labels through agent computation [8]. Stateful authorization mechanisms such as CapLease also maintain cross-step consistency, although their principal decisions are made at the tool/action boundary.

4.4.3 *OS/Process/Execution Boundary*

Seven studies (12.1%) enforce security mainly below the agent framework, through OS, process, container, sandbox, or comparable execution mechanisms.

Table 2. Operational definitions and corpus frequencies of the taxonomy categories ($N = 58$).

Dimension	Category	Operational Definition	Studies	%
Placement	Tool / action boundary	Intercepts proposed tool calls or externally effective actions immediately before execution.	33	56.9
	Agent runtime / harness	Enforces security within the runtime coordinating model calls, tools, state, memory, or multi-step execution.	18	31.0
	OS / process / execution boundary	Constrains execution through process, sandbox, container, kernel, filesystem, resource, or related systems mechanisms.	7	12.1
Mechanism	Authorization / privilege	Determines whether the agent is authorized to perform a proposed operation under the current task and security context.	41	70.7
	Action validation	Checks whether a proposed action, its parameters, and context satisfy security requirements.	34	58.6
	Execution isolation	Restricts resources or external effects available to agent-controlled computation.	9	15.5
	Information-flow control	Restricts how sensitive, untrusted, or labeled information may influence computation or external effects.	18	31.0
	Runtime monitoring	Observes security-relevant runtime state or behavior that informs execution-time control or intervention.	17	29.3
	Intervention / recovery	Blocks, modifies, terminates, escalates, revokes, rolls back, or otherwise responds to unsafe execution.	20	34.5
Assurance	Learned / probabilistic	Relies primarily on learned classifiers, predictors, or probabilistic risk estimates.	10	17.2
	Deterministic policy / systems	Relies primarily on explicit policies, rules, capabilities, reference-monitor logic, or deterministic systems controls.	31	53.4
	Formal / model-checked	Relies primarily on formal specification, proof, verified execution, or model checking.	8	13.8
	Cryptographic / attested	Protects authority, provenance, or execution evidence using proofs, signatures, authenticated records, or attestation.	9	15.5

Note: Placement and assurance use one primary category per study, whereas mechanism coding is multi-label. Mechanism counts and percentages therefore do not sum to 58 and 100%, respectively. Frequencies indicate prevalence rather than comparative effectiveness.

They address a different problem from semantic authorization: limiting what executing code or tools can actually do once execution has begun.

Examples include Sandlock, which confines agent-controlled code using unprivileged Linux primitives [20], and the on-demand behavioral sandbox approach [27], which isolates high-risk agent actions during execution. These mechanisms restrict what agent-controlled computation can reach even if higher-level authorization or validation is bypassed. Agent libOS [19] is classified primarily under the agent runtime/harness but also uses lower-level isolation, showing how enforcement can span several layers.

Although this is the smallest category, it supplies something that tool-level authorization cannot: semantic controls decide whether an action *should* execute, whereas systems isolation constrains what executing components *can* do.

Taken together, the placement results point to three complementary roles: tool/action controls mediate the step from proposal to effect, runtime/harness controls maintain security state across execution, and OS/process controls bound concrete execution capabilities.

4.5 What Is Enforced: Mechanism Families

Authorization and privilege control is the most common mechanism family, appearing in 41 of the 58 studies (70.7%), followed by action validation in 34 (58.6%). Intervention/recovery appears in 20 studies (34.5%), information-flow control in 18 (31.0%), runtime monitoring in 17 (29.3%), and execution isolation in 9 (15.5%). Because coding is multi-label, the percentages do not sum to 100%. As Figure 2 and Table 2 show, few systems rely on a single mechanism; many combine authority checks with

validation, information-flow restrictions, containment, monitoring, or intervention.

4.5.1 Authorization and Privilege Control

Authorization and privilege control appears in 41 of 58 studies (70.7%). The reviewed systems increasingly reject the assumption that holding a credential or tool capability is enough to justify an action. Authority is instead conditioned on task context, concrete parameters, provenance, delegation state, risk, or execution history.

OAP treats model-generated tool calls as authorization requests and returns an explicit ALLOW, DENY, or ESCALATE decision before execution [5]. SEAgent similarly uses mandatory access control so that agents cannot exercise privileges merely because a tool is available [7]. Other systems tie authority more closely to task intent, parameters, provenance, delegation ancestry, execution budgets, or durable workflow state [17, 22, 28–30].

The trend is clearest under retries, concurrency, and crash recovery, where reusable authorization artifacts can become unsafe. CapLease makes authorization state explicit across replay, retry, concurrency, and recovery [17], while runtime verification for the Agent Payments Protocol (AP2) and SessionBound bind approval to transaction or workflow state [21, 30]. Authorization in agent systems is moving away from long-lived credential possession toward execution-specific authority tied to the current task and state.

4.5.2 Action Validation

Action validation appears in 34 studies (58.6%). It differs from authorization by asking whether the *specific proposed action*, including its parameters and context, is acceptable.

The distinction matters when a tool is legitimately available but a particular invocation is unsafe. AgentSpec evaluates runtime predicates at action-relevant enforcement points [6], while StepGuard uses a learned model to judge proposed actions before execution [10]. Both validate actions, but on different assurance bases.

More context-sensitive approaches relate the proposed action to user intent, prior actions, provenance, trusted state, or downstream consequences. Within the corpus, intent-governed tool authorization and context-to-execution integrity checks compare a proposed call against the user's task and the provenance of the context that produced it [31, 32]. A related system, Explanation-Bound Tool Execution (EBTE), goes further and treats the model's own explanation as untrusted, verifying its security-relevant claims against trusted server-side state before execution [33]; it was identified after the search window closed and is therefore cited as related work rather than as a primary study. Neither approach uses the model's rationale as the sole basis for deciding whether its action is safe.

4.5.3 Information-Flow Control

Information-flow control appears in 18 studies (31.0%). It covers cases where an individually permitted action may still disclose or propagate information in an unauthorized way.

LLMbda applies dynamic IFC to agent computation and establishes a noninterference property [8]. Token-Flow Firewall mediates semantic flows before they reach privileged sinks [9], and other systems use labels or provenance to keep sensitive information away from disallowed destinations [34].

IFC is particularly relevant to indirect prompt injection and cross-context leakage. An agent may be separately authorized to read sensitive information and to communicate externally, yet the combination can open an unintended disclosure channel. IFC thus extends runtime enforcement from individual actions to the question of how information obtained in one security context may influence another.

4.5.4 Execution Isolation

Execution isolation appears in 9 studies (15.5%). Authorization and validation decide whether execution should proceed; isolation constrains what executing code or tools can affect once it has started.

This matters for generated code, shell commands, dynamically acquired tools, and external components whose behavior is hard to predict in advance. Sandboxes, containers, process isolation, filesystem restrictions, resource controls, and kernel-mediated mechanisms bound the effects of malicious or erroneous computation [19, 20, 27].

Isolation complements semantic enforcement. A validator may reject an unsafe command, while a sandbox limits the damage from a permitted command or dependency that misbehaves. Neither layer is a complete defense on its own.

4.5.5 Runtime Monitoring

Runtime monitoring appears in 17 studies (29.3%). We counted monitoring only when the observation had execution-time security significance; logging or auditing that could not affect execution did not qualify as an enforcement mechanism.

Monitors may track action sequences, provenance, policy state, behavioral trajectories, information flow, or deviations from expected execution. Unlike action validation, which evaluates a single proposed operation, monitoring observes security-relevant behavior or state across execution, and the two functions are coded as separate categories even when one system provides both. Monitoring is most valuable when it is coupled to intervention such as approval escalation, privilege reduction, process termination, or the blocking of subsequent actions [35–37].

4.5.6 Intervention and Recovery

Intervention and recovery appears in 20 studies (34.5%). Common responses are denial, modification, termination, privilege revocation, human escalation, or switching to a more restrictive policy. AgentSpec supports runtime intervention when predicates are violated [6], and OAP includes escalation as an explicit authorization outcome [5].

The reviewed literature puts much more weight on preventing unsafe execution than on recovering once effects have begun. Blocking before execution is fairly simple when the operation is atomic; recovery is hard after persistent state changes, information disclosure, external transactions, or other irreversible effects.

CapLease addresses part of the problem with durable authorization state and an Issue–Prepare–Commit structure that holds across retries, concurrency, and crashes [17]. AP2 runtime verification similarly binds authorization to replay-resistant transaction state [21]. These mechanisms improve execution consistency and replay resistance but do not provide general rollback for arbitrary external effects.

The gap is clear: authorization, validation, and pre-execution blocking are well represented, while transactional rollback, compensating actions, crash-consistent recovery, and the handling of irreversible effects receive comparatively little attention.

4.6 Assurance Models

The third taxonomy dimension addresses **RQ4**: on what basis should an enforcement decision be trusted? Deterministic policy/systems enforcement is the dominant primary assurance type, accounting for 31 of the 58 studies (53.4%). Learned/probabilistic assurance is primary in 10 studies (17.2%), cryptographic/attested assurance in 9 (15.5%), and formal/model-checked assurance in 8 (13.8%) (Figure 2; Table 2).

Assurance is distinct from mechanism. Authorization, monitoring, or validation may each be implemented with learned models, deterministic policies, formal specifications, or cryptographic controls. The assurance dimension records *why* an enforcement outcome should be trusted, as opposed to *what* security function is performed.

4.6.1 Learned and Probabilistic Assurance

Ten studies (17.2%) rely primarily on learned or probabilistic judgments of whether an action, state, or trajectory is unsafe. StepGuard evaluates tool-use steps before execution and reports a substantial reduction in attack success while largely preserving task utility [10]. Other learned approaches take trajectory context or evolving runtime state into account when deciding whether to intervene [36, 65].

Learned guards are attractive because agent behavior is semantic, context dependent, and hard to enumerate. Their decisions nevertheless remain probabilistic and can suffer from distribution shift or adversarial inputs. They reduce risk, but they do not give the same assurance as an independently enforced rule that makes a forbidden action unreachable.

4.6.2 Deterministic Policy and Systems Assurance

Deterministic policy/systems mechanisms form the largest category, with 31 studies (53.4%). These systems rely on explicit policies, access-control rules, capabilities, runtime predicates, reference-monitor logic, or other systems controls whose behavior is fixed by the current security state.

OAP performs deterministic pre-action authorization outside the model [5]; SEAgent constrains privileges through mandatory access control [7]; and AgentSpec applies explicit runtime predicates and enforcement actions [6]. Related systems extend deterministic enforcement to task-scoped permissions, execution budgets, session-bound authority, and durable authorization state [17, 28, 30].

The size of this category shows that many designers deliberately move selected security decisions out of generative reasoning. The goal is less to stop unsafe

proposals from being generated than to stop specified classes of proposals from crossing an independently enforced execution boundary.

4.6.3 *Formal and Model-Checked Assurance*

Eight studies (13.8%) rely primarily on formal or model-checked assurance. They specify security properties mathematically and support them through proof, model checking, verified semantics, probabilistic models, or related techniques. LLMbda formalizes information-flow behavior and provides machine-checked reasoning about its security properties [8]. Other studies use formal specifications and runtime verification for agent compliance, distributed workflows, and execution authority [24, 37, 57, 58].

Formal assurance can establish properties stronger than empirical attack testing, but only within the assumptions and abstractions of the model, and only to the extent that the implementation matches it. A verified information-flow mechanism, for instance, does not show that natural-language intent was interpreted correctly or that an external tool behaves as expected. The scope of the proved property must be kept in view when interpreting formal guarantees.

4.6.4 *Cryptographic and Attested Assurance*

Nine studies (15.5%) rely primarily on cryptographic or attested mechanisms that protect authority, provenance, delegation, workflow state, or execution evidence through signatures, proofs, authenticated records, or attestation.

Lineage, an extension of the User-Managed Access (UMA) 2.0 protocol, binds multi-hop delegation to signed ancestry and constrained authority [22]. Authenticated-workflow approaches cryptographically bind actions to authorized workflow structure and policy constraints [38], and proof-derived authorization associates execution decisions with verifiable evidence that the required authorization conditions were met [23].

These mechanisms strengthen integrity and verifiability; they do not predict semantic safety. They can show that authority was delegated, that workflow state is authentic, or that execution evidence was preserved, but not necessarily that the underlying action reflects the user's actual intent.

The four assurance types therefore support different security claims even when applied to similar runtime mechanisms. Reporting their outputs as a single, undifferentiated "safety" measure would hide real differences in what is guaranteed.

4.7 Evaluation Practices and Security–Utility Trade-offs

Evaluation practice varies widely across the corpus. Reported outcomes include attack success and unsafe-action blocking rates, false positives and negatives, task utility, policy compliance, information-flow violations, authorization latency, throughput, sandbox overhead, replay resistance, and formally established security properties. These metrics capture different aspects of runtime enforcement and are not directly comparable.

OAP evaluates authorization latency and adversarial execution under different policies [5]; StepGuard measures attack reduction together with benign-task utility [10]; AgentSpec evaluates policy compliance and runtime overhead across several agent domains [6]; and LLMbda combines empirical evaluation with formal information-flow guarantees [8]. Other studies emphasize attack blocking and interception latency [39], sandbox containment and overhead [27], or authorization correctness under replay, concurrency, and faults [17]. Each answers a different security question, and the results should not be placed on a common scale.

A recurring concern is the three-way trade-off among *security*, *utility*, and *overhead*. Restrictive policies block unsafe actions but also reject legitimate ones; learned mechanisms keep semantic flexibility at the price of classification errors; and sandboxing or cryptographic verification adds systems cost. StepGuard reports lower attack success with a modest loss of utility [10], while SessionBound and risk-adaptive authorization quantify the latency added by execution-time authorization [29, 30].

No standardized evaluation methodology covers all of these dimensions. Threat models, workloads, agent architectures, and definitions of attack success or safe completion differ considerably, so we interpret quantitative results within the placement, mechanism, assurance model, and threat assumptions of each study instead of pooling them.

Empirical evaluation and formal assurance are also only partly integrated. Attack-based testing shows behavior against concrete adversarial cases but cannot establish universal guarantees; formal analysis proves specified properties only within an explicit model and its assumptions. A few systems combine executable

evaluation with formal reasoning or bounded model checking [8, 40], but this remains uncommon.

4.8 Architectural Synthesis: From Model-Held Authority to Independently Mediated Execution

The strongest point of convergence across studies is architectural: reasoning is being separated from *execution authority*. In otherwise heterogeneous systems, the model proposes an action and a separate component decides whether that proposal may produce an external effect. The pattern appears in deterministic pre-action authorization [5], programmable runtime enforcement [6], mandatory access control [7], information-flow enforcement [8, 9], learned action guards [10], and execution-level isolation [19, 27].

These approaches share one control structure: *the model proposes an action, and an independently controlled runtime component decides whether that proposal may produce an external effect*. A compromised reasoning component may still generate an unsafe proposal without gaining the authority needed to execute it. Systems that bind authority to tasks, sessions, execution state, or delegation carry the separation beyond individual tool calls [17, 22, 28, 30].

The resulting shift is from *model-held authority* toward *independently mediated execution authority*. Figure 4 illustrates the separation between proposal generation, authorization, and effect commitment. Durable authorization, multi-agent delegation, cross-layer composition, and recovery from partially committed effects remain open and are discussed in Section 5.

5 Discussion

The results suggest that runtime security for AI agents is turning into a systems-security problem and is no longer just an extension of prompt filtering or model alignment. Whatever the mechanism or assurance model, the recurring question is which component finally decides whether a model-generated proposal may produce an external effect. This section discusses what follows from separating model reasoning from execution authority, and which gaps remain.

5.1 Beyond Guardrails: Execution Authority as the Security Boundary

A central finding of this review is the recurring separation between reasoning and execution authority. Model-centric defenses shape what an agent generates; runtime enforcement adds a control boundary between generated proposals and their consequences. This separation changes what a model failure costs. If the model both proposes and, in effect, authorizes actions, prompt injection, reasoning errors, or other compromise can lead directly to unauthorized execution. Deterministic pre-action authorization [5], programmable runtime enforcement [6], mandatory access control [7], and information-flow enforcement [8] move selected security decisions outside the model. Systems-level isolation adds a further boundary by limiting what agent-controlled computation can do once it is running [19, 27].

These mechanisms complement each other and cannot simply be swapped. Model-level safeguards make unsafe proposals less likely; runtime authorization and validation decide whether proposals may become effects; and systems isolation constrains what executing components can do. The principle mirrors conventional least-privilege and reference-monitor design: untrusted application logic should not determine its own effective authority.

For AI agents, though, authority may depend on natural-language tasks, dynamically generated plans, provenance, persistent state, retries, and delegation. Recent systems increasingly bind authority to task, session, execution, or delegation state instead of relying solely on long-lived credentials [17, 22, 28, 30]. Secure agent architectures should consequently treat model-generated actions as proposals that remain subject to independently enforced authorization.

5.2 Authorization as an Execution-State Property

Credential-based authorization alone is not enough for autonomous execution. An agent may hold a valid credential while attempting an action that goes beyond the user's current intent, so authorization has to consider the task, the proposed action, and the current execution state as well as identity and credential validity.

The reviewed systems show a progression toward more contextual authority. Runtime policies constrain individual tool calls [5]; mandatory access control restricts agent–tool privilege relationships [7]; task-scoped and risk-adaptive mechanisms incorporate intent and execution context [28, 29]; and lineage-aware mechanisms constrain the propagation of authority across agents [22]. SessionBound and CapLease go further by binding authority to sessions, execution budgets, or durable state [17, 30].

State matters most under retries, concurrency, crashes, and semantically equivalent requests. CapLease

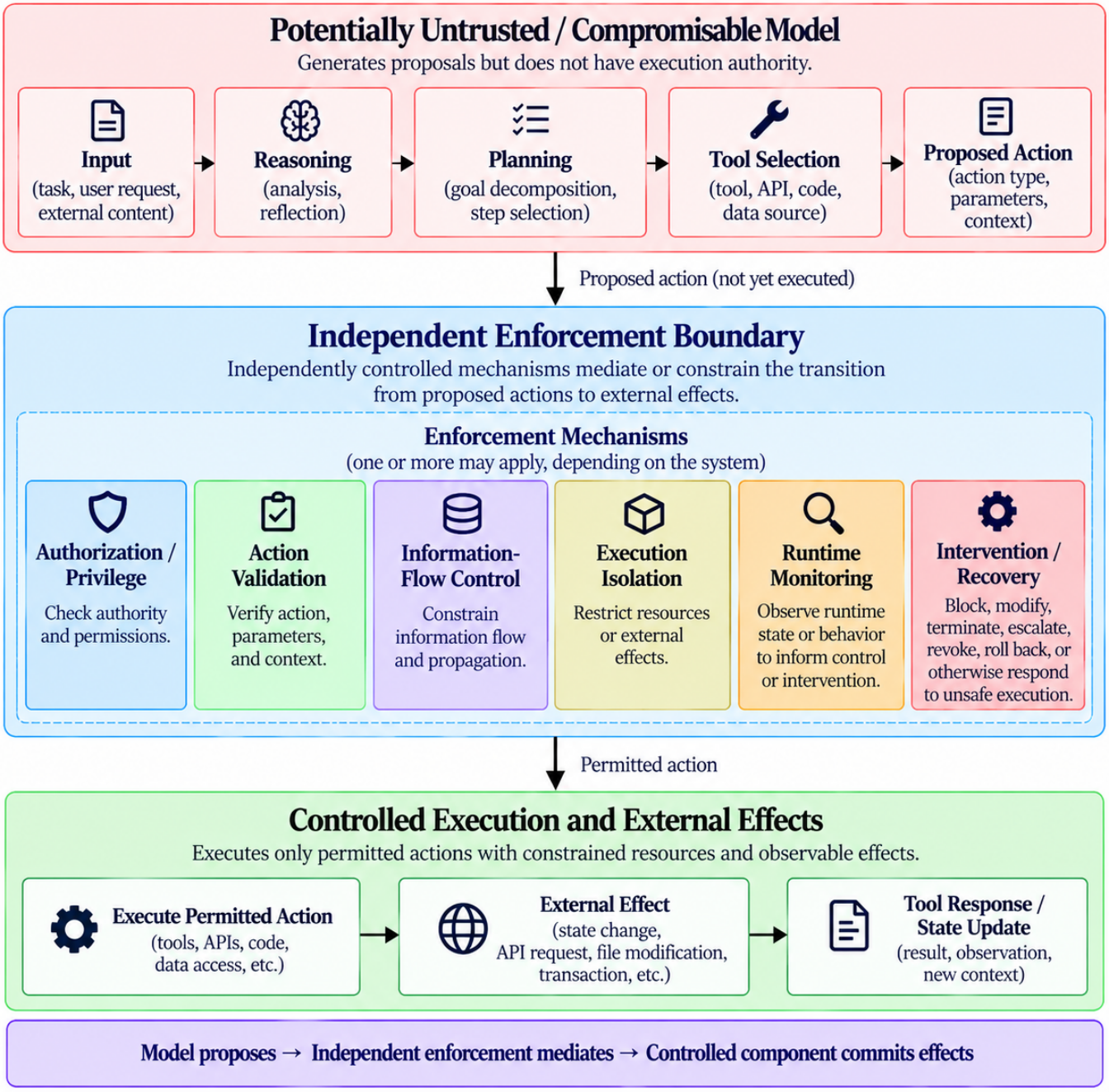


Figure 4. From model-held authority to independently mediated execution authority. Runtime controls may include authorization, validation, information-flow control, isolation, monitoring, and intervention.

maintains authorization state across these conditions [17], SessionBound converts task approval into constrained, short-lived database sessions [30], and runtime verification for agentic payments binds authorization to transaction context so that earlier approvals cannot be replayed [21].

Authorization is becoming a joint property of the credential, task, action, and execution state. Authority may need to be issued for a bounded purpose, attenuated or consumed during execution, and kept from being reused outside the context in which it was granted.

5.3 Composable Semantic and Systems Enforcement

No single mechanism in the reviewed literature provides a complete security boundary. Semantic controls can reason about user intent, task context,

provenance, information flow, and action parameters [5, 6, 8], but cannot guarantee that generated code, dependencies, or external tools will behave as expected once execution starts. Lower-level isolation, conversely, can constrain processes, resources, and external effects without knowing whether an operation matches the user's intent [19, 27, 41].

One way to describe this is as the difference between *permission* and *capability*. Semantic enforcement decides whether an operation should proceed; systems isolation constrains what approved computation can actually do. Information-flow controls add a third dimension by restricting how untrusted or sensitive information may influence an otherwise permitted effect, as LLMbda and Token-Flow Firewall do when they relate provenance or labels to downstream privileged operations [8, 9].

Assurance must also be considered compositionally. Learned guards offer semantic flexibility but probabilistic judgments [10]; deterministic authorization enforces explicit policy [5]; formal approaches establish specified properties under stated assumptions [8]; and cryptographic mechanisms protect authority, delegation, workflow state, or evidence integrity [22, 23, 38]. Similar empirical outcomes may therefore stand for quite different security claims.

Secure runtime architectures should combine these controls across architectural boundaries. The difficulty lies less in adding guards than in keeping security state and assumptions consistent across them, preventing one enforcement path from bypassing another, and ensuring that the failure of a single component does not open an unrestricted route from model output to external effect. A useful separation is among *generation*, *authorization*, and *execution*: the model proposes actions, independent mechanisms check whether the proposed effects satisfy the applicable constraints, and trusted or constrained components carry out approved effects.

5.4 Multi-Agent Delegation and Recovery

Two areas receive comparatively little attention in the reviewed literature: multi-agent delegation and recovery from partially committed effects.

In multi-agent systems, authority extends from individual agent–tool interactions to delegation chains. If privileges are copied instead of attenuated, downstream agents may acquire authority detached from the scope of the original task. Lineage constrains multi-hop delegation through signed ancestry and monotonic scope restriction [22], and task-scoped and durable capability mechanisms provide related building blocks [17, 28]. Revocation, preservation of user intent across delegation steps, cross-domain policy conflicts, and information-flow constraints across agents with different trust levels are still unresolved. Local authorization at each edge does not guarantee that the overall workflow is globally authorized or information-flow safe.

Recovery is a different, though related, state-management problem. Many systems can deny, validate, isolate, or escalate an action before it takes effect, but far fewer address what happens after an effect has been partly committed. A disclosure generally cannot be undone; a financial transaction may require a compensating operation; and file, database, or API actions come with application-specific concurrency, idempotency, and transaction semantics.

CapLease addresses authorization consistency across retries, concurrency, and crashes [17], and AP2 runtime verification binds authorization to replay-resistant transaction state [21]. These mechanisms reduce duplicate or inconsistent execution but do not offer general recovery from committed effects. Runtime enforcement needs clearer semantics for authorization consumption, idempotency, retries, partial failure, rollback where possible, and compensating actions where reversal is impossible, which ties agent security directly to transactions, crash recovery, distributed consistency, and fault-tolerant workflows.

5.5 Open Research Challenges

In answer to **RQ5**, the synthesis points to four research priorities.

1. **Stateful and replay-resistant authority.** Authorization should stay correct across retries, concurrency, crashes, long-running workflows, and semantically equivalent actions. Recent systems handle durable state, execution budgets, and context binding [17, 21, 30], but a general model that integrates the issuance, consumption, persistence, and revocation of authority is still lacking.
2. **Secure delegation and authority propagation.** Multi-agent workflows need mechanisms that preserve and attenuate authority along delegation chains. Task-scoped and lineage-aware

approaches are a useful starting point [22, 28], while revocation, dynamic agent graphs, cross-domain policies, and end-to-end information-flow constraints remain open.

3. **Composable and assurance-aware enforcement.** Tool-level authorization, semantic validation, information-flow control, execution isolation, monitoring, and recovery need compositional security models. Mechanisms that are each correct under their own assumptions may not yield end-to-end security when combined. Future architectures should state their probabilistic, deterministic, formal, and cryptographic claims explicitly and show how security state and guarantees carry across enforcement boundaries.
4. **Recovery and standardized evaluation.** Partial failure, irreversible effects, rollback, compensating actions, and crash-consistent execution need more systematic support. Evaluation is also fragmented across threat models, workloads, and metrics; common frameworks should report adversarial effectiveness, benign-task utility, false interventions, systems overhead, and the assumptions under which security claims hold.

All four challenges serve one systems objective: a failure in model reasoning should not automatically become a failure of execution control. Progress will depend on architectures that separate and compose generation, authorization, and execution behind independently enforceable boundaries.

6 Threats to Validity and Limitations

Several limitations affect the completeness and generalizability of this review. They stem mainly from literature coverage and study selection, the maturity of the evidence base, and the analytical abstractions used for corpus-level comparison.

6.1 Literature Coverage and Study Selection

The search was limited to Compendex and the Web of Science Core Collection. Targeted backward and forward snowballing identified four additional candidates, of which only three were retained. Snowballing was not exhaustive, and relevant work available only through preprint repositories, workshops, project websites, new venues, or terminology not covered by the search strings may have been missed. The corpus should be read as the literature identified by the stated search and snowballing procedure, not as a complete inventory of runtime-enforcement systems.

A further retrieval limitation concerns Compendex. The search interface reported 669 results, whereas 660 records were available in the detailed export used for screening. The nine-record gap could not be reconciled, so identified and imported counts are reported separately and the unavailable records are not treated as screened.

Screening was carried out in multiple passes, including a high-recall rescue audit, and studies that reached source-grounded eligibility assessment were checked against the primary sources and the predefined inclusion criteria. AI assistance was used to support screening, evidence extraction, and preliminary coding; final classifications were based on the predefined taxonomy and evidence from the primary studies. The review did not, however, use two independent human reviewers for all screening and extraction decisions, so inter-rater reliability statistics are not reported and some selection or coding error may remain.

6.2 Evidence and Coding Limitations

The evidence base is very recent: 54 of the 58 primary studies (93.1%) first appeared in 2026 and four (6.9%) in 2025. Of the 58 retained studies, approximately 45 are preprints that had not completed formal peer review by the search date; the remaining 13 appear in peer-reviewed journals or conference proceedings. This distribution reflects the pace of the field and is consistent with the eligibility criteria stated in Section 3.1, but readers should weigh quantitative findings from preprint and peer-reviewed sources with this disparity in mind. Technically substantive preprints were included because excluding them would omit much of the emerging literature, but their inclusion does not imply equal evidential maturity across the corpus. The review should be read as a snapshot of the field as of September 9, 2026.

The Placement–Mechanism–Assurance taxonomy also simplifies heterogeneous systems. Each study was assigned one primary placement and one primary assurance type, while mechanism coding was multi-label. Placement and assurance frequencies are therefore primary analytical classifications, not complete inventories of all components, and mechanism frequencies do not sum to 100%. Systems in the same category may still differ substantially in functionality or assurance strength.

Finally, some classifications involve interpretive

judgment, particularly for hybrid systems spanning several placements, mechanisms, or assurance types. Ambiguous cases were coded by the security function, placement, and assurance basis most central to the study's stated objective and execution-time control path. The taxonomy is best understood as a structured analytical lens, not a uniquely determined or exhaustive decomposition of runtime security, and future agent architectures may require its categories to be refined or extended.

7 Conclusion

This systematic review examined runtime-enforcement mechanisms for autonomous and tool-using AI agents. Although the search window began in January 2023, all 58 retained primary studies first became publicly available between 2025 and September 2026. The literature is concentrated at the tool/action boundary and inside the agent runtime or harness, with fewer systems enforcing security at the OS/process/execution boundary. Authorization and privilege control and action validation are the most common mechanism families, and the underlying assurance ranges from learned and probabilistic techniques to deterministic, formal, and cryptographic approaches. The proposed *Placement–Mechanism–Assurance* taxonomy captures these differences by separating where enforcement occurs, what security function it performs, and why its decisions can be trusted.

Beneath this technical diversity lies an architectural convergence: reasoning and execution authority are increasingly separated. An LLM may plan, select tools, and propose actions, while an independently controlled component decides whether those proposals may take external effect. Unsafe reasoning can still produce an unsafe proposal, but it is no longer sufficient on its own to authorize or commit the corresponding effect.

Runtime enforcement complements model-level safeguards and does not replace them. Model defenses reduce unsafe proposals, authorization and validation mediate their transition to external effects, information-flow controls limit how sensitive or untrusted data shape actions, and systems isolation bounds execution capabilities.

Durable authorization across retries and failures, secure multi-agent delegation, cross-layer composition, recovery from partially committed or irreversible effects, and standardized security–utility evaluation all remain open. Closing these gaps will require tighter integration of semantic security decisions with systems mechanisms for authority management, isolation, execution state, and recovery.

The central implication is that secure agent architectures should not depend on the generative model as the final security boundary. They should be designed so that failure or compromise in model reasoning does not automatically confer authority over consequential external effects; in practice, this means placing execution authority behind independently enforceable boundaries.

Data Availability Statement

Not applicable.

Funding

This work was supported without any funding.

Conflicts of Interest

The authors declare no conflicts of interest.

AI Use Statement

The authors declare that ChatGPT-5 was used for language editing of the manuscript. The authors have carefully reviewed, revised, and verified the AI-assisted output and take full responsibility for the content of the manuscript.

Ethical Approval and Consent to Participate

Not applicable.

References

[1] Ferrag, M. A., Lakas, A., Tihanyi, N., & Debbah, M. (2026). Securing LLM Agents: From Prompt Sanitization to Autonomous Red Teaming and Beyond. *Internet of Things and Cyber-Physical Systems, 6*. [CrossRef]

[2] Ling, Y., Yu, S., Chen, Z., & Fang, C. (2026). Toward Secure LLM Agents: Threat Surfaces, Attacks, Defenses, and Evaluation. *arXiv preprint arXiv:2606.10749*. [CrossRef]

[3] Hu, M., Wang, T., Qu, Y., Geng, Y., & Wang, H. (2026). A Focused Survey of Code Agent Security: Attacks, Defenses, and Evaluation. *Information and Software Technology, 199*, 108288. [CrossRef]

[4] Xu, Y., Zhuang, Y., Liu, X., Zhang, T., Xiao, B., Xu, X., ... & Hu, H. (2026). LLM Agents Security

Duality: A Comprehensive Survey of Self-Security and Empowered Cybersecurity. *Artificial Intelligence Review, 59*(8), 174. [CrossRef]

[5] Uchibeke, U. (2026). Before the Tool Call: Deterministic Pre-Action Authorization for Autonomous AI Agents. *arXiv preprint arXiv:2603.20953*. [CrossRef]

[6] Wang, H., Poskitt, C. M., & Sun, J. (2026, April). Agentspec: Customizable runtime enforcement for safe and reliable llm agents. In *Proceedings of the 2026 IEEE/ACM 48th International Conference on Software Engineering* (pp. 2938-2950). [CrossRef]

[7] Ji, Z., Wu, D., Jiang, W., Ma, P., Li, Z., Gao, Y., ... & Li, Y. (2026). Taming various privilege escalation in LLM-based agent systems: A mandatory access control framework. *arXiv preprint arXiv:2601.11893*. [CrossRef]

[8] Garby, Z., Gordon, A. D., & Sands, D. (2026). The llmbda calculus: Ai agents, conversations, and information flow. *arXiv preprint arXiv:2602.20064*. [CrossRef]

[9] Wang, P., Zhang, Y., Zhang, R., Guo, J., & Cheng, X. (2026). Token-Flow Firewall: Semantic Runtime Auditing for Persistent AI Agents. *arXiv preprint arXiv:2607.08395*. [CrossRef]

[10] Zheng, Z., Li, Y., Qian, C., Fu, Y., Fu, Y., Sheng, L., ... & Liu, D. (2026). Stepguard: Learning step-level guardrails with scalable supervision and safety-utility balancing. *arXiv preprint arXiv:2608.24777*. [CrossRef]

[11] Könighofer, B., Bloem, R., Ehlers, R., & Pek, C. (2022). Correct-by-construction runtime enforcement in AI–a survey. In Principles of Systems Design: Essays Dedicated to Thomas A. *Henzinger on the Occasion of His 60th Birthday* (pp. 650-663). Cham: Springer Nature Switzerland. [CrossRef]

[12] Wang, H., Poskitt, C. M., Wei, J., & Sun, J. (2026, October). Probguard: Proactive runtime monitoring for llm agent safety via probabilistic prediction. In *Proceedings of the 41st IEEE/ACM International Conference on Automated Software Engineering* (pp. 1470-1482). [CrossRef]

[13] Kitchenham, B., & Charters, S. (2007). Guidelines for Performing Systematic Literature Reviews in Software Engineering. Technical Report EBSE-2007-01, Keele University and Durham University. https://legacyfileshare.elsevier.com/promis_misc/525444systematicreviewsguide.pdf

[14] Wohlin, C. (2014). Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering* (*EASE*) (pp. 38:1-38:10). [CrossRef]

[15] Page, M. J., McKenzie, J. E., Bossuyt, P. M., Boutron, I., Hoffmann, T. C., Mulrow, C. D., ... & others (2021). The PRISMA 2020 statement: an updated guideline for reporting systematic reviews. *BMJ, 372*, n71. [CrossRef]

[16] Zhao, W., Li, Z., Zhang, P., & Sun, J. (2026). ClawGuard: A runtime security framework for tool-augmented LLM agents against indirect prompt injection. *arXiv preprint arXiv:2604.11790*. [CrossRef]

[17] Xu, J., Fan, L., Wang, Z., Li, X., & Liu, H. (2026). Beyond single-use tokens: Durable authorization state for replay-resistant LLM agent actions. *arXiv preprint arXiv:2608.01710*. [CrossRef]

[18] Zhang, W., Liu, Y., Yang, Z., Zhang, Z., Feng, H., Wang, X., ... & Hong, J. B. (2026). CARE: Pre-Execution Command Verification for Shell-Executing LLM Agents. *arXiv preprint arXiv:2607.21642*. [CrossRef]

[19] Zhang, Y. (2026). Agent libOS: A Runtime Substrate for Capability-Controlled Self-Evolving LLM Agents. *arXiv preprint arXiv:2606.03895*. [CrossRef]

[20] Wang, C., & Zheng, Y. (2026). Sandlock: Confining AI Agent Code with Unprivileged Linux Primitives. *arXiv preprint arXiv:2605.26298*. [CrossRef]

[21] Lan, Q., Kaul, A., Jones, S., & Westrum, S. (2026). Zero-trust runtime verification for agentic payment protocols: Mitigating replay and context-binding failures in AP2. *arXiv preprint arXiv:2602.06345*. [CrossRef]

[22] Stefanescu, S., & Aciobanitei, I. (2026). Lineage: A UMA2.0 Protocol Extension for Multi-Hop LLM Agent Delegation. *IEEE Access, 14*, 104574-104591. [CrossRef]

[23] He, J., & Yu, D. (2026). Verifiable Agentic Infrastructure: Proof-Derived Authorization for Sovereign AI Systems. *arXiv preprint arXiv:2605.15228*. [CrossRef]

[24] Rhodes, J., & Kang, G. (2026). Proof of Execution: Runtime Verification for Governed AI Agent Actions. *arXiv preprint arXiv:2607.05397*. [CrossRef]

[25] Dobrin, S., & Chmiel, Ł. (2026). The Unfireable Safety Kernel: Execution-Time AI Alignment for AI Agents and Other Escapable AI Systems. *arXiv preprint arXiv:2606.26057*. [CrossRef]

[26] Singh, G., & Madisetti, V. K. (2026). MCP-Secure: A Runtime Access Control Layer for Privilege-Aware LLM Agent Tooling. *IEEE Open Journal of the Computer Society, 7*, 504-513. [CrossRef]

[27] Jiang, J. (2026). Runtime Protection Practice for LLM Agent Security: An On-Demand Behavioral Sandbox Approach. In *2026 11th International Symposium on Advances in Electrical, Electronics and Computer Engineering* (*ISAEECE*) (pp. 587-590). [CrossRef]

[28] Sharma, R. K., Jiang, L., Lin, Z., & Chen, S. (2026). PAuth - Precise Task-Scoped Authorization For Agents. *arXiv preprint arXiv:2603.17170*. [CrossRef]

[29] Bhushan, B., Pappu, K., Jadav, V., & Jaiswal, N. (2026, February). Risk-adaptive authorization for agentic ai systems. In *SoutheastCon 2026* (pp. 1-6). IEEE. [CrossRef]

[30] Wu, M. (2026). SessionBound: Turning Enterprise Task Approval into Budgeted Database Sessions. *arXiv preprint arXiv:2607.00751*. [CrossRef]

[31] Zhu, G., & Wang, C. (2026). Intent-Governed Tool Authorization for AI Agents. *arXiv preprint arXiv:2606.22916*. [CrossRef]

[32] Santos-Grueiro, I. (2026). Context-to-Execution Integrity for LLM Agents. *arXiv preprint arXiv:2607.06000*. [CrossRef]

[33] Zhu, G., & Wang, C. (2026). Explanation-Bound Tool Execution for AI Agents: Server-Verified Action Claims Without Trusting Model Rationales. arXiv preprint arXiv:2607.25364. [CrossRef]

[34] Stanley, R., Verma, A., Tsai, L., Kallas, K., & Kumar, S. (2026). An AI Agent Execution Environment to Safeguard User Data. *arXiv preprint arXiv:2604.19657*. [CrossRef]

[35] Abaev, N., Klimov, D., Levinov, G., Mimran, D., Elovici, Y., & Shabtai, A. (2026). AgentGuardian: Learning Access Control Policies to Govern AI Agent Behavior. *arXiv preprint arXiv:2601.10440*. [CrossRef]

[36] Lin, J., Zhou, Z., Zheng, Z., Liu, S., Xu, T., Chen, Y., & Chen, E. (2026). VIGIL: Defending LLM Agents Against Tool-Stream Injection via Verify-Before-Commit. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics* (*Volume 1: Long Papers*) (pp. 9764-9785). Association for Computational Linguistics. [CrossRef]

[37] Kahani, N., Barati, M., & Addae, D. (2026, October). Runtime Compliance Verification for AI Agents. In *International Conference on Runtime Verification* (pp. 254-271). Cham: Springer Nature Switzerland. [CrossRef]

[38] Rajagopalan, M., & Rao, V. (2026). Authenticated Workflows: A Systems Approach to Protecting Agentic AI. *arXiv preprint arXiv:2602.10465*. [CrossRef]

[39] Yuan, A., Su, Z., & Zhao, Y. (2026). AEGIS: No Tool Call Left Unchecked - A Pre-Execution Firewall and Audit Layer for AI Agents. *arXiv preprint arXiv:2603.12621*. [CrossRef]

[40] Sharma, R. M. (2026). Ethical Hyper-Velocity (EHV): A Hardware-Rooted Zero-Trust Runtime Enforcement Architecture for Agentic AI Systems. *arXiv preprint arXiv:2605.17909*. [CrossRef]

[41] Zheng, Y., Fan, J., Fu, Q., Yang, Y., Zhang, W., & Quinn, A. (2026). AgentCgroup: Understanding and Controlling OS Resources of AI Agents. *arXiv preprint arXiv:2602.09345*. [CrossRef]

[42] Wang, S., Zhu, S., & Li, R. (2026). Runtime Policy Enforcement for MCP-Based LLM Agents. *Electronics*, *15*(13), 2829. [CrossRef]

[43] Zhu, J., Tseng, K., Vernik, G., Huang, X., Patil, S. G., Fang, V., & Popa, R. A. (2025). MiniScope: A Least Privilege Framework for Authorizing Tool Calling Agents. *arXiv preprint arXiv:2512.11147*. [CrossRef]

[44] Zheng, Y., Wu, T., Fu, Q., Yu, T., Mao, W., Ma, T., ... & Quinn, A. (2026). ActPlane: Programmable OS-Level Policy Enforcement for Agent Harnesses. *arXiv preprint arXiv:2606.25189*. [CrossRef]

[45] Mellafe Zuvic, D. (2026). Capability Gates Are Not Authorization: Confused-Deputy Failures in LLM Agent Frameworks. *arXiv preprint arXiv:2606.28679*. [CrossRef]

[46] Shi, T., He, J., Wang, Z., Li, H., Wu, L., Guo, W., & Song, D. (2025). Progent: Programmable Privilege Control for LLM Agents. arXiv preprint arXiv:2504.11703. [CrossRef]

[47] Kodathala, S. V. (2026). aiAuthZ: Off-Host, Identity-Bound Authorization for AI Agents. *arXiv preprint arXiv:2607.05518*. [CrossRef]

[48] Li, Y., Chen, Y., Wen, H., Zhang, B., Liu, H., Wang, P., ... & Tian, Y. (2026). VIGIL: Runtime Enforcement of Behavioral Specifications in AI Agent Skills. *arXiv preprint arXiv:2606.26524*. [CrossRef]

[49] Pan, S., Sun, X., Zhang, T., Liao, D., Si, M., & Xing, Z. (2026). SkillGuard: A Permission-Centric Framework for Agent Skill Security. arXiv preprint arXiv:2606.03024. [CrossRef]

[50] Mazzocchetti, A. (2026). Runtime Governance for Agentic AI: Action-Boundary Control with Trusted Provenance and Fail-Closed Execution. *arXiv preprint arXiv:2608.16891*. [CrossRef]

[51] Joshi, A., Finin, T., Joshi, K. P., & Kagal, L. (2026). Deontic Policies for Runtime Governance of Agentic AI Systems. In *2026 IEEE International Conference on Web Services* (ICWS). [CrossRef]

[52] Sidik, B., & Rokach, L. (2026). Beyond Static Sandboxing: Learned Capability Governance for Autonomous AI Agents. *arXiv preprint arXiv:2604.11839*. [CrossRef]

[53] Wang, P., Li, Y., & Tian, Y. (2026). Aligning Provenance with Authorization: A Dual-Graph Defense for LLM Agents. *arXiv preprint arXiv:2605.26497*. [CrossRef]

[54] El Helou, M., Ryder, B., Troiani, C., Diaconu, J., Muyal, H., & Yannuzzi, M. (2026). Hybrid Inspection and Task-Based Access Control in Zero-Trust Agentic AI. *arXiv preprint arXiv:2605.02682*. [CrossRef]

[55] Weng, J., Weng, J., Chen, M., Li, M., Liu, J.-N., Li, Z., & Zhang, Y. (2026). From Neural Intent to Cryptographic Authorization: Securing AI-Driven Enterprise Workflows. *arXiv preprint arXiv:2607.15596*. [CrossRef]

[56] Yang, C. (2026). AgentTrust: Runtime Safety Evaluation and Interception for AI Agent Tool Use. *arXiv preprint arXiv:2605.04785*. [CrossRef]

[57] Bollig, B. (2026). Causal Past Logic for Runtime Verification of Distributed LLM Agent Workflows. *arXiv preprint arXiv:2605.20923*. [CrossRef]

[58] Gill-Lakhowal, A. (2026). Deterministic Runtime Enforcement: The Execution Authority for

Autonomous AI Agents. *IEEE Access, 14*, 119764-119790. [CrossRef]

[59] Malik, A., Jayasinghe, D., Klemick, B., Shah, P., Talasu, N., & Trivedi, V. T. (2026). Decentralized Granular Access Control for Agentic AI Systems in Critical Infrastructure. *arXiv preprint arXiv:2607.22611*. [CrossRef]

[60] Gandikota, L. R., Jegathesan, V., & Reddy, C. D. R. (2026, May). Agentic AI in Cybersecurity: Threat Modeling and Containment Strategies for Autonomous LLM Agents. In *2026 2nd International Conference on Artificial Intelligence Systems (AIS)* (pp. 68-75). IEEE. [CrossRef]

[61] Nirban, H. S., Hinduja, B., Mangal, B., Chakka, V., Sharma, Y., Srivastava, S. R., ... & Bhatia, A. (2026, April). A Zero-Trust Identity Framework for Autonomous AI Agents: Cryptographic Attestation and Policy-Driven Authorization. In *International Conference on Advanced Information Networking and Applications* (pp. 258-269). Cham: Springer Nature Switzerland. [CrossRef]

[62] Li, F. (2026). OpenClaw PRISM: A Zero-Fork, Defense-in-Depth Runtime Security Layer for Tool-Augmented LLM Agents. *arXiv preprint arXiv:2603.11853*. [CrossRef]

[63] Shi, T., Mo, Y., Liu, Y., Hao, Z., Wang, Y., Hu, W., ... & Yu, J. (2026). Organizational Control Layer: Governance Infrastructure at the Execution Boundary of LLM Agent Systems. *arXiv preprint arXiv:2606.04306*. [CrossRef]

[64] Cai, Y., Wang, Z., Deng, Z., Yao, M., Liu, J., Hu, Y., ... & Li, D. (2025). Who Grants the Agent Power? Defending Against Instruction Injection via Task-Centric Access Control. In *Proceedings of the 2025 Workshop on Security and Privacy of AI-Empowered Mobile Super Apps* (SaTS). [CrossRef]

[65] Xiong, Y., Hao, L., He, S., Wang, Y., & Li, L. (2026). JANUS: Foreseeing Latent Risk for Long-Horizon Agent Safety. *arXiv preprint arXiv:2607.19913*. [CrossRef]

[66] Ntousakis, G., Stephen, J. J., Le, M. V., Chukkapalli, S. S. L., Taylor, T., Molloy, I. M., & Araujo, F. (2025, October). Securing mcp-based agent workflows. In *Proceedings of the 4th Workshop on Practical Adoption Challenges of ML for Systems* (pp. 50-55). [CrossRef]

[67] Betser, R., Bose, S., Giloni, A., Picardi, C., Padakandla, S., & Vainshtein, R. (2026). AgenTRIM: Tool Risk Mitigation for Agentic AI. *arXiv preprint arXiv:2601.12449*. [CrossRef]

[68] Zhou, Z. (2026). Governing Dynamic Capabilities: Cryptographic Binding and Reproducibility Verification for AI Agent Tool Use. *arXiv preprint arXiv:2603.14332*. [CrossRef]

[69] Xiang, R. (2026). Structural Enforcement of Goal Integrity in AI Agents via Separation-of-Powers Architecture. *arXiv preprint arXiv:2604.23646*. [CrossRef]

Appendix

A Classification of the Primary Studies

Table A1 lists the 58 primary studies included in the review and their classifications under the Placement–Mechanism–Assurance taxonomy. Each study is assigned one primary placement and one primary assurance type; mechanism categories are multi-label. Study identifiers follow the retained-study ledger and are therefore not consecutive.

Table A1. Classification of the 58 primary studies included in the review.

ID	Study title	Placement	Mechanism(s)	Assurance
S01	Runtime Policy Enforcement for MCP-Based LLM Agents [42]	Tool/action	Auth.; IFC; Monitoring	Deterministic
S02	MiniScope: A Least Privilege Framework for Authorizing Tool Calling Agents [43]	Tool/action	Auth.	Deterministic
S03	ActPlane: Programmable OS-Level Policy Enforcement for Agent Harnesses [44]	OS/process	Isolation; IFC; Monitoring	Deterministic
S04	Before the Tool Call: Deterministic Pre-Action Authorization for Autonomous AI Agents [5]	Tool/action	Auth.; Validation	Deterministic
S06	AgentGuardian: Learning Access Control Policies to Govern AI Agent Behavior [35]	Runtime/harness	Auth.; Monitoring; Intervention	Learned
S07	ClawGuard: A Runtime Security Framework for Tool-Augmented LLM Agents Against Indirect Prompt Injection [16]	Tool/action	Auth.; Validation; Intervention	Deterministic
S08	Capability Gates Are Not Authorization: Confused-Deputy Failures in LLM Agent Frameworks [45]	Tool/action	Auth.; Validation	Deterministic
S10	Progent: Programmable Privilege Control for LLM Agents [46]	Tool/action	Auth.	Deterministic
S11	aiAuthZ: Off-Host, Identity-Bound Authorization for AI Agents [47]	Tool/action	Auth.; Validation	Cryptographic
S12	Context-to-Execution Integrity for LLM Agents [32]	Tool/action	Validation; IFC; Auth.	Deterministic
S13	VIGIL: Runtime Enforcement of Behavioral Specifications in AI Agent Skills [48]	Runtime/harness	Monitoring; IFC; Validation	Formal
S14	VIGIL: Defending LLM Agents Against Tool-Stream Injection via Verify-Before-Commit [36]	Runtime/harness	Validation; Monitoring; Intervention	Learned
S15	CARE: Pre-Execution Command Verification for Shell-Executing LLM Agents [18]	Tool/action	Validation	Deterministic
S16	SkillGuard: A Permission-Centric Framework for Agent Skill Security [49]	Runtime/harness	Auth.; IFC; Monitoring	Deterministic
S17	Runtime Governance for Agentic AI: Action-Boundary Control with Trusted Provenance and Fail-Closed Execution [50]	Tool/action	Auth.; Validation; Intervention	Deterministic
S18	MCP-Secure: A Runtime Access Control Layer for Privilege-Aware LLM Agent Tooling [26]	Runtime/harness	Auth.; Intervention	Deterministic
S19	Deontic Policies for Runtime Governance of Agentic AI Systems [51]	Runtime/harness	Auth.; Intervention	Deterministic
S20	Beyond Static Sandboxing: Learned Capability Governance for Autonomous AI Agents [52]	Tool/action	Auth.; Validation; Isolation	Learned
S21	Aligning Provenance with Authorization: A Dual-Graph Defense for LLM Agents [53]	Tool/action	Validation; IFC	Learned
S22	Hybrid Inspection and Task-Based Access Control in Zero-Trust Agentic AI [54]	Tool/action	Auth.; Validation	Learned
S23	Beyond Single-Use Tokens: Durable Authorization State for Replay-Resistant LLM Agent Actions [17]	Tool/action	Auth.; Intervention	Deterministic

Continued on next page

Table A1 (continued)

ID	Study title	Placement	Mechanism(s)	Assurance
S24	From Neural Intent to Cryptographic Authorization: Securing AI-Driven Enterprise Workflows [55]	Runtime/harness	Auth.; Validation	Cryptographic
S25	AgentTrust: Runtime Safety Evaluation and Interception for AI Agent Tool Use [56]	Tool/action	Validation; Intervention	Deterministic
S26	Runtime Protection Practice for LLM Agent Security: An On-Demand Behavioral Sandbox Approach [27]	OS/process	Isolation; Validation; Monitoring	Deterministic
S27	AgentSpec: Customizable Runtime Enforcement for Safe and Reliable LLM Agents [6]	Runtime/harness	Validation; Intervention	Deterministic
S28	PAuth–Precise Task-Scoped Authorization For Agents [28]	Tool/action	Auth.; IFC	Deterministic
S29	Sandlock: Confining AI Agent Code with Unprivileged Linux Primitives [20]	OS/process	Isolation; IFC; Intervention	Deterministic
S30	Agent libOS: A Runtime Substrate for Capability-Controlled Self-Evolving LLM Agents [19]	Runtime/harness	Auth.; IFC; Isolation; Intervention	Deterministic
S31	Proof of Execution: Runtime Verification for Governed AI Agent Actions [24]	Tool/action	Auth.; Validation; Monitoring	Formal
S32	Taming Various Privilege Escalation in LLM-Based Agent Systems: A Mandatory Access Control Framework [7]	Tool/action	Auth.; IFC	Deterministic
S33	Token-Flow Firewall: Semantic Runtime Auditing for Persistent AI Agents [9]	Runtime/harness	IFC; Validation; Monitoring	Learned
S35	Runtime Compliance Verification for AI Agents [37]	Runtime/harness	IFC; Monitoring; Intervention	Formal
S36	Causal Past Logic for Runtime Verification of Distributed LLM Agent Workflows [57]	Runtime/harness	Monitoring; Validation	Formal
S37	Risk-Adaptive Authorization for Agentic AI Systems [29]	Tool/action	Auth.; Validation; Intervention	Deterministic
S38	Deterministic Runtime Enforcement: The Execution Authority for Autonomous AI Agents [58]	Runtime/harness	Auth.; Validation; Intervention	Formal
S39	Decentralized Granular Access Control for Agentic AI Systems in Critical Infrastructure [59]	Tool/action	Auth.; Intervention	Deterministic
S40	The LLMbda Calculus: AI Agents, Conversations, and Information Flow [8]	Runtime/harness	IFC	Formal
S42	An AI Agent Execution Environment to Safeguard User Data [34]	OS/process	IFC; Auth.; Isolation	Deterministic
S43	Agentic AI in Cybersecurity: Threat Modeling and Containment Strategies for Autonomous LLM Agents [60]	OS/process	Isolation; Validation; Monitoring	Learned
S44	A Zero-Trust Identity Framework for Autonomous AI Agents: Cryptographic Attestation and Policy-Driven Authorization [61]	Tool/action	Auth.	Cryptographic
S45	Intent-Governed Tool Authorization for AI Agents [31]	Tool/action	Auth.; Validation	Deterministic
S46	OpenClaw PRISM: A Zero-Fork, Defense-in-Depth Runtime Security Layer for Tool-Augmented LLM Agents [62]	Runtime/harness	Auth.; Validation; IFC; Monitoring; Intervention	Deterministic
S47	Organizational Control Layer: Governance Infrastructure at the Execution Boundary of LLM Agent Systems [63]	Tool/action	Auth.; Validation; Intervention	Deterministic
S48	Verifiable Agentic Infrastructure: Proof-Derived Authorization for Sovereign AI Systems [23]	Tool/action	Auth.; Validation; Monitoring	Cryptographic

Continued on next page

Table A1 (*continued*)

ID	Study title	Placement	Mechanism(s)	Assurance
S49	The Unfireable Safety Kernel: Execution-Time AI Alignment for AI Agents and Other Escapable AI Systems [25]	OS/process	Auth.; Isolation; Intervention	Formal
S50	Who Grants the Agent Power? Defending Against Instruction Injection via Task-Centric Access Control [64]	Runtime/harness	Auth.; Validation; Intervention	Deterministic
S51	Zero-Trust Runtime Verification for Agentic Payment Protocols: Mitigating Replay and Context-Binding Failures in AP2 [21]	Tool/action	Auth.; Validation	Cryptographic
S52	Ethical Hyper-Velocity (EHV): A Hardware-Rooted Zero-Trust Runtime Enforcement Architecture for Agentic AI Systems [40]	Tool/action	Auth.; Isolation; Monitoring	Formal
S53	StepGuard: Learning Step-Level Guardrails with Scalable Supervision and Safety-Utility Balancing [10]	Tool/action	Validation	Learned
S54	JANUS: Foreseeing Latent Risk for Long-Horizon Agent Safety [65]	Tool/action	Validation; Monitoring	Learned
S56	AEGIS: No Tool Call Left Unchecked – A Pre-Execution Firewall and Audit Layer for AI Agents [39]	Tool/action	Validation; Auth.; Intervention	Deterministic
S57	Lineage: A UMA2.0 Protocol Extension for Multi-Hop LLM Agent Delegation [22]	Tool/action	Auth.	Cryptographic
S58	Securing MCP-based Agent Workflows [66]	Runtime/harness	IFC	Deterministic
S59	AgenTRIM: Tool Risk Mitigation for Agentic AI [67]	Tool/action	Auth.; Validation	Learned
S60	Governing Dynamic Capabilities: Cryptographic Binding and Reproducibility Verification for AI Agent Tool Use [68]	Tool/action	Auth.; Monitoring	Cryptographic
S61	SessionBound: Turning Enterprise Task Approval into Budgeted Database Sessions [30]	OS/process	Auth.; IFC	Deterministic
S65	Authenticated Workflows: A Systems Approach to Protecting Agentic AI [38]	Runtime/harness	Auth.; IFC; Validation	Cryptographic
S67	Structural Enforcement of Goal Integrity in AI Agents via Separation-of-Powers Architecture [69]	Tool/action	Validation; Auth.	Cryptographic

Abbreviations: Auth., authorization and privilege control; IFC, information-flow control; Tool/action, tool or action boundary; Runtime/harness, agent runtime or harness; OS/process, OS/process/execution boundary; Learned, learned/probabilistic assurance; Deterministic, deterministic policy/systems assurance; Formal, formal/model-checked assurance; Cryptographic, cryptographic/attested assurance. Intervention denotes intervention/recovery.

Jimmy Ming-Tai Wu is with the Department of Information Management, National Kaohsiung University of Science and Technology, Kaohsiung, Taiwan. (Email: jimmywmt@nkust.edu.tw)

Lingping Kong is with the Department of Computer Science, VSB–Technical University of Ostrava, Ostrava, Czech Republic. (Email: lingping.kong@vsb.cz)

Matin Pirouz is with the College of Science and Mathematics, California State University, Fresno, CA, USA. (Email: mpirouz@csufresno.edu)

Mu-En Wu is with the Department of Information and Finance Management, National Taipei University of Technology, Taipei, Taiwan. (Email: mnasia1@gmail.com)