

Zero Trust Architecture for AI Agent Security

Response to NIST Docket NIST-2025-0035

Submitted by: Shapor Naghibzadeh, CEO, QueryStory, Inc.

Date: February 28, 2026

This response addresses questions: 1(a), 1(b), 1(c), 1(d), 2(a), 2(e), 3(a), 3(b), 4(a), 4(b), 4(d), 5(a), 5(b)

Contact: shapor@querystory.ai

About the Respondent

QueryStory builds and operates production AI agent systems for enterprise customers in a multi-tenant SaaS environment. The author co-founded Google's Threat Analysis Group (TAG) in the aftermath of Operation Aurora — the attack that proved perimeter-based network security was fundamentally broken and catalyzed the development of BeyondCorp and subsequently NIST SP 800-207. He later co-founded Chronicle and led engineering for AI-powered security agents in Google Cloud security operations from 2023 to 2025. That experience directly informs our approach to AI agent security, which applies the same principle: **just as zero trust abandoned trust in the network perimeter, AI agent architectures must abandon trust in model outputs.**

This response draws on both that institutional experience and our direct, daily work building and operating production AI agent systems. We focus on the priority questions identified in the RFI.

1. Security Threats, Risks, and Vulnerabilities

1(a) Unique security threats distinct from traditional software

The most consequential threat we observe in production AI agent systems is **the collapse of the traditional boundary between code and data**. In conventional software, instructions (code) and inputs (data) occupy clearly separated domains. In AI agent systems, a natural language instruction and a natural language data value are indistinguishable to the model. This creates an entirely new class of vulnerability with no analog in traditional software:

Indirect prompt injection via data. AI agents routinely read external data (data fields, content values, document contents) as context for their actions. If an adversary can place crafted text into a data source the agent will later read, the agent may interpret that data as an instruction. This is not a bug that can be patched; it is an inherent property of how current language models process input. Traditional SQL injection has well-understood mitigations (parameterized queries). Indirect prompt injection currently has no equivalent definitive solution. It is OWASP's #1 risk for LLM applications in 2025 [1], and OpenAI has publicly acknowledged that "prompt injection, much like scams and social engineering on the web, is unlikely to ever be fully 'solved'" [18].

The peer-reviewed research confirms the severity:

- Nasr, Carlini et al. tested 12 published defenses that originally reported near-zero attack success rates; using adaptive attacks, they bypassed all 12 with >90% success. Their accompanying red-teaming competition confirmed that "defenses evaluated against static attack sets give a false sense of security" [2].
- Google DeepMind's internal security evaluation found that an undefended Gemini model fell to >99% of prompt injection test scenarios. After applying adversarial fine-tuning for Gemini 2.5, attack success rates dropped to 53.6% in one scenario but remained at 94.6% for out-of-distribution attacks. Even combining adversarial training with their best external defense (Warning), the most effective attack still

succeeded 6.2% of the time, and creating a successful attack trigger cost less than \$10. Their conclusion: adversarial training is “a vital layer within a comprehensive defense-in-depth strategy” but “will not render the model immune” [3].

- An independent dynamic benchmark study tested 10 state-of-the-art defenses across prompting-based, filtering-based, alignment-based, and system-level categories and concluded that “nearly all existing defenses remain far from meeting the requirements for real-world deployment” [4].
- A systematic review of 128 peer-reviewed studies (2022-2025) found that input preprocessing achieves 60-80% detection rates, advanced architectural defenses up to 95% against known patterns, but “significant gaps persist against novel attack vectors” [5].
- An analysis of 45 key sources documenting real-world exploits concluded that prompt injection is “a fundamental architectural vulnerability requiring defense-in-depth approaches rather than singular solutions” [6].

This body of research leads to a central design principle that we believe should be central to any NIST guidance: AI agent frameworks must protect against agent outputs rather than attempt to trust them.

The parallel to network security is instructive. NIST SP 800-207 defines zero trust architecture as one where “no implicit trust is granted to assets or user accounts based solely on their physical or network location” and establishes that “trust in the requester is evaluated before the access is granted” on a per-request basis [7]. Before zero trust, organizations relied on perimeter defenses (firewalls, VPNs) to create a trusted internal network. Google’s BeyondCorp demonstrated that this model was fundamentally flawed: once the perimeter was breached, attackers had unrestricted lateral movement [8]. The solution was not better perimeter defenses, but eliminating the concept of a trusted perimeter entirely and authenticating every request independently.

AI agent security is at a similar inflection point. The current industry approach relies on “model perimeter defenses” (prompt engineering, detection classifiers, instruction hierarchy) to create a trusted agent output. The research cited above demonstrates that this perimeter is equally unreliable. The solution is the same: **eliminate the concept of trusted model output entirely, and treat every LLM output as untrusted input at every point where it touches infrastructure.**

In practice, this means applying the same discipline the security community already uses for other untrusted inputs, but systematically, at every stage of the agentic reasoning chain. A web application sanitizes user input before rendering it (XSS prevention). A database driver uses parameterized queries rather than trusting string concatenation (SQL injection prevention). An AI agent system must apply the same principle: when the LLM generates a database query, parse and validate it against an allowlist before execution. When it produces rendering code, execute it in a sandbox that cannot access the host application. When it requests data, scope the request to what the current user is authorized to see, enforced by the data layer, not by the model’s instructions. Each boundary independently validates what crosses it, regardless of where it originated. This is SP 800-207’s principle that “access to individual enterprise resources is granted on a per-session basis” [7] made concrete: the LLM is an untrusted client, and every resource it touches verifies independently.

Amplified blast radius through tool access. A traditional software vulnerability typically allows an attacker to do what the compromised component was already authorized to do. AI agent systems are different: a single compromised planning step can chain together multiple tools in novel sequences that were never anticipated by developers. An agent with read access to a database and write access to an output channel can be manipulated into exfiltrating data — not because any individual tool is misconfigured, but because the agent’s ability to compose tools creates emergent capabilities that exceed the sum of the individual tool permissions. ToolHijacker [9], the first prompt injection attack targeting tool selection, demonstrated 96.7% attack success in no-box settings by injecting malicious tool documents into

tool libraries, highlighting that the attack surface extends beyond prompt content to the tool ecosystem itself.

Trust boundary ambiguity. In traditional systems, trust boundaries are explicit: user input is untrusted, internal function calls are trusted. AI agent systems blur this because the model's output is simultaneously "internal logic" (it decides what to do next) and "influenced by external input" (it was shaped by user prompts and retrieved data). There is no clean architectural point where you can say "everything past here is trusted." This is precisely why the zero trust model, which per SP 800-207 assumes that "an attacker is present in the environment" [7], is the correct framework.

1(b) How threats vary by agent capability and deployment context

The security risk of an AI agent system is not primarily determined by the model's capability but by the agent's **effective action space**: the set of real-world state changes the agent can cause. A highly capable model with read-only access to scoped data, executing in a sandboxed environment with no outbound network access, has a narrow effective action space regardless of model sophistication. A less capable model with write access to production databases, the ability to send emails, and credentials to external APIs has a vastly larger effective action space — and correspondingly larger blast radius when compromised.

This means the most important variables for threat assessment are not model-level (parameter count, capability benchmarks) but system-level:

- **Tool access breadth:** Read-only vs. read-write. Internal-only vs. external communication. Single-tool vs. multi-tool composition.
- **Data scope:** Per-user scoped access vs. cross-tenant access. Infrastructure-enforced vs. model-instructed boundaries.
- **Execution environment:** Sandboxed and isolated vs. running in the application's security context with access to credentials and network.
- **Autonomy level:** Human-in-the-loop for consequential actions vs. fully autonomous operation.
- **Deployment context:** Single-tenant on-premises (constrained blast radius) vs. multi-tenant SaaS (cross-tenant risk) vs. consumer-facing (adversarial users).

NIST guidance that provides a structured methodology for analyzing effective action space, and calibrating security requirements proportionally, would be more useful than guidance organized around model capability alone. An agent that can only read scoped data and render sandboxed visualizations requires different controls than one that can send emails and modify files, regardless of the underlying model.

1(c) Security barriers to adoption

These security concerns are creating measurable barriers to enterprise adoption. In our experience working with enterprise customers, security review is the primary gate for procurement of AI agent systems. Organizations are reluctant to deploy agents that interact with production data when the security model depends on probabilistic model behavior. Providing deterministic, infrastructure-level security guarantees, the kind that security teams are accustomed to evaluating, is essential for moving AI agent adoption beyond experimentation into production enterprise use.

1(d) How threats have changed and are likely to evolve

Early AI agent systems (2023-2024) were largely experimental, with limited tool access, typically generating text responses with perhaps a web search or code interpreter. The threat surface was narrow.

The current generation (2025-2026) has substantially broader tool access: database queries, API calls, file system operations, code execution, and multi-step planning. We have observed a corresponding shift in the threat landscape:

- **Prompt injection has moved from theoretical to practical.** Published demonstrations now achieve reliable agent hijacking in realistic settings. NIST’s own research on agent hijacking evaluations documents this progression [10]. Anthropic’s pilot of Claude for Chrome with 1,000 users found a 23.6% attack success rate in autonomous mode, reduced to 11.2% after implementing site blocking, advanced classifiers, and mandatory confirmations — yet acknowledged prompt injection is “far from a solved problem” [17].
- **Multi-step attacks are emerging.** Rather than single-turn injections, attackers can plant payloads that activate only when the agent reaches a specific step in a multi-step plan, making detection significantly harder.
- **Supply chain risks are growing.** As agent systems increasingly rely on third-party tools, plugins, and sub-agents, each integration point becomes a potential injection surface. The model cannot distinguish between a legitimate API response and a crafted one containing adversarial instructions. ToolHijacker [9] demonstrated this concretely with attacks that manipulate the tool selection layer rather than the prompt itself.

Looking forward, we expect the most significant evolution to be in **cross-agent attacks** as multi-agent architectures become common. When agents delegate to other agents, a compromise in one agent can propagate through the entire system via the natural language messages they exchange — the equivalent of a worm that spreads through conversation rather than network protocols.

2. Security Practices

2(a) Technical controls that ensure or improve security

We implement security controls at all three levels identified in the RFI, and offer observations on their relative maturity and effectiveness:

i. Model-level controls:

We have found model-level robustness to prompt injection to be the least mature and least reliable category of control. While instruction hierarchy and system prompt prioritization improve baseline resistance, they remain probabilistic. No model-level defense currently provides the deterministic guarantees that security-critical systems require.

Detection-based approaches face a fundamental asymmetry. The first head-to-head comparison of open-source detection tools [11] found significant tradeoffs:

Tool	Recall	False Positive Rate	Tradeoff
LLM Guard	99.9%	12.7%	Catches everything but blocks legitimate use
Rebuff	98.1%	3.4%	Balanced, but LLM-based detection is itself vulnerable to injection
Vigil	83.8%	0%	Zero false positives, but 1 in 6 attacks gets through

The best-performing open-source classifier (ProtectAI DeBERTa) self-reports 99.9% accuracy but independent evaluations tell a different story: approximately 90% real-world accuracy per Knostic.ai [12], 28% FPR and 22% FNR in agentic settings per PromptArmor [13], and “significant over-defense which drastically diminishes utility” per AgentDyn [4]. Training data bias is the core limitation: it works well on similar prompts, poorly on novel attacks.

More promising approaches are emerging as defense-in-depth layers: PromptArmor achieved 0.56% FPR and 0.13% FNR on AgentDojo [13]; DataFilter reduced attack success rate to near-zero across multiple

benchmarks while maintaining utility [14]; CaMeL treats the LLM as an untrusted component with capability-based security, solving 77% of AgentDojo tasks with provable guarantees [15]. These are valuable additions to a layered defense, but none eliminate the threat. Detection raises the cost of attack; it does not provide deterministic guarantees. We treat model-level controls as defense-in-depth, never as primary security boundaries.

ii. Agent system-level controls:

This is where we concentrate our security investment. Given the research demonstrating that prompt injection detection is unreliable, our architecture treats every agent output as potentially adversarial. Rather than asking “is this output safe?” (a question we cannot answer reliably), we ask “even if this output is malicious, what damage can it cause?” and engineer the answer to be “none.”

We achieve this through layered infrastructure controls, each enforced by deterministic systems independent of the model. This directly implements SP 800-207’s tenet that “access to individual enterprise resources is granted on a per-session basis” and that “trust in the requester is evaluated before the access is granted” [7]:

- **Database-enforced access control.** Rather than relying on the AI agent to respect data access boundaries, we enforce access control at the database layer. Agent-generated operations execute within sessions scoped to the current user’s permissions. Even if the agent is manipulated into requesting unauthorized data, the database engine itself refuses to return it. This moves the trust boundary from the model (probabilistic) to the database engine (deterministic). In zero trust terms, the agent is an untrusted client; the database is the resource that authenticates and authorizes every request independently.
- **Execution environment sandboxing.** Agent-generated operations execute in a locked-down environment where unnecessary capabilities (extensions, file system access, outbound network calls) are disabled at the engine level. Resource limits prevent denial-of-service. Configuration is locked at initialization, preventing runtime tampering. The agent cannot escape the sandbox because the sandbox is enforced below the layer the agent can influence. This applies well-established sandboxing principles, the same approach used for running untrusted code in browser engines and container runtimes, to the AI agent context.
- **Allowlist-based output validation.** Agent-generated operations are parsed and validated before execution against a strict allowlist of permitted constructs. Operations that do not conform are rejected regardless of what the model produces. This treats the model’s output the same way a web application treats user-submitted form data: parse it, validate it against a strict schema, reject anything that doesn’t conform.
- **Isolated execution of agent-generated code.** When agents produce executable output (e.g., rendering logic), that code runs in a sandboxed execution context with no access to the host application, authentication state, or storage. Established sandboxing techniques (iframe sandboxing with restricted capabilities, containerized execution) provide well-understood, battle-tested mechanisms. The rendered output is a visual artifact with no side effects, not executable content in the application’s security context. Even if an attacker uses prompt injection to make the agent generate malicious code, the sandbox constrains its impact to the isolated environment.
- **Principle of least privilege for tool access.** Agent capabilities are restricted to the minimum required for their function (read-only data access, for example, with no ability to modify source data or access administrative functions). This is enforced at the API layer, not by instructing the model to avoid these actions. The distinction matters: instructions to a model are suggestions; API-level restrictions are guarantees.

- **Structured output channels.** Agent actions flow through typed, validated interfaces, not arbitrary text execution. The agent returns structured objects that are validated against schemas before being acted upon. This constrains the agent’s effective action space even if its reasoning is compromised.

The practical result: **these controls enable rather than restrict.** The goal is not to prevent AI agents from doing useful work; it is to let them operate freely within a well-defined safe boundary. When the sandbox is transparent and the controls are invisible to normal operation, agents and users have no incentive to work around them.

iii. Human oversight controls:

Infrastructure controls work well for constraining agent actions that are deterministically verifiable (data access, code execution, rendering). But agents increasingly operate in domains where infrastructure alone cannot fully constrain the impact: composing emails, generating customer-facing content, publishing analyses, or making recommendations that influence decisions. For these actions, the appropriate control is **mandatory human sign-off before the action takes effect.** The agent drafts; a human reviews and approves before the output is published, sent, or shared.

We implement this through explicit workflow states: agent-generated outputs are created in a draft state and must be reviewed and promoted by a human before they become visible to others or trigger external actions. This is not a blanket approval requirement; it is targeted at the boundary where agent output crosses from internal analysis to external impact. The infrastructure controls described above ensure that the agent cannot cause harm during the analysis phase; human oversight ensures that consequential outputs receive human judgment before they take effect.

This distinction matters because blanket approval requirements introduce their own security risks. Approval fatigue, where users routinely approve actions without review, is well-documented in security research and is specifically noted as a limitation of architectural approaches like CaMeL [15]. If users are asked to approve every agent action, the approval becomes a rubber stamp and provides no real security value. By confining human review to the consequential boundary (the moment an output moves from draft to published, from internal to external), the review is high-signal and meaningful rather than high-volume and exhausting.

2(e) Relevant cybersecurity frameworks

The most directly applicable existing framework is **NIST SP 800-207: Zero Trust Architecture** [7]. SP 800-207 establishes several tenets that map directly to AI agent security:

- *“No implicit trust is granted to assets or user accounts based solely on their physical or network location”* — Analogously, no implicit trust should be granted to agent outputs based solely on their origination from a model within the system boundary.
- *“Access to individual enterprise resources is granted on a per-session basis”* — Each resource an agent accesses (databases, execution engines, rendering environments) should independently authenticate and authorize every agent action.
- *“The enterprise monitors and measures the integrity and security posture of all owned and associated assets”* — Agent actions should be comprehensively logged and monitored, with the understanding that monitoring is a detection layer, not a prevention layer.
- *“The enterprise assumes that the internal network is hostile”* (paraphrasing the threat model) — The system should assume that the agent’s outputs may be adversarial at any time.

The precedent is well-established. Google’s BeyondCorp [8] demonstrated that moving access controls from the network perimeter to individual resources, authenticating and authorizing every request independently, produced a more robust security posture than any perimeter defense. The same transformation is needed for AI agent systems: move security controls from the model (the “perimeter”) to

the individual resources the agent accesses. Each resource independently validates, authorizes, and constrains every agent action.

The impediment to adoption is largely conceptual: development teams think of the AI model as an internal component (trusted) rather than as an external client (untrusted). Reframing the model as an untrusted actor operating within a zero trust perimeter is the most important mental shift for AI agent security.

NIST's Secure Software Development Framework (SP 800-218A) [16] is also relevant but would benefit from specific guidance on the unique supply chain challenges of AI agent systems, particularly the risks introduced by third-party tool integrations and model updates that can silently change agent behavior.

3. Assessing Security

3(a) Methods to anticipate, identify, and assess security threats during development

Pre-deployment assessment methods we employ:

- **Adversarial prompt testing.** Before deploying agent capabilities, we test with crafted inputs designed to trigger prompt injection, data exfiltration, and privilege escalation. This is analogous to penetration testing for traditional systems but requires domain-specific expertise in LLM behavior.
- **Tool-chain analysis.** We enumerate the sequences of tool calls the agent can make and identify combinations that could cause harm. This is similar to threat modeling but applied to the agent's action space rather than traditional attack surfaces.
- **Hermetic integration testing.** Every change to our agent system runs against isolated environments with the full security stack (access control policies, API restrictions, execution sandboxing, output validation) active. Tests specifically validate that security controls hold under adversarial conditions, not just that the happy path works.
- **Infrastructure control validation.** Because our security model relies on deterministic infrastructure rather than model behavior, we can write deterministic tests for it. We verify that sandboxed environments reject prohibited operations, that rendering contexts cannot access the host application, and that API-layer restrictions prevent unauthorized tool invocations. These tests are not probabilistic (unlike testing model robustness to prompt injection); they either pass or fail, which gives us high confidence in our security boundaries.

Post-deployment detection:

- **Comprehensive activity logging.** Every agent action (every operation generated, every tool invoked, every output produced) is logged with full context (user, session, input that triggered the action). This creates an audit trail that enables both real-time monitoring and forensic analysis.
- **Anomaly detection on agent behavior.** Unusual patterns in agent actions (accessing resources the user has never touched before, sudden changes in action complexity, actions that reference system metadata) can be surfaced through activity log analysis and AI-assisted review of agent behavior.
- **Pre-success detection of adversarial patterns.** The comprehensive activity log enables detection of exfiltration attempts and malicious activity patterns before they succeed. Real-world attacks against AI agent systems typically require multiple attempts: probing for data boundaries, testing what the agent can access, iterating on injection payloads. Analysis of activity logs, whether through automated workflows or AI-assisted review, can identify these patterns in aggregate, flagging suspicious sequences even when individual actions appear benign. This is a detection advantage that agent systems have over traditional software: the conversational nature of the interaction produces a rich behavioral trace that makes multi-step attacks observable.

Maturity assessment: Pre-deployment adversarial testing is still largely ad hoc across the industry, without standardized benchmarks or coverage metrics. However, testing infrastructure-level controls (sandboxing, access control, output validation) is well-understood from traditional software security and can use existing testing methodologies. This is another advantage of the “protect against outputs” approach: the security controls themselves are testable with deterministic methods, even when the model’s behavior is not.

3(b) Assessing the security of a particular agent system

The most useful framework for assessing a specific system’s security posture is analyzing the agent’s **effective action space** as described in Section 1(b). For each dimension (tool access breadth, data scope, execution environment, autonomy level, deployment context), the key question is not just *what can the agent do* but *where is the constraint enforced*: at the model layer (unreliable) or the infrastructure layer (reliable)?

Security assessments should be proportional to the effective action space. A system where the agent can only read scoped data and render outputs in an isolated context requires different controls than one that can send emails, call arbitrary APIs, and modify files. NIST guidance that provides a structured methodology for this analysis, mapping effective action space to required controls, would be valuable.

4. Deployment Environments

4(a) Constraining the agent’s deployment environment

The most effective constraint we have implemented is **enforcing security at infrastructure boundaries rather than within the agent itself**, the zero trust approach applied to every layer of the stack:

- **Database-level access control:** The agent operates within a session that cannot access unauthorized data. The constraint is enforced by the data layer, not by the LLM.
- **Execution environment sandboxing:** The execution environment for agent-generated operations disables unnecessary capabilities at the engine level. Configuration is locked at initialization, creating a tamper-resistant boundary below the layer the agent can influence.
- **Rendering isolation:** Agent-generated output executes in a sandboxed context (e.g., sandboxed iframes, isolated containers) with no access to the host application’s authentication, storage, or navigation capabilities. This leverages well-established browser and OS sandboxing primitives.
- **Network isolation:** The agent’s execution environment has no outbound network access beyond the specific services it requires. Data exfiltration is blocked at the network layer, not by model instructions.
- **API-level capability restrictions:** Tools available to the agent are explicitly registered; the agent cannot discover or invoke unregistered capabilities. New tool access requires code changes and security review, not a prompt update.

An agent instructed “never access other users’ data” provides no security guarantee. An agent operating within a session that enforces access control at the infrastructure layer provides a deterministic one.

4(b) Environment modifications to mitigate threats

Rollback and undo capabilities are critical but underdeveloped across the industry. Our current state:

- **Read-only data operations.** Restricting agents to read-only data access eliminates the need for rollback in the data access path. This is itself a design choice that constrains the agent’s effective action space.
- **Stateless output rendering** — sandboxed execution produces output with no persistent side effects. There is nothing to roll back because the execution environment cannot modify external state.
- **Reversible output creation** — agent-generated outputs support soft-delete and version history.
- **Transparent, auditable agent context.** A subtle but important threat is poisoning the agent’s persistent context: the accumulated knowledge, preferences, and domain-specific guidance that shape its behavior

across sessions. If this context is an opaque internal state, a compromised session can plant instructions that affect all future sessions. We mitigate this by making all agent context fully transparent and introspectable: every piece of context has a complete version history showing who added it and when, supports rollback to any prior state, and is visible for human review at any time. This transforms agent context from an invisible attack surface into an auditable, governable artifact.

- **Multi-step trajectory rollback remains an open challenge.** If an agent produces a flawed analysis based on a manipulated intermediate result, undoing the final output does not address the compromised reasoning chain. This is an area where further research and guidance would be valuable.

4(d) Monitoring deployment environments

We monitor through:

- **Full action audit trails** with structured logging of every agent decision and tool invocation
- **Real-time alerting** on anomalous patterns (unusual data access, unexpected tool sequences, high-frequency actions)
- **User-facing activity feeds** that surface what the agent did, enabling end-users to verify agent behavior

Challenge unique to AI agents: Traditional monitoring looks for known-bad patterns (malware signatures, known exploit payloads). AI agent monitoring must also detect **semantically anomalous behavior** — an agent doing something that is technically permitted but contextually inappropriate. This requires understanding the agent’s intent, not just its actions, which is a fundamentally harder problem than signature-based detection. However, the conversational nature of agent interactions produces a rich behavioral trace — adversarial attempts typically require multiple probes, creating observable patterns that log analysis can surface (as described in Section 3(a)).

When infrastructure controls are properly implemented, monitoring becomes a detection layer for improving response time rather than a primary security control — the infrastructure constraints limit damage regardless of whether anomalous behavior is detected promptly.

5. Additional Considerations

5(a) What would aid rapid adoption of security practices

The single most impactful resource NIST could provide is a **reference architecture for secure AI agent deployment** built around the principle of **protecting against agent outputs rather than trusting them**.

Many development teams currently rely on prompt-level instructions (“do not access unauthorized data”) or detection classifiers as their primary security control. The research record demonstrates these are insufficient: all published detection defenses have been bypassed by adaptive attackers [2]. This mirrors the pre-zero-trust era of network security, when organizations relied on VPNs and firewalls to create a trusted internal network — a model that advanced persistent threats proved fundamentally inadequate.

A clear, authoritative NIST publication establishing that:

1. **Agent outputs must be treated as untrusted** — analogous to how user input is treated as untrusted in web application security, and how network traffic is treated as untrusted in zero trust architectures per SP 800-207 [7]
2. **Security controls must be enforced at the infrastructure layer** (databases, query engines, rendering environments, network boundaries), not at the model layer
3. **Each resource the agent accesses must independently validate and authorize every action** — the zero trust model applied to AI agents
4. **Detection and model-level defenses are valuable defense-in-depth layers** but must never be the primary security boundary

...would significantly raise the baseline across the industry. The mental model shift from “make the agent trustworthy” to “make the environment safe regardless of what the agent does” is the most important conceptual contribution NIST can make.

Critically, this framing is **enabling rather than restrictive**. When security is enforced at the infrastructure layer and is invisible to normal operation, agents can operate freely within well-defined safe boundaries. Users and agents have no incentive to circumvent controls they never encounter during legitimate use — the same insight that made zero trust networking successful.

5(b) Where government collaboration is most urgent

The most urgent need is standardized evaluation frameworks for AI agent security. Currently, every organization assesses their agent security ad hoc, with no common benchmarks, test suites, or coverage metrics. NIST is uniquely positioned to develop shared evaluation methodology. Just as NIST’s cryptographic validation programs (CAVP/CMVP) provide standardized security evaluation for cryptographic modules, an **AI Agent Security Validation Program** could evaluate both layers:

- **Infrastructure controls** (deterministic, testable): Does the system enforce data access control at the infrastructure layer rather than the model layer? Is the execution environment sandboxed? Are agent-generated outputs rendered in isolated contexts using established sandboxing techniques? Is outbound network access restricted at the infrastructure level? These can be evaluated with traditional security testing methods.
- **Model-level controls** (probabilistic, defense-in-depth): How robust is the system to adversarial prompt injection? What detection mechanisms are in place? These should be evaluated with standardized adversarial benchmarks, with the explicit understanding that no passing score guarantees security, which is why the infrastructure layer must be the primary control.

An equivalent program for AI agent security would accelerate industry maturity significantly and help organizations understand which layer of their security stack actually provides guarantees versus which layer provides probabilistic resistance.

Beyond evaluation frameworks, we identify several open challenges where the field lacks established practices and NIST guidance would be particularly valuable:

Agent identity and caller context propagation. When an AI agent accesses a resource, it acts on behalf of a specific user, and the security constraints applied to that action must reflect that user’s permissions, not the agent’s. This is straightforward for single-agent systems, but becomes a critical design challenge as agents delegate to other agents or call external services. The calling user’s identity and authorization context must propagate through the entire chain — if Agent A delegates to Agent B, Agent B must operate under Agent A’s caller’s permissions, not its own. Systems that do not propagate caller context through delegation chains will inevitably create privilege escalation paths. NIST guidance establishing that role-based and policy-based access control (RBAC/PBAC) must be foundational to agent architectures — designed in from inception, not bolted on — would prevent a generation of insecure multi-agent systems.

Multi-agent trust boundaries. When agents communicate via natural language, every message between them is a potential prompt injection vector. An outer agent delegating a task to an inner agent is, architecturally, passing untrusted input across a trust boundary — but most multi-agent frameworks treat inter-agent messages as trusted internal communication. NIST guidance on trust propagation models for multi-agent systems, including how to authenticate inter-agent messages and constrain delegation authority, would address a gap that the industry is only beginning to recognize.

Model supply chain and behavioral drift. AI agent systems depend on third-party models that change without notice. A model provider ships an update, and the agent’s behavior — including security-relevant behavior — changes silently. There is no changelog, no semantic versioning, no way to diff behavioral changes between model versions. New model versions may be stronger on some tasks but regress on

others, reasoning depth can vary as providers optimize for cost, and capabilities that worked reliably may degrade. SP 800-218A [16] addresses AI supply chain at a high level, but specific guidance on model versioning, behavioral change detection, and regression testing for AI agent systems would help organizations manage a risk that is currently invisible to most.

Incident response for AI agent systems. Existing incident response frameworks (NIST SP 800-61, NIST CSF) provide well-established methodologies for detection, containment, eradication, and recovery. These frameworks apply directly to AI agent incidents and should not be reinvented, but they need adaptation for agent-specific scenarios. When an agent is compromised via prompt injection, what constitutes “containment”? When a model update causes behavioral regression, how does an organization “eradicate” a threat that is embedded in a third-party service? How do you conduct forensics on a probabilistic system whose outputs are non-deterministic? NIST guidance that extends existing incident response frameworks to cover these agent-specific scenarios would give organizations a practical playbook built on foundations they already understand.

QueryStory, Inc. appreciates the opportunity to contribute to NIST's work on AI agent security. We are available to provide additional technical detail, participate in workshops, or contribute anonymized case studies from our production deployment experience.

References

- [1] OWASP, “Top 10 for Large Language Model Applications,” 2025. <https://owasp.org/www-project-top-10-for-large-language-model-applications/>
- [2] M. Nasr, N. Carlini, et al., “The Attacker Moves Second,” arXiv:2510.09023, Oct 2025. Researchers from OpenAI, Anthropic, and Google DeepMind. <https://arxiv.org/abs/2510.09023>
- [3] Google DeepMind, “Lessons from Defending Gemini Against Indirect Prompt Injections,” arXiv: 2505.14534, May 2025. <https://arxiv.org/abs/2505.14534>
- [4] “AgentDyn: A Dynamic Open-Ended Benchmark,” arXiv:2602.03117, Feb 2026. <https://arxiv.org/abs/2602.03117>
- [5] “Prompt Injection Attacks on Large Language Models: A Survey,” Computers, Materials & Continua, Dec 2025. Systematic review of 128 peer-reviewed studies (2022-2025). <https://www.techscience.com/cmc/online/detail/25276>
- [6] “Prompt Injection Attacks in LLMs and AI Agent Systems: A Comprehensive Review,” MDPI Information, vol. 17, no. 1, Jan 2026. Analysis of 45 key sources. <https://www.mdpi.com/2078-2489/17/1/54>
- [7] NIST, “Zero Trust Architecture,” NIST SP 800-207, Aug 2020. <https://csrc.nist.gov/pubs/sp/800/207/final>
- [8] R. Ward and B. Beyer, “BeyondCorp: A New Approach to Enterprise Security,” ;login:, vol. 39, no. 6, Dec 2014. <https://research.google/pubs/pub43231/>
- [9] “ToolHijacker: Prompt Injection Attack Targeting Tool Selection,” NDSS 2026. First prompt injection attack targeting tool selection, achieving 96.7% attack success in no-box settings. <https://arxiv.org/abs/2504.19793>
- [10] NIST, “Technical Blog: Strengthening AI Agent Hijacking Evaluations,” Jan 2025. <https://www.nist.gov/news-events/news/2025/01/technical-blog-strengthening-ai-agent-hijacking-evaluations>
- [11] “Enhancing Security in LLM Applications: A Performance Evaluation of Early Detection Systems,” arXiv:2506.19109, Jun 2025. Comparison of LLM Guard, Vigil, and Rebuff. <https://arxiv.org/abs/2506.19109>
- [12] Knostic.ai, “Revolutionizing Prompt Injection Detection,” 2025. Independent evaluation of ProtectAI DeBERTa. <https://www.knostic.ai/blog/revolutionizing-prompt-injection-detection-a-leap-to-99-accuracy>
- [13] “PromptArmor,” arXiv:2507.15219, Jul 2025. Submitted to ICLR 2026. <https://arxiv.org/abs/2507.15219>
- [14] “DataFilter,” arXiv:2510.19207, Oct 2025 (revised Feb 2026). Fine-tuned filter model, open source on HuggingFace. <https://arxiv.org/abs/2510.19207>
- [15] “CaMeL: Capability-based Security for AI Agents,” arXiv:2503.18813, Mar 2025. Google DeepMind. <https://arxiv.org/abs/2503.18813>
- [16] NIST, “Secure Software Development Practices for Generative AI and Dual-Use Foundation Models,” NIST SP 800-218A, 2024. <https://csrc.nist.gov/pubs/sp/800/218/a/final>
- [17] Anthropic, “Piloting Claude for Chrome,” Aug 2025. <https://www.anthropic.com/news/claude-for-chrome>
- [18] OpenAI, “Continuously Hardening ChatGPT Atlas Against Prompt Injection Attacks,” Dec 2025. <https://openai.com/index/hardening-atlas-against-prompt-injection/>