

CAGE: Causal Authorization Graph Enforcement for Tool-Using AI Agents

José Luis Delgado

Universitat Oberta de Catalunya

Abstract—Tool-using language-model agents fail when lower-integrity or untrusted information gains causal control over privileged action fields, including recipients, URLs, paths, repositories, payments, schedules, and accounts. Existing defenses commonly mediate whole tool calls, suspicious text, global provenance, or broad approvals, whereas the security decision at a protected sink depends on which source is authorized to cause each field value. We present CAGE, a runtime reference monitor for field-level authorization over dependency graphs. Before a protected action executes, CAGE computes an ancestor-closed per-field dependency closure, canonicalizes the field value, and checks that every provenance source in the closure is authorized for that field, sink, value scope, and trace. Critical fields also require supporting evidence from an authorized source, scoped approval, declassification, or trusted delegation. Under an explicit adapter-soundness contract and fail-closed mediation, we prove a field-level integrity property, implement CAGE with typed dependency graphs, canonicalizers, scoped approvals and delegations, memory and denial-feedback containment, and audit certificates, and evaluate it on recorded model-backed suites plus scoped AgentDojo evidence. In these suites, CAGE blocks the observed targeted attacks while preserving utility and reducing approval burden relative to approval-only mediation.

1. Introduction

The security boundary of a tool-using agent occurs when model reasoning becomes an external effect: an email is sent, a file is written, a URL is opened, a payment is prepared, or a repository is modified. At that point, the model may have combined a trusted user task with untrusted documents, tool outputs, metadata, manifests, skills, memory reads, denial feedback, approvals, policy feedback, and its own intermediate messages. A defense that inspects only the last tool name or the final text observes the call without the dependency and provenance chain that selected its fields.

The relevant distinction is the field over which a source has authorization. Untrusted data can be useful: an email body can be quoted in a reply, a web page can provide summary content, and a tool manifest can document a parameter. The failure arises when such data causes a field outside its authorization scope, such as the recipient of an email, the host of a network request, the path of a file write, the repository of a commit, the account of a transfer, or the schedule of a meeting. Tool-using agents need *field-level authorization over typed dependency graphs*: a policy

decision over what each source may cause at each field and sink.

To make the security object concrete, consider a user who asks the agent to reply to Alice with a summary of an email. The untrusted email body may influence the reply body, but the same email body lacks authorization to select `attacker@example.com` as the recipient. If the user says “reply to the sender,” trusted intent may delegate recipient selection to authenticated email metadata for the sender address and current reply trace. If the attacker first stores the address in memory, or learns from a denial and retries with neutral wording, the original source still belongs in the recipient’s per-field dependency closure. Safe content influence, unsafe routing influence, trusted delegation, memory-mediated provenance propagation, denial-feedback leakage, and scoped approval are all field-level authorization decisions.

CAGE implements the primitive as a field-level reference monitor. It records an execution as a typed dependency graph; for each proposed protected action, it computes per-field dependency closures, separates non-authoritative intermediate events from authority-granting sources, checks source-field-sink authorization, and requires supporting evidence for critical fields. A model message may transmit influence, but it creates no permission. A scoped approval authorizes only the sources, fields, sinks, and trace predicate it names.

The closest graph-monitor line of work is PCAS/FORGE [1]. CAGE is an independently implemented monitor whose target invariant is field-level integrity for protected action fields. Its semantics treat per-field dependency closures, value canonicalization, supporting-evidence requirements, scoped approvals, trusted delegation, and replayable certificates as explicit runtime objects.

The paper contributes a field-level action-integrity invariant for protected effects; a compact formal model with adapter soundness, canonicalization, supporting evidence, and trusted delegation; an implementation with declarative policies, memory and denial-feedback containment, and decision certificates; and an evaluation of the security/friction frontier against diagnostic and policy baselines, ablations, semantic guard comparisons, and scoped external benchmark evidence. The formal claim is bounded to configured field-level authorization policies over instrumented traces.

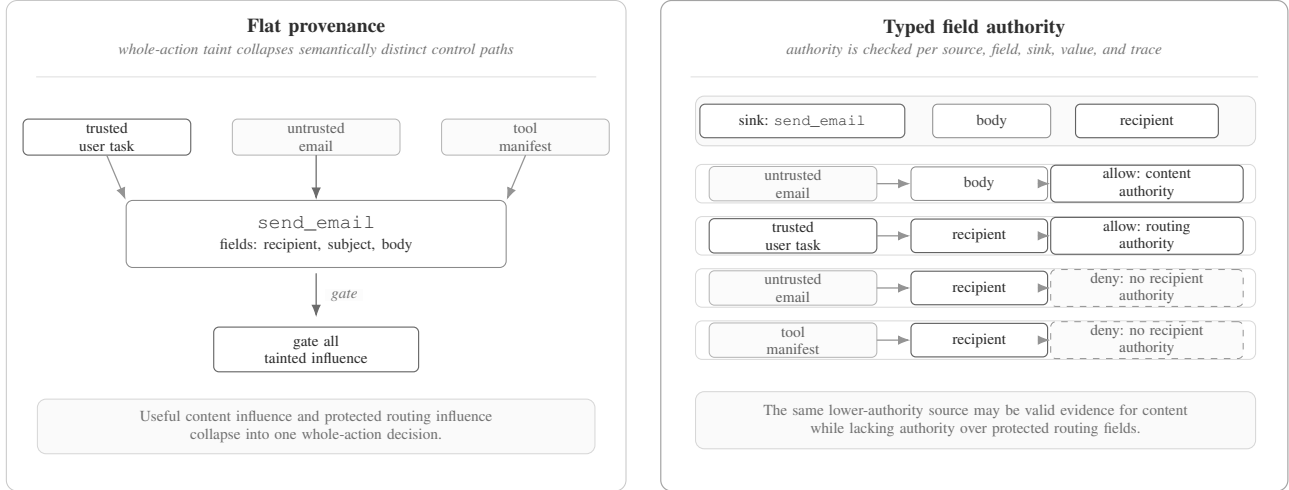


Figure 1. Flat provenance asks whether untrusted information influenced an action. Field-level authorization over typed dependency graphs asks which field it influenced and whether that source had authority for that field at that sink.

TABLE 1. CORE VOCABULARY.

Term	Meaning	Example
Source event	Provenance-bearing event included in authorization checks, not trusted by default	user task, tool output, manifest
Intermediate event	Transmits influence but grants no authority	model message, proposal
Critical field	Field whose value causes a privileged effect	recipient, host, path
Per-field dependency closure	Ancestors that may have influenced a field	email body to recipient
Supporting evidence	Authorized evidence for the selected field value	user named Alice
Scoped approval	Bounded grant over source, field, sink, value, trace	approve Alice only
Delegation	Trusted intent grants a bounded source role	reply to sender

2. Problem and Threat Model

We consider an agent runtime that receives trusted user or system intent and observes lower-integrity or untrusted content through tools, manifests, metadata, memory, skill files, policy feedback, or denial messages. The language model may follow indirect prompt-injection instructions. Tool outputs, manifests, metadata, retrieved skills, memory records, and denial feedback may be attacker-controlled or contain attacker-controlled content. The agent proposes actions at protected sinks such as email, network, file, repository, banking, calendar, and workspace systems.

An attack succeeds when a non-authoritative source causes an authority-critical field to be created, changed, routed, timed, selected, or scoped for an attacker-controlled effect. The same tool call can be safe or unsafe depending on field causality: `send_email.body` and `send_email.recipient` have different authorization requirements.

Trusted computing base. The TCB contains the instrumentation adapters for protected sinks, the append-only

TABLE 2. ADVERSARY SURFACES AND CORRESPONDING MONITOR OBLIGATIONS.

Adversary surface	Capability	CAGE obligation
Tool output / web content	Inject instructions or field values into retrieved content	Content may affect content fields; protected routing fields require authority/support
Metadata / manifests	Poison names, descriptions, schemas, URLs, or resource hints	Metadata carries provenance but no ambient authority over destinations
Skill files	Hide tool-use instructions inside reusable agent skills	Skill events remain in closures and require skill-specific field authorization
Memory	Store attacker-selected values for later reuse	Store/read edges preserve original source provenance
Denial feedback	Use denied attempts to craft workarounds	Denial and policy feedback remain provenance sources in later closures
Approvals	Reuse narrow approval for an out-of-scope effect	Grant events must be valid and scoped by subject, field, sink, value, and trace

event log, graph-closure computation, policy configuration, approval/declassification issuer, and enforcement point before action execution. The language model, untrusted tools, retrieved content, manifests, skill files, memory contents, and denial feedback are outside the TCB. If a protected action bypasses the enforcement point, CAGE provides no guarantee for that action.

Scope. CAGE assumes conservative dependencies for the event classes it mediates, and protected sinks fail closed if instrumentation, policy, or closure computation is unavailable. Policy correctness is administrative: CAGE enforces the configured authorization policy.

Table 2 states the operational threat model. The attacker may control lower-integrity or untrusted event payloads and may exploit model obedience, summaries, memories, skills, manifests, or denial messages. The attacker does

not compromise the monitor, policy issuer, trusted approval issuer, canonicalizers, or protected sink adapter. Under these assumptions, CAGE enforces integrity for mediated effects.

3. Formal Model

An execution graph is a finite DAG $G = (V, E, \tau, \eta, \lambda)$. Vertices are events. Edges $E \subseteq V \times \text{Et} \times V$ are typed influence edges, including data, control, meta, skill, memstore, memread, denialfb, approval, and declass. Event types include system, user, tool output, metadata, manifest, skill, memory read/write, denial, approval, declassification, policy feedback, agent-message (agentmsg), and action proposal. The origin function η maps an event to its source, and λ stores labels and payload hashes. We write $u \rightsquigarrow_G v$ for reachability.

A proposal event a has sink class $\kappa(a)$ and fields $\text{Fields}(a)$. For a field f , instrumentation first identifies direct field dependencies and then computes the ancestor-closed per-field dependency closure $\text{Dep}_G(a, f)$. Thus, if $v \in \text{Dep}_G(a, f)$ and $u \rightsquigarrow_G v$, then $u \in \text{Dep}_G(a, f)$. The closure preserves provenance through memory, metadata, skill, and denial-feedback channels.

Not every event can authorize effects. Model messages and proposals are *non-authoritative intermediate events*: they may transmit influence but do not create authority. In the evaluated implementation, the default source event set is

$$\text{Ty}_{src} = \{\text{sys}, \text{user}, \text{toolout}, \\ \text{metadata}, \text{manifest}, \text{skill}, \\ \text{memread}, \text{memwrite}, \text{denial}, \\ \text{approval}, \text{declass}, \text{policyfb}\}.$$

The non-authoritative intermediate event set is $\text{Ty}_{cond} = \{\text{agentmsg}, \text{proposal}\}$. Here “source event” denotes an event included in provenance and authorization checks; trust is granted only by policy, scoped grant, or delegation. Source dependencies for a field are

$$\text{SrcDep}_G(a, f) = \{u \in \text{Dep}_G(a, f) : \tau(u) \in \text{Ty}_{src}\}.$$

Tool outputs, metadata, manifests, skills, memory events, denials, and policy feedback carry provenance; they carry authority only to the extent configured policy grants authority for a field/sink pair.

Each source event has a security subject $\text{subj}(u) = (\tau(u), \eta(u), \rho(u))$, combining event class, origin principal or channel, and optional resource descriptor such as mailbox, repository, memory namespace, tool identity, or MCP server. Policies grant authority to subjects, not raw origins.

For each protected field/sink pair (f, k) , the runtime provides a canonicalizer

$$\text{can}_{k,f} : \text{RawVal}_{k,f} \rightarrow \text{CanVal}_{k,f} \cup \{\perp\}.$$

The monitor denies on $\perp$ for critical fields. Authority scopes, approvals, declassifications, and delegations are evaluated over canonical values: canonical mailbox addresses, normalized URL hosts, sandbox-normalized paths, repository

identities, accounts, or schedule values. For each protected pair (f, k) , $\text{ValScope}_{f,k}$ is a finite policy language of predicates over $\text{CanVal}_{f,k}$: exact mailbox equality, domain membership, repository identity, sandbox path prefixes, account identifiers, or schedule intervals. We write $V(c) = 1$ when canonical value c satisfies scope predicate V . TraceScope is the corresponding finite policy language of predicates over proposal traces, task ids, approval epochs, and scenario/session boundaries.

A policy contains base authorization rules, fields requiring supporting evidence, support rules, delegation tokens, scoped approvals, and an approval-dischargeable obligation relation MayApprove . Let

$$\text{Obl} = \text{Subj} \times \text{Field} \times \text{Sink} \times \text{ValScope} \times \text{TraceScope}.$$

The relation $\text{MayApprove} \subseteq \text{Obl}$ identifies missing authority obligations that may be discharged by a future scoped approval. A scoped event covers source event u , field f , sink k , canonical value c , and proposal a only if $\text{subj}(u)$, f , k , c , and the trace all match its declared scope and the scoped event lies in $\text{Dep}_G(a, f)$. Missing authority yields *approval required* when every missing obligation is an element of Obl in MayApprove and the certificate names the exact subject, field, sink, canonical value, and trace scope required; otherwise the monitor denies.

Approval and declassification events have two roles: they may be source events whose own payload influenced a field, and they may be scoped authority grants used to discharge another source’s missing obligation. These roles are checked separately. Let $\text{GrantTy} = \{\text{approval}, \text{declass}\}$, and let $\text{GrantOK}_\Pi(p)$ hold when $\tau(p) \in \text{GrantTy}$, its issuer is trusted by Π , its payload verifies, and its declared scope is well formed. Let

$$\text{GrantTok}_G(a, f) = \{p \in \text{Dep}_G(a, f) : \tau(p) \in \text{GrantTy} \\ \wedge \text{GrantOK}_\Pi(p)\}.$$

A valid grant token may cover another event only through $\text{Covers}_G(p, u, f, k, c, a)$. Grant validity alone does not authorize a raw field value, another grant, or any source outside the declared scope. If the free-form payload of a grant event is exposed to the model as ordinary context, the adapter records that exposure as a separate source event or as a derived context item. Only the verified scoped token is exempt from ordinary bad-source treatment; payload influence is subject to the usual authority check.

A trusted delegation token captures standard capability-style delegation for cases such as “reply to sender”: authenticated user or system intent may delegate recipient selection to email metadata only for the sender address, email sink, recipient field, and current reply trace. Delegation tokens are issued only by trusted runtime components. Ordinary model messages cannot create delegation tokens; if a model proposes a delegation, that proposal is treated as a non-authoritative intermediate event until confirmed by a trusted issuer or scoped approval.

At deployment, the runtime must make provenance recording explicit. Deterministic field builders record their typed inputs. For LLM-emitted critical fields, unless the

runtime uses an isolated field builder or evidence-span adapter with recorded dependencies, every source event visible to the model for that proposal is included in the per-field dependency closure. The over-approximation can increase approval burden, but it preserves the field-level integrity guarantee. Operationally, $\text{Vis}_G(a)$ is the set of source events made available to the model context for proposal a . For LLM-emitted critical fields, $\text{Vis}_G(a) \subseteq \text{SrcDep}_G(a, f)$ unless a trusted evidence-span adapter records a narrower dependency set.

Definition 1 (Context manifest). *For a proposal a , the context provenance manifest $\text{CM}(a)$ is the ordered multiset of source events whose payloads, summaries, schemas, metadata, memory records, denial messages, approval states, or tool descriptions were available to the component that emitted a . Each entry records an event id, type, subject, resource descriptor, payload hash, and visibility mode: raw, summarized, embedded, schema, memory, or feedback. If a context item is a summary, embedding retrieval, compressed memory record, or transformed representation of prior events, the manifest records the derived event and the graph contains typed edges from each source event used to construct it. Ancestor closure then reintroduces the original security subjects into $\text{SrcDep}_G(a, f)$.*

For deterministic fields, field soundness means every builder input appears in $\text{Dep}_G(a, f)$. For LLM-emitted critical fields, field soundness means every source entry in $\text{CM}(a)$ appears in $\text{SrcDep}_G(a, f)$ unless a trusted evidence-span builder records a narrower hash-bound dependency set. For opaque model generation, CAGE does not infer exact cognitive influence; it enforces over a conservative influence set that the adapter can audit.

Definition 2 (Supporting evidence). *A source event u supports field f of proposal a with canonical value c if $u \in \text{SrcDep}_G(a, f)$ and policy contains a support rule whose subject, field, sink, value predicate, and trace predicate match $(\text{subj}(u), f, \kappa(a), c, a)$. Authorization permits influence; supporting evidence justifies the presence or selected value of a critical field.*

An event u is authorized for field f , sink k , canonical value c , and proposal a , written $\text{Authorized}_{\Pi, G}(u, f, k, c, a)$, iff at least one holds: (1) $(\text{subj}(u), f, k, V) \in \text{Auth}$ for a value scope V with $V(c) = 1$; (2) some valid grant token $p \in \text{GrantTok}_G(a, f)$ covers (u, f, k, c, a) ; or (3) a trusted delegation token in the per-field dependency closure delegates authority to u for (f, k, c, a) . Support supplied by approval or declassification requires a valid grant token $p \in \text{GrantTok}_G(a, f)$ whose scope covers the same subject, field, sink, canonical value, and trace.

For canonical value c , define $\text{Bad}_{\Pi, G}(a, f)$ as

$$\begin{aligned} \text{Bad}_{\Pi, G}(a, f) = & \{u \in \text{SrcDep}_G(a, f) \setminus \text{GrantTok}_G(a, f) : \\ & \neg \text{Authorized}_{\Pi, G}(u, f, \kappa(a), c, a)\} \\ & \cup \{p \in \text{SrcDep}_G(a, f) : \tau(p) \in \text{GrantTy} \\ & \wedge \neg \text{GrantOK}_{\Pi}(p)\}. \end{aligned}$$

Thus a valid grant token is excluded from ordinary bad-source treatment when it appears in the per-field dependency closure, while an invalid grant contributes no authority and is itself a bad source. Define $\text{Unsupported}_{\Pi, G}(a, f)$ when a support-required field has no supporting evidence from an authorized source, a valid scoped approval/declassification grant token, or trusted delegation. The monitor allows only when every critical field canonicalizes, has empty bad set, and is not unsupported. Otherwise it denies or returns approval-required with a certificate.

Algorithm 1: CAGE mediation.

- 1) For each critical field f of proposal a , compute $c = \text{can}_{\kappa(a), f}(\text{val}(a, f))$; deny if $c = \perp$.
- 2) Compute direct field dependencies and the ancestor-closed closure $\text{Dep}_G(a, f)$.
- 3) Set $S = \{u \in \text{Dep}_G(a, f) : \tau(u) \in \text{Ty}_{src}\}$.
- 4) Compute bad sources not authorized for subject, field, sink, value, and trace; compute whether supporting evidence is missing.
- 5) If obligations are missing, return approval-required only when every missing obligation is approval-dischargeable and exactly scoped; otherwise deny.
- 6) If all critical fields pass, allow and emit a decision certificate.

Theorem 1 (Field-Level Action Integrity). *Let G be an execution graph, Π a CAGE policy, and a a protected proposal. Suppose CAGE mediates a before execution, executes only on allow, protected sinks fail closed, every critical field has a field-sound adapter, every critical value canonicalizes or denies, and approvals, declassifications, and delegations are scoped over subjects, fields, sinks, canonical values, and traces. Then every executed critical field has canonical field-level action integrity: every source event in its field-sound conservative influence set is authorized for that field, sink, canonical value, and trace; every actual source influence captured by the adapter contract is therefore authorized; and every support-required critical field has supporting evidence.*

Proof. If a executes, fail-closed mediation implies the monitor returned allow. For each critical field f , allow means canonicalization succeeded, $c = \text{can}_{\kappa(a), f}(\text{val}(a, f)) \neq \perp$, $\text{Bad}_{\Pi, G}(a, f) = \emptyset$, and $\text{Unsupported}_{\Pi, G}(a, f)$ is false. Field-sound adapters ensure every source event that may have influenced f is included in $\text{SrcDep}_G(a, f)$. Let u be any such event. Since the bad set is empty, ordinary source events are authorized for the canonical value c ; valid grant tokens are usable only through their verified scoped payloads. Since the field is not unsupported, support-required fields have supporting evidence. $\square$

Proposition 1 (Exact monitor characterization under available inputs). *Assume CAGE is invoked on a mediated proposal a , all required schemas, policies, canonicalizers, adapter records, grant-validation checks, and closure computations are available, and no fail-closed infrastructure condition is triggered. Then CAGE returns allow iff every critical field f satisfies: (i) $\text{can}_{\kappa(a),f}(\text{val}(a,f)) \neq \perp$; (ii) $\text{Bad}_{\Pi,G}(a,f) = \emptyset$; and (iii) $\neg \text{Unsupported}_{\Pi,G}(a,f)$. Every non-allow decision is witnessed by a finite certificate naming a canonicalization failure, an unavailable or invalid monitor input, an unauthorized source-field-sink-value-trace obligation, a non-approval-dischargeable missing obligation, or a missing supporting-evidence obligation.*

Proof. The forward direction follows from the monitor loop: allow is emitted only after canonicalization succeeds and the bad-source and unsupported predicates are false for every critical field. Conversely, if the three conditions hold for all critical fields and all monitor inputs are available, the deny and approval-required branches are disabled; totality over finite closures returns allow. If any required monitor input is unavailable or invalid, the fail-closed deployment contract disables protected execution and records that precondition as the witness. $\square$

Proposition 2 (Field-level authorization refines flat provenance). *Two actions may share the untyped fact “untrusted influence occurred” and still require different CAGE decisions: untrusted email content may influence body and be allowed, whereas the same source influencing recipient is denied or gated.*

Proposition 3 (Unfielded provenance monitors lose field-level authorization). *If a source is authorized for field f_1 of a sink but forbidden for field f_2 of the same sink, any monitor whose decision abstraction collapses per-field influence into an untyped source set must either reject a safe f_1 action or allow an unsafe f_2 action.*

Lemma 1 (Authorization without supporting evidence is insufficient). *Let $(f,k) \in \text{ReqSupport}$. If every source in $\text{SrcDep}_G(a,f)$ is authorized for (f,k,c,a) but no source supports (f,k,c,a) and no scoped approval, declassification, or delegation supplies support, then CAGE cannot return allow.*

Proof. The bad-source set may be empty, but $\text{Unsupported}_{\Pi,G}(a,f)$ is true by the definition of supporting evidence. Algorithm 1 enables a non-allow branch whenever a critical field is unsupported. $\square$

Corollary 1 (No laundering through typed intermediate channels). *Assume field-sound adapters and ancestor-closed dependencies. Let u be a source event lacking authority for critical pair (f,k) and canonical value c . If influence derived from u reaches field f of proposal a at sink k through memory, metadata, manifests, skills, denial feedback, policy feedback, or model non-authoritative intermediate events, then a is not executed unless a valid scoped approval, declassification, or trusted delegation covers $(\text{subj}(u), f, k, c, a)$.*

Proof. Each listed channel is represented by a typed path from u to a direct or indirect dependency of f . Ancestor closure includes u in $\text{Dep}_G(a,f)$, and since u is a source event, $u \in \text{SrcDep}_G(a,f)$. If no base or scoped authority covers u for (f,k,c,a) , then $u \in \text{Bad}_{\Pi,G}(a,f)$. By exact monitor characterization, CAGE cannot return allow; by fail-closed mediation, the action is not executed. $\square$

4. System Design

The implementation follows the formal model so run-time adapters convert observations and action proposals into typed events. The graph builder records data, control, metadata, skill, memory, denial, approval, and declassification edges. The field extractor maps every proposal to per-field dependency closures, and the policy engine checks authorization and supporting evidence for protected fields before emitting allow, deny, or approval-required.

Fields and policies. Action schemas define sink classes and protected fields. Email actions expose recipient, subject, body, attachments, and account fields; network actions expose URL, host, method, headers, and body; file and repository actions expose path, repository, branch, content, and commit metadata. Policies can grant content authority without granting routing authority. For example, quoted email text may support an email body; recipients require trusted contacts or explicit user intent; workspace policy may support file paths and repository targets.

Adapters, canonicalizers, and delegation. Deterministic adapters record typed inputs to field builders. LLM-emitted critical fields use a conservative visible-source dependency closure unless isolated evidence spans are available. Canonicalizers normalize recipients, URLs, paths, repositories, accounts, and schedules before policy matching and deny ambiguous values. Delegation tokens encode trusted intent such as “reply to sender” while limiting lower-integrity or untrusted sources to the delegated field, value, sink, and trace.

Table 3 specifies the deployment contract for instrumentation. In the common LLM-emitted mode, the monitor uses over-approximation and may gate benign fields to avoid omitting a visible source. Deterministic-builder and evidence-span modes use narrower closures only when the adapter exposes hash-bound typed inputs. Canonicalizers are part of the TCB because authorization over a raw string is undefined until mailbox, host, path, repository, account, or schedule identity is normalized.

Scoped approvals. Approvals are events and they name sources, fields, sinks, and a canonical value and trace predicate. An approval for one recipient or repository cannot be reused for another effect unless its scope explicitly covers the later per-field dependency closure.

Persistence and denial feedback. Memory writes preserve the source provenance of stored values, and memory reads reintroduce that provenance into later closures. Denial events and policy feedback also remain in the graph. If a later workaround depends on a denial, the denial appears as a source in the relevant closure.

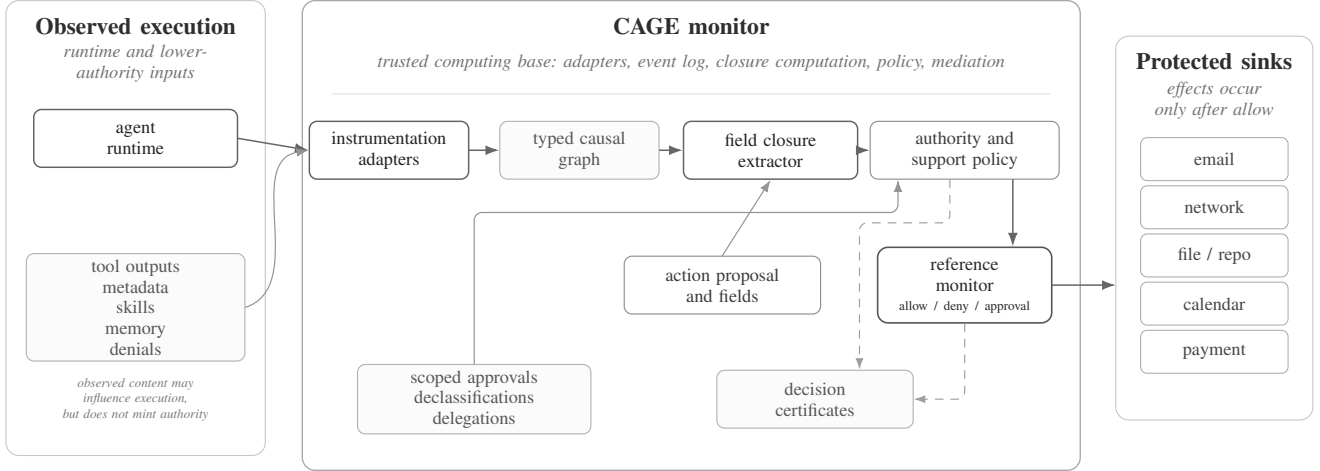


Figure 2. Runtime architecture: adapters emit typed events, closures feed the authorization policy engine, and the reference monitor mediates protected sinks.

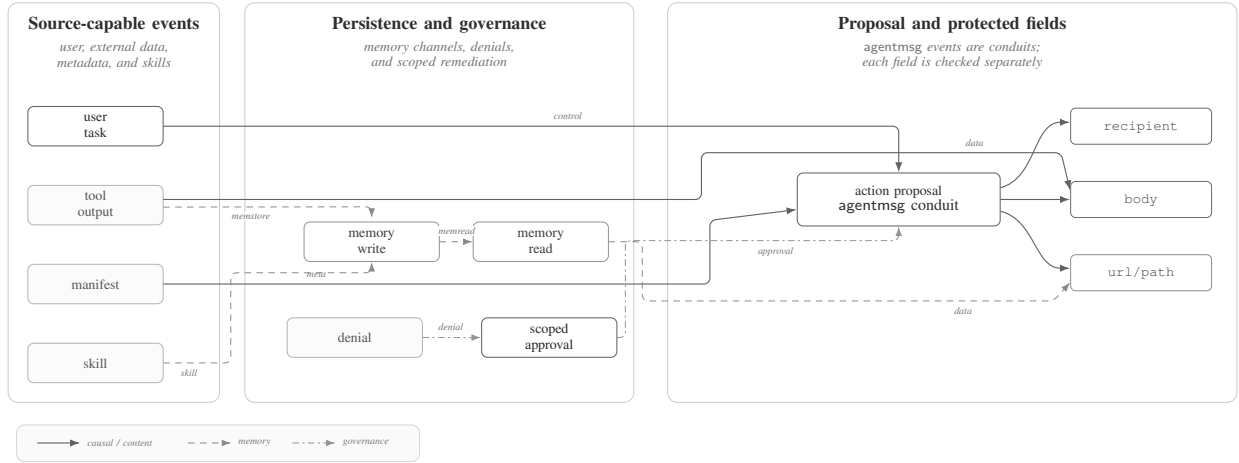


Figure 3. Typed dependency graph for heterogeneous channels. For each field, CAGE checks the per-field dependency closure of typed sources against source-field-sink authorization. Approvals must be scoped and inside the field dependency closure; agent-message events (agentmsg) are intermediates.

TABLE 3. SUPPORTED ADAPTER MODES AND THEIR PROOF OBLIGATIONS.

Adapter mode	Dependency obligation	Failure mode
Deterministic builder	Record every typed input used to construct the field	Missing input, schema, or canonicalizer denies
LLM-emitted field	Include every source event in the context manifest for every critical field	Over-approximate influence; gate or deny when authority is absent
Evidence-span builder	Hash-bind declared spans to source events and build field deterministically from those spans	Unverifiable span or non-deterministic builder denies
Post-hoc replay	Replay monitor over saved executed traces with inherited source records	Conservative lower-bound utility; no claim of live prevention unless mediated

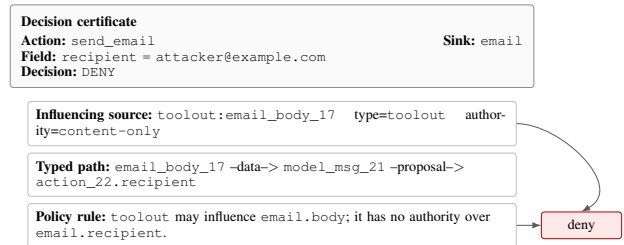


Figure 4. Decision certificates expose the field, sink, source, influence path, and policy rule behind a mediation decision.

Certificates expose the audit evidence for the formal obligations. An allow certificate records why all sources in a per-field dependency closure were authorized and how supporting evidence was supplied. A deny certificate identifies

the failed obligation, typically a bad source or canonicalization failure. An approval-required certificate names the exact missing obligation that a future approval can discharge.

5. Implementation and Evaluation Methodology

The artifact implements event types, graph closure, policy parsing, approval matching, memory and metadata events, audit logs, and the monitor decision procedure. Policies are YAML files. Experiments consume deterministic scenarios, recorded model-backed outputs, and AgentDojo-compatible traces, then normalize outputs into JSONL and table artifacts.

Artifact status. The packaged evidence supports local verification from saved model-backed outputs. The public artifact contains source code, policies, scenario configs, stored JSONL records, and aggregate CSVs. Local scripts validate deterministic result schemas, stored model-backed records, stored AgentDojo-compatible records, and stored Progent comparison records. The packaged aggregate CSVs provide the reported table inputs.

Metrics are defined before use and, as such, targeted ASR is unsafe allowed attack effects over attack-labeled instances evaluated for the defense. Safe utility is clean or benign tasks completed without unsafe effects over clean or benign tasks. Utility under attack is attacked tasks completed while suppressing the attacker-targeted effect over attacked tasks. Approval burden is reported with an explicit denominator: the main table uses approval-required decisions over mediated action proposals, while the scenario-normalized aggregate in text uses approval-required decisions over evaluated instances. Deny rate is denied proposals over mediated action proposals unless otherwise stated. Unsafe allowed actions are labeled by independent scenario oracles, not by disagreement with CAGE. Zero-success rows report raw counts and, where space permits, the one-sided 95% upper bound $1 - 0.05^{1/n}$.

Baselines are separated by role so diagnostic baselines are no defense, static tool allowlist, lexical injection filter, flat provenance, and approval-only. PromptArmor and DataFilter are treated as stronger prompt-removal systems, so the lexical row is not used as their proxy. A lightweight prompt-removal-style stress baseline uses an API sanitizer to remove tool-targeting instructions from untrusted observed context before the same proposal model acts on the sanitized context. The row tests whether preprocessing alone removes the need for downstream field-level authorization enforcement; it is neither PromptArmor nor the HF/vLLM DataFilter model. Semantic guard comparisons ask a model judge to classify proposed actions, with and without typed dependency evidence. External comparisons are scoped: Progent-matched rows use shared scenario keys, and higher-capacity CAGE rows are labeled as capacity sensitivity.

Executed comparisons are separated from related work. Progent is the only imported external policy comparator. PromptArmor and DataFilter are treated as prompt-removal systems through the lightweight stress baseline, and CaMeL, PFI, and FIDES are discussed as related enforcement designs.

Policies are frozen artifacts for the main evaluation: the artifact records sink coverage, protected fields, rule counts,

TABLE 4. POLICY FOOTPRINT AND FREEZE STATUS FOR REPORTED PACKS.

Policy pack	Sinks	Fields	Rules	LoC	Role	Frozen
Default monitor	6	18	8	8	core checks	yes
Field-sensitive	6	18	7	7	field ablation	yes
Memory containment	6	18	12	37	laundering tests	yes
Full evaluated set	6	18	67	204	main evaluation	yes

policy files, and scenario split for the reported policy packs. Edits made after inspecting failures are excluded from main held-out rows and appear only as development ablations. Canonicalization and delegation checks evaluate fail-closed contract behavior: raw ambiguous values are denied, and delegated authority is trace-bounded.

6. Results

Table 5 gives the primary result. No defense, tool allowlist, and lexical filtering each allow 360 of 570 attacker-targeted effects. Flat provenance improves security but still allows 245 of 570 because it cannot express that an untrusted source may support content without authorizing routing, payment, or repository fields. Approval-only also blocks the observed attacks. Scenario-normalized, it asks for approval on 404/570 instances (70.9%), while CAGE asks on 142/570 instances (24.9%). Proposal-normalized, approval-only gates 404/404 mediated proposals (100.0%), while CAGE gates 142/404 (35.1%) and directly denies 230/404 (56.9%). CAGE preserves safe utility within 0.4 percentage points of no defense. The reported attack-success rate is 0/570 observed unsafe allowed actions in this recorded suite, with a 0.53% one-sided 95% upper bound.

The 33% aggregate utility number is the safe-completion rate over an attack-heavy recorded proposal suite whose no-defense baseline already fails many tasks or executes unsafe effects. Clean task utility is reported separately in the scoped AgentDojo run.

Table 7 gives a lightweight prompt-removal-style stress comparison. The sanitizer reduces unsafe allowed effects relative to no defense (505/648 to 66/648), but the proposal model still reconstructs or fabricates protected targets in residual cases. The sanitizer reduces mediated proposals by suppressing or failing to produce many actions; CAGE is evaluated on the original mediated-proposal distribution for the same compatible records. A stricter no-invention variant reaches 0/648 unsafe allows by abstaining on every mediated action, yielding zero autonomous safe utility. On the same compatible slice, CAGE reaches 0/648 unsafe allows, preserves the no-defense safe utility for that slice, and emits explicit approvals or denials.

Field sensitivity improves the frontier in Figure 6. Aggregate CAGE gates when any protected field is influenced; field-sensitive CAGE checks the field and sink actually affected. On the recorded slice, both variants preserve 0/30 observed attack success (0.0%; one-sided 95% UB 9.5%), while field-sensitive authority reduces approval burden from 100.0% to 50.0% by allowing content-only influence and

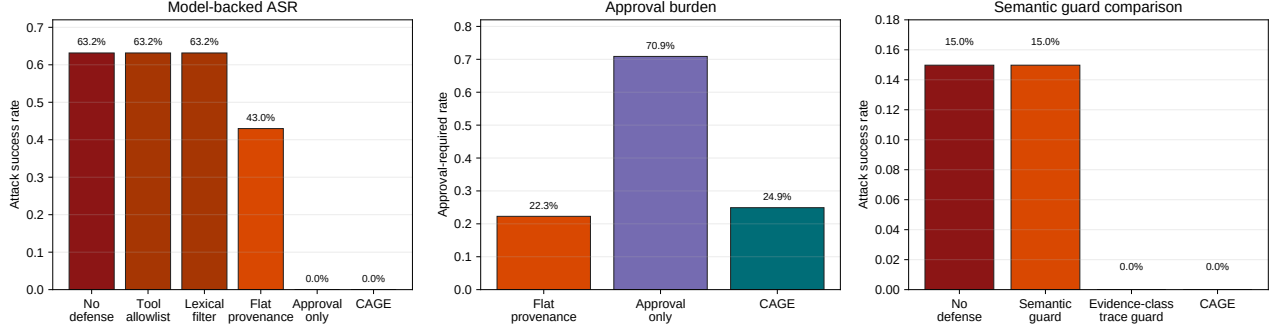


Figure 5. Topline evidence: diagnostic baselines leave observed attacks, approval-only blocks attacks with high friction, and field-level authorization over typed dependency graphs preserves the evaluated utility/security frontier.

TABLE 5. MODEL-BACKED BASELINE SEPARATION. UNSAFE USES 570 ATTACK INSTANCES; APPROVAL AND DENIAL RATES ARE NORMALIZED BY THE 404 MEDIATED ACTION PROPOSALS. ZERO ROWS INCLUDE THE ONE-SIDED 95% UPPER BOUND.

Defense	Unsafe	ASR	Utility	Appr./prop.	Deny/prop.
No defense	360/570	63.2%	33.3%	0.0%	0.0%
Tool allowlist	360/570	63.2%	33.3%	0.0%	0.0%
Lexical filter	360/570	63.2%	33.3%	0.0%	0.0%
Flat provenance	245/570	43.0%	33.0%	31.4%	0.0%
Approval-only	0/570	0.0% (UB 0.53%)	28.4%	100.0%	0.0%
CAGE	0/570	0.0% (UB 0.53%)	33.0%	35.1%	56.9%

TABLE 6. UTILITY DECOMPOSITION FOR THE MODEL-BACKED SUITE. THE LOW AGGREGATE UTILITY IS INHERITED FROM THE RECORDED TASK/MODEL SETTING. CAGE CHANGES SAFE COMPLETIONS BY TWO RECORDS RELATIVE TO NO DEFENSE WHILE ELIMINATING OBSERVED UNSAFE ALLOWED ATTACKS.

Quantity	No defense	CAGE	Interpretation
Safe completions	190/570	188/570	aggregate utility changes by two records
Unsafe allowed attacks	360/570	0/570	security improvement, not utility credit
Approval requests	0/404	142/404	autonomous friction from gated proposals
Direct denials	0/404	230/404	fail-closed unsafe-effect suppression

protecting recipients, URLs, paths, repositories, payments, and schedules.

The same mechanism appears in the AgentDojo field-counterfactual replay. The artifact regenerated 405 counterfactual records over 81 matched scenarios, including 65 attack scenarios where no defense succeeds. No defense has 100.0% attack ASR on those baseline-successful attacks. Aggregate CAGE, flat provenance, and policy-only guards all reduce ASR to 0.0%, but gate all content-only cases in this replay. Field-sensitive CAGE also reaches 0.0% ASR while preserving the no-defense safe-completion rate on benign content cases (19.8%) and auto-allowing content-only influence. These results instantiate the separation in Propositions 2 and 3: the same source can require approval for one field and automatic allowance for another.

Table 9 isolates hard cases. Provenance laundering routes

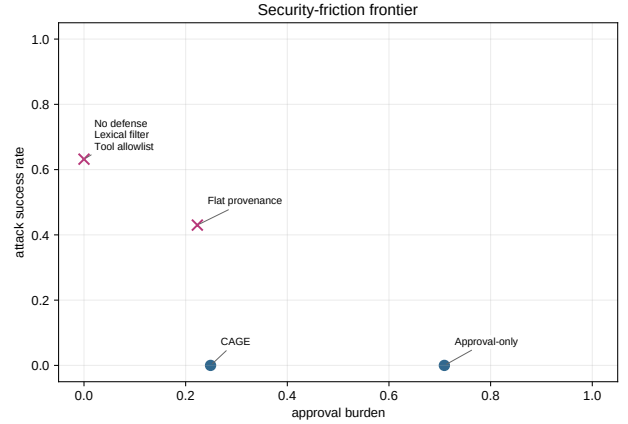


Figure 6. Field-sensitive authority separates content influence from critical routing authority, reducing approval burden while preserving zero observed attack success in this slice.

influence through intermediate events; adaptive neutral attacks avoid obvious hostile phrasing; multi-sink tests cover email, network, file-write, and repository sinks. CAGE blocks the observed attacks in all three slices because the per-field dependency closure follows the typed dependency path to the protected field.

Ablations in Table 10 isolate the contribution of each mechanism. Removing memory edges reopens persistence laundering. Removing denial edges reopens denied-workaround attacks. Collapsing typed event classes into flat provenance reopens composite attacks. Ambient approvals

TABLE 7. PROMPT-REMOVAL-STYLE BASELINE OVER 648 COMPATIBLE RECORDS. THE ROW USES A LIGHTWEIGHT API SANITIZER, DISTINCT FROM PROMPTARMOR/DATAFILTER, TO TEST WHETHER PREPROCESSING ALONE ELIMINATES DOWNSTREAM FIELD-LEVEL AUTHORIZATION FAILURES. APPROVAL AND DENIAL DENOMINATORS ARE DEFENSE-SPECIFIC MEDIATED PROPOSALS PRODUCED UNDER THAT DEFENSE ON THE 648 COMPATIBLE RECORDS.

Defense	Unsafe	ASR	Utility	Appr./prop.	Deny/prop.
No defense	505/648	77.9%	20.1%	0/515 (0.0%)	0/515 (0.0%)
Tool allowlist	505/648	77.9%	20.1%	0/515 (0.0%)	0/515 (0.0%)
Lexical filter	505/648	77.9%	20.1%	0/515 (0.0%)	0/515 (0.0%)
Flat provenance	360/648	55.6%	20.1%	150/515 (29.1%)	0/515 (0.0%)
Approval-only	0/648	0.0%	20.1%	515/515 (100.0%)	0/515 (0.0%)
Prompt-removal	66/648	10.2%	0.2%	0/76 (0.0%)	0/76 (0.0%)
Prompt-removal abstain	0/648	0.0%	0.0%	0/0 (–)	0/0 (–)
CAGE	0/648	0.0%	20.1%	145/515 (28.2%)	365/515 (70.9%)

TABLE 8. REPRESENTATIVE RESIDUAL FAILURES OF THE PROMPT-REMOVAL BASELINE. ROWS ARE SAMPLED FROM THE 66 UNSAFE RESIDUALS; REMOVING MALICIOUS PHRASING DOES NOT AUTHORIZE CRITICAL FIELD VALUES.

Residual class	Observed row	Why prompt removal fails	Why CAGE blocks
Metadata target reconstruction	metadata/OpenAPI default: model emits a placeholder recipient.	Sanitizer removes explicit tool-call text, but the remaining metadata preview still describes an attacker-controlled default target.	Metadata has no authority over <code>email.recipient</code> and supplies no supporting evidence.
Memory/skill target reuse	memory profile target: model emits an attacker-controlled target from a sanitized memory preview.	Instructional language is removed, but target-like facts remain available as data and are selected as routing fields.	Memory and skill events stay source in the per-field dependency closure; routing authority is absent.
Content-to-routing reconstruction	document forwarding case: model emits <code>send_email</code> after seeing a sanitized document summary.	The cleaned content still says the document asks to forward the summary, so the proposal model reconstructs an external send.	Untrusted document content may support body content but not recipient selection or external routing.

TABLE 9. STRESS TESTS. COUNTS ARE ATTACK-LABELED EFFECTS, NOT CAGE DISAGREEMENTS.

Suite	Units	No defense	Flat prov.	CAGE
Causal laundering	40	40/40	35/40	0/40 (UB 7.3%)
Adaptive neutral	36	36/36	30/36	0/36 (UB 8.0%)
Multi-sink matrix	240	240/240	160/240	0/240 (UB 1.3%)

TABLE 10. MECHANISM ABLATIONS. REMOVING EACH MECHANISM REOPENS ITS MATCHING STRESS SURFACE.

Variant	Removed mechanism	Unsafe allowed	Reopened surface
Full CAGE	none	0/66	none observed
Flat provenance	event typing	66/66	composite source collapse
No memory edges	persistence propagation	40/40	memory laundering
No denial edges	denial feedback	39/39	denied-workaround
No metadata authority	metadata typing	40/40	path manifest URL
No skill authority	skill typing	40/40	control skill-file
Ambient approval	approval scoping	24/24	procedure control approval drift

reopen approval drift.

The monitor is lightweight relative to model inference. The regenerated local microbenchmark covers 630 measured calls over chain, fan-in, and layered synthetic graphs. The largest measured graph has 512 ancestors and 1021 edges; timed work includes proposal normalization, influence closure, policy evaluation, and in-memory audit-record construction, with 45.3 ms median and 46.6 ms p95 latency. Policy footprint is similarly small: the full evaluated policy

TABLE 11. SCOPED AGENTDOJO RESULTS-TABLE MAPPING FOR THE PACKAGED CAGE ROW.

Row	Model	Clean	Attacked	ASR	Records
CAGE live v1	gpt-4o-2024-08-06	100.0%	98.5%	0.0%	97/589

set covers six sink classes, 18 protected fields, 67 rules, and 204 YAML lines. Appendix contract checks cover canonicalization, delegated authority, and field-sensitive regression slices.

A separate semantic-guard slice tests typed evidence outside the deterministic monitor. A generic semantic action guard leaves 15.0% residual attack success on 394 focal proposals; a trace guard that receives typed source and field evidence reaches 0/394 observed unsafe allows. These results show that certificates can supervise learned guards but the main claim is deterministic enforcement.

The external grounding row maps a live AgentDojo run to public results-table columns: 97 clean records, 589 attacked records, 100.0% clean utility, 98.5% utility under attack, and 0.0% targeted ASR for the stated model and attack configuration. Table 12 records the metadata needed to replay the row: benchmark version, attack, model, run id, policy pack, scorer/protocol scripts, exact attacked-record coverage, implementation-file presence, and absence of task-id literals or task-id conditionals in the adapter files audited by the artifact.

Two additional replay rows delimit the external evidence. The AgentDojo replay-policy-sensitivity run imports 40 no-defense traces and replays default and field-sensitive CAGE policies, validating 120 records; both replay policies reduce attack ASR from 100.0% to 0.0% on 23 baseline-

TABLE 12. AGENTDOJO PROTOCOL AUDIT FOR THE PACKAGED LIVE ROW.

Protocol check	Status	Evidence
Version/model	ok	AgentDojo v1.1.2; important_instructions; gpt-4o-2024-08-06
Run/traces	ok	cage_live_v1; saved live traces; replayable from artifact outputs
Policy pack	ok	policies/field_sensitive.yaml, sha256 ffd3e85a
Scorer/protocol	ok	scorecard script f3b1e101; protocol a7d8c3d9
Full-suite selection	ok	no user-task or injection-task filters declared
Suite coverage	ok	banking 144, slack 105, travel 100, workspace 240 records
Expected records	ok	589 observed / 589 expected attacked records
Implementation files	ok	adapter, events, monitor, and policy files present
Task-id leakage audit	ok	no exact task-id literals or task-id conditionals found

TABLE 13. MATCHED EXTERNAL POLICY COMPARISON ON SHARED SCENARIO KEYS. THE GPT-4O ROW REPORTS CAPACITY SENSITIVITY.

Run	Model	Records	ASR	Safe util.	Approval
No defense	gpt-4o-mini	32	71.9%	15.6%	0.000
Progent manual	gpt-4o-mini	32	0.0%	71.9%	0.000
CAGE matched	gpt-4o-mini	32	0.0%	84.4%	0.000
CAGE capacity	gpt-4o	32	0.0%	93.8%	0.000

successful attacks, with 50.0% safe utility over the replayed scenario set. The Progent external run imports 96 records with zero schema validation errors. The CAGE replay row is a post-hoc interruption lower bound, so its safe utility is reported conservatively and separated from live CAGE rows.

The AgentDojo row is externally legible and is reported as scoped external evidence. Public-context rows use different models, dates, and defenses and are left to the artifact narrative. The matched Progent slice is the direct comparison: no defense, Progent manual policy, and model-matched CAGE share scenario keys. Progent blocks the evaluated attacks and preserves 71.9% safe utility on that slice; model-matched CAGE also blocks them and preserves 84.4% safe utility. A higher-capacity CAGE run reaches 93.8% safe utility with 0.0% ASR and is labeled as capacity sensitivity. The matched slice is small and same-key and it is included as the strongest executed policy comparator.

7. Related Work and Limitations

AgentDojo is an extensible environment for attacks and defenses in tool-using agents [2]. AgentDyn, Skill-Inject, MCP-SafetyBench, and MCPTox expand the surface to dynamic tasks, skill files, MCP servers, and tool poisoning [3], [4], [5], [6]. ToolHijacker studies prompt injection against tool retrieval and selection [7]. TRUSTDESC hardens tool descriptions by generating trusted descriptions from implementations [8]. CAGE mediates the downstream protected action, preventing lower-integrity or untrusted metadata and manifests from authorizing protected fields.

CaMeL and PFI separate trusted control from untrusted data flows [9], [10]. FIDES explores information-flow control for agent planners [11]. Progent provides programmable privilege policies over tool calls [12]. PCAS is closest in architecture: it compiles declarative policies over dependency graphs and enforces them through a reference monitor [1]. CAGE is independently implemented and targets a different invariant: field-level action integrity over protected action fields, with value scopes, supporting evidence, scoped approvals, trusted delegation, heterogeneous event classes, and replayable certificates. The distinction from PCAS is the target invariant and runtime object. CAGE specifies field-level authorization for protected action fields, the adapter-soundness contract for LLM-emitted fields, and the runtime machinery needed to enforce those objects.

PromptArmor and DataFilter are strong prompt-injection detection/removal defenses, not lexical filters [13], [14]. The prompt-removal run tests that preprocessing defense class on compatible saved traces; true DataFilter with its HF/vLLM model is future adapter work. ARM identifies denial-feedback leakage and uses denial-aware provenance [15]. AttrGuard uses counterfactual causal attribution to ask why a tool call was produced [16]. Recent formal frameworks emphasize contextual security, including source authorization and data isolation [17]. CAGE defines deterministic field-level authorization semantics at the tool boundary.

Limitations follow directly from the theorem. CAGE enforces configured field-level policies over instrumented dependencies. Dependencies outside the event stream, semantic truth, task optimality, and unmediated sinks are outside the guarantee. Incorrect policies are enforced incorrectly. Deterministic builders must record typed inputs, LLM-emitted critical fields must use conservative visible-source dependency closure unless isolated evidence spans are available, and critical values must canonicalize before policy matching. The empirical claims are scenario-evaluation rates over saved outputs and live traces.

Ethical Considerations

The experiments use controlled scenarios, saved traces, and benchmark-compatible agent environments. The artifact supports defensive evaluation of tool-using agents; attack examples reproduce the reported mediation decisions and are scoped to sandboxed sinks or saved outputs.

8. Conclusion

CAGE treats protected action fields as the enforcement unit for tool-using agents. A source may legitimately support one field while lacking authority over another, so the monitor checks each critical field against its dependency closure, canonical value, sink, and trace before execution. Under mediated sinks, conservative adapters, canonicalized critical values, and fail-closed execution, the implemented monitor

enforces canonical field-authority integrity and emits certificates naming the field, source path, policy rule, and missing or satisfied obligation. The evaluated suites show that this field-level invariant blocks the observed targeted field-control attacks with lower approval burden than approval-only mediation, and the packaged artifact provides saved traces, policies, validators, and aggregate outputs for reproducing the reported decisions.

LLM Usage Statement

LLMs were used as model-backed components in the experimental methodology. Model-backed experimental outputs used as evidence are saved in the artifact, scored by scripts or scenario oracles, and reported with the stated model identifiers, denominators, and scope limits.

References

- [1] N. Palumbo, S. Choudhary, J. Choi, G. Amir, P. Chalasani, and S. Jha, “Formal policy enforcement for real-world agentic systems,” 2026. [Online]. Available: <https://arxiv.org/abs/2602.16708>
- [2] E. Debenedetti, J. Zhang, M. Balunovic, L. Beurer-Kellner, M. Fischer, and F. Tramèr, “AgentDojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents,” in *Advances in Neural Information Processing Systems 37 (NeurIPS 2024), Datasets and Benchmarks Track*, 2024. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2024/hash/97091a5177d8dc64b1da8bf3e1f6fb54-Abstract-Datasets_and_Benchmarks_Track.html
- [3] H. Li, R. Wen, S. Shi, N. Zhang, Y. Vorobeychik, and C. Xiao, “AgentDyn: Are your agent security defenses deployable in real-world dynamic environments?” 2026. [Online]. Available: <https://arxiv.org/abs/2602.03117>
- [4] D. Schmotz, L. Beurer-Kellner, S. Abdelnabi, and M. Andriushchenko, “Skill-inject: Measuring agent vulnerability to skill file attacks,” 2026. [Online]. Available: <https://arxiv.org/abs/2602.20156>
- [5] X. Zong, Z. Shen, L. Wang, Y. Lan, and C. Yang, “MCP-SafetyBench: A benchmark for safety evaluation of large language models with real-world MCP servers,” in *International Conference on Learning Representations*, 2026. [Online]. Available: <https://openreview.net/forum?id=7XYjeL46co>
- [6] Z. Wang, Y. Gao, Y. Wang, S. Liu, H. Sun, H. Cheng, G. Shi, H. Du, and X. Li, “MCPTox: A benchmark for tool poisoning on real-world MCP servers,” in *Proceedings of the AAI Conference on Artificial Intelligence*, vol. 40, no. 42, 2026, pp. 35 811–35 819. [Online]. Available: <https://ojs.aaai.org/index.php/AAAI/article/view/40895>
- [7] J. Shi, Z. Yuan, G. Tie, P. Zhou, N. Z. Gong, and L. Sun, “Prompt injection attack to tool selection in LLM agents,” in *Network and Distributed System Security (NDSS) Symposium*, 2026. [Online]. Available: <https://www.ndss-symposium.org/ndss-paper/prompt-injection-attack-to-tool-selection-in-llm-agents/>
- [8] H. Ye, Z. Zhang, J. Jia, and H. Hu, “TRUSTDESC: Preventing tool poisoning in LLM applications via trusted description generation,” 2026. [Online]. Available: <https://arxiv.org/abs/2604.07536>
- [9] E. Debenedetti, I. Shumailov, T. Fan, J. Hayes, N. Carlini, D. Fabian, C. Kern, C. Shi, A. Terzis, and F. Tramèr, “Defeating prompt injections by design,” in *4th IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)*, 2026. [Online]. Available: <https://arxiv.org/abs/2503.18813>
- [10] J. Kim, W. Choi, and B. Lee, “Prompt flow integrity to prevent privilege escalation in LLM agents,” 2025. [Online]. Available: <https://arxiv.org/abs/2503.15547>
- [11] M. Costa, B. Köpf, A. Kolluri, A. Paverd, M. Russinovich, A. Salem, S. Tople, L. Wutschitz, and S. Zanella-Béguelin, “Securing AI agents with information-flow control,” 2025. [Online]. Available: <https://arxiv.org/abs/2505.23643>
- [12] T. Shi, J. He, Z. Wang, H. Li, L. Wu, W. Guo, and D. Song, “Progent: Securing AI agents with privilege control,” 2025. [Online]. Available: <https://arxiv.org/abs/2504.11703>
- [13] T. Shi, K. Zhu, Z. Wang, Y. Jia, W. Cai, W. Liang, H. Wang, H. Alzahrani, J. Lu, K. Kawaguchi, B. Alomair, X. Zhao, W. Y. Wang, N. Gong, W. Guo, and D. Song, “PromptArmor: Simple yet effective prompt injection defenses,” 2025. [Online]. Available: <https://arxiv.org/abs/2507.15219>
- [14] Y. Wang, S. Chen, R. Alkhudair, B. Alomair, and D. Wagner, “Defending against prompt injection with DataFilter,” in *4th IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)*, 2026. [Online]. Available: <https://arxiv.org/abs/2510.19207>
- [15] M. H. Chinaei, “Causality laundering: Denial-feedback leakage in tool-calling LLM agents,” 2026. [Online]. Available: <https://arxiv.org/abs/2604.04035>
- [16] Y. He, H. Zhu, Y. Li, S. Shao, H. Yao, Z. Liu, and Z. Qin, “AttriGuard: Defeating indirect prompt injection in LLM agents via causal attribution of tool invocations,” 2026. [Online]. Available: <https://arxiv.org/abs/2603.10749>
- [17] V. Siu, J. He, K. Montgomery, Z. Wang, N. Gong, C. Wang, and D. Song, “A framework for formalizing LLM agent security,” 2026. [Online]. Available: <https://arxiv.org/abs/2603.19469>

Appendix A. Proof Details

Lemma 2 (Scoped approvals do not grant ambient authority). *An approval or declassification event covers only subjects, fields, sinks, canonical values, and traces named by its scope predicate. It authorizes no subject-field-sink-value tuple outside that scope.*

Proof. Coverage requires simultaneous subject membership, field membership, sink membership, value-predicate satisfaction, trace-predicate satisfaction, and inclusion of the scoped event in the per-field dependency closure. If any component is outside scope, coverage is false. $\square$

Lemma 3 (Grant events neither self-authorize nor poison valid approvals). *A valid grant event authorizes only source-field-sink-value-trace obligations covered by its verified scope. It is not authorized by coverage. However, a valid grant token is not counted as a bad ordinary source merely because the monitor requires it to appear in the per-field dependency closure. An invalid grant token is counted as an ordinary source event and grants no authority.*

Proof. By definition, a grant contributes authority only if $\text{GrantOK}_{\Pi}(p)$ holds and $\text{Covers}_G(p, u, f, k, c, a)$ holds for the target obligation. Coverage is evaluated for target event u and does not imply coverage of p itself. The bad-source definition separates valid grant tokens from ordinary source influence, so the presence of a valid token in $\text{Dep}_G(a, f)$ cannot by itself create a bad-source failure. If $\text{GrantOK}_{\Pi}(p)$ fails, p belongs to the invalid-grant component of $\text{Bad}_{\Pi, G}$ and cannot discharge any obligation. $\square$

Lemma 4 (Persistence does not launder authority). *Assume conservative memory-store and memory-read instrumentation. If a source lacks authority for (f, k) , information derived from it is written to memory, and a later read influences field f at sink k , the action is not executed unless a scoped approval, declassification, or trusted delegation covers the original source for (f, k) and the canonical field value.*

Proof. Memory instrumentation creates paths from the original source to the write and from the write to the read. Ancestor-closed field dependencies include the original source in $\text{SrcDep}_G(a, f)$. Without base authority or scoped approval, the source is in $\text{Bad}_{\Pi, G}(a, f)$, so allow is impossible. $\square$

Lemma 5 (Denial feedback does not launder authority). *If denial-feedback instrumentation records paths from denials to later events that observe or summarize them, and policy grants no denial authority for (f, k) , then a field depending on an uncovered denial cannot be executed at sink k .*

Proof. The denial event is in the ancestor-closed source dependency set. With no base authority or scoped approval, it is a bad source, so the monitor denies or gates the action. $\square$

Lemma 6 (Supporting evidence closes vacuity). *If $(f, k) \in \text{ReqSupport}$ and $\text{SrcDep}_G(a, f) = \emptyset$, the field is unsupported and cannot be allowed.*

Proof. Support requires an authorized source event in the per-field dependency closure or a scoped approval, declassification, or trusted delegation covering such a source. An empty source set satisfies neither existential condition. $\square$

Lemma 7 (Canonicalization and delegation are fail-closed). *If a critical field fails canonicalization, or if a delegation token does not match subject, field, sink, canonical value, and trace, the monitor cannot use that value or delegation to return allow.*

Proof. The monitor denies on $\perp$ before authority matching. Delegated authorization requires all scoped predicates to hold; otherwise the delegation contributes no authorization and any missing authority stays in the bad or unsupported set. $\square$

Appendix B. Artifact Tables

TABLE 14. EXAMPLE POLICY INTENTS.

Field	Authority examples	Support?	Decision example
email body	user task, quoted content	no	allow quoted text
email recipient	user, contacts, delegated sender	yes	deny attacker recipient
network host	user, trusted config, scoped approval	yes	deny manifest host
file path	user, workspace policy	yes	deny path rewrite
repository target	user, CI policy	yes	deny skill-controlled repo

TABLE 15. CANONICAL OBJECTS USED FOR POLICY MATCHING.

Sink	Critical field	Canonical object	Ambiguity denied
Email	recipient/account	mailbox address, account id	display-name spoofing, malformed mailbox
Network	host/URL	scheme, IDNA host, port, path authority	host confusion, unsupported parse
File	path	sandbox-normalized path	traversal, unresolved alias
Repository	repo/branch	owner, remote, repository, branch tuple	fork/alias mismatch
Calendar	attendee/time	attendee id, normalized time interval	timezone or attendee ambiguity
Payment	beneficiary/account	beneficiary rail/account identity	alias or account mismatch

TABLE 16. DECISION CERTIFICATE MODES.

Decision	Certificate witness	Reviewer/auditor check
Allow	field, canonical value, dependency closure, support rule	Every source event is authorized and supporting evidence exists
Deny	bad source or canonicalization failure	Named source lacks field/sink/value/trace authority or value is $\perp$
Approval required	missing approval-dischargeable obligation	Required subject, field, sink, value scope, and trace are explicit

TABLE 17. EXTERNAL COMPARATOR STATUS. EXECUTED EVIDENCE IS SEPARATED FROM FEASIBLE BUT UNEXECUTED ADAPTERS.

Comparator	Overlap	Executed evidence	Use in paper
Progent	AgentDojo / ASB / EhrAgent	96 imported records, 0 validation errors; 32 scenario-key matched slice	Main external policy slice
CaMeL	AgentDojo / indirect injection	archive feasible; no executed CAGE-matched run	Related work and future adapter candidate
PFI	AgentDojo / AgentBench	archive feasible; container required; no executed CAGE-matched run	Related work and future adapter candidate
FIDES	IFC tutorial	archive feasible; separate integration required	Conceptual IFC comparison
PromptArmor/DataFilter	prompt-removal defenses	API prompt-removal baseline executed over 648 compatible records; DataFilter HF/vLLM integration deferred	Main prompt-removal stress comparison plus related work

TABLE 18. CANONICALIZATION, DELEGATION, AND FIELD-CONTRACT CHECKS.

Contract check	Records	ASR	Utility effect	Evidence
Content-only vs. critical rewrite	10 deterministic	0.0% under field policies	benign content auto-allowed by permissive pack	policy-sensitivity field cases
Model-backed field sensitivity	30 proposals	0/30 unsafe allows	approval 96.7% to 46.7%	field-sensitive model slice
Slack canonical tool arguments	5 live records	0.0%	safe utility +20.0 pp	canonical rank slice
Banking delegated authority	144 live records	0.0%	safe utility +8.3 pp, 0 regressions	delegated authority slice

TABLE 19. ARTIFACT EVIDENCE SURFACES.

Claim surface	Artifact evidence	Reproduction status
Main baseline table	saved JSONL traces, policy files, score scripts	regenerated locally from saved outputs
Mechanism ablations	ablation configs and certificates	deterministic replay available
External grounding	saved AgentDojo-compatible scorecards	protocol and mapping included
Policy freeze	policy packs, rule counts, scenario split metadata	artifact records frozen packs
Decision certificates	per-field source, path, rule, and decision records	replay/audit checks included