Security Boundaries in AI Operating Systems: Prompt Injection, Data Isolation, and the Protection of Execution Authority

November 2026

Abstract

An enterprise agent must read information from customers, suppliers, documents, and external services while retaining the authority limits established by the business. This creates a security problem when instructions embedded in that material influence tool use or access to protected data. This paper develops a threat model for an AI operating system that connects information retrieval with consequential business actions. It examines indirect prompt injection, tenant separation, credential use, persistent memory, and approval integrity. Drawing on published agent security research and database access control documentation, it proposes layered controls at the boundaries between content, interpretation, authorisation, and execution. The contribution is an operational evaluation design that measures whether an attack reaches a protected effect, including delayed and indirect effects, while also measuring legitimate work blocked by the controls. MatruOS provides an architectural context. The paper reports no penetration test or product assurance result. Its conclusion is that containment must remain effective even when an agent interprets hostile material incorrectly.

Keywords: agent security, prompt injection, tenant isolation, execution authority, data protection, security evaluation.

1. The security consequence of reading external material

An accounts assistant receives a supplier document that includes payment information and a request to send internal account details to an unrelated destination. The document is relevant to the user’s task, but the embedded instruction does not come from the person who authorised the task. If the agent treats all text as equivalent instruction, reading the document can redirect the workflow.

The problem is not limited to visibly suspicious wording. A document can claim to contain a required verification step, a tool response can assert that a different service must be contacted, or a retrieved note can appear to be an internal policy. The attacker’s opportunity is to place content where the agent expects useful evidence.

An AI operating system increases the importance of this distinction because the same environment may connect customer information, inventory, finance, and communication tools. Integration creates useful operational context, but it can also create paths from an untrusted input to a protected action. Each path needs an explicit security boundary.

The research question is how an enterprise AI operating system can preserve authorised intent and data separation when agents process untrusted material. The focus is defensive architecture and evaluation. The examples describe attack conditions without providing instructions for compromising a real system.

2. Evidence from agent security research

AgentDojo evaluates agents using tools over untrusted data in an environment that supports prompt injection attacks and defences. It demonstrates why security assessment needs to examine the interaction between legitimate tasks, retrieved content, and attempted redirection. A benchmark result is evidence about its tested setting, not a guarantee about a business deployment. [1]

InjecAgent examines indirect prompt injection in agents that use tools, including attempts to cause harmful actions or extract private information. Its research design supports separating attacker objectives and evaluating how external content influences the agent's behaviour. [2]

NIST's zero trust guidance provides a broader access control principle: resource access requires explicit decisions rather than assumed trust from location. For an enterprise agent, this means that content retrieved from an internal system and actions initiated inside the company still need appropriate verification and permission checks. [3]

PostgreSQL's row security documentation describes controls that restrict row access and modification, together with important exceptions involving owners and privileged roles. It illustrates why enabling a feature is only one part of establishing effective isolation. The actual execution role and policy configuration matter. [4]

This paper synthesises these foundations into a threat model for integrated business execution. Its proposed controls and tests are architectural recommendations to be evaluated. No claim is made that prompt injection can be eliminated solely through model instructions or that the framework constitutes a security certification.

3. Assets, actors, and protected boundaries

The protected assets include business records, credentials, approval decisions, financial limits, and the integrity of customer commitments. Some assets require confidentiality; others require protection against unauthorised modification or misleading use. A correct invoice sent to the wrong recipient is a different failure from an incorrect invoice sent to the right recipient.

The threat model assumes that an attacker can influence some material the agent reads, such as an incoming message, uploaded document, or external page. It does not assume control of the application's authorisation service. Separate tests are needed for compromised accounts, privileged insiders, and infrastructure vulnerabilities.

The main boundaries are between user instruction and external content, one tenant and another, a retrieved fact and an executable policy, a proposed action and an approved action, and the application's internal state and external destinations. An implementation should identify every tool path crossing those boundaries.

The analysis uses a consequential effect as the unit of security outcome. An agent producing suspicious text is relevant, but the central question is whether protected information was disclosed, a record was altered, authority was expanded, or an unauthorised commitment was made. Controls may contain an incorrect model interpretation before it creates such an effect.

4. Content must retain its source and status

4.1 Preserving the instruction boundary

The system should distinguish the user's request from material retrieved to fulfil it. Tool outputs and documents should retain their origin and trust status. This supports interpretation, but the boundary must also be enforced through constrained actions because a model can still misread the material.

For example, an instruction found inside a supplier invoice should not authorise a change to payment destination. The relevant service should require a separate approved process for that change. The document can supply evidence for review, while authority remains with the established workflow.

An agent may legitimately need to follow instructions contained in a document, such as assembly steps or a customer's requested delivery details. The application must define which fields the workflow is allowed to extract and act upon. Treating all document instructions as forbidden would block useful work; treating them all as authority would create an uncontrolled channel.

4.2 Data minimisation during retrieval

The agent should receive the information required for the task within its authorised scope. Retrieval should apply tenant and resource permissions before returning content. A final response filter is insufficient if protected information has already been sent to an external model or tool.

Search indexes, document stores, caches, and analytical replicas all need consistent access rules. An application can correctly protect its primary database while leaking information through a search result or cached summary. Isolation testing should therefore follow every retrieval route used by the agent.

The system should also consider derived information. A summary of another customer's terms can disclose protected content even when the original document remains hidden. Derivative records need a defined relationship to source permissions and a procedure for responding to changes in access.

4.3 Persistent memory as a security surface

An agent may store an extracted instruction or summary for later use. If untrusted content becomes an approved policy through that process, an attack can survive beyond the original session. The memory lifecycle should separate captured material from validated and authorised knowledge.

A change to executable policy should identify its source, approver, and scope. Routine summarisation should not activate a new instruction. Later workflows should be able to inspect how a rule entered the system and suspend it if its origin is disputed.

This creates a temporal security requirement. Testing only the immediate task may miss a harmful instruction stored for future retrieval. Evaluation should include a later session that encounters the stored material and attempts a related action.

5. Protecting execution and credentials

5.1 Constrained action services

Agents should call services with defined inputs and business operations. A service that creates a draft quotation can enforce different permissions from one that confirms an order. Broad tools that execute

arbitrary queries or unrestricted network requests require particular scrutiny because they can bypass intended workflow boundaries.

Input validation should establish entity identity, permitted fields, and business scope. The application should derive sensitive context such as tenant identity from authenticated state where possible. A tenant identifier supplied by the model should not be accepted as sufficient proof of access.

Allowing a tool does not mean allowing every parameter combination. A messaging tool may be permitted to contact an identified customer about an existing order while being prohibited from sending internal records to an arbitrary address. Destination and content scope belong in the permission decision.

5.2 Credential handling

Credentials should be managed by the application and exposed to tools only within the necessary scope. Placing reusable secrets in model context increases the consequences of a disclosure or malicious instruction. The agent can request an authorised operation without needing the credential value itself.

Service accounts need explicit ownership and review. A background agent should not inherit broad administrative permissions merely to simplify integration. Where a tool requires elevated access, the application should isolate that capability and enforce the business conditions before invoking it.

Revocation must cover active jobs and queued work. Removing access from an interface is insufficient if a worker retains credentials that remain valid. The implementation should define how authority changes propagate and how pending operations are checked before execution.

5.3 Approval integrity and external destinations

An approval should bind to the action the reviewer actually saw, including recipient, amount, data scope, and transaction version. If an agent changes those fields after approval, the service should require the applicable revalidation. This prevents approval of an innocuous proposal from authorising a materially different effect.

External destinations deserve explicit controls. A workflow may need to send documents to a customer, but that permission should be tied to a verified business relationship and authorised purpose. Newly supplied addresses, redirected endpoints, or altered account details should follow a separate validation process where the consequence warrants it.

These controls also need usable explanations. A blocked action should identify the unresolved condition without disclosing protected information. The user should be able to correct a genuine mistake through the proper process rather than being encouraged to find a workaround.

6. Tenant isolation in an integrated environment

A platform serving several businesses must keep their records and authority distinct across every component. The tenant scope should travel through the request, job queue, storage access, retrieval, and audit record. Missing scope should produce a defined failure, not a default tenant chosen for convenience.

Row security can contribute to database isolation, but privileged execution paths require inspection. [4] Tests should use the actual application roles and background services. Testing only a restricted user in an interactive session does not establish the behaviour of administrative jobs or connectors.

Shared caches should include the relevant authorisation scope in their design. A response generated for one user may contain material another user cannot access. Reuse must account for permissions, data changes, and

the purpose of the original request. An apparently harmless performance optimisation can otherwise cross a confidentiality boundary.

Exports, logs, and support tools also belong in the threat model. A support employee may need to investigate an error without receiving unrestricted customer data. The platform should provide scoped diagnostic information and record exceptional access. Isolation is an operational responsibility extending beyond the main application screen.

7. Worked scenario and MatruOS application

Consider an illustrative MatruOS workflow that prepares a supplier payment proposal from an invoice and a purchase record. The invoice contains a request to use a new bank account and to forward a ledger extract for “verification.” The purchase record contains the approved supplier and established payment details.

The agent may extract the invoice amount and identify the discrepancy. The payment service should require the approved account or a separately authorised change. The communication service should reject disclosure outside the permitted purpose and recipient scope. The user receives a clear exception explaining which information requires verification.

Suppose the model nevertheless proposes sending the ledger. The security outcome depends on whether the service permits that effect. A constrained tool should prevent the disclosure and record the denied attempt. The workflow can then pause or continue with the permitted portion of the task according to policy.

If the agent stores a summary saying that this supplier “requires ledger verification,” that summary should remain unapproved source information. A subsequent workflow should not treat it as policy. This case connects immediate containment with memory controls and later evaluation.

The proposed architecture should also test whether the same attack behaves differently when delivered through an email, a scanned invoice, or a tool response. Different ingestion paths may preserve different metadata. The implementation needs evidence that the boundary holds across the routes it actually supports.

8. A security evaluation protocol

Build a test environment with synthetic businesses, users, suppliers, documents, and external destinations. Define the legitimate task and attacker objective independently. The legitimate task might be preparation of a payment proposal; the attacker objective might be disclosure to an unauthorised recipient or modification of protected payment details.

Measure completed legitimate tasks, attempted unauthorised actions, blocked actions, and protected effects that actually occurred. Also record false blocking of permitted work. These measures should remain separate because a system can achieve containment by refusing every task, which would not establish useful operation.

Tests should vary document placement, wording, file type, tool output, and timing. Include attacks that affect stored memory, queued work, or a second session. Repeat trials because one successful refusal does not establish consistent behaviour. Preserve the model version, tool configuration, policy version, and test inputs.

The evaluator should inspect state and external effects independently of the agent’s own account. A response claiming that no data was sent is not proof. The test harness should observe the communication endpoint, modified records, and audit events. Any mismatch between reported and observed action should be recorded.

Testing should also exercise privileged service roles, cross tenant identifiers, cache reuse, and export routes. These are application security tests as well as model tests. Results should identify whether a failure originated in interpretation, permission enforcement, identity propagation, or an external integration.

9. Limitations and conclusion

The proposed threat model covers attacker controlled content interacting with an otherwise functioning authorisation layer. It does not replace broader security engineering, infrastructure review, or incident response preparation. An attacker with privileged administrative access can create different failure paths.

No collection of test cases proves the absence of future attacks. Results should state the covered workflows, attacker capabilities, and configuration. Changes to tools, retrieval, memory, or models can change the attack surface and require targeted reassessment.

An AI operating system should preserve a clear boundary between information encountered during work and authority to act. Source attribution helps interpretation, while application controls protect resources when interpretation fails. Tenant isolation, credential scope, approval binding, and memory governance must operate together.

The proposed evaluation focuses on protected effects and legitimate work, including delayed consequences. This provides a practical basis for assessing whether an integrated agent environment contains untrusted instructions without making ordinary business operation unusable.

Author affiliation and disclosure

The author is affiliated with Code Colonies Private Limited, Ahmedabad, and leads the MatruOS team. The paper proposes security requirements and tests; it does not report an independent security assessment of MatruOS.

References

[1] Debenedetti, E., et al. (2024). AgentDojo: A Dynamic Environment to Evaluate Prompt Injection Attacks and Defenses for LLM Agents. arXiv:2406.13352, version 3.

https://arxiv.org/abs/2406.13352v3

[2] Zhan, Q., Liang, Z., Ying, Z., and Kang, D. (2024). InjecAgent: Benchmarking Indirect Prompt Injections in Tool-Integrated Large Language Model Agents. arXiv:2403.02691, version 3.

https://arxiv.org/abs/2403.02691v3

[3] Rose, S., Borchert, O., Mitchell, S., and Connelly, S. (2020). Zero Trust Architecture. NIST Special Publication 800-207.

https://csrc.nist.gov/pubs/sp/800/207/final

[4] PostgreSQL Global Development Group. PostgreSQL 18 Documentation. Section 5.9, Row Security Policies.

https://www.postgresql.org/docs/18/ddl-rowsecurity.html