Nikxius Inc.

NIKXIUS RESEARCH · SEPTEMBER 2026

Agentic AI Safety: Prompt Injection, Authority Boundaries and Uncertain Outcomes

A proposed action-level safety contract for admission, revocation and provider-specific recovery

REPORT
NX-RESEARCH-AUTHORITY-2026-01

DATE
2026-09-26

VERSION
1.0

CLASSIFICATION
PUBLIC

ABSTRACT

Prompt-injection defenses must be complemented by controls over what an agent can cause outside the model. This paper analyzes exact-action authority, atomic admission, revocation cutoffs, native deferred approvals and uncertain outcomes, with fair comparison to OAuth, AWS and Stripe.

Contents

Abstract

An agent can produce an unsafe tool request even when it authenticates correctly. A valid approval can become stale before execution. A timed-out write can have succeeded, and a later revocation may not stop work already accepted by a provider. These are different safety problems. This paper proposes an action-level contract that separates untrusted model output, verified identity, exact authority, admission and external outcome. We examine how the contract limits consequences of prompt injection without claiming to eliminate injection itself. A hypothetical refund timeline shows why short-lived local authority, native deferred approval and provider idempotency must be interpreted together. OAuth, MCP, AWS AgentCore and Stripe already provide substantial relevant mechanisms; the remaining task is to establish their combined behavior for the specific operation. The analysis is a proposed engineering framework, not a report of a deployed authority server, a new protocol or a completed security validation.

Nikxius Research · 26 September 2026 · Version 1.0 · Prepared with AI assistance. This technical working paper has not been peer reviewed.

Research questions and method

We address three questions. Which boundaries must remain effective when an agent follows malicious instructions? At what point can a system truthfully say that approval, expiry or revocation prevents an action? What evidence permits safe recovery when the provider's outcome is uncertain?

The method is a selective analysis of primary documentation refreshed on 26 September 2026: OWASP prompt-injection guidance, OAuth RFCs, the dated MCP authorization specification, AWS AgentCore policy documentation and Stripe's approval, MCP, idempotency and error contracts. Relevant source sections were inspected directly. Documentation establishes the published mechanism and its stated limits. It does not establish availability in every account, correct configuration, independent security assurance or successful integration.

The refund scenario is a **hypothetical worked example**. No payment, provider acceptance test, customer deployment, attack experiment or new production observation was performed for this paper. It extends the earlier When AI Gets Write Access analysis by concentrating on admission, revocation and deferred provider effects. The regression and governance evidence companion papers address different questions; their public synthetic response tests do not validate the execution guarantees proposed here.

Prompt injection changes intent; authority controls consequences

OWASP distinguishes direct prompt injection from indirect injection through external material such as websites or files. Its guidance recommends measures including input/output controls, privilege restriction, human review and adversarial testing. It does not present retrieval augmentation or fine-tuning as complete prevention.[1]

A support agent may need to read a customer's message in order to do its job. That message is task data, not authority to change the application's security policy. The same is true of retrieved documents, tool descriptions and prior model output. If any of those sources can instruct the agent to widen its own permissions, the application has allowed untrusted content to cross a control boundary.

Prompt defenses can reduce the chance of a malicious request. An action boundary instead asks whether the requested effect is permitted even if the request was maliciously produced. Both matter. A well-enforced refund ceiling can still permit an unwanted refund within that ceiling; it cannot determine whether every permitted business decision is wise. Conversely, an apparently well-behaved model with unrestricted credentials can exceed a policy whenever its behavior fails.

The proposed design therefore treats model-authored action arguments as untrusted input. Identity, customer-to-payment relationships, policy, approval and current resource state must come from appropriate trusted sources. Natural-language explanations can help a reviewer understand the proposal, but they cannot supply the authorization facts they merely assert.

Distinguish identities before granting authority

At least five identities can participate in an action: the agent workload, the application operating it, a delegating user or human approver, the execution component, and the downstream credential. They should not be merged into one display label.

For example, the customer receiving a refund is normally the affected subject, not the employee authorizing it. An argument such as `behalfOf: customerId` proves neither consent nor delegation. A verified delegation needs an authenticated origin, an identified actor, accepted scope and lifetime. An autonomous service action may legitimately rely on service authority instead; it should not invent a human principal to make the event appear user-approved.

OAuth already supplies important foundations. RFC 9700 addresses replay, privilege restriction and audience/sender constraints. RFC 9396 defines structured rich authorization requests and includes fine-grained authorization data. RFC 8693 provides token exchange with actor and subject semantics. It is inaccurate to say that OAuth can carry only coarse permissions or cannot express transaction detail.[2][3][4]

However, carrying a field is not the same as enforcing it. The resource server or a trusted intermediary must interpret the relevant authorization details and reject disallowed requests. An exact amount in an unexamined token claim does not limit a provider. An audience-bound token limits where a token is accepted; it does not independently establish that a particular refund is appropriate.

The inspected MCP revision requires servers to validate that presented tokens were issued for their use and prohibits token passthrough. Those are meaningful security boundaries. They do not eliminate the MCP server's responsibility to implement its business rules and recovery behavior.[5][6]

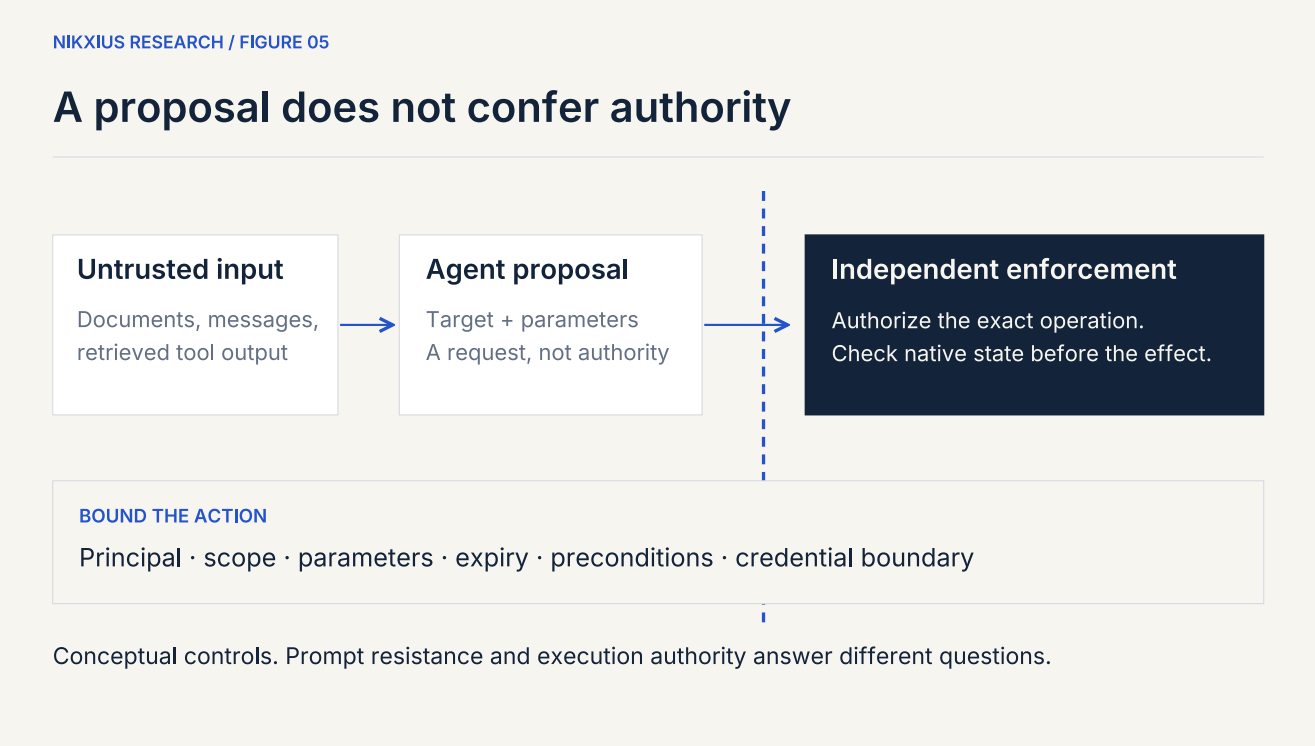
A proposed authority intersection

For one consequential action, we define effective authority as the intersection of independently established bounds:

```
workload rights ∩ application rights ∩ delegation ∩ current policy ∩ issued grant
∩ executor ceiling ∩ downstream permissions
```

An inapplicable delegation term is omitted for an explicitly authorized service action; an unknown required term is not treated as unrestricted permission. This expression is a design model, not a new interoperable token format.

An exact operation should bind the action type and version, tenant, account and environment, target resource, parameters, relevant business context, policy and approval references, validity and logical operation identity. A refund example needs an integer amount and currency, the correct payment/account mapping and whatever eligibility rules the organization actually requires. The agent must not select trusted values merely by placing them in its request.



Untrusted documents and tool output can shape an agent proposal, while independent enforcement authorizes the exact operation and checks native state before an effect.

Figure 1. Proposed independent enforcement boundary. Its preventive value depends on the agent lacking an alternate route to equivalent provider authority and on the executor and policy sources remaining within their stated trust assumptions.

A mediated executor can keep the provider credential out of model context. That is useful only if the agent also cannot read the credential file, access the secret broker, mint an equivalent token or call a less restrictive endpoint. A wrapper beside an all-powerful agent is not a meaningful security boundary.

The executor remains a privileged component. A compromised executor may misuse its credential or misreport observations. Narrow native permissions, host isolation, constrained egress and independent customer limits reduce its power. A signature on an instruction does not sandbox malicious executor code, and a receipt signed by that executor does not independently establish the provider's truth.

Admission is a specific event, not a synonym for execution

We propose an explicit **admission point** at which the service makes its final decision to accept one logical operation. In a suitable architecture, one authoritative transaction checks current policy, delegation, approval, validity and any promised

budget; consumes the one-use grant; and records the immutable execution ticket. Concurrency and revocation rules must refer to that transaction, rather than to an imprecise statement that checks happen “before execution.”

The executor then persists the ticket and original provider request identity before a possible network send. Recording possible dispatch before network I/O allows recovery to recognize ambiguity after a crash. It also has a cost: a crash after that record but before the actual send may leave an operation unresolved even though nothing happened. A conservative design can sacrifice availability to avoid an unsupported duplicate write.

Keep authority state separate from execution knowledge:

Record	Meaning	What it does not establish
Granted	A bounded authorization exists	Current admission checks will pass later
Admitted	One logical operation passed the final admission rule	A provider request was sent or succeeded
Possibly dispatched	The process crossed the durable possible-send boundary	The provider received or applied the request
Pending provider approval	The provider reports a known approval lifecycle	The action has completed or will respect local cancellation
Confirmed effect	Sufficient operation-specific evidence establishes the stated effect	Every later business outcome or settlement is complete
Confirmed no effect	Sufficient evidence establishes no effect for this operation	A generic timeout or empty search would have been enough
UNKNOWN	Available evidence does not settle the outcome	Failure, safety to repeat, or successful rollback

One-use authority means one admission of a logical operation, not necessarily one transport attempt. Report delivery and read-only reconciliation may be retried without authorizing a new effect. Whether a mutating request may be repeated under the original provider key is a separate, provider-specific rule.

Expiry and revocation have a cutoff

A grant expiry can stop new admissions when checked by a trusted clock. It does not undo an admitted operation, revoke every derived credential or cancel work retained by another system. The truthful interface must say where the prevention guarantee ends.

Consider a revocation racing admission. If revocation commits first under the documented serialization rule, admission should be denied. If admission commits first, the result must disclose that work was already admitted and cancellation is not guaranteed. Both events should remain recorded. Rewriting the operation as “revoked” after an external effect would conceal what happened.

Token exchange has similar limits. RFC 8693 explicitly says exchange does not create a tight lifecycle linkage between input and output tokens; propagation of revocation is not a general property of that protocol.[4] Short lifetime narrows exposure, but does not establish one-use execution, exact object constraints or immediate termination of an already accepted action.

Clock and policy assumptions also need examination. Signed expiry is not permission to add an undisclosed grace period. A measured skew allowance should make dispatch more conservative where necessary. A newly broadened policy must not silently widen an already approved operation. If the policy or prepared action changes materially, the design should require an explicitly defined reauthorization step rather than stretching old approval to fit new work.

Worked example: two native approval lifecycles

Imagine a support application requesting a USD 84.20 refund under a proposed local grant valid for ten minutes. The amount and lifetime are illustrative design values, not provider guarantees. The application has verified the payment mapping and admitted the exact request. Stripe then requires a native approval.

The inspected Stripe documentation describes two distinct paths. For certain actions through MCP using user credentials, the agent receives a confirmation URL; approval provides a token and the agent must retry the operation. An unapproved action expires after 24 hours. Separately, account approval rules can interrupt an action, create an approval request and complete the action automatically when a reviewer approves. Requests not approved within 14 days expire. These are documented public-preview behaviors and should be verified for the actual account and API integration.[7][8]

Time in the hypothetical account-approval path	Observation	Required interpretation
T0	Local grant is issued	Local authority has bounded admission validity
T1	Current checks pass and the request is admitted	Grant consumption survives later uncertainty
T2	Provider creates a pending approval request	Record its identity and known pending state; do not report a completed refund
T3	Local grant expires or is revoked	New local admission stops; provider cancellation is a separate question
T4	Provider reviewer approves the pending request	Native documentation allows automatic execution; local expiry alone cannot be advertised as preventing it
T5	Outcome event or authoritative query is received	Reconcile the original operation and record the specific confirmed state

The key problem is semantic composition. A local promise that “nothing can happen after ten minutes” is stronger than the account-approval path supports unless an effective downstream cancellation or execution-time check has been established. The truthful promise may instead be “no new local admission after ten minutes,” with a clearly disclosed provider-pending lifecycle.

The application should not respond to the pending state by creating another refund request. Nor should it treat the MCP token-and-retry flow as equivalent to native automatic execution. The connector must model the actual path, retain native identifiers and preserve uncertainty when cancellation or outcome cannot be established. This is a requirement derived from the published contracts, not a claim that a Nikxius server has implemented or passed it.

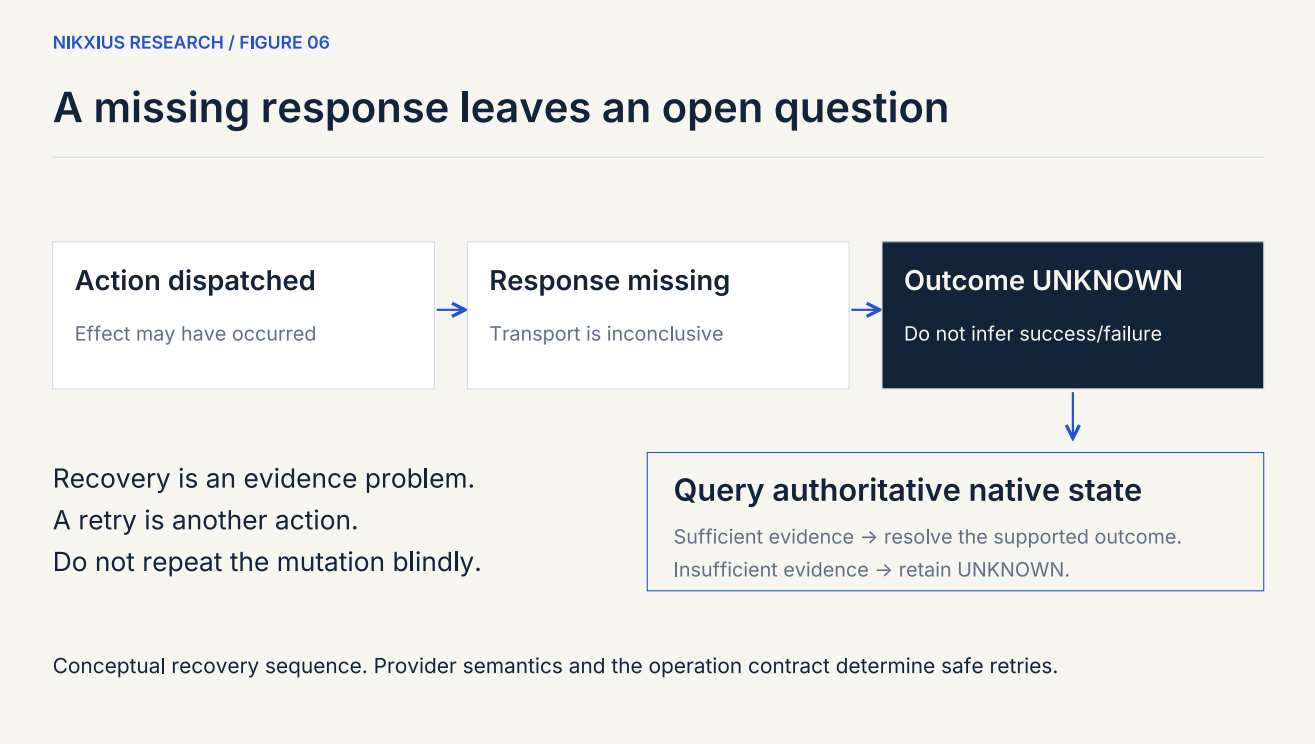
Idempotency, retries and UNKNOWN

Stripe’s idempotency documentation states that it stores the first result for a key after execution begins, including errors, and compares parameters on reuse. Keys may be pruned after at least 24 hours; reuse after pruning creates a new request. Validation

failures and concurrent conflicts have different caching behavior.[9] These details make idempotency valuable, but bounded.

Stripe also distinguishes network and server failures. Its network-error guidance recommends retrying with the same idempotency key and parameters to obtain a result. For a `500` response, it advises treating the result as indeterminate and warns against using a new key because the first request may have produced side effects. Reconciliation and later events may reveal outcomes that were absent from the original response.[10]

A correct action contract must preserve that distinction. “Never retry any write” discards useful native guarantees. “Retry until success” ignores expiry, cached errors, retention, restored state and duplicate admission. A profile can permit a same-key retry only when its provider contract, retained operation identity and authorization rules justify it. Otherwise, conservative query-based reconciliation may be necessary.



After a possible provider send, a lost reply leads to reconciliation of the original operation; sufficient evidence confirms an effect or no effect, while insufficient evidence retains UNKNOWN.

Figure 2. Outcome knowledge after a lost response. Reconciliation preserves the original operation identity; an empty lookup or transport error is not automatically proof of no effect.

UNKNOWN requires an owner and an operational path. Preserve the request identity, possible-send record, provider references, observations and next review time. Fence conflicting new operations where the business risk requires it. An unrelated new

request key must not bypass that fence. If the evidence remains inconclusive, escalation is a legitimate outcome; a fabricated success or failure is not.

This is not universal exactly-once execution. The design's claim is narrower: avoid admitting duplicate logical work, use documented provider mechanisms, and refuse to infer an outcome that the retained evidence cannot support. Backups, multiple executors and restored journals require their own fencing and recovery analysis.

Fair comparison with native controls

AWS AgentCore Policy documents deterministic policy enforcement at a gateway, including identity and tool-input conditions. Its temporal policies can require earlier events, correlate an approval with current parameters, and express counts and sums over session history.[11][12] These are substantive preventive controls, not merely after-the-fact logging.

Their documented scope matters. Temporal history is associated with a caller-supplied session, and starting a new session begins a new count. The inspected documentation also describes account/region and identity-propagation constraints. A per-session limit must not be presented as a provider-wide financial budget unless another control establishes that wider scope.[12]

Stripe's native rules already support amount, actor and rate conditions, outright blocking and two-party review. OAuth and MCP already supply authorization and token boundaries. A narrowly coded handler using these mechanisms may satisfy the entire required action contract. The paper offers no basis for replacing an adequate native design or claiming that a separate authority vendor is necessary.

The meaningful comparison is operation-specific: trusted context, exact bounds, approval lifecycle, alternate access, admission concurrency, cancellation cutoff and authoritative recovery. When an existing stack meets those requirements, the work is configuration and verification. When it does not, the missing piece may be a small integration rather than a new control plane.

Threat analysis and practical verification

The proposed boundary should be challenged with cases that can falsify its claims. A poisoned support record should not change tenant identity or customer-to-payment mapping. A valid agent credential should not permit arbitrary provider methods. A changed amount under an existing operation key should be rejected. Two workers should not admit the same one-use grant twice.

Additional cases must cover time and failure. Revoke authority on both sides of admission. Lose the provider response after a possible send. Restart the executor with its original journal. Exercise the native pending-approval lifecycle, including approval after local expiry. Verify that a missing webhook is not treated as proof of failure and that a stale backup cannot silently reopen consumed authority.

These are proposed acceptance cases, not reported passed tests. Their results must retain environment, account configuration, provider/API version, injection point and observations. A successful prompt-injection test suite cannot substitute for the concurrency cases, and a correct journal cannot substitute for domain authorization. Compromised identity issuers, privileged administrators, executors and provider infrastructure remain separate trust dependencies that the stated controls may not prevent.

Limitations and practical conclusion

The analysis is selective, English-language and documentation-based. It measures no attack prevalence, recovery frequency, customer savings or production safety. Public-preview behavior and living documentation can change. A real implementation needs current provider verification, domain review and evidence that alternative credential paths are closed. The proposed contract is not a formal proof or an interoperable standard.

Agentic AI safety at the action boundary depends on more than persuading the model to behave. Verify who may request the exact effect, where current authority is consumed, what revocation can still prevent, and how the original operation will be reconciled. Keep native mechanisms wherever they satisfy that contract. Preserve UNKNOWN when they do not supply enough evidence. The evaluation page and research collection connect this action analysis with the separate tasks of behavioral testing and accountable review.

References

1. OWASP Gen AI Security Project, LLM01:2025 Prompt Injection, 2025 edition. Relevant sections: direct/indirect injection and prevention/mitigation strategies. Guidance. Refreshed 26 September 2026. Threat guidance, not a measured guarantee for a particular defense.↩
2. IETF, Best Current Practice for OAuth 2.0 Security, RFC 9700, January 2025. Relevant sections: 2.2–2.3 and 4.10. RFC. Refreshed 26 September 2026. Token security does not independently establish business-action appropriateness.↩

3. IETF, OAuth 2.0 Rich Authorization Requests, RFC 9396, May 2023. Relevant sections: 1–2 and 9. RFC. Refreshed 26 September 2026. Fine-grained authorization data requires agreed semantics and enforcement.↩

4. IETF, OAuth 2.0 Token Exchange, RFC 8693, January 2020. Relevant sections: 1.1, 2.1 and actor claims. RFC. Refreshed 26 September 2026. Exchange does not generally propagate token lifecycle or revocation.↩ ↩

5. Model Context Protocol, Authorization, specification revision 2026-07-28. Relevant sections: scope, resource parameters and token validation. Specification. Refreshed 26 September 2026. The inspected revision is pinned here; client and server support require verification.↩

6. Model Context Protocol, Authorization Security Considerations, specification revision 2026-07-28. Relevant sections: token audience binding, validation and token theft. Specification. Refreshed 26 September 2026. Resource binding and the prohibition on token passthrough are distinct from application business rules.↩

7. Stripe, Model Context Protocol (MCP), undated living documentation. Relevant sections: authentication and human confirmation of actions. Documentation. Refreshed 26 September 2026. Inspected confirmation path uses an approval token and retry, with a 24-hour unapproved-action expiry; actual account/path support was not exercised.↩

8. Stripe, Set up two-party approvals, undated living documentation, labelled Public preview. Relevant sections: rule conditions, approval review, agent-key requests and events. Documentation. Refreshed 26 September 2026. Native approval can complete actions automatically; unapproved requests expire after 14 days. No provider acceptance test was performed here.↩

9. Stripe, Idempotent requests, undated living API documentation. Relevant sections: saved results, parameter comparison, pruning and pre-execution errors. Documentation. Refreshed 26 September 2026. These are bounded provider semantics, not universal exactly-once execution.↩

10. Stripe, Advanced error handling, undated living documentation. Relevant sections: network errors, server errors and reconciliation. Documentation. Refreshed 26 September 2026. Network retries and indeterminate server errors require different handling.↩

11. Amazon Web Services, Policy in Amazon Bedrock AgentCore: Control Agent Interactions, undated living documentation. Relevant sections: deterministic gateway policy and identity/input conditions. Documentation. Refreshed 26 September 2026. Published capability, not an independently validated deployment.↩

12. Amazon Web Services, Temporal policies, undated living documentation. Relevant sections: policy sessions, invalidation, considerations and security considerations. Documentation. Refreshed 26 September 2026. Session-scoped history and caller-supplied session identity limit the scope of counts and budgets.↩ ↩

Source method

Selected primary security, protocol and provider documentation refreshed on 26 September 2026; proposed architecture and a hypothetical refund timeline, with no new execution experiment.

Provenance. Nikxius Research, prepared with AI assistance. Technical working paper; not peer reviewed.