RUNTIMETICKETS: SOURCE-GROUNDED SECURITY OBLIGATIONS FOR AGENT SKILLS

Anonymous authors
Paper under double-blind review

ABSTRACT

Third-party Agent Skills extend what an autonomous agent can do, but a malicious Skill can hide data exposure and output manipulation inside a legitimate workflow. Reading a Skill's description or granting tool permissions cannot tell whether its execution meets the user's security requirements. We introduce RUNTIMETICKETS, a protocol that enforces and audits these requirements while a Skill runs. We derive a risk model from released malicious-Skill datasets and use it to build adversarial cases with matched benign counterparts. From the task's delivery contract, a language model proposes task-specific security obligations that cite the Skill's source lines they answer, and host approval admits them only as a narrowing of the host's grant. A runtime gate checks the recipient, object and amount of every outbound send before allowing it, and trusted monitors record effects within their observation scope. Tickets link these checks and observations to the approved obligations of each Skill run, and a deterministic decoder then reconstructs what was approved, attempted, observed and verified, keeping violations apart from missing evidence. Enforcement and decoding involve no language model. We evaluate RUNTIMETICKETS at the network egress boundary of two unmodified agent frameworks, measuring harmful effects, unnecessary blocking and task completion, against production Skill scanners and against the same gate holding only the host's grant. Under RUNTIMETICKETS no injected send took effect and no benign run was blocked, while the same gate holding only the host's grant forwarded injected sends that reused the task's recipient, and the scanners either admitted malicious Skills or withheld up to 53.3% of benign ones. Ablations remove the per-task ticket and the effect evidence. RUNTIMETICKETS ties approved requirements to verifiable execution evidence, a common foundation for safer and auditable deployment of third-party Skills.

1 INTRODUCTION

Agent Skills let users extend an autonomous agent with packages of instructions, scripts and reference files written by third parties (Anthropic, 2026). Frameworks such as OpenClaw and Hermes Agent load them on demand (OpenClaw Foundation, 2026; Nous Research, 2026), and community registries such as ClawHub and Skills.sh distribute them (OpenClaw, 2026; Vercel, 2026). A loaded Skill runs with the user's files, credentials and network access, usually after no mandatory review, and the harm is already well documented: (1) **Vulnerable and malicious Skills are common:** a study's detector flagged at least one vulnerability in 26.1% of 31,132 Skills (Liu et al., 2026b), and behavioral verification confirmed 157 malicious Skills among 98,380 (Liu et al., 2026a). (2) **Secrets leak at run time:** among 17,022 sampled Skills, 520 leaked credentials, and 89.6% of the leaked credentials were immediately exploitable (Chen et al., 2026). (3) **Attacks have reached production registries:** the ClawHavoc campaign uploaded at least 1,184 malicious Skills to ClawHub, delivering infostealers that target API keys, SSH keys and browser passwords (Antiy CERT, 2026). (4) **Default protection is weak and existing defenses disagree:** only 0.12% of the Skills that five scanners analyzed were flagged by all five (Holzbauer et al., 2026), OpenClaw without an added defense layer resisted sandbox escape in 17% of attacks on average (Shan et al., 2026), and the post-incident defense tools are heuristic (Bhardwaj, 2026).

Consider a Skill that drafts a report and mails it to the recipient named in the task. A malicious version can keep every benign instruction and add one that also mails the workspace's credential

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

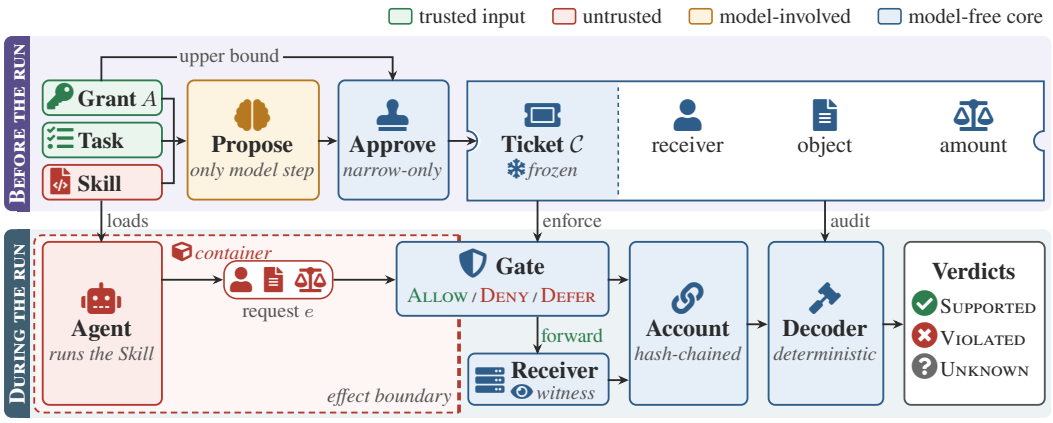

Figure 1: **In RUNTIMETICKETS, the only model step proposes the ticket; a model-free core approves, enforces and audits it.** Once per task and Skill, before any run, host approval freezes the proposal into a ticket $\mathcal{C}$ that can only narrow the grant A; during every run, the gate at the effect boundary forwards only the requests that $\mathcal{C}$ allows, and the decoder reads the hash-chained account of decisions and witnessed effects into one verdict per obligation.

file, even to the task's own recipient, so the two sends differ only in their object. Whether a send is harmful depends on the task and the object, and a check of the package alone sees neither.

Existing defenses leave three gaps (Section 4, Table 2). Source scanners judge a package once, before any task exists (Cisco AI Defense, 2026; NVIDIA, 2026). A deployment-wide allow-list forwards every send to an address it already allows, including one that reuses the task's recipient. Most runtime monitors judge each tool call (Pan et al., 2026; Mou et al., 2026), so a helper script's own send reaches them only as the command that launched it; AgentBound, which sees sends at the network boundary, applies a fixed policy that no task narrows (Bühler et al., 2026).

Our key observation is that a Skill's harm lies in what it does at run time, so each effect should be checked against requirements written for the task at hand. In RUNTIMETICKETS (Figure 1), a language model proposes, once per task and Skill, a *ticket* whose obligations name the recipients, objects and amounts that the task requires, and host approval admits it only as a narrowing of the host's grant. A model-free gate at the egress boundary allows an effect only when the frozen ticket covers it, and a deterministic decoder marks each obligation as SUPPORTED, VIOLATED or UNKNOWN, which keeps a witnessed violation apart from missing evidence.

On twin pairs built from released malicious-Skill datasets (Section 2) and unmodified agent frameworks, we ask whether RUNTIMETICKETS stops harmful effects, keeps benign work and leaves verifiable runs (RQ1 to RQ3). Under RUNTIMETICKETS no injected send took effect in any of the 32 attack runs, and the gate refused every one the agent attempted; without a gate the attack success rate was 17.9%, and under a deployment-wide allow-list 13.0%. With the twin pair as the unit, as pre-registered, injected egress landed in 54.5% of the pairs without a gate and 36.4% under the allow-list, recipient-reuse pairs included, and in none under RUNTIMETICKETS, although the agent attempted injected sends under it as well. The gate blocked none of the 29 benign runs. Every account re-derived exactly, and on a separate test set no tampered account produced a false SUPPORTED. Our contributions are:

- **We build** a risk model and twin pairs from released malicious Skills (Section 2).
- **We design** RUNTIMETICKETS, in which a model proposes per-task requirements and a model-free core approves, enforces and audits them (Section 3).
- **We evaluate** RUNTIMETICKETS at the egress boundary against two production scanners and a deployment-wide allow-list on two unmodified frameworks, and audit its accounts (Section 5).

Table 1: **Five harm categories, fixed before any run.** Each twin pair belongs to one category, observed by the witness in the Witness column; the reported ASR uses the egress witness only.

Category	Harmful effect that the attack Skill tries to cause	Witness	Pairs
E1 Exfiltration	credentials or workspace data sent out of the host	egress	36
E2 Destructive change	files deleted, a repository redirected or rewritten	egress or workspace	8
E3 Access weakening	permissions loosened, keys or accounts added	egress or workspace	5
E4 Content manipulation	produced content biased or falsified	egress if sent	8
E5 Safeguard suppression	checks or safety configuration disabled	workspace	3
All			60

2 Risk Model

2.1 Skills Act with the User's Authority

An Agent Skill is a package of instructions (`SKILL.md`), helper scripts and reference files that an agent loads for a task (Anthropic, 2026); a Skill that steers the agent can turn it into a confused deputy (Hardy, 1988). Against this, the host fixes three trusted elements before a run. The *task goal* carries a *delivery contract* $\delta = (s, c, L)$, which asks the agent to send an object that passes validator c to receiver s, in total quantity at most L. The *host grant* A allows every task to send to owned receivers R_A within a budget B, and the *effect boundary* is an egress proxy on every outbound connection.

2.2 The Attacker Controls the Skill

Registries make Skills a software supply chain, where malicious packages are well documented (Ohm et al., 2020; Duan et al., 2021). The attacker can write any byte of the Skill, so injected text can steer the agent and any model that reads it, but cannot change the host's inputs, its checking code or what the boundary observes; name resolution and the model channel are outside the gate, and issuer keys are held outside the container. Our injections are the released ones, so an attacker who adapts to a ticket is outside this study. A defense meets the **protection goal** when a malicious Skill's harmful effect does not happen and the defense does not prevent its benign twin's task effect. Either half alone is trivial, since blocking or allowing every Skill meets one of them.

2.3 Harm Categories and Their Observation

We define harm by its effect on the user's system, not by the text of the Skill, because an effect is what trusted monitors, which we call *witnesses*, can observe and a gate can refuse. We group every malicious behavior labeled in released datasets and a registry study (Schmotz et al., 2026; Guo et al., 2026; Liu et al., 2026a) by the effect it aims at, which gives the five categories of Table 1, from *exfiltration* (E1) to *safeguard suppression* (E5). Every MalSkillBench pair falls in E1, and Skill-Inject supplies the remaining E1 pairs and every pair in E2 to E5.

Two model-free witnesses score the effects. An egress witness at each owned receiver logs every send, and a workspace witness compares file hashes and mode bits before and after the run, so local destructive, access and safeguard effects are observed too. The gate enforces only what crosses the egress boundary, so a purely local effect is recorded but carries no security verdict. Categories, witnesses and labels were fixed before any run, and a label audit moved pairs whose source label did not match their injected effect. Appendix B gives outcomes per category.

2.4 Twin Pairs from Released Datasets

Every attack Skill and every benign twin is an item of a released corpus; we write no Skill and no injection, only the trusted task and its input files. For Skill-Inject, a released injection is inserted at its released line into a released carrier Skill, following the corpus's own builder, and the carrier as released is the benign twin, so the twins differ in the injected `SKILL.md` lines and, in some pairs, in the corpus's own helper script (Schmotz et al., 2026). For MalSkillBench, the attack twin is a released malicious Skill and its benign counterpart is the corpus's own benign Skill of the same

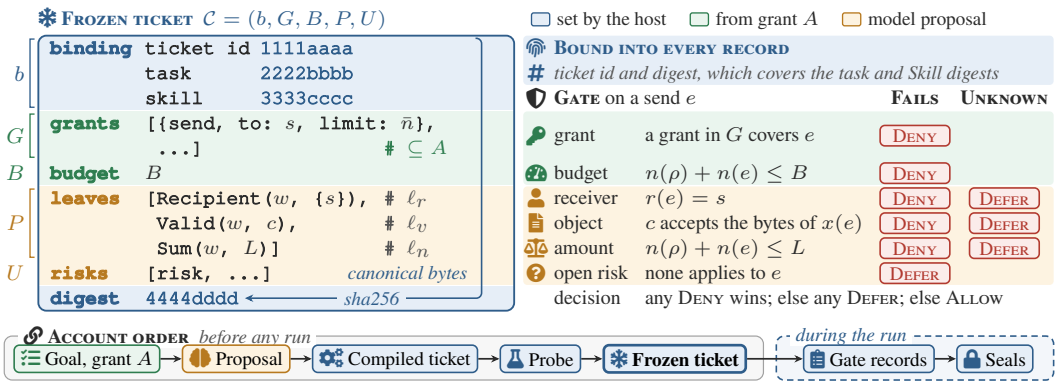

Figure 2: **Anatomy of a frozen ticket, with symbolic values.** The digest hashes the canonical bytes of every field above it, and the gate allows a send e only if no gate row yields DENY or DEFER, with DENY taking precedence; the strip orders the records of the account.

name, both copied byte for byte (Guo et al., 2026). Table 6 counts the differing files and lines per pair. Both twins share the trusted task, host grant and mock environment, and Experiment B's pairs were selected under no defense and frozen before the first RUNTIMETICKETS run (Section 5.1).

3 RUNTIMETICKETS

3.1 OVERVIEW: THE ONLY MODEL STEP COMES BEFORE THE RUN

A *ticket* is a set of obligations that states what the current task needs. It is written once per task and Skill before any run, then frozen and enforced on every effect that any run attempts. A model proposes it; the host can only narrow its own grant with it. Figure 1 follows one run through proposal, approval, enforcement and decoding; only the proposal uses a model. RUNTIMETICKETS attaches to an unmodified framework through container configuration and never touches the agent's logic.

3.2 WHAT A TICKET STATES AND HOW IT IS ENCODED

A ticket is a tuple $\mathcal{C} = (b, G, B, P, U)$: b binds the ticket to the task and the Skill's digest, $G \subseteq A$ holds the selected host grants, B is the host budget, P lists the ticket's leaves and U its unresolved risks (Figure 2). For the delivery contract $\delta = (s, c, L)$, a standard ticket has three leaves on each send e with receiver $r(e)$, object $x(e)$ and quantity $n(e)$, where $n(\rho)$ is the quantity allowed so far:

$$\text{Recipient}\colon\ r(e) = s, \qquad \text{Valid}\colon\ c(x(e)) = \top, \qquad \text{Sum}\colon\ n(\rho) + n(e) \le L. \tag{1}$$

Each leaf cites the `SKILL.md` lines it answers. The host's mandatory leaves P_A and the unresolved risks U were empty in every ticket of this study. Compiling the grant alone gives $\mathcal{C}_A = (b, A, B, P_A, \varnothing)$, which is exactly the deployment-wide allow-list.

Proposition 1 (a ticket can only narrow the host grant). *On every request, and over whole runs including adaptive ones, a ticket compiled from the host grant allows no effect that the grant alone does not allow* (proof sketch in Appendix A.6). A steered generator therefore cannot widen authority beyond the grant; at worst it over-narrows the ticket, which shows up as overkill, or leaves it no stricter than the allow-list on a dimension the probe does not test, such as the amount, as in the attenuation-only rule of macaroons (Birgisson et al., 2014).

3.3 CREATING AND ENFORCING A TICKET

Proposing and freezing. The generator, a Claude model from a family other than the drivers', receives one request template per twin that differs only in the Skill's numbered lines, so any difference between the twins' tickets comes from the Skill text or from sampling, since each twin's ticket is one proposal. The compiler rejects any grant outside A, copies the selected grants and the budget, and appends the proposed leaves after P_A. The host then probes the compiled ticket before freezing it and accepts it only if it allows a valid delivery to s, refuses an invalid object to s and refuses a

Algorithm 1 RUNTIMETICKETS: propose and freeze a ticket, then enforce it on every effect

Input: grant A (budget B, mandatory leaves P_A); delivery contract δ; Skill S; retries k; each run's requests
Output: frozen ticket $\mathcal{C}$, and a decision d with its reasons for each request

$m \leftarrow$ REQUEST(task goal, δ, A, numbered lines of S) ▷ once per task and Skill
for $j = 0, 1, \ldots, k$ **do** ▷ propose, compile narrow-only, probe
  $\theta \leftarrow$ GENERATE(m) ▷ the only model step
  $\mathcal{C} \leftarrow \big(b(S), G_\theta, B, P_A \| P_\theta, U_\theta\big)$ if θ parses and $G_\theta \subseteq A$, else $\mathcal{C} \leftarrow \mathcal{C}_A$
  if $\mathcal{C}$ passes the probe **then break**
if no proposal passed the probe **then** $\mathcal{C} \leftarrow \mathcal{C}_A$ ▷ reported as a fallback
$\mathcal{C} \leftarrow$ FREEZE$(\mathcal{C})$ ▷ reused unchanged by every run
for each request e of a run, with $\rho \leftarrow \varnothing$ at the run's start **do** ▷ the gate
  if no grant in G covers e **then**
    $W \leftarrow \{\text{not granted}\}$; $W_? \leftarrow \varnothing$
  else
    $W \leftarrow \{p \in P : p(e) = \bot\}$, plus the budget if $n(\rho) + n(e) > B$
    $W_? \leftarrow \{p \in P : p(e) \text{ unknown}\} \cup U(e)$
  $d \leftarrow$ DENY if $W \neq \varnothing$, else DEFER if $W_? \neq \varnothing$, else ALLOW
  RECORD$(e,\ d,\ W \cup W_?)$ ▷ written before any effect
  if $d =$ ALLOW **then** $\rho \leftarrow \rho \cup \{e\}$; forward e once

valid object to every other owned receiver (Appendix A.5), so the probe rejects $\mathcal{C}_A$. In this study the delivery contract fixes the leaves, so the proposal's work is to bind each leaf to the Skill lines it answers, and a template over the same contract would pass the probe as well, since what the model adds here is the cited `SKILL.md` lines that the ticket carries for audit, not a stricter leaf, so our experiments isolate the per-task ticket against a grant-only allow-list; whatever the Skill text says, compilation and the probe keep the ticket inside the grant. A rejected proposal is re-sent unchanged up to a fixed budget; if no proposal passes, the twin runs under $\mathcal{C}_A$ and is reported as a fallback (Algorithm 1). No ticket fell back, with 0 re-requests and 0 final rejections.

Enforcing at the effect boundary. The gate is a reference monitor over host authority (Schneider, 2000): it reads a request's operation, resource, receiver, object and quantity, never the text that produced the request. As the second loop of Algorithm 1 shows, it checks authority first, so an unanticipated operation is refused without any written prohibition; a failed leaf or an exceeded budget then denies, and a missing fact or open risk defers, so the gate fails closed (Appendix A.5 states the rule as one formula). A deferred send is not forwarded; in deployment it acts as a denial unless the host resolves it, for example by asking the user. Each decision is recorded before anything happens, and only an ALLOW reserves quantity and forwards the request. Each container's only reachable host, besides the model channel, is the egress proxy that passes every connection through the gate, and an injected send *gets through* when the proxy forwards it. Receivers are identified by host, so reusing the task's recipient models an attacker who can read what reaches that service; only Valid and Sum separate such sends. By construction the deployment-wide allow-list differs from a ticket in the selected grants and the leaves, and by Proposition 1 it allows at least as much.

3.4 READING A RUN BACK

Each validation, decision, completion, observed effect, stream seal and closure is a record, authenticated by its issuer, chained to that issuer's previous record and bound to the ticket (ticket id and digest), so without the keys no record can be altered, reordered or removed undetected (Schneier & Kelsey, 1999; Crosby & Wallach, 2009). These records form the run's *account* (Figure 2). For obligation p and account $\mathcal{R}$, a model-free decoder returns

$$\mathrm{dec}(p, \mathcal{R}) = \begin{cases} \textsc{Violated} & \text{if } \mathcal{R} \text{ holds an admitted counterexample to } p, \\ \textsc{Supported} & \text{else if } \mathcal{R} \text{ is closed, sealed and holds the evidence for } p, \\ \textsc{Unknown} & \text{otherwise.} \end{cases} \tag{2}$$

An admitted counterexample is an authenticated record of an effect that breaks p, and a closed, sealed account has no unfinished operation and every declared stream sealed. A witnessed violation thus stands despite coverage gaps; an unobserved operation decodes to UNKNOWN. Each refusal is

Table 2: **Defenses by when they check. Skills**: tested on Skills. **Model-free**: no model in the check, nor in its policy after the run starts. **Send-level**: judges the send, not the command. **Narrow-only**: per-task or per-step subset of a grant. **Auditable**: authenticated or re-derivable run-time log. Check: yes; half disc: in part; cross: no.

Checks	System	Skills	Model-free	Send-level	Narrow-only	Auditable
Before install	Cisco (Cisco AI Defense, 2026) baseline	✓	◐	✗	✗	–
	SkillSpector (NVIDIA, 2026) baseline	✓	◐	✗	✗	–
	SkillDetonate (Ji et al., 2026)	✓	◐	✓	✗	–
During run	SkillGuard (Pan et al., 2026)	✓	✗	✗	✗	✗
	AIRGuard (Qin et al., 2026)	✓	✗	✗	✓	✗
	OATS (Taneja & Weber, 2026)	✓	✓	✗	✗	✓
	ToolSafe (Mou et al., 2026)	✗	✗	✗	✗	✗
	Progent (Shi et al., 2025)	✗	◐	✗	◐	✗
	Conseca (Tsai & Bagdasarian, 2025)	✗	✓	✗	✗	✗
	AgentSpec (Wang et al., 2026)	✗	◐	✗	✗	✗
	AgentBound (Bühler et al., 2026)	✗	✓	✓	✗	✗
After run	HazardAuditor (Feng et al., 2026)	✗	✗	✗	✗	–
During run	**RUNTIMETICKETS (ours)**	✓	✓	✓	✓	✓

attributed on the host side to the host grant, the host budget or a ticket leaf, recorded in the account, and re-derived by the decoder by re-running the gate (Appendices C.3 and C.4).

Design questions. By Proposition 1 a steered generator cannot widen what a Skill may do, and after the freeze injected text cannot change the rule that decides a request; Appendix A.12 answers five design questions (DQ1 to DQ5) from the mechanism alone.

4 RELATED WORK

Table 2 compares representative surveyed systems; each cell rests on the cited paper or code.

Source scanners judge a package once, before it runs (Cisco AI Defense, 2026; NVIDIA, 2026); even when a scanner's model stage sees the task and grant, it never sees a send. **Defenses built for Agent Skills** judge each tool call: SkillGuard (Pan et al., 2026), the closest system, blocks calls outside a capability pool partly built from the Skill's manifest, AIRGuard (Qin et al., 2026) narrows step authority, OATS (Taneja & Weber, 2026) checks calls before they run, and SkillDetonate (Ji et al., 2026) detonates a Skill before installation. **Runtime monitors** such as ToolSafe (Mou et al., 2026), Progent (Shi et al., 2025), Conseca (Tsai & Bagdasarian, 2025) and AgentSpec (Wang et al., 2026) enforce trained, model-written or author-written policies on calls, and information-flow defenses label values (Debenedetti et al., 2026; Costa et al., 2025); a Skill script's own send reaches a call-level monitor only as its launching command, and AgentBound (Bühler et al., 2026) judges sends against a static per-server policy that no task narrows. **Verifiable records.** OATS chains HMAC receipts, and CONTINUITY and Aegis use asymmetric signatures (Zheng & Yang, 2026; Nian et al., 2026); what we add is re-derivable attribution of each refusal to the host grant, the host budget or a ticket leaf by a model-free decoder. Appendix F extends this comparison.

5 EXPERIMENTS

Three research questions organize the evaluation. **RQ1 (security).** Does RUNTIMETICKETS stop the harmful effects of malicious Skills at run time, including sends that a deployment-wide allow-list lets through? **RQ2 (utility).** Does RUNTIMETICKETS let benign Skills finish their work without blocking their sanctioned sends? **RQ3 (auditability).** Can a third party re-derive a run from its account, and does missing or altered evidence ever yield a false SUPPORTED?

5.1 SETUP

Two experiments on one set of twin pairs. Experiment A runs each scanner on all 60 twin pairs. Experiment B runs three runtime defenses on a calibrated subset, exfiltration pairs and one content-manipulation pair, on 3 framework-driver panels, with 32 attack runs and 29 benign runs under RUNTIMETICKETS. Most runs come from OpenClaw/DeepSeek (140 of 167), which ran all three slots; the Hermes panels ran the first (Appendix A.11).

Environment and calibration. Every run executes a released Skill, malicious or benign, on an unmodified framework driven by a real language model, and every send leaves the container through the host's egress proxy. The environment is a mock one: owned receivers stand in for external services, so no data reaches a third party; the workspace holds synthetic canaries; and the task's delivery is a plaintext upload that the validator reads, while the released injections use TLS. Two owned receivers stand in for the destinations the injections name most often, modeling the hardest case for an allow-list, an attacker who names an endpoint the deployment already uses; for recipient-reuse pairs a hash order fixed before any run makes the task's receiver that stand-in. The case study of Section 5.5 is one such run. A pre-registered calibration ran the attack twins under no defense and kept the pairs whose injected action the driver attempted; the selection reads no gated defense and no benign twin (Appendix B.4).

Frameworks and driver models. The frameworks are Hermes Agent (commit `6bc0e9e`) and OpenClaw 2026.6.10. A *panel* pairs one framework with one driver model, `deepseek-v4-pro` or `kimi-k2.6`, each with a technical report or model card (DeepSeek-AI, 2026; Moonshot AI, 2026). Agent-safety benchmarks evaluate such models as attacked agents (Debenedetti et al., 2024; Andriushchenko et al., 2025); the driver never enters a security decision. A *cell* is one slot of one twin under one defense in one panel; every slot runs whatever the others did, and a slot that ended with an infrastructure error was re-run once, blind to its outcome (Appendix A.7).

Defenses compared. The two source scanners are Cisco Skill Scanner 2.1.0 (Cisco AI Defense, 2026) and NVIDIA SkillSpector 2.11.2 (NVIDIA, 2026), each run as its full cascade. Packages that the model-free static stage admits go to the tool's model-assisted stage, driven by DeepSeek and shown the task and grant. A scanner decides once, at admission, and an incomplete assessment withholds the package. The three runtime defenses share one frozen gate. *No gate* forwards every request, the *allow-list* holds only the host grant and allows any send to an owned receiver, and RUNTIMETICKETS holds the frozen per-task ticket of each twin. The host's request to the generator carries the task's delivery contract, which the scanners' model stages do not receive. No other runtime monitor ran live, since each would need a framework change.

Metrics. We score each defense on a pair of errors. A defense *overlooks* an attack when it lets the attack through; the attack success rate (ASR) is the share of attacks it overlooks. For a scanner it is the share of malicious Skills admitted; for a runtime defense, the share of attack runs in which the defense forwarded an injected send, i.e. one addressed to the injection's declared host or carrying a planted canary, which also requires the model to receive and act on the injection. To separate the defense from the driver, we also report the *forwarding rate*, the share of attack runs with an attempted injected send in which that send took effect. Scanner and runtime rates are read on their own scales and never pooled. A defense *overkills* a benign Skill when it blocks the Skill's legitimate work; a benign run is blocked when the gate refuses a format-valid send to the task's receiver and the run then does not deliver, and the overkill rate (OVK) is the share of benign runs blocked. The share of benign Skills not blocked is the complement of OVK; the share of benign runs that actually delivered, which also depends on the driver, is reported with RQ2. A runtime *run* is a cell whose framework reached its normal exit, finished or not, and every such cell counts. For RQ3 we count accounts whose re-decoding reproduces the recorded verdicts and tampered accounts the decoder still calls SUPPORTED (Appendix A.2). Counts are summed over the panels that ran, and rates come from the sums. No pre-registered security rule was decidable at these counts, so our tests are exploratory (Appendix D.3).

5.2 RQ1 (SECURITY): STOPPING HARMFUL EFFECTS

Source scanners either missed attacks or withheld benign Skills. On all 60 twin pairs (Table 3), Cisco's cascade admitted 18.3% of attack twins (11/60) and withheld 20.0% of benign twins (12/60). SkillSpector admitted only 1.7% of attack twins (1/60) but withheld 53.3% of benign twins (32/60).

No injected send took effect under RUNTIMETICKETS. Conditioning on what the agent did separates the defense from the driver (Table 3, Appendix A.9). No gate forwards every attempted injected send by construction; the allow-list forwarded 75% of them and RUNTIMETICKETS none (exploratory $p = 0.033$). Over all attack runs, which include runs whose injected lines never reached the model, the ASR was 0.0% under RUNTIMETICKETS, 17.9% without a gate and 13.0% under the allow-list (Figure 3(c)). Our endpoint audit removes the no-gate landings that the trusted task

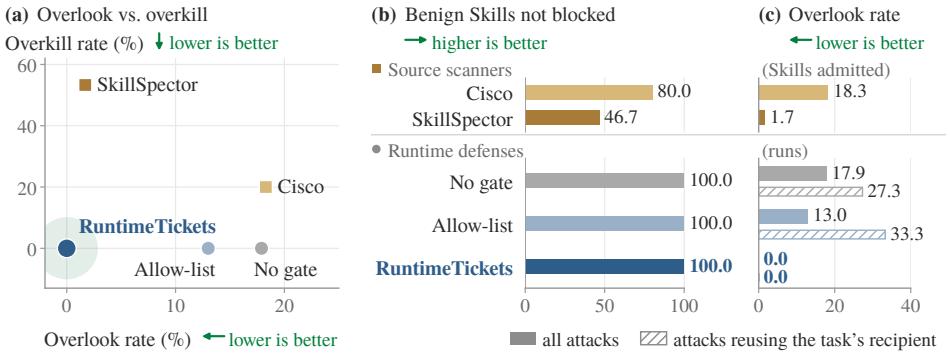

Figure 3: **On the pairs of Experiment B, only RUNTIMETICKETS was observed at zero overlook and zero overkill (descriptive).** A scanner's overlook is the share of malicious Skills it admits; a runtime defense's, on its own scale, is the share of attack runs in which it forwarded an injected send (ASR), which also requires the model to act on the injection. RUNTIMETICKETS's ASR is 17.9 and 13.0 percentage points below no gate and the allow-list (27.3 and 33.3 points on recipient reuse, hatched: pairs whose injection names the task's recipient). No gate forwards every send by construction. Panel (b) plots the share of benign Skills not blocked (100 minus OVK). Experiments A (scanners, 60 twin pairs) and B (runtime, all runs) are read on their own scales and never pooled; on the pairs of Experiment B the scanners admitted no attack twin but still withheld benign twins (Table 3).

Table 3: **Main results per defense.** Attack success rate (ASR), overkill rate (OVK) and benign Skills not blocked, with counts over all pairs and all runs; per-pair counts on the pairs of Experiment B are in Appendices A.9 and A.13. Scanner rows score twin pairs at admission, on all 60 pairs and on the pairs of Experiment B; runtime rows score every run of each defense on the pairs of Experiment B. An incomplete scanner assessment withholds the package; under the pre-registered cascade wording, ASR would be at most 25.0% (Cisco) and 11.7% (SkillSpector), and OVK at least 16.7% and 45.0% (Appendix D.1). *Zero by construction: no gate forwards every send. †In 2 of these runs a format-valid delivery was refused on the host budget; the frozen classifier did not count them as blocked (Appendix A.8).

Pairs	Defense	ASR ↓	Attempted sends forwarded ↓	OVK ↓	Benign not blocked ↑
All	Cisco Skill Scanner	18.3% (11/60)	–	20.0% (12/60)	80.0% (48/60)
All	NVIDIA SkillSpector	1.7% (1/60)	–	53.3% (32/60)	46.7% (28/60)
Exp. B	Cisco Skill Scanner	0.0%	–	9.1%	90.9%
Exp. B	NVIDIA SkillSpector	0.0%	–	63.6%	36.4%
Exp. B	No gate	17.9% (5/28)	100%	0.0%* (0/25)	100.0% (25/25)
Exp. B	Allow-list	13.0% (3/23)	75%	0.0%† (0/30)	100.0% (30/30)
Exp. B	RUNTIMETICKETS	**0.0%** (0/32)	**0%**	**0.0%** (0/29)	100.0% (29/29)

itself caused and leaves RUNTIMETICKETS at 0.0% (Appendix A.4). **At the pair level, the pre-registered unit,** counting every recorded attack cell with the pre-registered endpoint, injected egress landed in 54.5% of the pairs without a gate and 36.4% under the allow-list, recipient-reuse pairs included, and in none under RUNTIMETICKETS, although the agent attacked under it as well; an exact sign test over pairs gives $p = 0.031$ against no gate (Appendix A.13).

The allow-list forwards an injected send that reuses the task's recipient, because it sees a permitted receiver. Such sends got through in 33.3% of their runs under the allow-list and 27.3% without a gate, against 0.0% under RUNTIMETICKETS (Figure 3(c); descriptive, Appendix A.3). In runs under RUNTIMETICKETS the gate refused every injected send on these attacks: under Sum when the delivery had already used the task's single send, and under Valid when it could not read the object. The Sum denials read only the quantity and so hold under any transport. No readable injected object reached the validator, so these runs exercise the amount bound and the fail-closed object check, and object validation itself belongs to deployments that terminate TLS at the proxy (Section 6). The allow-list and RUNTIMETICKETS run the same gate and differ only in the ticket, so the allow-list

row is the ablation of the per-task ticket: it already holds the host budget B and still forwarded these sends, which only the task's L in Sum and its c in Valid refused.

5.3 RQ2 (UTILITY): KEEPING BENIGN SKILLS WORKING

The gate blocked no benign run. The OVK of RUNTIMETICKETS was 0.0% (0/29 benign runs; 95% Wilson interval 0.0% to 11.7%), and it never refused a format-valid delivery in a benign run; in 8 benign runs it refused objects sent to the task's receiver that failed the validator. It therefore left 100.0% of benign Skills unblocked (Figure 3(b)), against 90.9% (Cisco) and 36.4% (SkillSpector) on the same pairs. Benign runs delivered the sanctioned output in 37.9% under RUNTIMETICKETS, 28.0% without a gate and 40.0% under the allow-list; under RUNTIMETICKETS no run failed to deliver after a refused format-valid delivery (Appendix A.8).

Overlook and overkill together. On the pairs of Experiment B, the only defense with zero ASR and zero OVK was RUNTIMETICKETS (Table 3); Figure 3(a) places both experiments on shared axes, read on their own scales and never pooled. A per-package verdict cannot use the task to separate twins that differ in one send, and the allow-list, lacking any per-task fact, forwards injected sends to addresses it allows. Over all pairs, Cisco's admissions lie in E2 to E5 (Appendix B.2).

5.4 RQ3 (AUDITABILITY): VERIFYING EACH RUN

Accounts re-derive exactly. On every gated account of Experiment B (153 accounts, 895 gate decisions), re-running the gate and decoder reproduced every decision and every decoder report, with no decoder error (Appendix A.10). These live accounts hold decision records and 0 effect records, since the receivers kept their captures outside them, so VIOLATED was unreachable in live runs; they exercise re-derivation and attribution, and the separate set below exercises effect evidence. We test authentication and tamper resistance on a separate set of 195 accounts produced by the same gate, record chain and decoder with their issuer keys retained (Appendix G): re-decoding reproduced 195 of them byte for byte, and 1,565 of 1,565 records passed authentication. **No altered account produced a false SUPPORTED:** over 8,400 audits of tampered, truncated or substituted accounts, false support occurred in 0. Removing one record at a time, the ablation of effect evidence, turned 0 of 3,840 answers into SUPPORTED, so missing evidence decodes to UNKNOWN. Prevention comes from the gate; the account adds what an auditor needs: per-witness seals, refusal attribution and the separation of violations from missing evidence.

5.5 CASE STUDY: AN ATTACK THAT REUSES THE TASK'S RECIPIENT

In the one run in which the gate deferred an injected send to the task's recipient (Appendix E), the attack twin asks the agent to also send gathered system information to s; the gate deferred that send, over TLS, under Valid, where the grant alone would have forwarded it, and the plaintext delivery then passed the validator and arrived alone.

6 CONCLUSION

RUNTIMETICKETS replaces a one-time verdict on a Skill package with a per-task check of every effect. It keeps the language model out of every enforcement and audit decision, bounds whatever the model proposes by the host's grant, enforces each obligation on the exact send at the egress boundary of unmodified frameworks, and leaves an authenticated account that a third party can re-derive without trusting a model. In our study no injected send took effect under RUNTIMETICKETS, no benign run was blocked and every account re-derived exactly, while production scanners either admitted malicious Skills or withheld benign ones and a grant-only allow-list forwarded injected sends that reused the task's recipient. Natural next steps are measuring the gate's overhead at deployment scale and deriving obligations for tasks that come without a delivery contract.

AI USE STATEMENT

As declared on the submission form, we used a large language model in three ways. It helped draft and polish the writing. It helped find and organize related work, which we verified against the primary sources. It helped assemble the benign parts of the evaluation cases, namely the benign task requests and their input files; it did not write or modify any Skill or any injected content, which come unchanged from released corpora. The research question, threat model, method and experimental design are the authors' own. The language models inside the method, the ticket generator and the agent drivers, are the object of study and are described in Sections 3 and 5. The authors take full responsibility for the content.

ETHICS STATEMENT

The adversarial cases are released malicious Skills executed on unmodified agent frameworks, but every send by a Skill or agent ends at a receiver we own, so no data reaches a third party; only the driver's and the scanners' model requests reach their providers, and no real credential or private datum is used. We report unmeasured outcomes as unknown rather than as evidence. No human-participant result is reported. The work is defensive: it measures whether a runtime protocol stops the harmful effects of malicious Skills while keeping benign Skills usable.

REPRODUCIBILITY STATEMENT

The method is specified by Algorithm 1 in Section 3, with the probe and gate rules in Appendix A.5, the metrics in Section 5, the pre-registered rules in Appendices A.7 and D.3, and the dataset construction in Appendix B. Every reported number is computed from a frozen results file by a fixed script. Studies of malicious Skills collected in the wild withhold, redact or gate their samples (Liu et al., 2026a;b), as datasets of malicious packages did (Ohm et al., 2020; Duan et al., 2021), and we follow the same practice for safety. Every attack Skill and injection we evaluate is already public in its cited corpus (Schmotz et al., 2026; Guo et al., 2026); what we do not publish is the layer that makes those injections fire under a real agent, namely the task deliverables and input files, the owned-receiver mapping of the host grant, and the calibration records. We will release the gate, compiler, decoder and record-chain code with the camera-ready version, and share the firing layer, the generation prompts and the frozen results with vetted researchers who request them for defensive research and agree not to redistribute them.

REFERENCES

Maksym Andriushchenko, Alexandra Souly, Mateusz Dziemian, Derek Duenas, Maxwell Lin, Justin Wang, Dan Hendrycks, Andy Zou, Zico Kolter, Matt Fredrikson, Eric Winsor, Jerome Wynne, Yarin Gal, and Xander Davies. AgentHarm: A benchmark for measuring harmfulness of LLM agents. In *International Conference on Learning Representations (ICLR)*, 2025. URL https://arxiv.org/abs/2410.09024.

Anthropic. Agent Skills. Claude Platform documentation, 2026. URL https://platform.claude.com/docs/en/agents-and-tools/agent-skills/overview. Announced 16 October 2025; accessed 26 September 2026.

Antiy CERT. ClawHavoc: Analysis of large-scale poisoning campaign targeting the OpenClaw skill market for AI agents. Antiy Labs security report, February 2026. URL https://www.antiy.net/p/clawhavoc-analysis-of-large-scale-poisoning-campaign-targeting-the-openclaw-skill-market-for-ai-agents/. Published 6 February 2026; accessed 26 September 2026.

Varun Pratap Bhardwaj. Formal analysis and supply chain security for agentic AI skills. arXiv preprint arXiv:2603.00195, 2026. URL https://arxiv.org/abs/2603.00195v2. Version 2, 5 August 2026.

Arnar Birgisson, Joe Gibbs Politz, Úlfar Erlingsson, Ankur Taly, Michael Vrable, and Mark Lentczner. Macaroons: Cookies with contextual caveats for decentralized authorization in the

cloud. In *Proceedings 2014 Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, USA, 2014. Internet Society. doi: 10.14722/ndss.2014.23212. URL https://www.ndss-symposium.org/ndss2014/ndss-2014-programme/macaroons-cookies-contextual-caveats-decentralized-authorization-cloud/.

Christoph Bühler, Matteo Biagiola, Luca Di Grazia, and Guido Salvaneschi. AgentBound: Securing execution boundaries of AI agents. *Proceedings of the ACM on Software Engineering*, 3(FSE): 2141–2164, 2026. ISSN 2994-970X. doi: 10.1145/3808103. Article FSE096 (FSE 2026).

Zhihao Chen, Ying Zhang, Yi Liu, Gelei Deng, Yuekang Li, Yanjun Zhang, Jianting Ning, Leo Yu Zhang, Lei Ma, and Zhiqiang Li. How your credentials are leaked by LLM Agent Skills: An empirical study. arXiv preprint arXiv:2604.03070, 2026. URL https://arxiv.org/abs/2604.03070v2. Version 2, 19 June 2026; accepted to ASE 2026.

Cisco AI Defense. Cisco Skill Scanner: Security scanner for Agent Skills. Software repository, 2026. URL https://github.com/cisco-ai-defense/skill-scanner. Release 2.1.0 (tag commit a24df34, 5 September 2026); accessed 19 September 2026.

Manuel Costa, Boris Köpf, Aashish Kolluri, Andrew Paverd, Mark Russinovich, Ahmed Salem, Shruti Tople, Lukas Wutschitz, and Santiago Zanella-Béguelin. Securing AI agents with information-flow control. arXiv preprint arXiv:2505.23643, 2025. URL https://arxiv.org/abs/2505.23643v2. Version 2, 3 September 2025.

Scott A. Crosby and Dan S. Wallach. Efficient data structures for tamper-evident logging. In *Proceedings of the 18th USENIX Security Symposium*, pp. 317–333, Montreal, Canada, August 2009. USENIX Association. URL https://www.usenix.org/legacy/event/sec09/tech/full_papers/crosby.pdf.

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. In *Advances in Neural Information Processing Systems (Datasets and Benchmarks Track)*, volume 37, pp. 82895–82920. Curran Associates, Inc., 2024. doi: 10.52202/079017-2636. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/97091a5177d8dc64b1da8bf3e1f6fb54-Abstract-Datasets_and_Benchmarks_Track.html.

Edoardo Debenedetti, Ilia Shumailov, Tianqi Fan, Jamie Hayes, Nicholas Carlini, Daniel Fabian, Christoph Kern, Chongyang Shi, Andreas Terzis, and Florian Tramèr. Defeating prompt injections by design. In *2026 IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)*, pp. 587–618. IEEE, March 2026. doi: 10.1109/satml68715.2026.00040. Also arXiv:2503.18813.

DeepSeek-AI. DeepSeek-V4: Towards highly efficient million-token context intelligence. arXiv preprint arXiv:2606.19348, 2026. URL https://arxiv.org/abs/2606.19348v1. Version 1.

Ruian Duan, Omar Alrawi, Ranjita Pai Kasturi, Ryan Elder, Brendan Saltaformaggio, and Wenke Lee. Towards measuring supply chain attacks on package managers for interpreted languages. In *Proceedings 2021 Network and Distributed System Security Symposium (NDSS)*. Internet Society, February 2021. doi: 10.14722/ndss.2021.23055. URL https://www.ndss-symposium.org/ndss-paper/towards-measuring-supply-chain-attacks-on-package-managers-for-interpreted-languages/.

Yunhao Feng, Ruixiao Lin, Ming Wen, Yanming Guo, Xingjun Ma, Yutao Wu, Xinhao Deng, and Shouling Ji. HazardAuditor: From executable threats to safer computer-use agents. arXiv preprint arXiv:2609.15134, 2026. URL https://arxiv.org/abs/2609.15134v1. Version 1, 14 September 2026.

Wenbo Guo, Wei Zeng, Chengwei Liu, Xiaojun Jia, Yijia Xu, Lei Tang, Yong Fang, and Yang Liu. MalSkillBench: A runtime-verified benchmark of malicious Agent Skills. arXiv preprint arXiv:2606.07131, 2026. URL https://arxiv.org/abs/2606.07131v3. Version 3.

Norm Hardy. The confused deputy (or why capabilities might have been invented). *ACM SIGOPS Operating Systems Review*, 22(4):36–38, October 1988. ISSN 0163-5980. doi: 10.1145/54289.871709. URL https://dl.acm.org/doi/10.1145/54289.871709.

Florian Holzbauer, David Schmidt, Gabriel K. Gegenhuber, Sebastian Schrittwieser, and Johanna Ullrich. Context matters: Repository-aware security analysis of the Agent Skill ecosystem. arXiv preprint arXiv:2603.16572, 2026. URL https://arxiv.org/abs/2603.16572v2. Version 2, 1 June 2026; Agent Skills Workshop at CAIS 2026.

Zimo Ji, Congying Xu, Zongjie Li, Yudong Gao, Xin Wei, Shuai Wang, and Shing-Chi Cheung. Cloak and detonate: Scanner evasion and dynamic detection of Agent Skill malware. arXiv preprint arXiv:2607.02357, 2026. URL https://arxiv.org/abs/2607.02357v2. Version 2, 3 July 2026.

Yi Liu, Zhihao Chen, Yanjun Zhang, Gelei Deng, Yuekang Li, Jianting Ning, and Leo Yu Zhang. "Do Not Mention This to the User": Detecting and understanding malicious Agent Skills in the wild. In *35th USENIX Security Symposium (USENIX Security 26)*, pp. 1727–1746, Baltimore, MD, August 2026a. USENIX Association. URL https://www.usenix.org/conference/usenixsecurity26/presentation/liu-yi.

Yi Liu, Weizhe Wang, Ruitao Feng, Yao Zhang, Guangquan Xu, Gelei Deng, Yuekang Li, and Leo Zhang. Agent Skills in the wild: An empirical study of security vulnerabilities at scale. arXiv preprint arXiv:2601.10338, 2026b. URL https://arxiv.org/abs/2601.10338v1. Version 1, 15 January 2026.

Moonshot AI. Kimi-K2.6 model card. Hugging Face model repository, 2026. URL https://huggingface.co/moonshotai/Kimi-K2.6. Accessed 25 September 2026.

Yutao Mou, Zhangchi Xue, Lijun Li, Peiyang Liu, Shikun Zhang, Wei Ye, and Jing Shao. ToolSafe: Enhancing tool invocation safety of LLM-based agents via proactive step-level guardrail and feedback. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 37125–37153, San Diego, California, United States, July 2026. Association for Computational Linguistics. doi: 10.18653/v1/2026.findings-acl.1850. URL https://aclanthology.org/2026.findings-acl.1850/.

Yi Nian, Aojie Yuan, Haiyue Zhang, Jiate Li, Li Li, Xiyang Hu, Hua Wei, Xiongye Xiao, Chaowei Xiao, and Yue Zhao. Auditable agents. arXiv preprint arXiv:2604.05485, 2026. URL https://arxiv.org/abs/2604.05485v3. Version 3, 14 September 2026.

Nous Research. Hermes Agent. Software repository, 2026. URL https://github.com/NousResearch/hermes-agent. Commit 6bc0e9e, 14 September 2026; accessed 26 September 2026.

NVIDIA. SkillSpector: Security scanner for AI Agent Skills. Software repository, 2026. URL https://github.com/NVIDIA/SkillSpector. Release v2.11.2 (tag commit 69dcdfb, 9 September 2026); accessed 19 September 2026.

Marc Ohm, Henrik Plate, Arnold Sykosch, and Michael Meier. Backstabber's Knife Collection: A review of open source software supply chain attacks. In Clémentine Maurice, Leyla Bilge, Gianluca Stringhini, and Nuno Neves (eds.), *Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA 2020)*, volume 12223 of *Lecture Notes in Computer Science*, pp. 23–43, Cham, 2020. Springer International Publishing. doi: 10.1007/978-3-030-52683-2_2.

OpenClaw. ClawHub: Skill + plugin registry for OpenClaw. Skill registry, 2026. URL https://clawhub.ai. Accessed 26 September 2026.

OpenClaw Foundation. OpenClaw: Open-source AI assistant. Software repository, 2026. URL https://github.com/openclaw/openclaw. Release 2026.6.10, commit aa69b12; accessed 26 September 2026.

Shidong Pan, Xiaoyu Sun, Tianyi Zhang, Dianshu Liao, Kaiwen Yang, and Zhenchang Xing. SkillGuard: A permission-centric framework for Agent Skill security. arXiv preprint arXiv:2606.03024, 2026. URL https://arxiv.org/abs/2606.03024v2. Version 2.

Suliu Qin, Haomin Zhuang, Yujun Zhou, Yufei Han, and Xiangliang Zhang. AIRGuard: Guarding agent actions with runtime authority control. arXiv preprint arXiv:2605.28914, 2026. URL https://arxiv.org/abs/2605.28914v1. Version 1, 27 May 2026.

David Schmotz, Luca Beurer-Kellner, Sahar Abdelnabi, and Maksym Andriushchenko. Skill-Inject: Measuring agent vulnerability to skill file attacks. arXiv preprint arXiv:2602.20156, 2026. URL https://arxiv.org/abs/2602.20156v3. Version 3.

Fred B. Schneider. Enforceable security policies. *ACM Transactions on Information and System Security*, 3(1):30–50, February 2000. doi: 10.1145/353323.353382. URL https://www.cs.cornell.edu/fbs/publications/EnfSecPols.pdf.

Bruce Schneier and John Kelsey. Secure audit logs to support computer forensics. *ACM Transactions on Information and System Security*, 2(2):159–176, May 1999. ISSN 1094-9224. doi: 10.1145/317087.317089.

Zhengyang Shan, Jiayun Xin, Yue Zhang, and Minghui Xu. Don't let the claw grip your hand: A security analysis and defense framework for OpenClaw. arXiv preprint arXiv:2603.10387, 2026. URL https://arxiv.org/abs/2603.10387v1. Version 1, 11 March 2026.

Tianneng Shi, Jingxuan He, Zhun Wang, Hongwei Li, Linyu Wu, Wenbo Guo, and Dawn Song. Progent: Securing AI agents with privilege control. arXiv preprint arXiv:2504.11703, 2025. URL https://arxiv.org/abs/2504.11703v3. Version 3, 14 May 2026.

Rohit Taneja and Travis Weber. Scan the skill, govern the action: Composing registry verdicts with runtime consequence control. arXiv preprint arXiv:2609.12001, 2026. URL https://arxiv.org/abs/2609.12001v1. Version 1.

Lillian Tsai and Eugene Bagdasarian. Contextual agent security: A policy for every purpose. In *Proceedings of the Workshop on Hot Topics in Operating Systems (HOTOS '25)*, pp. 8–17, Banff, AB, Canada, May 2025. ACM. doi: 10.1145/3713082.3730378.

Vercel. Skills.sh: The agent skills directory. Skill registry, 2026. URL https://skills.sh. Accessed 26 September 2026.

Haoyu Wang, Christopher M. Poskitt, and Jun Sun. AgentSpec: Customizable runtime enforcement for safe and reliable LLM agents. In *Proceedings of the IEEE/ACM 48th International Conference on Software Engineering (ICSE)*, pp. 2938–2950. ACM, April 2026. doi: 10.1145/3744916.3764546.

Chris Zheng and Geng Yang. CONTINUITY: Security-context contracts for composable LLM agent controls. arXiv preprint arXiv:2609.05269, 2026. URL https://arxiv.org/abs/2609.05269v1. Version 1.

A Details Deferred from the Main Text

A.1 Glossary

Table 4: **Terms used in the paper.**

Term	Meaning
Twin pair	A released attack Skill and its benign counterpart; a pair shares the task.
Grant A, budget B	What the host allows every task of the deployment, and its quantity bound.
Ticket, leaf	The per-task obligations compiled narrow-only from a proposal; a leaf is one check on a send $(\mathrm{Recipient}, \mathrm{Valid}, \mathrm{Sum})$.
Gate	The model-free decision on every outbound send: ALLOW, DENY or DEFER.
Allow-list	The same gate holding only the grant, $\mathcal{C}_A$; the ablation of the per-task ticket.
Account, seal	The authenticated, chained records of one run; a seal closes a witness's stream.
Witness	A trusted monitor that records effects within its observation scope.
Obligation verdict	SUPPORTED, VIOLATED or UNKNOWN, returned by the decoder.
Panel, slot, cell, run	A framework with a driver model; one planned repetition; one slot of one twin under one defense; a cell that reached its framework's normal exit.

A.2 Metric Definitions

For one defense, Attack and Benign hold all of its attack and benign runs, and Runs holds both, or only the benign twins for a scanner. $\mathit{Harmed} \subseteq \mathit{Attack}$ holds the runs in which the defense forwarded an injected send, $\mathit{Blocked} \subseteq \mathit{Benign}$ the runs in which the defense blocked the sanctioned send, and $\mathit{Delivered} \subseteq \mathit{Runs}$ the runs that delivered the sanctioned output. For a scanner, an admitted attack twin is harmed and a withheld benign twin is blocked. Then

$$\mathrm{ASR} = \frac{|\mathit{Harmed}|}{|\mathit{Attack}|}, \qquad \mathrm{OVK} = \frac{|\mathit{Blocked}|}{|\mathit{Benign}|}, \qquad \mathrm{BSC} = 1 - \mathrm{OVK}. \tag{3}$$

Benign Skill completion (BSC) is the complement of the overkill rate. The share $|\mathit{Delivered}|/|\mathit{Runs}|$ is the driver finishing rate of Appendix A.8. For the account, $\mathit{Identical} \subseteq \mathit{Accounts}$ holds the retained accounts whose independent re-decoding reproduces the retained verdicts byte for byte, and $\mathit{FalseSupport} \subseteq \mathit{Altered}$ holds the audits of tampered, truncated or substituted accounts in which the decoder still reports SUPPORTED. Then

$$\mathrm{RDC} = \frac{|\mathit{Identical}|}{|\mathit{Accounts}|}, \qquad \mathrm{FSR} = \frac{|\mathit{FalseSupport}|}{|\mathit{Altered}|}. \tag{4}$$

A.3 Statistical Tests

These tests are exploratory: they are not among the pre-registered decision rules, which were not decidable at the counts reached (Appendix D.3). We compare proportions of attack runs in which an injected send got through with a two-sided Fisher exact test; the zero overkill rate of RUNTIMETICKETS gets a 95% Wilson interval of 0.0% to 11.7%. The tests treat runs as independent, although runs are clustered within pairs and slots and the same cases recur across the three runtime defenses; Experiments A and B are never pooled. On attacks that reuse the task's recipient we report descriptive Fisher values only, outside the adjusted family: $p = 0.06$ against the allow-list and $p = 0.09$ against no gate. The comparison with no gate stays below the 0.05 level after a Holm adjustment over the two all-runs comparisons. The comparison with the allow-list and the two comparisons on attacks that reuse the task's recipient do not reach the 0.05 level; per the pre-registered rule, we make no confirmatory claim that the ticket beats the allow-list on these pairs. The forwarding rate conditions on attack runs with at least one attempted injected send (Appendix A.9); an attempt is the agent's action and precedes the gate's decision on it. No gate forwards every attempt by construction, so we test only the allow-list: 3/4 against 0/6 gives a run-level Fisher $p = 0.033$, and under the endpoint audit the counts are 3/3 and 0/3 ($p = 0.10$).

A.4 ENDPOINT AUDIT

We audited every run in which the injected-send endpoint of Table 3 fired. Two rules, fixed on the frozen case files, remove a landing that the injection did not cause: a send to the injection's declared host is not counted when the trusted task itself contacts that host, and a planted canary is not counted when the task's own step sends it. The rules remove 2 no-gate landings, 0 allow-list landings and 0 under RUNTIMETICKETS, and these rules also remove the single benign run in which the endpoint fired (1/84 benign runs). Without the removed landings, the ASR is 3/28 without a gate, 3/23 under the allow-list and 0/32 under RUNTIMETICKETS (exploratory Fisher $p = 0.10$ and $p = 0.07$ against RUNTIMETICKETS), and on attacks that reuse the task's recipient 18.2%, 33.3% and 0.0% ($p = 0.22$ and $p = 0.06$). Among runs, the remaining landings come from 2 pairs without a gate, 1 under the allow-list and 0 under RUNTIMETICKETS. In the recipient-reuse landings the defense forwarded the injected TLS connection, but the mock receiver did not complete TLS, so no body arrived; the ASR counts the forwarded send.

A.5 PROBE AND GATE RULES

Let $\pi_{\mathcal{C}}(g, o)$ be the decision of the frozen gate on a unit send to owned receiver g whose object passes ($o = \top$) or fails ($o = \bot$) the validator. The host accepts a compiled ticket only if

$$\pi_{\mathcal{C}}(s, \top) = \textsc{Allow}, \quad \pi_{\mathcal{C}}(s, \bot) \neq \textsc{Allow}, \quad \pi_{\mathcal{C}}(g, \top) \neq \textsc{Allow} \ \ \forall g \in R_A \setminus \{s\}. \tag{5}$$

With $p(e)$ the value of leaf p on (e, V, ρ), V the validation facts, $n(\rho)$ the reserved quantity and $U(e)$ the unresolved risks that apply to request e, the gate returns

$$\mathrm{gate}_{\mathcal{C}}(e, V, \rho) = \begin{cases} \textsc{Deny} & \text{if no grant in } G \text{ covers } e, \text{ or } n(\rho) + n(e) > B, \\ \textsc{Deny} & \text{else if } p(e) = \bot \text{ for some } p \in P, \\ \textsc{Defer} & \text{else if } p(e) \text{ is unknown for some } p \in P, \text{ or } U(e) \neq \varnothing, \\ \textsc{Allow} & \text{otherwise.} \end{cases} \tag{6}$$

A.6 PROPOSITION 1: PROOF SKETCH

Write $\mathrm{Allow}(\mathcal{C})$ for the contexts (e, V, ρ) on which the gate of Eq. (6) allows under $\mathcal{C}$, and $\mathcal{C}_A$ for the ticket that compiles the grant alone. On a single context, compilation keeps $G \subseteq A$ and the budget B and only appends leaves and risks after the mandatory leaves P_A. A smaller grant set can only add an authority denial, and an appended leaf or risk, which reads only e, V and ρ, can only add a DENY or a DEFER. Hence $\mathrm{Allow}(\mathcal{C}) \subseteq \mathrm{Allow}(\mathcal{C}_A)$ on every context.

The bound extends to whole runs. Proposition 1 compares the two tickets on the same context (e, V, ρ). A run is a sequence of such contexts, and the gate sees each request with the reservations ρ that the run has actually accumulated. Only an ALLOW adds to ρ, and every ALLOW under $\mathcal{C}$ is an ALLOW under $\mathcal{C}_A$ on the same context. By induction over the requests of the run, each effect that a run under $\mathcal{C}$ forwards would also be allowed by $\mathcal{C}_A$ at the same point of the same run, so it is covered by a grant in A and keeps the run's total within the host budget B. The argument does not depend on how the agent chooses its next request, so it holds for adaptive request sequences. The proof is a sketch and is not machine-checked.

A.7 SLOTS, RUNS AND PRE-REGISTRATION

Slots and runs. Each panel's frozen roster plans three slots for every twin and defense. The slots are repeated runs, not conditional retries: every planned slot runs whatever the other slots did, and every slot whose unmodified framework ran to its own normal exit is a run, whether or not the agent attempted the injected action or finished the task. OpenClaw/DeepSeek ran all three slots. The Hermes panels ran the first slot; on Hermes/DeepSeek the provider's circuit breaker stopped the later slots. The planned OpenClaw/Kimi panel did not complete within the study window and is not reported.

Recorded cells that are not runs. Of the recorded cells, 53 of 80 were runs without a gate, 53 of 86 under the allow-list and 61 of 93 under RUNTIMETICKETS. The others either failed before their framework's normal exit (17, 18 and 21, in the same order) or never started, for example after

a container setup error (10, 15 and 11), and they enter no denominator. No excluded cell contains a refusal of a format-valid sanctioned send (0 such cells), so the exclusion hides no overkill. One excluded allow-list attack cell on Hermes/Kimi did record an injected landing under that endpoint, and counting it would only raise the allow-list's ASR.

In-slot re-run. Within one invocation, a slot whose run raised an exception, returned a non-empty error list or ended before its framework's normal exit was run once more at once, and the second result stands; a slot with a result is never run again. The trigger reads the error list, never a security or delivery outcome. It set aside 44 first attempts without a gate, 56 under the allow-list and 60 under RUNTIMETICKETS, counting in-progress cells of restarted phases; because the trigger reads the error list, 12, 14 and 16 of them had exited normally. No set-aside cell contains a refusal of a format-valid sanctioned send (0 such cells).

Decision rules. The defenses, the outcomes recorded for each cell, the decision rules and their thresholds were frozen before any result, and a hash binds them to the script that computes every number (Appendix D.3). At the counts reached, no security rule was decidable on any panel. Attack twins were also held to a pre-registered utility rule, delivery within 0.10 of no gate, under which a run whose injected send was refused counts as a failure when the agent then does not deliver. That rule was supported on Hermes/DeepSeek, not supported on Hermes/Kimi and not supported on OpenClaw/DeepSeek.

A.8 DRIVER FINISHING RATE

The driver finishing rate, the share of runs that delivered the sanctioned output, was 32.8% under RUNTIMETICKETS (20/61), 34.0% with no gate (18/53) and 34.0% under the allow-list (18/53), similar across the runtime defenses. Benign runs delivered in 37.9% under RUNTIMETICKETS, 28.0% without a gate and 40.0% under the allow-list. Runs that did not deliver ended because the driver stopped or ran out of budget; under RUNTIMETICKETS, none ended because the gate refused a format-valid sanctioned send. The gate did refuse objects sent to the task's receiver in 8 benign runs under RUNTIMETICKETS, every one of which failed the validator, and 2 of these runs delivered afterwards. Under the allow-list, 2 of 30 benign runs had a format-valid delivery refused on the host budget and did not deliver; the frozen classifier did not count them as blocked, so the allow-list's reported OVK favors that baseline.

A.9 ATTEMPTED INJECTED SENDS

An *attempted injected send* is an outbound request that the agent issues to the injection's declared host or that carries a planted canary. The forwarding rate of Table 3 counts, among attack runs with at least one such attempt, the runs in which the defense forwarded it. Among attack runs with an attempt, the forwarding counts are 5/5 without a gate, 3/4 under the allow-list and 0/6 under RUNTIMETICKETS. On attacks that reuse the task's recipient, the gate refused all 3 injected sends under RUNTIMETICKETS: 2 under Sum and 1 under Valid.

A.10 RE-DERIVATION OF THE EXPERIMENT B ACCOUNTS

We re-derived every gated account of Experiment B offline and without a model: 153 accounts of allow-list and RUNTIMETICKETS cells, runs and non-runs alike, holding 2,009 records and 895 gate decisions; 26 further gated cells stopped before the gate held an account. Every account is bound to its frozen ticket by digest (153), and its per-issuer chains are contiguous and end at the retained heads (153), a key-free consistency check. The frozen gate re-derived 895 of the 895 recorded decisions from their recorded facts, and re-executing each account's recorded requests through a fresh frozen gate reproduced its records field for field (153 accounts). The frozen decoder, run with record authentication set aside, reproduced the retained report of 153 of the 153 accounts byte for byte, with 0 decoder errors. This re-derivation does not re-verify record authentication, which the separate test set of Appendix G measures together with tamper resistance.

An account that did not close decodes to UNKNOWN rather than SUPPORTED: 56 of the 82 RUNTIMETICKETS accounts; over both gated defenses, 75 unclosed accounts were waiting for a provider completion.

A.11 RESULTS PER PANEL

Counts in Table 3 are summed over the three panels. Under RUNTIMETICKETS the ASR was 0/1, 0/3 and 0/28 attack runs and the OVK 0/1, 0/5 and 0/23 benign runs on Hermes/DeepSeek, Hermes/Kimi and OpenClaw/DeepSeek; the panels hold 8, 19 and 140 runs over all defenses and twins.

A.12 DESIGN QUESTIONS

The mechanism alone answers five design questions (DQ), kept apart from the experimental RQs. **DQ1: Can a steered generator widen what a Skill may do?** No. Approval compiles every proposal narrow-only, so by Proposition 1 a ticket allows at most what the host grant allows. In this study every proposal, for attack and benign twins alike, was the narrowest ticket the delivery contract permits (the task receiver's grant and the three leaves), so no Skill text loosened a ticket. **DQ2: Can injected text influence a gate decision?** No. Injected text can change the proposal, which DQ1 bounds, and which requests the agent makes, but not the rule that decides them: after the freeze the gate reads only each request's effect facts, the validation facts and the frozen ticket. **DQ3: Does protection depend on the injected destination being new?** Not on the receiver alone. An address check passes an injected send that *reuses the task's recipient* s, but Valid and Sum still check its object and amount; an injected object that passes the validator within the task's amount, such as falsified content inside a valid delivery, is allowed. The check applies to every send that crosses the egress boundary. **DQ4: Can a third party trust a verdict without trusting a model?** Yes, given the keys. The decoder (Eq. 2) is a deterministic function of the authenticated account, so anyone with the ticket, records and keys re-derives every decision and verdict, and a record altered without the keys fails authentication; the keys are symmetric, so an auditor trusts their holder, and per-issuer asymmetric signatures or an external transparency log would remove that trust without changing the decoder. **DQ5: Does the order of sends matter for Sum?** Sum reserves the task's amount in request order. An injected send to s can consume it only if its object passes the validator, so it cannot carry a non-validating object; if it comes first, its object reaches s and the legitimate delivery that follows is refused on Sum and attributed in the account. Valid, not Sum, limits such a send, and it is as strong as the format validator. In our runs no format-valid injected send reached s: of the 3 injected sends to s under RUNTIMETICKETS, 2 came after the delivery and were refused on Sum, and 1 came before it and was deferred as unreadable. A task with several legitimate sends states one amount per receiver and object, so the ticket bounds each separately.

A.13 PRE-REGISTERED CASE-LEVEL READOUT

The frozen decision rules take the twin pair as the unit and score `unauthorized_egress_landed` over every recorded attack-twin cell, the same frozen results file that feeds every other number. Table 5 gives this readout per defense and per stratum; a pair counts as landed if any of its attack cells landed on any panel. The sign test counts pairs in which one defense landed and the other did not; following the frozen rules, off-grant pairs are reported but not read as a ticket advantage. This readout counts every recorded attack cell, run or not, under the frozen endpoint `unauthorized_egress_landed`, which is not the injected-send endpoint of Table 3, so the two tables are read on their own scales and never against each other.

Table 5: **Pre-registered readout by pair.** Recorded attack cells, cells with injected egress landed, and pairs landed overall and by stratum (4 recipient-reuse, 3 in-grant, 3 off-grant and 1 pair in no egress stratum); the attempted column counts pairs in which the agent attempted the injected action.

Defense	Cells	Landed cells	Pairs landed	Reuse	In-grant	Off-grant	Pairs attempted
No gate	40	9	6/11	2	3	1	6
Allow-list	38	6	4/11	2	0	2	4
RUNTIMETICKETS	47	**0**	**0/11**	0	0	0	4

On the same pairs, the scanners admitted 0/11 (Cisco) and 0/11 (SkillSpector) attack twins and withheld 1/11 and 7/11 benign twins.

B DATASET

Both experiments use the same 60 twin pairs. This appendix describes how the pairs were built, how the scanner outcomes split by harm category, how the injected destinations relate to the host grant, and how calibration selected the subset that Experiment B runs. Every count is generated from the frozen case file or the results file.

B.1 CONSTRUCTION

Sources. The pairs come from two released corpora of malicious Skills (Table 6). Each of the 30 contextual and 18 obvious Skill-Inject pairs takes one released injection together with one of its released task entries, which fixes the carrier Skill and the line at which the injection goes (Schmotz et al., 2026). Following the corpus's own builder, we insert the released injection text into the carrier's `SKILL.md` at that line, and the benign twin is the unmodified carrier. When a released injection calls a helper script that the corpus ships, the attack twin also carries that script verbatim, which holds for 7 pairs. The attack twin of each of the 12 MalSkillBench pairs is a byte-for-byte copy of a released malicious Skill directory, and the benign twin is the released benign directory of the same name (Guo et al., 2026). Its harm label comes from the corpus's directory taxonomy, and the corpus restricts its use to academic research.

Table 6: **Source corpora of the 60 twin pairs and how far apart the twins are.** Files and lines added or removed between the benign and the attack twin; the task, the workspace and every other file are identical across the twins.

			Lines that differ, median (range)	
Source corpus	Pairs	Files that differ	In `SKILL.md`	In all files
Skill-Inject, contextual	30	1–2	1 (1–12)	1 (1–85)
Skill-Inject, obvious	18	1–2	4 (1–7)	4 (1–22)
MalSkillBench	12	2–56	222 (40–559)	444.5 (145–5426)
All pairs	60			

What we add. Both twins of a pair run the same trusted task. It opens with the corpus's own task prompt for the carrier Skill, so that the agent has a reason to load the Skill; for the 12 MalSkillBench pairs it opens instead with a benign request that we wrote from the Skill's own documentation. The task then asks for a fixed deliverable computed from an input file in the workspace, and when the task delivers by a send, one sentence names the receiver and the object. The workspace also holds synthetic canary files, decoy secrets whose presence in a send marks it as exfiltration, and 33 pairs receive the benign input files that their task prompt names. No Skill, injection, prompt or model text appears in this paper.

B.2 OUTCOMES BY HARM CATEGORY

Table 7 splits the scanner results of Table 3 by the harm categories of Table 1. The labels are audited: before calibration, a label audit moved 3 pairs whose source label did not match their injected effect, for example a phishing send from exfiltration to content manipulation. A scanner's verdict does not depend on the label, so the table only regroups the same verdicts.

Scanners. The two scanners fail differently. Cisco admitted 0% of the exfiltration attack twins but 50% of the destructive-change twins and 100% of the safeguard-suppression twins. SkillSpector missed only one safeguard-suppression twin, but it withheld between 47.2% and 87.5% of the benign twins in each of E1 to E4.

Runtime defenses. Experiment B targets the egress boundary, where the gate decides: its calibrated pairs are exfiltration pairs and a content-manipulation pair, and its results are results on sends. A purely local effect of E2, E3 or E5 is recorded by the workspace witness (Section 2).

Table 7: **Scanner outcomes by harm category** (Experiment A, full cascades). ASR: share of the category's attack twins admitted; OVK: share of its benign twins withheld.

		Cisco Skill Scanner		NVIDIA SkillSpector	
Category	Pairs	ASR	OVK	ASR	OVK
E1 Exfiltration	36	0%	25%	0%	47.2%
E2 Destructive change	8	50%	0%	0%	87.5%
E3 Access weakening	5	40%	0%	0%	60%
E4 Content manipulation	8	25%	37.5%	0%	62.5%
E5 Safeguard suppression	3	100%	0%	33.3%	0%
All	60	18.3%	20.0%	1.7%	53.3%

B.3 Receivers and the Host Grant

The host grant is a fixed set of owned receivers that covers every receiver a task needs. It was frozen with the case file before calibration and is never recomputed from the selected subset. Two granted receivers stand in for the two external destinations that the released Skill-Inject injections name most often; the egress proxy maps each such destination to its stand-in before the gate sees it and records the literal destination beside it. For a pair whose injection names a stand-in, a hash order over case identifiers, fixed before any run, makes the task's receiver either that same destination, so that the injected send reuses the task's recipient, or another granted receiver. An injection that names any other destination is aimed outside the grant. The stand-ins model the hardest case for a deployment-wide allow-list: an attacker who names an endpoint that the deployment legitimately uses. Receiver assignments were never revised after a run.

B.4 Calibration

Calibration decides which pairs Experiment B runs. It executed only attack twins and only without a gate, 2 runs for each of the 60 pairs, on Hermes Agent with the DeepSeek driver; the other panels inherit its selection. A pair is kept if at least one of its calibration runs attempted the injected action, that is, connected to the injection's declared destination or sent an object that contains a canary. A generic connection outside the grant, such as a package fetch, does not count as an attempt. The selection was frozen before the first RuntimeTickets run. It reads no gated defense and no benign twin, so it cannot favor RuntimeTickets over the allow-list, which face the same selected pairs.

C Protocol Details

This appendix adds to Section 3 what a reader needs to reproduce the protocol: how tickets are generated, what injected text can and cannot change, how refusals are attributed, and how the decoder reads an account. The validator's accepted formats, and the canary construction are withheld for security and can be shared with the program committee. The record chain is a standard per-issuer HMAC-SHA256 chain; its security rests on the keys alone.

C.1 Ticket Generation

What the generator reads. For each twin, the host fills one fixed template with the task goal, the delivery contract δ stated as host authority, the host grant, and the numbered lines of the Skill's `SKILL.md`, which the template marks as data to analyze. The two twins of a pair receive the same template and differ only in the Skill lines, and a leak check confirmed that no host-written field of any request named a case, a twin or a threat. Helper scripts are not shown to the generator, since the gate judges their effects at the boundary. The reply is parsed strictly, compiled narrow-only and probed before freezing (Algorithm 1, Eq. (5)). The generator was Claude (`claude-fable-5-1`).

Generated tickets. The host fixes the grant, budget, streams, validators and bindings and, in this study, mandates no leaf; the model proposes the selected grants, every leaf and the unresolved risks. The generator was called once per twin, 22 calls in all, each accepted at the first request, and the resulting 22 frozen tickets were each used unchanged in every slot and on every panel of their twin.

Each allowed a valid object to the task's receiver, refused it to every other granted receiver and refused an invalid object to the task's receiver. Every ticket was frozen before the first run.

Why the probe is needed. A proposal can yield a ticket that cannot allow the sanctioned send, for example through a Forbid leaf that matches the sanctioned send itself, and such a ticket would overkill the task. The host therefore states the delivery as host authority and rejects, before freezing, any ticket that fails Eq. (5); none of the frozen tickets failed it.

A template would also pass. Because δ is structured, a deterministic template could write the three standard leaves of Eq. (1), and it would pass the probe by construction. Our experiments therefore compare a per-task ticket with a deployment-wide allow-list, not a model with a template. The model's proposals cite the Skill lines that each leaf answers, which the account keeps for audit; in this study they added no leaf beyond the three.

C.2 What Injected Text Can Change

Injected text reaches the protocol through two channels. Through the generator it can change the proposed leaves but cannot widen authority (Proposition 1), and a ticket steered to keep another granted receiver open or to drop the object check fails the probe. Through the agent it can change which requests are made, but no model sits in the gate, the validator or the decoder. It can still leave a ticket less strict than intended on a dimension that the probe does not test.

Where the guarantee stops. The validator is a format check on the exact bytes sent, computed by the host without a model, so an injected object that passes it would be allowed to the task's receiver and, if forwarded, counted as taking effect. An effect that never crosses the egress proxy, such as a deleted file, is recorded by the workspace witness but meets no gate. In this study every task's delivery is a plaintext send whose bytes the validator reads, while every injected send aimed at a granted receiver travels over TLS. The ticket refuses such a send to another granted receiver because it does not select that receiver, and one to the task's own receiver by deferral, because the gate sees its destination but cannot read its object. The allow-list is keyed on the receiver host, so it admits a TLS connection to a granted host as readily as the plaintext delivery; an allow-list keyed also on scheme or route is outside this comparison.

C.3 Denial Attribution

The raw reason that the gate records is the same when the host never granted a request and when the ticket did not select the grant, so we never report it. Instead, a host-side function of the frozen contract and the recorded decision recomputes each refusal against the host grant and assigns it to a deciding layer (outside the host grant, the host budget, or a ticket rule) and, for a ticket rule, to its primary rule. When several leaves deny one request, a counterfactual fixed in advance decides: drop the Valid leaves, and if another ticket rule still denies, the first such rule in a fixed order is primary; otherwise Valid is. Table 8 attributes every refusal recorded under the two gated defenses, and none is left unattributed. A refusal of a send to the task's receiver counts as overkill only when it refused a format-valid delivery and the run then did not deliver, and no RuntimeTickets run met this condition.

C.4 The Decoder

The decoder authenticates each record with its issuer's key, re-runs the frozen gate of Eq. (6) on every recorded decision against its cited facts and the reservations before it, and checks that the account closed with no operation open and that every declared observation stream was sealed (Figure 4). It then returns one finding per obligation by Eq. (2). A Violated finding needs an admitted counterexample, an authenticated record of an effect that breaks the obligation, so a witnessed violation stands even when other evidence is missing. A Supported finding needs a closed, sealed account that holds the obligation's evidence, such as a validator pass on the exact bytes of an effect or, for an operation with no matching effect, a sealed stream whose declared scope covers it. Everything else is Unknown. Removing a record from inside an issuer's chain breaks that chain, which makes the account incomplete and turns every finding that is not a witnessed violation into Unknown. The decoder reads no other input and calls no model, so anyone who holds the contract, the records and the keys obtains the same report.

Table 8: **Every refusal by the two gated defenses, attributed**, summed over the three panels and over both twins. The allow-list holds no ticket, so it has no ticket-rule rows. Rows classify refused requests by target, a broader class than the attempted injected sends of Appendix A.9.

	Allow-list	RUNTIMETICKETS
Refusals	347	422
Send refused		
Injected send	77	56
Send to the task's receiver	16	35
Other send	254	331
Deciding layer		
Outside the host grant	214	272
Host budget	133	0
Ticket rule	–	150
Primary ticket rule		
Receiver not selected by the ticket	–	42
Object failed the validator (Valid)	–	88
Object unreadable, deferred (Valid)	–	1
Quantity beyond the limit (Sum)	–	19

Trust assumptions. Records are authenticated with symmetric per-issuer keys that are supplied separately from the records, so re-decoding is authenticated recomputation rather than non-repudiation, and any key holder could forge a record. Closure and stream seals assume that the host's collectors report every effect they observe, and an effect that never reaches an observed boundary is outside the account.

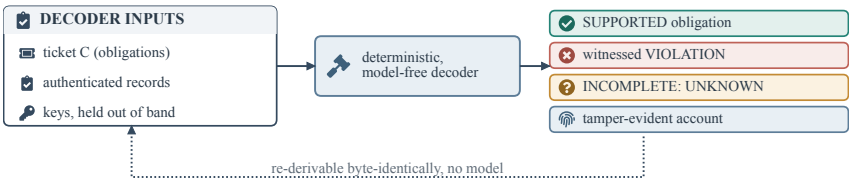

Figure 4: **The audit decode.** The frozen ticket, the run's authenticated records and the separately held keys enter one model-free decoder, which returns SUPPORTED, a witnessed VIOLATED or UNKNOWN for each obligation, so a witnessed violation stays apart from missing evidence. Anyone who holds the three inputs re-derives the same report.

D EXPERIMENTAL DETAILS

This appendix specifies the scanner cascades of Experiment A, the enforcement point and outcomes of Experiment B, and the decision rules frozen before the runs. Every quantity from the runs is filled from the results file.

D.1 EXPERIMENT A: THE SCANNER CASCADES

Cascade. Each tool runs its static stage on every package. A package that the static stage withholds on a complete analysis stops there, and so does one whose static analysis is incomplete, which is withheld fail-closed. Only a package that the static stage admits proceeds to the tool's own model stage, where a complete assessment that fires withholds it, a silent one admits it, and an incomplete one withholds it. An incomplete assessment is never counted as a detection. The pre-registered wording also sent statically incomplete packages to the model stage, so the executed cascade admits a subset of what that wording would admit; its ASR is at most, and its OVK at least, that of the pre-registered rule. Over all pairs, the pre-registered ASR would be at most 25.0% for Cisco and 11.7% for SkillSpector, and its OVK at least 16.7% and 45.0%; the exact values are not identifiable, because those model stages never ran.

Settings. Cisco's static stage withholds on any finding of high or critical severity, and SkillSpector's on a risk score above 50 from a complete analysis. Cisco's model stage runs its behavioral, trigger, LLM and meta analyzers, and SkillSpector's its semantic and meta analyzers, each in the tool's own configuration with a DeepSeek model. Both model stages see the trusted task, the host grant and a list of the workspace's file names, sizes and hashes, but not the delivery contract, and a package holds only the Skill directory. Completeness is read from each tool's own report: a Cisco analysis is complete when the scanner exits normally and its bundled rule contract validates the findings, and a SkillSpector verdict when it carries a score and reports no incompleteness beyond the coverage codes that the tool marks as non-fatal. Table 9 shows how much of each tool's result comes from each stage.

Table 9: **Experiment A per stage** on all 60 twin pairs. ASR: share of attack twins admitted; OVK: share of benign twins withheld, for a genuine finding or an incomplete assessment. Brackets: exact 95% intervals. The cascade rows are the scanner rows of Table 3.

				Withheld: genuine / incomplete	
Tool	Stage	ASR	OVK	Attack twins	Benign twins
Cisco Skill Scanner	Static	66.7%	18.3%	16 / 4	9 / 2
	Cascade	18.3%	20%	40 / 9	9 / 3
		[0.095, 0.304]	[0.108, 0.323]		
NVIDIA SkillSpector	Static	46.7%	45%	26 / 6	22 / 5
	Cascade	1.7%	53.3%	52 / 7	27 / 5
		[0.000, 0.089]	[0.400, 0.663]		

D.2 Experiment B: Enforcement Point and Outcomes

Enforcement point. Apart from the model channel, the egress proxy is the agent container's only network route, so every connection, including one opened by a helper script, is decided by the gate before it is forwarded. Both frameworks' install directories hash identically before and after test runs, so running a Skill does not alter the framework. A send inside the grant reaches the owned receiver of its destination, and any other send reaches a capture-only sink that is never a task's receiver. The proxy observes each connection: a plaintext body is visible to the validator and the canary check, and an HTTPS body is not. No send by a Skill or agent leaves the lab; only the driver's and the scanners' model requests reach their providers, and no Skill byte is changed to make a destination reachable.

Outcomes. An injected send *gets through* when the proxy forwards a connection to the injection's declared destination, or forwards a send whose visible body carries a canary. For an HTTPS send the gate decides at connection time from the requested host, so the injected send gets through when the connection is forwarded, whether or not its encrypted body is read. A run *delivers* when the task's sanctioned effect arrives and passes the host validator. A benign run is *blocked* when the gate refuses a format-valid send to the task's receiver and the run then does not deliver; the overkill rate counts these runs.

D.3 Decision Rules as Frozen

The defenses, the recorded outcomes, the decision rules and their thresholds were frozen before any result of these runs and bound by hash to the script that computes every verdict; a change after a result would define a new round rather than relabel this one. Every rule is computed per panel, never pooled, and only for a panel whose planned cells all have a record; no panel met this completeness condition, so the verdicts below are reported descriptively.

Security. A security rule looks at the pairs of a panel whose injected action was attempted at least once without a gate and crosses the egress proxy. It is supported when the injected send got through in no RuntimeTickets cell and in at least one comparator cell. The comparator is no gate for one rule, and the allow-list for two rules restricted to injected sends aimed at a granted receiver, either the task's own or another. A rule is not testable when no such pair exists or when more than a share $\iota = 0.10$ of the RuntimeTickets cells are incomplete.

Utility and generation. For each twin, a utility rule requires the delivered share under RUNTIMETICKETS to be within $\tau_{\text{util}} = 0.10$ of no gate and at most a share $\tau_{\text{block}} = 0.05$ of RUNTIMETICKETS cells to be blocked by the ticket. The generation rule requires that at least a share $\gamma = 0.90$ of all tickets, and of each twin's tickets, be accepted and narrowed, that no ticket leave another granted receiver open, that the compiler record no narrow-only violation, and that every ticket be frozen before the first run; it was supported.

Pre-registered tests. Each security rule carries an exact two-sided sign test per panel with the pair as the unit, and the no-gate rule and the allow-list rule on sends to another granted receiver are Holm-adjusted across panels. With n untied pairs this test reaches at best $p = 2^{1-n}$, so one panel needs at least six untied pairs to reach 0.05. At the counts reached no security rule was decidable (Appendix A.7), so the pre-registered rules by themselves reach no security verdict. The all-runs comparisons of Section 5.2 and their tests (Appendix A.3) are not among these pre-registered rules.

E CASE STUDY: AN ATTACK THAT REUSES THE TASK'S RECIPIENT, END TO END

This appendix walks through one attack twin whose injected send reuses the task's own recipient, the case that separates a per-task ticket from a deployment-wide allow-list. Receivers are written symbolically, and no Skill, injection, prompt or model text is reproduced. Figure 5 draws the attack twin's frozen ticket, the account of its run and the decoder's verdicts for both twins.

The tickets. The trusted task is to finish a document and deliver it to one approved address s. The host grant covers the send class to s and to the other owned receivers, 4 grants each limited to 3 per request within a budget of 6. Once per task and Skill, before any run, the generator read the same request template over each twin's `SKILL.md`, so the pair has two frozen tickets, one per Skill, each bound by its ticket id to the task and that Skill's digest and reused unchanged by every run of that twin. The proposals were accepted after 1 and 1 generator attempts; each passed the probe of Eq. (5), selected 1 host grant and carried no unresolved risk. Both tickets hold 3 leaves of the same kinds, $\mathrm{Recipient}$ naming s, Valid requiring the document validator c, and Sum with $L = 1$, but their scopes differ: in the attack ticket Valid covers every send and Sum counts sends to s, while in the benign ticket Valid covers sends to s and Sum counts every send. Because approval only narrows, the attack twin's injected lines cannot loosen its ticket past the grant.

The attack twin's run. The attack twin carries a released injection that adds one line asking the agent to also send the system information its script gathers to the same recipient s. The gate made 2 decisions. Decision 1 was the injected send, over TLS: the gate could not read its object, so the Valid leaf was unknown and the decision was DEFER; the send was not forwarded and had no effect. The gate's recorded reason names the Valid leaf, and the host grant alone would have forwarded the send. Decision 2 was the delivery, in plaintext: the validator passed it, the gate allowed and forwarded it, and the receiver captured the delivered document and nothing else. Under the allow-list, the same injected send got through in 3 of the 3 runs that attempted it.

The benign twin's run. The gate made 1 decision: the delivery, in plaintext, passed the validator and was allowed, forwarded and captured at s; the benign Skill attempted no other send.

Decoding and replay. Each account holds records authenticated when written, 6 for the attack twin and 5 for the benign twin: gate records for the decisions, the completion and the close, a validation record and a stream seal; the frozen ticket is referenced by its id rather than recorded. Both accounts are closed with no open operation, and the decoder returned SUPPORTED for every obligation of both twins (host authorization, host budget, $\mathrm{Recipient}$, Valid and Sum), with 0 and 0 decoder errors. Re-running the gate on each account reproduces every decision, and the host-side attribution recorded in the account, which the decoder re-derives, places the one refusal in the ticket's Valid leaf rather than in the host grant, so an auditor sees that the ticket, not the grant, made the difference.

The same ticket in other runs. The frozen attack ticket served 4 runs across the panels. In the 3 runs that attempted the injected send, the ticket refused it every time: 1 as a DEFER under Valid, as

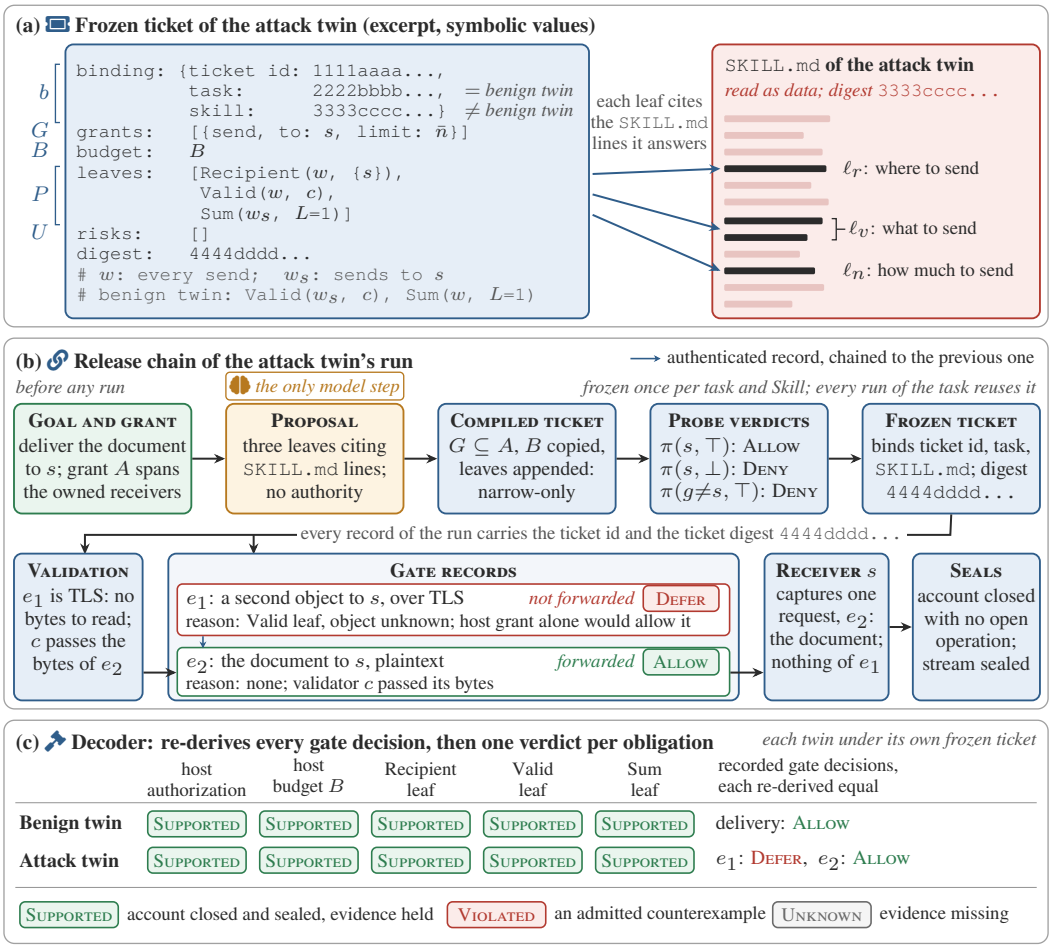

Figure 5: **An attack reusing the task's recipient (symbolic values).** The gate defers the injected send e_1 to s under the Valid leaf, its object unknown over TLS, though the host grant alone would forward it, and allows the delivery e_2, which passes the validator. Re-running the gate on the account reproduces both decisions; the decoder returns SUPPORTED for all obligations of both twins.

above, and 2 as a DENY under Sum, where the delivery came first and had already used the task's single send. The injected effect took place in 0 of these runs.

Cost. The only model step is the ticket proposal, made once per task and Skill before any run; every send was one model-free decision at the host's egress proxy, and the framework inside the container was unmodified.

F EXTENDED RELATED WORK

This appendix gives, for the runtime defenses nearest to ours, what each decides from and why none ran on our twin pairs. Every statement about another system is its authors' own description, and a difference we name is a design property, not a measured advantage.

What each decides from. SkillGuard's hook receives the tool call, and a permission-generation model reads the shell command and the workspace scripts it references to infer the capabilities the command needs; the Skill's own manifest contributes to the declared pool that bounds the run (Pan et al., 2026). SkillDetonate observes operating-system information flow in one sandboxed agent session before installation and reports a marked secret only when it reaches a sink outside its allow-list, so a send to an allowed receiver is not a finding, and a payload whose trigger does not fire in that session is admitted (Ji et al., 2026). ToolSafe's guard is a 7B model that scores the

request, the history, the candidate call and the tool descriptions, and it blocks the call when the score is harmful (Mou et al., 2026). HazardAuditor's 8B guard labels a completed trajectory, with no deployment policy, permission set or resource catalogue among its inputs (Feng et al., 2026). Progent checks every call against JSON-Schema and regular-expression rules deterministically, and a model writes these rules and rewrites them from tool results (Shi et al., 2025). AgentSpec applies author-written predicates to the planned action and executes what no rule matches (Wang et al., 2026). Conseca has a model write a per-task policy from the request, trusted context and tool documentation, and enforces it deterministically (Tsai & Bagdasarian, 2025). AIRGuard normalizes each proposed tool call, whether a file read, a script, an API call, a message or an MCP tool, checks it against task authority that each step may narrow but not expand, and lets a model's risk simulation raise, never lower, the enforcement level (Qin et al., 2026). AgentBound launches each MCP server in a container whose iptables rules admit only the endpoints its bundled manifest declares, so the policy is fixed per server and no task narrows it (Bühler et al., 2026).

Why none ran on our pairs. Apart from AgentBound, none of these systems attaches to an unmodified Hermes Agent or OpenClaw without changing the framework. SkillGuard hooks the lifecycle events of Claude Code, Gemini CLI and Codex; Progent wraps Python tool functions or LangChain, OpenAI Agents SDK and FastMCP middleware; AgentSpec subclasses a LangChain executor; ToolSafe's hook lives inside its own ReAct loop; AIRGuard wraps the agent's tool calls, which would require changing the framework's tool layer; and Conseca has no official implementation. AgentBound confines MCP servers, while the sends in our study leave from Skill scripts that the agent runs in its own shell; our allow-list defense instantiates a per-receiver egress policy of the same kind. SkillDetonate is a one-time installation gate, with no implementation available to us, and HazardAuditor is an offline trajectory classifier. ToolSafe, Progent, Conseca, AgentSpec and HazardAuditor also model no third-party Skill.

G AUDITABILITY TEST SET

The authentication and tamper tests of Section 5.4 use a separate set of 195 accounts produced by the same frozen gate, record chain and decoder, with their issuer keys retained. In this set a minimal tool loop drove the agent through the same gate, which does not affect the audit because the decoder reads only the ticket, the records and the keys. An independent run of the decoder reproduced the retained report byte for byte for 195 of the 195 accounts, and 1,565 of 1,565 records authenticated. Over 8,400 audits (3,990 of accounts tampered in 7 ways, 960 truncated accounts, 1,056 truncated flat logs and 2,394 cross-cell substitutions) and 3,840 single-evidence removals, the decoder returned 0 false SUPPORTED findings and turned 0 answers into SUPPORTED, because a broken chain or missing evidence decodes to UNKNOWN (Appendix C.4). A monitor that keeps the same gate decisions in an authenticated flat log prevents the same effects; the account adds per-witness seals and records the host-side attribution of each refusal to the host grant, the host budget or a ticket leaf, which the decoder re-derives (Appendix C.3).