Fail-Closed Authorization for Tool-Using AI Agents: A Capability Contract and Case Study

Andrey Lesnikov
Independent Researcher
justrunme@gmail.com
ORCID: 0009-0004-7583-4314
https://justrunme.com https://github.com/justrunme

October 2026

Abstract

For tool-using LLM agents the decisive question is whether a concrete tool call is *authorized* before it executes. We describe the Agent Security Contract (ASC), a deny-by-default, pre-execution gate for agent and MCP tool calls. An ASC policy pins a SHA-256 digest of the approved tool catalog, declares per-agent tool allowlists with argument constraints, marks high-impact tools that require out-of-band approval, and bounds call rates; each decision (ALLOW, BLOCK, REQUIRE_APPROVAL) is emitted as an integrity-checked (unkeyed) JSON evidence envelope, and every error path we enumerate and test resolves to BLOCK. We evaluate the open-source implementation on *static, recorded tool-call traces only; no live agent is run.* Bypasses of the v0.1.0 demo policy found in review were fixed in v0.2.0; on a 60-case adversarial probe, allowed attacks fall from 32 to 1 (a policy gap), at the cost of 3 new benign over-blocks. On the author-constructed 66-scenario v0.2.1 corpus, designed around these checks, ASC returns the expected verdict in 66/66 cases; tool-name RBAC (Python and OPA/Rego) admits 30 of its 33 attacks by construction, and mutation testing shows the corpus exercises few policy constraints (8 of 62 single-constraint mutants killed). As a held-out check we replay 132 AgentDojo v1.2 ground-truth traces under policies frozen beforehand. With tool-level approval policies, 25 of 26 non-empty attack traces but also 60 of 97 benign tasks are held for approval. v0.2.0's strict-argument rule makes these unadapted policies block 94 of 97 benign tasks. Policies derived mechanically from AgentDojo's tool signatures first hit v0.2.0's lack of optional arguments (31 benign blocks); with v0.2.1 they reproduce the tool-level verdicts on all 132 traces, with no blocks and no newly admitted attack, so type-only declarations add no protection here. Median in-process latency is about 38 µs per trace.

1 Introduction

Agents that operate real systems—reading repositories, calling APIs, running shell commands, deploying services—turn prompt-injection and tool-poisoning attacks [7, 11, 30, 6] into concrete incidents: exfiltration to an attacker host, reads of a cloud metadata endpoint, or an unapproved production deploy. Detection-oriented defences, such as injection detection by masked re-execution [32] or LLM-based guards [29, 3], and red-teaming tools [20, 16] are essential for finding and flagging such behaviour, but they do not by themselves *prevent* a given call. Classical access-control practice suggests the complementary measure: fail-safe defaults and complete mediation [22], enforced where the agent's intent becomes an action.

Platform teams already enforce this principle for Kubernetes objects through RBAC [25] and admission controllers such as OPA/Gatekeeper [15, 14] and Kyverno [9]. We ask what an analogous admission check should look like for agent tool calls, and what a tool-level allowlist leaves uncovered.

Contributions. This is a systems/engineering contribution rather than a new enforcement mechanism; closely related enforcers exist (§9, Table 5). (1) A deny-by-default *capability contract* combining catalog-digest pinning, argument constraints, approval gating and rate limits, with fail-closed handling of every error path we enumerate (§3). (2) An integrity-checked *evidence envelope* that makes each decision a CI artifact bound to policy hash and commit (§3.4). (3) An open implementation [10] (v0.1.0, the hardened v0.2.0, and v0.2.1 with optional arguments) and an evaluation on static traces: its own corpus with a mutation-based ablation, an adversarial probe, and a held-out replay of AgentDojo ground-truth traces under frozen policies (§5). (4) A before/after account of bypasses found in review and fixed in v0.2.0 and v0.2.1, remaining limitations, and a reproducibility package (§6–§8).

ASC is deliberately narrow: it is not a prompt-injection detector, sandbox, or signer, and is intended to sit alongside red-teaming (e.g., Promptfoo), regression harnesses (OWASP), runtime guardians aligned with the OWASP Agent Control Standard [19], and model-level defences.

2 Threat Model

System and deployment. An agent runtime proposes tool calls; a tool catalog (names, descriptions, input schemas), possibly served by MCP servers [12], is presented to the model. ASC runs in two modes. *Runtime*: the runtime calls the gate (library or loopback HTTP sidecar) once per proposed call, passing the calls already admitted in the episode plus the new call; only an ALLOW lets that call execute, so the decision precedes execution and never depends on future calls. *CI/admission*: a recorded trace is evaluated as a whole, as a regression or admission check. All results in this paper exercise the second mode on recorded traces; the AgentDojo harness (§5.4) exercises the per-call path only offline.

Adversary. The adversary can (i) inject instructions into data the agent reads [7], thereby choosing arbitrary proposed calls and arguments, i.e., fully controlling model output; (ii) alter tool descriptions or schemas in the presented catalog (tool poisoning, including a "rug pull" after initial approval [21, 24, 8, 13]); (iii) induce excessive call volume.

Trusted. The policy author; the approval channel; the evaluator process; the CI system. **The `approved` flag must come from an out-of-band, authenticated approval workflow, never from the model or any field the agent can write.** In all released versions (v0.1.0–v0.2.1) `approved` is a boolean field of the call inside the trace; ASC itself does not authenticate it, so a runtime that lets the model populate it voids approval gating. We treat binding approvals to an authenticated principal as future work.

Catalog pinning. The digest is computed by the gate over the catalog *as presented to the model at evaluation time* (the trace's `tool_catalog`, or the policy's own catalog if none is supplied). It detects a catalog that differs from the approved one, including a post-approval change, only if the runtime passes the catalog actually shown to the model on every call; a runtime that supplies a stale or no catalog defeats this check.

Out of scope. Compromise of the evaluator host, CI or approval channel; forgery of envelopes by a party with write access (the digest is unkeyed); attacks expressible entirely within allowed calls and arguments (e.g., exfiltration through an allowlisted host); semantic misuse no syntactic constraint captures; OS-level isolation.

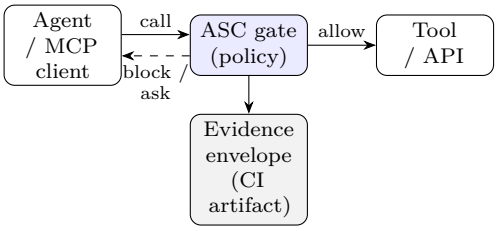


Figure 1: ASC as a pre-execution gate. Only ALLOW reaches the tool; BLOCK/REQUIRE_APPROVAL return to the agent; every decision emits an envelope.

Security goal. No proposed call outside the declared contract executes, and any high-impact call without approval is held. *The guarantee is only as strong as the contract's constraints*: a permissive constraint admits whatever it permits (§6).

3 Design

3.1 Policy contract

A policy (YAML) contains: a **capability catalog** with a pinned `capability_digest`; per-agent **allowed tools**, each with **argument constraints** (type, regex `pattern`, `deny_patterns`, `allow_hosts`, `max_length`); **high-impact tools**; and **rate limits** per trace and per tool. The demo policy declares 8 capabilities, allowlists 7 for `demo-agent` (`secrets.read` is omitted), and marks `filesystem.delete` and `deploy.production` high impact; v0.2.0 tightens its `shell.exec` deny patterns.

3.2 Decision procedure

Evaluation is ordered and short-circuiting (`src/asc/evaluator.py`):

1. Unknown `agent_id` ⇒ BLOCK.
2. Catalog digest mismatch ⇒ BLOCK. The digest is `sha256` over `json.dumps` of the list of {`name`, `description`, `input_schema`} objects, sorted by `name`, with `sort_keys=True`, separators `(",",":")` and `ensure_ascii=False`, UTF-8 encoded (`src/asc/digest.py`). No Unicode normalization is applied, so visually identical but differently encoded descriptions produce different digests (fail-closed), and any other catalog field (e.g., annotations) is *not* covered.
3. Rate limits exceeded ⇒ BLOCK.

4. Per call: tool not allowlisted ⇒ BLOCK; a *declared* argument missing, of wrong type, too long, not matching `pattern`, matching a `deny_pattern`, or with a non-allowlisted host ⇒ BLOCK. In v0.1.0 undeclared extra arguments are not checked (§6).
5. Any unapproved high-impact call ⇒ REQUIRE_APPROVAL; else ALLOW.

v0.2.0 additions. The hardened release (`3f7ce98`; `CHANGELOG.md`) changes step 4 as follows. *Policy-driven* (configured per tool/argument in the YAML): (a) **schema-strict arguments**: any argument not declared for the tool ⇒ BLOCK; (b) **every declared argument is required**: v0.2.0 has no notion of an optional argument, so a call omitting a declared argument ⇒ BLOCK ("missing required argument")—this rule produces every BLOCK in the v0.2.0 schema-derived AgentDojo replay (§5.4) and is relaxed in v0.2.1; (c) the demo policy's `shell.exec` deny patterns are extended (redirection, expansion, quoting and escape characters; `.aws`, `.ssh`, `.kube`, `/etc/shadow`, `/proc/*/environ`, `credentials`). *Hard-coded in evaluator code* (not removable by editing the policy): (d) **lexical path canonicalization** of path arguments before the pattern checks (repeated percent-decoding, NUL rejection, slash collapsing, POSIX `normpath`); (e) **`shell.exec` confinement**: only `ls`/`cat`/`echo`, a fixed set of forbidden characters (redirection, brackets/braces, quotes, backslash), and path tokens confined to `/workspace`; a fixed sensitive-path denylist (`.aws`, `.ssh`, `.kube`, `/etc/shadow`, `/etc/passwd`, `/proc/*/environ`, `/root`, ...) is applied to shell path tokens and to canonicalized path arguments; (f) destructive filesystem operations whose path resolves to a parent of the allowlisted directory or to the root are BLOCK even when approved. The hard-coded checks (d)–(f) use built-in assumptions (the `/workspace` root, the `shell.exec` tool name, a fixed denylist), which is also why policy-level mutation testing (§5) cannot remove them.

v0.2.1 additions. v0.2.1 (`f14e93e`) adds a per-argument `required: true|false` key (boolean, default `true`, so v0.2.0 policies keep their meaning): an omitted optional argument passes, a present one is fully validated, and undeclared arguments still BLOCK. It also closes three ways the new key could be abused (§6): policy load now rejects any unknown argument-constraint key (e.g., a `deny_pattern` typo or an unsupported `default`) and a non-boolean `required`; an argument carrying string constraints (`pattern`, `deny_patterns`, `allow_hosts`, `max_length`) is BLOCK when its value is not a string, even if typed `any`; and a destructive filesystem call without a string `path` is BLOCK even when `path` is optional and the call approved. The demo policy gains an `email.send` capability with an optional `attachments` argument (9 capabilities, 8 allowlisted). BLOCK dominates REQUIRE_APPROVAL: a trace with both an unapproved deploy and a disallowed tool is blocked.

3.3 Fail-closed semantics

The top-level `evaluate` converts any exception into BLOCK. Enumerated and tested error paths: missing, invalid, empty, or digest-mismatched policy; invalid-JSON, non-mapping or malformed trace; missing trace file; non-boolean `approved`; injected evaluator exception; sidecar malformed body; client timeout and unreachable sidecar (15 fail-closed and 6 sidecar/client tests at v0.1.0 and v0.2.0; 16 fail-closed at v0.2.1). Not covered: resource exhaustion (very large traces, pathological regexes in a policy), a hung in-process evaluator without a caller-side timeout, and faults inside the HTTP server loop. The CLI maps verdicts to exit codes 0/1/2 so a CI step fails unless ALLOW; verdicts map to ACS dispositions ALLOW/DENY/ASK [19].

3.4 Evidence envelope

Each decision emits JSON validated against a published schema: `schema_version`, `policy_version`, `policy_sha256`, `scenario_id`, `verdict`, `reason`, `trace_id`, `commit_sha`, `timestamp`, and `envelope_sha256` over the canonical JSON of all other fields. `asc verify` rejects missing fields, invalid values, verdict/disposition inconsistency, digest mismatch and, optionally, a policy hash differing from a given file (8 envelope tests). The digest is *unkeyed*: it is an integrity check against accidental or naive edits, not tamper-evidence against a party who can recompute it; trust derives from the CI run that produced the artifact. `commit_sha` is taken from the environment; envelopes generated outside CI carry `"unknown"` (as in the committed `results/results.json` at v0.1.0). At v0.2.0 the committed `results/results.json` recorded `81b4af0`, a pull-request merge ref, instead of the tag commit `3f7ce98`. v0.2.1 prefers `ASC_COMMIT_SHA` over `GITHUB_SHA` and sets it in CI from the pull-request head (or the pushed commit), so envelopes now record a real commit: the push run 39315964590 records the tag commit `f14e93e`, and the committed `results/results.json` at v0.2.1 records `96482a8`, the head of the pull-request branch, a real commit whose tree was later squash-merged as `f14e93e` (so it is not the tag commit). The CI run remains the evidence of record.

4 Implementation

ASC is a Python 3.11+ package (Apache-2.0; one runtime dependency, PyYAML) of about 1,520 lines in 13 source files at v0.1.0, 1,880 lines in 14 files at v0.2.0 and 1,931 lines in 14 files at v0.2.1: loaders, evaluator, digest, envelope, corpus runner, CLI (`check`, `verify`,

corpus), an HTTP sidecar, and a client that fails closed on timeout. A composite GitHub Action exposes the gate; CI pins actions by commit SHA, uses permissions: contents: read, and self-tests that an attack trace fails the step and that a shell-injection-shaped action input is treated as data. The corpus runner regenerates results/results.json and docs/EVIDENCE.md, so reported numbers are not hand-typed.

5 Evaluation

All results are on static, recorded traces; no live agent or LLM is run. Corpus numbers are on the repository corpus of the stated version (28 scenarios at v0.1.0, commit 77e3aa0; 58 at v0.2.0, 3f7ce98; 66 at v0.2.1, f14e93e) and do not generalize beyond it. Every number is mapped to its source in traceability.md. The primary results are for v0.2.1, against which every experiment was re-run; v0.1.0 (and, where it differs, v0.2.0) results are kept for comparison, and all v0.2.0 outputs remain in the package.

Corpus. v0.1.0: 14 attacks (9 critical, 5 high) and 14 benign controls: unknown tool, disallowed secrets read, goal hijack via injected tool output, MCP catalog poisoning, path traversal, shell metacharacters, out-of-path write, type violation, SSRF to cloud metadata, exfiltration to a non-allowlisted host, two rate-abuse traces, two unapproved high-impact actions. v0.2.0 adds negatives for the bypasses of §6 and their encoding/case variants, shell-syntax abuse, and benign controls for over-blocking regressions, for 29 attacks (23 critical, 6 high; 27 expected BLOCK, 2 expected REQUIRE_APPROVAL) and 29 benign. Each attack is annotated with an OWASP Agentic risk [17] and ACS controls. The v0.2.0 additions were written after, and in response to, the review findings; they are regression tests, not a held-out set. v0.2.1 adds 4 benign email.send controls and 4 attacks on optional-argument handling (missing required, invalid optional, undeclared, wrong type): 33 attacks (23 critical, 10 high; 31 expected BLOCK, 2 expected REQUIRE_APPROVAL) and 33 benign.

5.1 RQ1: Contract conformance on the corpus

v0.2.1: locally (Python 3.13.5) all 66/66 expected verdicts, critical false-ALLOW = 0, benign false-BLOCK = 0/33, suite time 22 ms; push CI run 39315964590 on commit f14e93e (CPython 3.12.15, 2026-10-19 22:02 CEST) reports the same counts (suite times 34 ms and 113 ms in its two jobs). The unit suite (105 collected tests, including 5 Hypothesis property-based tests of the path canonicalizer) passes locally and in CI, and ruff is clean. v0.2.0: 58/58, 0, 0/29, 85 tests (CI run 39312778958). v0.1.0: 28/28, 0, 0/14, 49/49 tests (CI run 38871805381).

5.2 RQ2: Which mechanism accounts for which verdict?

We compare tool-name RBAC (admit iff every call's tool is allowlisted—Kubernetes-style verb/resource granularity) and cumulative additions, plus *leave-one-out* variants of the full policy produced by policy mutation (experiments/ablation_mutation.py); all run the unmodified evaluator of the given version on its own corpus. RBAC was also written in Rego and evaluated with OPA 1.21.2; on all three corpora (28, 58, 66 traces) it gave identical allow/deny decisions to the Python RBAC (REQUIRE_APPROVAL counted as allow). Table 1: RBAC stops only the three scenarios that call a tool absent from the allowlist. At v0.1.0, removing argument constraints re-admits 6 attacks; at v0.2.0 only 5 (type violation, exfiltration, workspace-parent traversal, SSRF, out-of-path write) and at v0.2.1 the same 5 plus the new email.send wrong-type case, because most of the new v0.2.0 negatives are caught by checks implemented in evaluator code (canonicalization, shell confinement, sensitive-path denylist, schema strictness) that a policy-level ablation does not remove. Rate limits account for the 2 rate-abuse traces, approval for the 2 unapproved high-impact actions, and digest pinning for the poisoned-catalog scenario. This attribution is by construction; it shows that tool-level granularity is insufficient for these attack classes, not a detection rate.

Corpus adequacy (mutation testing). Removing each single constraint of the demo policy in turn, only **8 of 45** mutants change any verdict at v0.1.0, **9 of 51** at v0.2.0 and **8 of 62** at v0.2.1. At v0.2.1 none of the 11 new email.send mutants is killed, and http.fetch.url.type is no longer killed: the new rule that a string-constrained argument must be a string makes the type check redundant for the type-violation scenario. Survivors at v0.2.1 still include every max_length, all deploy.production argument constraints and all 11 shell.exec deny patterns, the six new ones included: the shell negatives are caught by code-level confinement whichever policy pattern is removed, so the policy-level patterns are redundant on this corpus, and some could break unnoticed.

5.3 RQ3: Fail-closed behaviour

The 15 fail-closed, 6 sidecar/client and 8 envelope tests listed in §3 pass at all three versions; v0.2.0 adds 14 path-canonicalization tests (5 property-based) and evaluator tests for the new checks, and v0.2.1 adds tests for optional, invalid-optional, missing-required, undeclared and wrong-type arguments and for malformed required values. This is evidence for the enumerated paths only, not a proof over all paths.

Gate	v0.1.0 corpus (28 scenarios)				v0.2.1 corpus (66 scenarios)			
	Atk FA	Crit FA	Ben FB	Exact	Atk FA	Crit FA	Ben FB	Exact
RBAC (Python / OPA-Rego)†	11/14	7/9	0/14	17/28	30/33	21/23	0/33	36/66
RBAC+Approval	9/14	5/9	0/14	19/28	28/33	19/23	0/33	38/66
RBAC+Approval+Digest	8/14	4/9	0/14	20/28	27/33	18/23	0/33	39/66
ASC − argument constraints‡	6/14	4/9	0/14	22/28	6/33	3/23	0/33	60/66
ASC − rate limits	2/14	0/9	0/14	26/28	2/33	0/23	0/33	64/66
ASC − approval	2/14	2/9	0/14	26/28	2/33	2/23	0/33	64/66
ASC (full)	0/14	0/9	0/14	28/28	0/33	0/23	0/33	66/66

Table 1: Each version's gate on its own corpus, which was designed around these checks. FA = attack returned ALLOW; FB = benign not ALLOW. †Python and OPA agree on allow/deny; "Exact" is from the Python baseline (OPA returns a boolean); 2 of the RBAC misses are unapproved high-impact cases returned as allow. ‡Argument names (and, at v0.2.1, their `required` flag) kept, all other constraints removed; deleting the declarations instead makes schema-strict v0.2.x block all benign traces (29/29 at v0.2.0, 33/33 at v0.2.1; v0.1.0: same as shown). v0.2.0 values for all rows are in `anc/experiments/v020/`.

5.4 RQ4: Held-out replay of AgentDojo ground-truth traces

To obtain traces not written by the author, we replay the ground-truth tool-call sequences shipped with AgentDojo [6] (package 0.1.35, benchmark v1.2): user-task ground truths as benign traces and injection-task ground truths (the attacker goal's calls) as attack traces, over the banking, slack, travel and workspace suites. The ASC policies (one per suite) allowlist every suite tool and mark state-changing, outbound or credential tools as high impact; they were written from tool semantics without reading injection goals, and contain no argument declarations or constraints. **Protocol.** Policies, gate adapter and replay script were hashed into `frozen.sha256` (2026-10-19 21:24 CEST) before the first replay. Disclosure: an LLM-free dry run over the same ground truths with the same, unmodified policy file was performed about five minutes before freezing; the policy was not changed afterwards. The replay was first run with the v0.1.0 gate and then re-run with the v0.2.0 and v0.2.1 gates after re-verifying all frozen hashes: *only the ASC library version changed*; policies, adapter, script and AgentDojo package files are byte-identical. No LLM is involved; these are recorded *goal* traces, not full hijacked agent trajectories.

Suite	Benign (user tasks)		Attack (inj. goals)	
	ALLOW	REQUIRE_APPROVAL	ALLOW	REQUIRE_APPROVAL
banking	4	12	0	9
slack	1	20	1	4
travel	14	6	0 (+1 empty)	6
workspace	18	22	0 (+8 empty)	6
total (v0.1.0 gate)	37/97	60/97	1/26	25/26
total (v0.2.0 / v0.2.1 gate)*	3/97	0/97	0/26	0/26

Table 2: Offline replay of 132 AgentDojo v1.2 ground-truth traces (386 calls) under the frozen policies. With the v0.1.0 gate no trace was BLOCK. "Empty": 9 injection tasks with no ground-truth tool call in this release, excluded from denominators. Tool-name RBAC returns ALLOW for all 132. *With the v0.2.0 and v0.2.1 gates (identical results) the other 94 benign and all 26 non-empty attack traces are BLOCK, all with reason "undeclared arguments".

v0.1.0 gate (Table 2, upper part): 25 of 26 non-empty attack goals are held for approval; the remaining one (`slack/injection_task_3`) needs only `get_webpage` on an attacker URL. However, 60 of the 97 benign tasks are also held, so ASC converts these attacks into *approval decisions* rather than stopping them, at a high approval load (§5.5).

v0.2.0 gate, unadapted policies (finding): the same frozen policies now BLOCK 94 of 97 benign and all 26 non-empty attack traces, every one with the reason "undeclared arguments"; the 3 benign ALLOW traces call only argument-free tools. This is not protection: the frozen policies declare no arguments, and v0.2.0's schema-strict rule rejects any undeclared argument. Strict-argument checking thus breaks policies that were not adapted to it—a real deployment cost of the hardening. v0.2.1 keeps schema strictness, and the unadapted row is unchanged under it.

v0.2.0 gate, schema-derived policies: we then generated a new policy per suite *mechanically* (`gen_schema_policy.py`): every suite tool stays allowlisted; every parameter of each tool's published AgentDojo function signature is declared with its JSON-schema type (unions/optionals → `any`; 118 parameters over 74 tools); no further constraints; the high-impact lists are copied verbatim from the frozen v0.1.0 policies. No trace was read. Generator, generated policy and replay script were hashed into a new freeze file `frozen_v020_schema.sha256` (2026-10-19 21:50 CEST) before the first run; the original `frozen.sha256` is unchanged and still verifies. Results (Table 3): of 97 benign tasks, 30 ALLOW, 36 REQUIRE_APPROVAL and 31 BLOCK; of 26 non-empty attack goals, 15 REQUIRE_APPROVAL, 10 BLOCK and

1 ALLOW (`slack/injection_task_3`, `get_webpage`). All 41 BLOCK verdicts have the reason "missing required argument" (e.g., `search_emails.sender` 8×, `send_email.attachments` 4 benign / 7 attack, `create_calendar_event.location` 6 / 1): v0.2.0 treats every declared argument as required and has no notion of optional parameters, so calls that omit an optional parameter are blocked. The 10 blocked attacks are therefore stopped by this artifact, not by a security rule.

v0.2.1 gate, schema-derived policies with optional arguments (main AgentDojo result): the generator was extended in one rule, fixed before any v0.2.1 run (`gen_schema_policy_v021.py`): a parameter with a default in the signature, or absent from its JSON-schema `required` list, gets `required: false` (26 of 118 parameters, e.g., `get_most_recent_transactions.n`, `send_email.attachments/cc/bcc`, `search_emails.sender`); nothing else changed. It writes one complete ASC policy per suite, each loaded with the repository's own validating loader (`load_policy`, including unknown-key rejection and the digest check); generator, policies and replay script were hashed into `frozen_v021_schema.sha256` (2026-10-19 22:06 CEST) before the first run, and the older freeze files still verify. Result (Table 3, last row): 37 benign ALLOW, 60 REQUIRE_APPROVAL, 0 BLOCK; 1 attack ALLOW, 25 REQUIRE_APPROVAL, 0 BLOCK. All 41 v0.2.0 blocks disappear: 24 benign become REQUIRE_APPROVAL, 7 benign become ALLOW, and all 10 artifact-blocked attacks become REQUIRE_APPROVAL. **No attack goal becomes** ALLOW: the only admitted non-empty attack is still `slack/injection_task_3` (`get_webpage`), plus the 9 empty ground truths. The per-trace verdicts are identical to the v0.1.0-gate replay on all 132 traces: on these ground truths, type-only argument declarations add no protection, and the remaining protection is the tool-level approval list (§5.5). Live utility and attack-success rates remain unmeasured; a ready-to-run harness is included (`experiments/agentdojo/`).

	Benign (97)			Attack (26 non-empty)		
Suite	allow	ask	block	allow	ask	block
banking	4	6	6	0	8	1
slack	1	20	0	1	4	0
travel	14	1	5	0	2	4
workspace	11	9	20	0	1	5
v0.2.0	30	36	31	1	15	10
v0.2.1	37	60	0	1	25	0

Table 3: Mechanically schema-derived AgentDojo policies. Per-suite rows and the v0.2.0 total: v0.2.0 gate (frozen in `frozen_v020_schema.sha256`); every BLOCK is "missing required argument" for an optional parameter. v0.2.1 total: v0.2.1 gate with optional parameters marked `required: false` (frozen in `frozen_v021_schema.sha256`); per suite it equals the v0.1.0-gate counts of Table 2. ask = REQUIRE_APPROVAL.

5.5 Discussion: why 60 of 97 benign tasks are held (v0.1.0 and v0.2.1 gates)

A post-hoc analysis of the v0.1.0-gate replay output (`experiments/agentdojo/hold_breakdown.py`; policies unchanged, `frozen.sha256` re-verified) (the v0.2.1 schema-derived replay yields the same 60 holds, trace for trace) shows that every hold comes from a single rule type, the high-impact list—there are no argument-constraint or rate-limit hits. Per suite, 12/16 banking, 20/21 slack, 6/20 travel and 22/40 workspace benign tasks are held. 50 of the 60 held traces contain exactly one high-impact tool (8 contain two, 2 contain three), so the holds are caused not by long chains but by the task's one intended side effect. The triggering tools (first unapproved high-impact call) are, in decreasing order: `create_calendar_event` (8 workspace, 4 travel), `send_channel_message` (6), `send_money` (5), `send_direct_message` (5), `invite_user_to_slack` and `add_user_to_channel` (4 each), `append_to_file` and `create_file` (4 each), `update_scheduled_transaction` (3), and 13 holds spread over 11 other tools (1–2 each). The attack goals concentrate on a subset of the same tools: `send_money` (7 of 25 held attacks), `send_email` (7), `post_webpage` (2), `reserve_hotel` (2). In AgentDojo, then, the dangerous action and the requested action are frequently the *same tool*, differing only in arguments (recipient, amount, URL): a tool-granularity approval list cannot separate them, and any reduction in benign holds without loss of protection must come from argument-level or context-dependent rules—which is precisely what Progent and Conseca generate per task. As a diagnostic only (it uses knowledge of the attack goals and was *not* applied to any policy), 15 of the 60 held benign traces involve only high-impact tools that no attack goal uses. We deliberately did not re-tune the frozen policies on this corpus.

5.6 RQ5: Overhead

On an 8-vCPU Intel Xeon VM (Python 3.13.5, no CPU pinning), in-process `evaluate` over each corpus's parsed traces was measured in 10 independent runs, each with 20 discarded warm-up passes and 200 timed passes. v0.2.1 (66 traces, 13,200 samples per run): per-run medians 37.3–40.9 µs, p99 101.1–120.9 µs; v0.2.0 (58 traces): medians 34.9–37.3 µs, p99 102.7–112.1 µs. v0.1.0 (28 traces): medians 23.9–25.7 µs, p99 35.9–56.4 µs. The corpora differ, so the two are not a like-for-like comparison of evaluator cost. The loopback HTTP sidecar round trip (shipped client, 10 runs, 0 verdict mismatches vs. in-process) had per-run

	v0.1.0	6be4b5c	v0.2.0	v0.2.1
Attack probes ALLOW (of 60)	32	4	1	1
Benign probes ALLOW (of 26)	24	18	21	21
As authored, of 61	33	5	2	2
As authored, of 25	23	17	20	20

Table 4: Reviewer's single-call probe, re-run by us at each commit. Upper rows count `echo\tx` as benign (relabelled); lower rows give the as-authored attack and benign ALLOW counts.

medians of 0.89–0.97 ms and p99 1.45–1.54 ms at v0.2.1 (v0.2.0: 0.86–0.93 ms, 1.40–1.56 ms; v0.1.0: 0.88–0.91 ms, 1.34–1.49 ms). A CLI cold start (`python -m asc check`, 30 runs) had a median of 76.3 ms (68.2–98.2 ms) at v0.2.1 (v0.2.0: 75.5 ms; v0.1.0: 73.6 ms), dominated by interpreter start-up. These are single-machine measurements.

6 Bypasses Found and Fixed

Round 1 (v0.1.0). An independent review found that the v0.1.0 demo policy admitted: `shell.exec cat /root/.aws/credentials`, `cat /etc/shadow` and `cat /proc/self/environ`; `filesystem.list /workspace/..`; `filesystem.read /workspace/..%2fetc/passwd`; approved `filesystem.delete /workspace/out/..`; and an `http.fetch` with an undeclared `headers` argument. We re-verified all seven at `77e3aa0`; all seven are BLOCK at v0.2.0 and v0.2.1 (`experiments/v010_bypass_check.py`). An eighth case, data in the query string of an allowlisted host, remains ALLOW and is out of scope (allowed-channel exfiltration).

Probe. The reviewer then ran a 61-case attack / 25-case benign single-call probe (`experiments/v020/probe/asc-bypass-probe.py`) against an intermediate fix commit (`6be4b5c`, branch `fix/policy-bypasses`) and against v0.2.0. We re-ran the probe ourselves at `77e3aa0`, `6be4b5c` and `3f7ce98`; our verdicts match the reviewer's files case for case.
Relabel. One case authored as an attack, `echo\tx` (`echo`, a tab, `x`), is harmless: executed by `bash` or `sh` it prints `x` and writes no file, and it contains no redirection, substitution, separator or path. The gate's command pattern `^(ls|cat|echo)(\s|$)` accepts any whitespace by design, and adding a redirect, `$(...)` or `;` to the same tab-separated command is BLOCK at v0.2.0. The probe author confirmed the mislabel; we therefore count it as benign (60 attack / 26 benign cases) and show the as-authored counts alongside.

Round 2. At `6be4b5c` four (relabelled) attack probes were still ALLOW; three were shell bypasses—output redirection (`echo pwned > /etc/cron.d/x`), brace expansion (`cat /workspace/{README.md,../../etc/hostname}`) and backslash-escaped traversal (`cat /workspace/..\/..\/etc/hostname`)—and all three are BLOCK at v0.2.0. One attack probe remains ALLOW at v0.2.0 and is a *residual limit*: `filesystem.read /workspace/.env`, a gap in the demo policy rather than in the gate (see §7). Over-blocking: three benign probes that `6be4b5c` blocked are ALLOW again at v0.2.0 (`filesystem.list /workspace/`, `ls -la /workspace`, `echo rotate credentials`). Five benign probes remain BLOCK at v0.2.0: in-workspace `..` segments for read and write (also blocked at v0.1.0), and three new over-blocks relative to v0.1.0, first introduced at `6be4b5c`, which we report as *trade-offs of v0.2.0*: `cat README.md` with a relative path, bare `ls`, and reading `/workspace/certs/ca.pem`. These probes were written by the reviewer and not used to tune the fixes beyond the cases listed. The round-2 fixes (the three shell bypasses and three over-block fixes; intermediate commit `f939361`, squashed into the single v0.2.0 release commit) were developed in the same review iteration that produced the probe, so the before/after comparison is not independent of the probe author either. At v0.2.1 the probe gives verdicts identical to v0.2.0, case for case.

Round 3 (v0.2.1, optional arguments). The CHANGELOG lists three abuse paths of the new `required: false` key found and fixed before release. We verified each by running the same checks (`experiments/v021/abuse/required_false_abuse.py`) against the pre-fix commit `d85f6b6` (branch `feat/optional-args`; not in the tag history, which is one squashed commit) and against `f14e93e`: (1) *silently ignored keys*: a policy with a `deny_pattern` typo or an unsupported `default` key loaded at `d85f6b6`, so the intended constraint was dropped without notice (in our test case the value was still blocked by `pattern`, so the drop was silent but not exploitable there); both are rejected at load in v0.2.1 (a string-valued `required` was already rejected); (2) *non-string values*: with an optional, string-constrained argument typed `any`, an explicit `null` or a list value (`["/etc/passwd"]`) was ALLOW at `d85f6b6` because the string constraints were skipped; both are BLOCK in v0.2.1; (3) *destructive call without a path*: with `filesystem.delete.path` marked optional, an approved delete with `path` omitted (or `null`, if `path` is typed `any`) was ALLOW at `d85f6b6`; it is BLOCK in v0.2.1. These are reachable only through policies that use the new key in these ways; the shipped demo policy marks only `email.send.attachments` optional.

7 Limitations and Threats to Validity

Static traces only: no live agent was run; we report no attack-success or utility numbers under a model. **Corpus construction**: both corpora were written by the same author as the policy and evaluator, each scenario targeting one check, and the v0.2.0 additions respond to known findings; Table 1 is by construction, and mutation testing shows most constraints are unexercised. **Text-only path checks**: v0.2.x canonicalization is lexical and never touches the filesystem, so a symlink inside `/workspace` pointing elsewhere is not detected; the tool executor must resolve the real path (e.g., `realpath` plus prefix check, or `O_NOFOLLOW`) at execution time. **Demo-policy secrets**: the demo policy still permits `filesystem.read /workspace/.env`, a common location for secrets; deployers must deny such paths explicitly. **Residual shell surface**: `shell.exec` remains a regex-and-tokenizer check over a string, not a parsed command; other shells or locales may tokenize differently, and it has not been fuzzed. **Over-blocking trade-offs**: v0.2.0 blocks three benign probe cases that v0.1.0 admitted (relative paths, bare `ls`, `.pem` read; §6). **Optional arguments**: v0.2.0 treats every declared argument as required, which accounts for all BLOCK verdicts in its schema-derived AgentDojo replay; v0.2.1 adds `required: false`, but its three pre-release abuse paths show that each new policy feature widens the surface the loader and evaluator must defend, and only the paths listed in its CHANGELOG were checked. **Held-out replay**: AgentDojo ground truths are attacker *goals*, not hijacked trajectories; 9 of 35 are empty; the original frozen policies are incompatible with v0.2.x schema strictness; the schema-derived policies declare only types, so argument-level protection on this corpus is untested; and the policies were written by the author (frozen before the run, but after a dry run with the same file). **Approval**: protection on the replay rests on approvals; the `approved` flag is not authenticated by ASC, and human approvers are fallible and fatigable. **Allowed-channel misuse** (exfiltration via an allowlisted host's query string, semantically harmful but well-formed calls) is not stopped. **Fail-closed** coverage is limited to enumerated paths. **Envelopes are unsigned**. **No comparison** against Progent, AgentSpec, CaMeL or other enforcers on a common benchmark. **Timing** is a single-machine measurement.

8 Reproducibility

Artifacts: `https://github.com/justrunme/agent-security-contract`; v0.2.1 = commit `f14e93efebdc1ea64863e66f912c688376430a73`, push CI run `https://github.com/justrunme/agent-security-contract/actions/runs/39315964590`; v0.2.0 = commit `3f7ce986ff1aeecaba426ba1591097a6d946739b`, CI run `https://github.com/justrunme/agent-security-contract/actions/runs/39312778958`; v0.1.0 = commit `77e3aa0983825e70331d232088f4a813654efa90`, CI run 38871805381. Experiment scripts and outputs are not in the repository; they ship in this paper's source package under `anc/experiments/` (v0.1.0 outputs at the top level, v0.2.0 outputs in `v020/`, v0.2.1 outputs in `v021/`) with a README listing exact commands and expected outputs, together with `traceability.md`. Versions: Python 3.13.5 (local), CPython 3.12.15 (CI), PyYAML 6.0.3, pytest 9.1.1, ruff 0.17.1, OPA 1.21.2, agentdojo 0.1.35, hypothesis 6.170.2.

```
PAPER=/path/to/unpacked/anc # contains experiments/
git clone https://github.com/justrunme/agent-security-contract
cd agent-security-contract && git checkout v0.2.1 # or v0.2.0, 77
     e3aa0
python -m venv .venv && . .venv/bin/activate
pip install -e ".[dev]" agentdojo==0.1.35
ruff check src tests && pytest -q
python -m asc corpus --policy policy/demo-agent.policy.yaml \
  --corpus corpus/scenarios --results /tmp/r.json \
  --summary /tmp/s.md --evidence /tmp/e.md
E=$PAPER/experiments; P=policy/demo-agent.policy.yaml
python $E/baselines.py $P corpus/scenarios /tmp/b.json
python $E/ablation_mutation.py $P corpus/scenarios /tmp/m.json
python $E/latency.py $P corpus/scenarios /tmp/l.json
python $E/v010_bypass_check.py $P /tmp/v.json
python $E/v020/probe/asc-bypass-probe.py /tmp/probe.json
for f in corpus/scenarios/*.json; do jq .trace $f > /tmp/in.json
  echo "$(basename $f) $(opa eval -f raw -d $E/v021/rbac.rego \
    -i /tmp/in.json data.agent.rbac.allow)"; done
cd $E/agentdojo && sha256sum -c frozen.sha256 --ignore-missing \
  && python replay.py && python gen_schema_policy.py \
  && sha256sum -c frozen_v020_schema.sha256 && python
       replay_schema.py
# v0.2.1 only (v0.2.0 commands above need git checkout v0.2.0):
python gen_schema_policy_v021.py && sha256sum -c
     frozen_v021_schema.sha256 \
  && python replay_schema_v021.py
```

9 Related Work

Attacks and benchmarks. Indirect prompt injection was characterized by Greshake et al. [7] and formalized by Liu et al. [11]; InjecAgent [30], AgentDojo [6], ASB [31] and AgentHarm [1] benchmark agents and provide the dynamic evaluation ASC still lacks.

MCP security. Studies document tool poisoning, rug pulls and related MCP risks [8, 24, 21] and propose enterprise mitigations [13]; digest pinning is one concrete such control.

System-level defences. CaMeL [5] extracts control and data flow from the trusted query and enforces capability-based policies at tool calls; design patterns [2] restructure agents so untrusted data cannot alter control flow; FIDES [4] applies information-flow control; IsolateGPT [28] isolates apps; MELON [32] detects injection by masked re-execution; GuardAgent [29] and ShieldAgent [3] use LLM-based guards. Closest to ASC are policy

enforcers (Table 5): Progent [23] deterministically checks every tool call against symbolic rules over tool names and arguments, blocks unmatched calls by default, and uses LLM-generated, task-specific policies whose SMT-detected expansions require approval; AgentSpec [27] is a DSL of trigger/predicate/enforcement rules, including user inspection, integrated with LangChain; Conseca [26] has an isolated LLM generate just-in-time policies with per-argument regex constraints and human-readable rationales. Progent and CaMeL are evaluated on AgentDojo; ASC is not yet evaluated live and is less expressive than Progent or AgentSpec. ASC's distinguishing elements are engineering ones: static reviewed contracts, catalog-digest pinning, uniform fail-closed error handling, and CI-grade evidence envelopes.

Industry guidance and tools. OWASP's LLM and Agentic Top 10 [18, 17] provide the taxonomy we map to; the Agent Control Standard [19] defines hooks and dispositions to which ASC's verdicts map; the OWASP Agent Security Regression Harness [16] and Promptfoo [20] generate and run attacks and report findings, a role ASC does not attempt. They are complementary: red-team findings can become corpus scenarios, and ASC can be the enforcement point they test.

Policy engines. OPA [15], Gatekeeper [14] and Kyverno [9] provide admission control for cloud-native resources, and Kubernetes RBAC [25] authorizes API verbs on resources. ASC transfers the admission-control pattern to agent tool calls; its contract could be compiled to Rego.

10 Conclusion

Fail-closed, pre-execution authorization is a simple, low-latency control for tool-using agents; its deployment cost lies in policy authoring and approval load, not in latency. On its own author-constructed corpora, by construction, argument constraints, catalog pinning, approval gating and rate limits each account for attacks that tool-level RBAC admits. Review-driven hardening in v0.2.0 closed all seven original bypasses and three further shell bypasses, at the cost of three new over-blocks, and left one probe case admitted (`/workspace/.env`, a demo-policy gap); v0.2.1's optional arguments came with three abuse paths that were closed before release. On held-out AgentDojo goal traces, coarse approval gating holds nearly all attack goals but also most benign tasks; the stricter v0.2.x semantics invalidate a policy written for v0.1.0; and a mechanically schema-derived policy, once optional arguments exist, reproduces the tool-level verdicts exactly, so its type checks add no protection there. The evaluation remains static. The next steps are live evaluation in front of real agents, execution-time path resolution, independently authored and fuzzed corpora, authenticated approvals, and signed envelopes.

References

[1] Maksym Andriushchenko, Alexandra Souly, Mateusz Dziemian, Derek Duenas, Maxwell Lin, Justin Wang, Dan Hendrycks, Andy Zou, et al. AgentHarm: A benchmark for measuring harmfulness of LLM agents, 2024. Published at ICLR 2025.

[2] Luca Beurer-Kellner, Beat Buesser, Ana-Maria Creţu, Edoardo Debenedetti, Daniel Dobos, Daniel Fabian, Marc Fischer, David Froelicher, et al. Design patterns for securing LLM agents against prompt injections, 2025.

[3] Zhaorun Chen, Mintong Kang, and Bo Li. ShieldAgent: Shielding agents via verifiable safety policy reasoning, 2025.

[4] Manuel Costa, Boris Köpf, Aashish Kolluri, Andrew Paverd, Mark Russinovich, Ahmed Salem, Shruti Tople, Lukas Wutschitz, et al. Securing AI agents with information-flow control, 2025.

[5] Edoardo Debenedetti, Ilia Shumailov, Tianqi Fan, Jamie Hayes, Nicholas Carlini, Daniel Fabian, Christoph Kern, Chongyang Shi, et al. Defeating prompt injections by design, 2025.

[6] Edoardo Debenedetti, Jie Zhang, Mislav Balunović, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. AgentDojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents, 2024. Published at NeurIPS 2024 Datasets and Benchmarks Track.

[7] Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection, 2023. Published at AISec 2023.

[8] Xinyi Hou, Yanjie Zhao, Shenao Wang, and Haoyu Wang. Model Context Protocol (MCP): Landscape, security threats, and future research directions, 2025.

[9] Kyverno. Kyverno documentation. `https://kyverno.io/docs/`. Accessed 2026-10-19.

[10] Andrey Lesnikov. agent-security-contract, versions v0.1.0, v0.2.0 and v0.2.1. `https://github.com/justrunme/agent-security-contract`, 2026. Tag v0.1.0: commit 77e3aa0983825e70331d232088f4a813654efa90; tag v0.2.0: commit 3f7ce986ff1aeecaba426ba1591097a6d946739b; tag v0.2.1: commit f14e93efebdc1ea64863e66f912c688376430a73.

System	Policy origin	Argument-level rules	Approval / escalation	Unmatched call	Evaluation (as reported)
ASC (this work)	static, human-written YAML (§3)	regex, type, host allowlist, length on declared args; v0.2.0 rejects undeclared args, canonicalizes paths; v0.2.1 adds optional args (§3)	REQUIRE_APPROVAL for listed high-impact tools; flag not authenticated (§2)	BLOCK (§3)	own 28/58/66-scenario corpora; adversarial probe; offline AgentDojo ground-truth replay (§5.4)
Progent [23]	LLM-generated initial policy, LLM-proposed updates (Sec. 5); manual policies also expressed (Sec. 8.1)	symbolic conditions over tool arguments (Sec. 4.1), implemented with JSON Schema (Sec. 7)	expansions (SMT-checked) need approval; auto-deny / auto-approve / manual approver (Sec. 5, 7, 8.2)	blocked by default, fallback message to agent (Sec. 4.2)	live AgentDojo (ASR 39.9%→1.0%) and ASB (70.3%→3.9%) (Sec. 1, 8.1); MCP benchmark, framework integrations (Sec. 8.3)
AgentSpec [27]	DSL rules written by users (Sec. 3.1) or generated by LLMs (Sec. 5.3)	predicates evaluated on trigger events, domain-specific (Sec. 3.2)	enforcements: `user_inspection`, `llm_self_examine`, `invoke_action`, `stop` (Sec. 3.2)	enforcement applies only when a rule's trigger and predicates hold (Sec. 3.1); default otherwise not stated	code agents, embodied agents, autonomous vehicles (Sec. 5.1–5.4); runtime overhead (Sec. 5.5)
Conseca [26]	generated just-in-time by an isolated LLM from trusted context (Sec. 3.1–3.2)	per-argument regex constraints plus rationale (Sec. 3.2, 4.1)	users may optionally approve a task's policy; policies can be logged and audited (Sec. 3.2)	denied calls return rationale to planner (Sec. 3.3, 4.1); default for unlisted calls not stated	20 synthetic filesystem/e-mail tasks, Gemini 1.5 Pro: 12.0/20 vs. 14.0/20 without policy (Sec. 5)
CaMeL [5]	policies as Python functions defined by the CaMeL engine (Sec. 5.2)	over tool name, arguments and value capabilities/dependencies (Sec. 5.2–5.4)	violations blocked in evaluation; user confirmation envisaged for deployment (Sec. 5.4)	not stated	live AgentDojo: 77% tasks solved vs. 84% undefended (Abstract, Sec. 6)

Table 5: Qualitative comparison from a reading of each full paper (arXiv 2504.11703, 2503.18666, 2501.17070, 2503.18813; section numbers refer to those versions). "Not stated" means we could not find it stated, not that it is absent. Tool-catalog / description pinning is not stated in any of the four. No quantitative comparison was run; reported numbers are from the respective papers and are not comparable to ours.

[11] Yupei Liu, Yuqi Jia, Runpeng Geng, Jinyuan Jia, and Neil Zhenqiang Gong. Formalizing and benchmarking prompt injection attacks and defenses, 2023. Published at USENIX Security 2024.

[12] Model Context Protocol. Model Context Protocol specification, revision 2025-06-18. `https://modelcontextprotocol.io/specification/2025-06-18`. Accessed 2026-10-19.

[13] Vineeth Sai Narajala and Idan Habler. Enterprise-grade security for the Model Context Protocol (MCP): Frameworks and mitigation strategies, 2025.

[14] Open Policy Agent. OPA gatekeeper documentation. `https://open-policy-agent.github.io/gatekeeper/website/docs/`. Accessed 2026-10-19.

[15] Open Policy Agent. Open policy agent documentation. `https://www.openpolicyagent.org/docs/`. Accessed 2026-10-19.

[16] OWASP. Agent security regression harness. `https://github.com/OWASP/Agent-Security-Regression-Harness`. Accessed 2026-10-19.

[17] OWASP GenAI Security Project. OWASP top 10 for agentic applications for 2026. `https://genai.owasp.org/resource/owasp-top-10-for-agentic-applications-for-2026/`, 2025. Accessed 2026-10-19.

[18] OWASP GenAI Security Project. OWASP top 10 for large language model applications. `https://genai.owasp.org/llm-top-10/`, 2025. Accessed 2026-10-19.

[19] OWASP GenAI Security Project. Agent control standard (ACS). `https://genai.owasp.org/resource/agent-control-standard-acs/`, 2026. Accessed 2026-10-19.

[20] Promptfoo. MCP security testing (promptfoo documentation). `https://www.promptfoo.dev/docs`

/red-team/mcp-security-testing/. Accessed 2026-10-19.

[21] Brandon Radosevich and John Halloran. MCP safety audit: LLMs with the Model Context Protocol allow major security exploits, 2025.

[22] Jerome H. Saltzer and Michael D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, 1975.

[23] Tianneng Shi, Jingxuan He, Zhun Wang, Hongwei Li, Linyu Wu, Wenbo Guo, and Dawn Song. Progent: Securing AI agents with privilege control, 2025.

[24] Hao Song, Yiming Shen, Wenxuan Luo, Leixin Guo, Ting Chen, Jiashui Wang, Beibei Li, Xiaosong Zhang, et al. Beyond the protocol: Unveiling attack vectors in the Model Context Protocol (MCP) ecosystem, 2025.

[25] The Kubernetes Authors. Using RBAC authorization. https://kubernetes.io/docs/reference/access-authn-authz/rbac/. Accessed 2026-10-19.

[26] Lillian Tsai and Eugene Bagdasarian. Contextual agent security: A policy for every purpose, 2025. Published in Proc. Workshop on Hot Topics in Operating Systems (HotOS '25), doi:10.1145/3713082.3730378.

[27] Haoyu Wang, Christopher M. Poskitt, and Jun Sun. AgentSpec: Customizable runtime enforcement for safe and reliable LLM agents, 2025. Published at ICSE 2026.

[28] Yuhao Wu, Franziska Roesner, Tadayoshi Kohno, Ning Zhang, and Umar Iqbal. IsolateGPT: An execution isolation architecture for LLM-based agentic systems, 2024. Published at NDSS 2025.

[29] Zhen Xiang, Linzhi Zheng, Yanjie Li, Junyuan Hong, Qinbin Li, Han Xie, Jiawei Zhang, Zidi Xiong, et al. GuardAgent: Safeguard LLM agents by a guard agent via knowledge-enabled reasoning, 2024.

[30] Qiusi Zhan, Zhixiang Liang, Zifan Ying, and Daniel Kang. InjecAgent: Benchmarking indirect prompt injections in tool-integrated large language model agents, 2024. Published at Findings of ACL 2024.

[31] Hanrong Zhang, Jingyuan Huang, Kai Mei, Yifei Yao, Zhenting Wang, Chenlu Zhan, Hongwei Wang, and Yongfeng Zhang. Agent security bench (ASB): Formalizing and benchmarking attacks and defenses in LLM-based agents, 2024. Published at ICLR 2025.

[32] Kaijie Zhu, Xianjun Yang, Jindong Wang, Wenbo Guo, and William Yang Wang. MELON: Provable defense against indirect prompt injection attacks in AI agents, 2025.