35th USENIX Security Symposium · 2026

PatchWeaver: Risk-Bounded Autonomous Vulnerability Remediation Under Change-Management Policies

Rui Li · **Shuang Cao**

Dawn Computing

“Policy-bounded autonomy — proposals are cheap, execution is expensive: simulate first, act second.”

ARTIFACT EVALUATED AVAILABLE

Cloud remediation is a continuous, adversarial race

Exploitation lands in days or hours — and every fix is governed

Patch the wrong way and you cause an **outage or a compliance failure**.

Enterprise remediation is a **governed workflow**:

maintenance windows · staged rollout · blast-radius caps

approval gates · least-privilege · artifact provenance

These rules are often **required for compliance** (SOC 2, PCI-DSS).

The question this paper asks

Not “can an agent patch?” but:

“Can an agent predictably patch without violating policy under uncertainty?”

Today’s “autonomous” remediation is brittle

Two ends of a spectrum — each fails the governed workflow

Scripted playbooks

Safe but rigid

Deterministic and auditable.

Cannot cover the long tail: evolving dependency graphs, heterogeneous environments, exception handling.

LLM-driven agents

Flexible but unsafe

Can produce the command — but lose track of evolving state + governance.

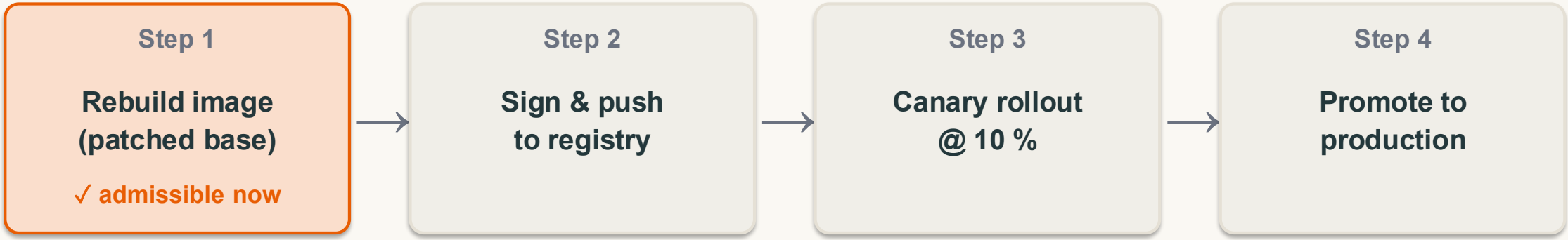
Patch outside windows, take too many replicas offline, skip mandatory validation.

The gap

The failure isn’t that an LLM can’t emit a command — it’s that no layer scores a candidate remediation trajectory against evolving policy before acting.

The trap: every step is legal, the trajectory is not

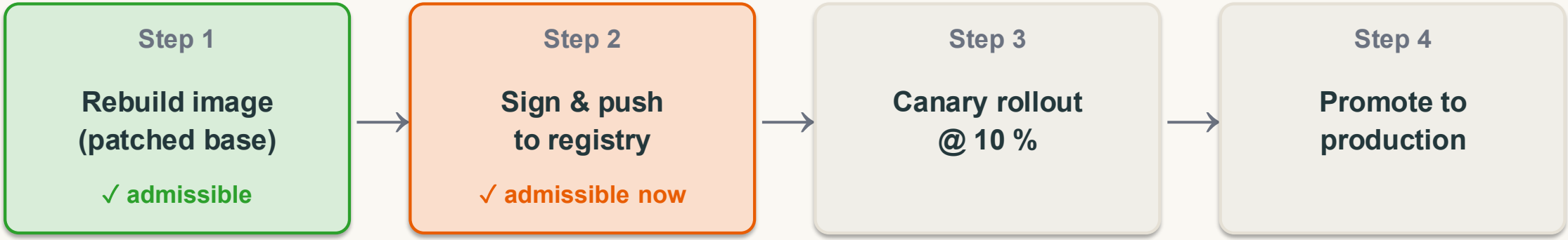
Running example — a 4-step canary remediation



Precondition: the production-approval ticket is issued only AFTER step 2 completes and the staging gate passes.

The trap: every step is legal, the trajectory is not

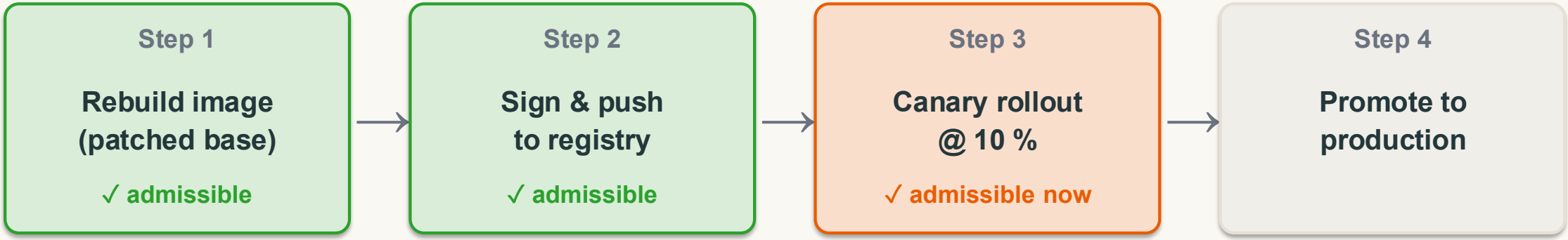
Running example — a 4-step canary remediation



Precondition: the production-approval ticket is issued only AFTER step 2 completes and the staging gate passes.

The trap: every step is legal, the trajectory is not

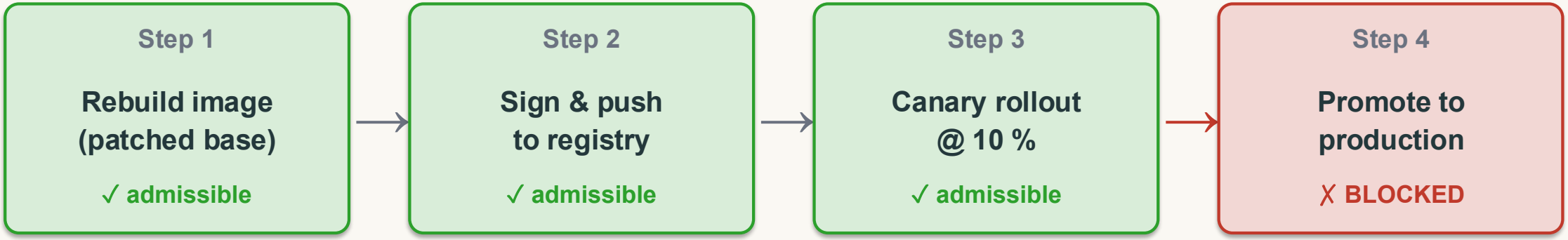
Running example — a 4-step canary remediation



Precondition: the production-approval ticket is issued only AFTER step 2 completes and the staging gate passes.

The trap: every step is legal, the trajectory is not

Running example — a 4-step canary remediation



Precondition: the production-approval ticket is issued only AFTER step 2 completes and the staging gate passes.

TRAPPED

Can't promote (no approval); can't roll back (policy forbids reverting a canary). Every step passed admission — the trajectory still dead-ends.

→ LLM-Agent falls in here in 34 of 50 canary episodes, adding 47% to time-to-remediate.

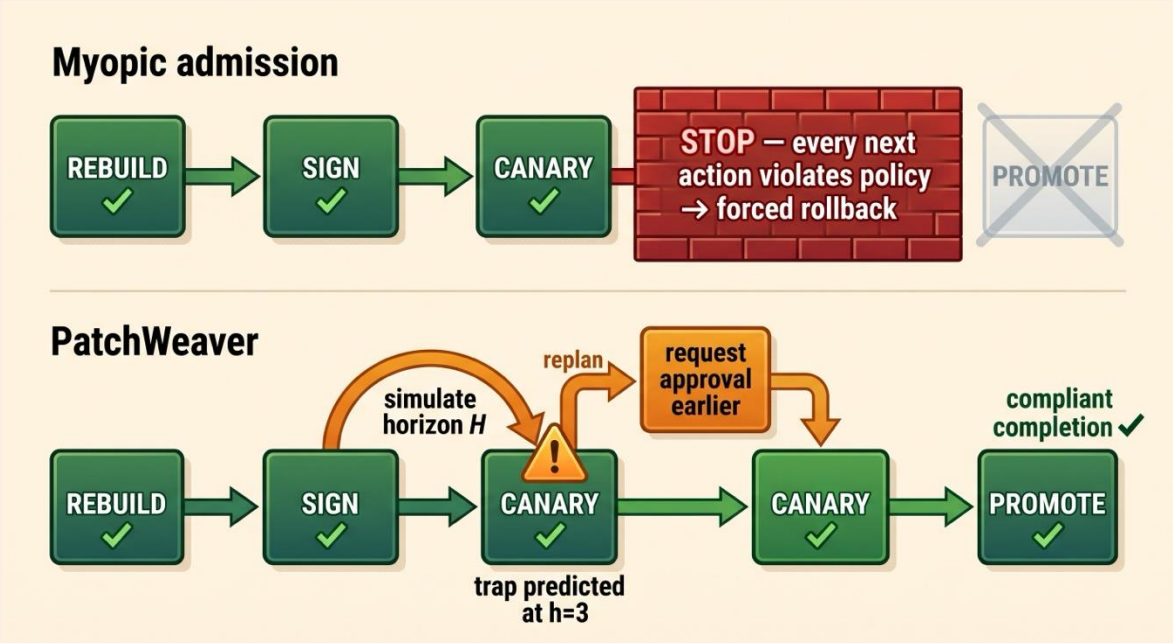
Point-in-time checks cannot avoid constraint traps

Key insight: remediation needs trajectory-level compliance

Existing tools ask: **"Is this action admissible now?"**

PatchWeaver asks: **"Will this plan remain compliant through completion, given what we know about state evolution?"**

FIGURE D4 · Point-in-time vs trajectory-level compliance



Best-in-class tools each solve a different layer

None scores a candidate trajectory against evolving policy

Policy-as-code

OPA · Cedar · Kyverno

Evaluate an individual request at admission time.

Point-in-time — no notion of a multi-step plan.

Rollout controllers

Argo Rollouts · Flagger

Gate progressive delivery stages (canary / blue-green).

Control delivery, not policy over evolving state.

Workflow frameworks

LangGraph · LangChain

Model stateful agent flow + human-in-the-loop.

Encode control flow, not trajectory compliance.

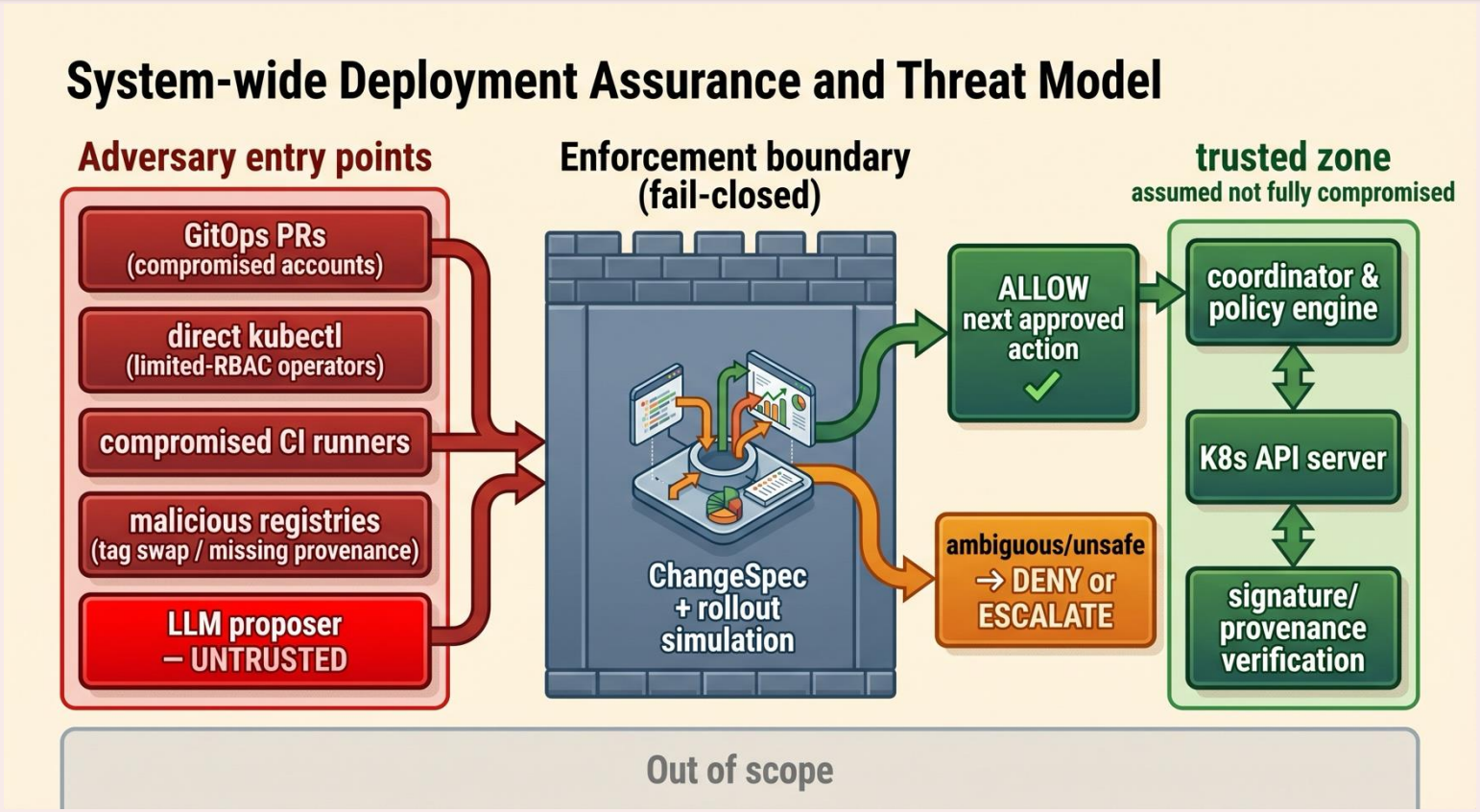
We build ON these, not against them

PatchWeaver's contribution is the missing layer: executable governance + short-horizon trajectory scoring + fail-closed execution.

Threat model: the LLM is an untrusted proposer

Fail closed — ambiguous or unsafe actions are denied or escalated

FIGURE D5 · Threat model & trust boundary



Adversary can:

- Manipulate observations (drift)
- Achieve partial compromise
- Exploit partial remediation
- Submit adversarial inputs

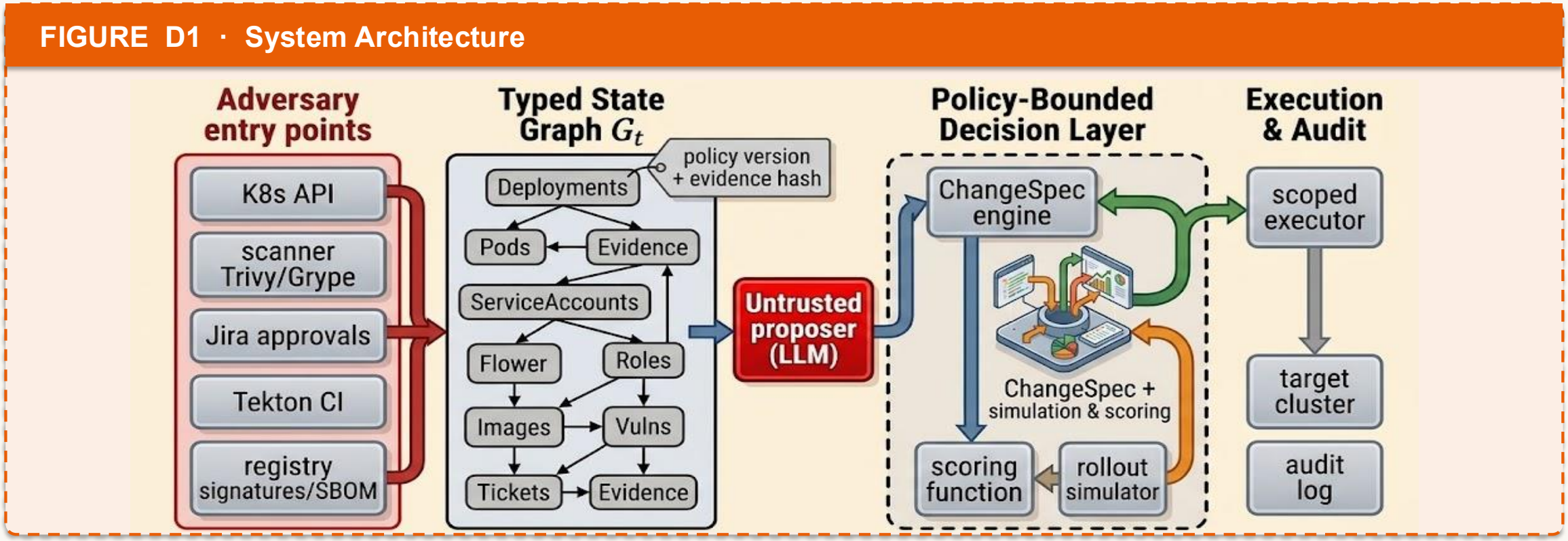
Security goals:

- Policy compliance
- Least-privilege execution
- Auditability · Drift-robustness
- Attack resistance

PatchWeaver: Propose → Simulate → Act

Three elements behind one control plane

FIGURE D1 · System Architecture



① Typed state graph of assets, deps, vulns, evidence

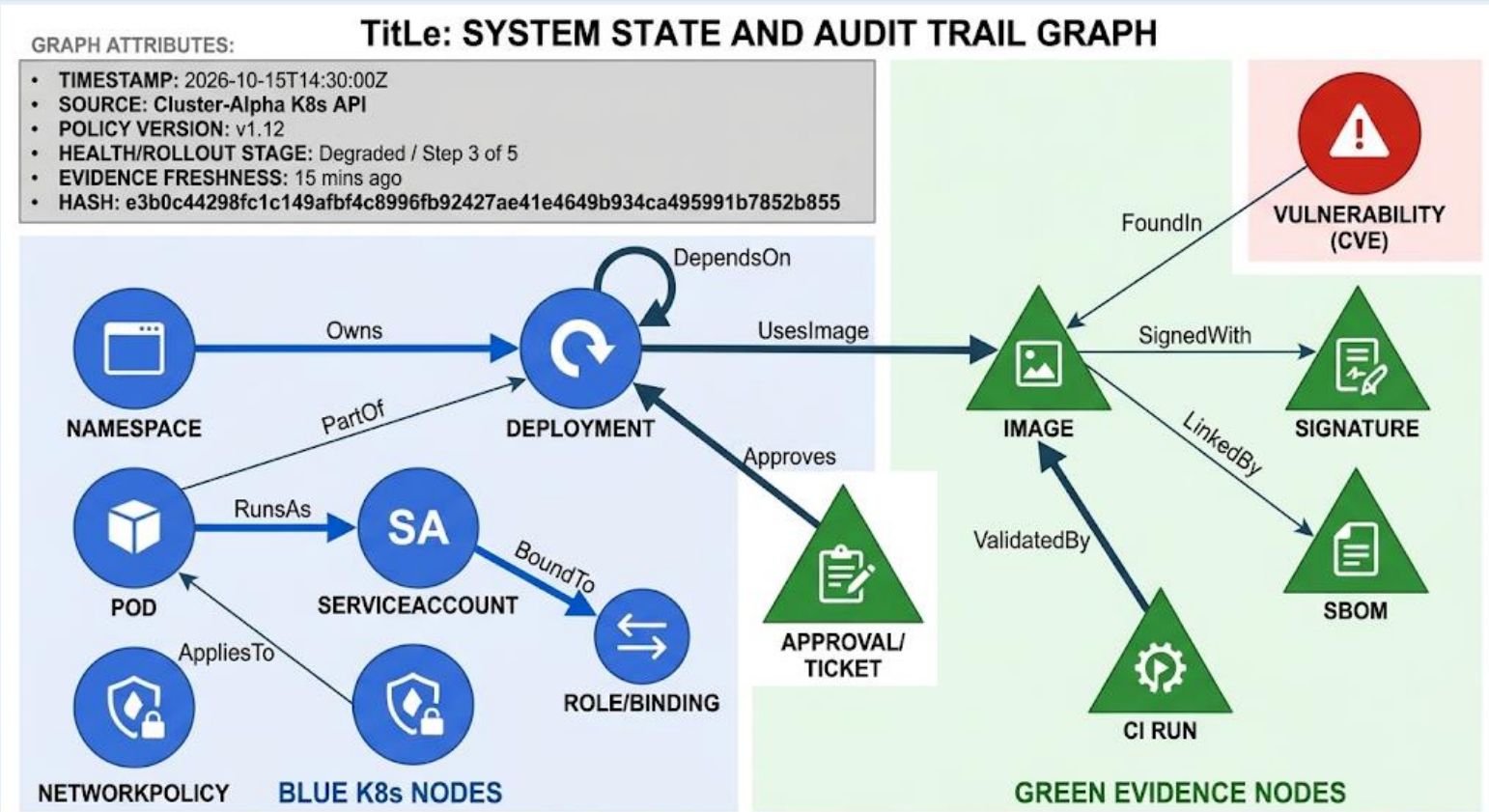
② ChangeSpec executable policy predicates

③ Rollout decision layer score plans before acting

① Typed state graph $G_t = (V_t, E_t, X_t)$

A graph of entities & evidence — source-labeled, versioned, hashed

FIGURE D2 · Typed state graph



Not a workflow graph

Nodes = operational entities + evidence

Edges = RunsAs, BoundTo, UsesImage, Approves, ValidatedBy, DependsOn

Attributes = versions, health, rollout stage, evidence freshness + hashes

Refreshed continuously via K8s watch streams (delta-merge)

Same schema scored on current AND predicted graphs

② ChangeSpec: executable change-management policy

Typed predicates that return a decision + an audit-grade explanation

```
rule deny_rbac_wildcard {
  match: action.type in
    {ApplyRole, UpdateClusterRole}
  when: action.object.rules[*]
    .verbs contains "*"
   OR action.object.rules[*]
    .resources contains "*"
  decision: DENY
  priority: 90
  explain: "RBAC wildcards
    violate least privilege"
}
```

Four decisions (terminal order):

DENY | **ESCALATE** | **REQUIRE(evid.)** | **ALLOW**

DENY > ESCALATE > REQUIRE > ALLOW

A high-priority escalation cannot override a hard deny.

32 representative predicates across 7 categories · authored in 4.5 person-hours

- **Time windows (3)**
- **Ordering / gates (6)**
- **Blast radius (5)**
- **Least-privilege (6)**
- **Approvals (4)**
- **Admission safety (4)**
- **Artifact trust (4)**

③ Decision layer: score plans before acting

Decouple proposal from scoring — the LLM proposes, policy + rollouts decide

Score each of K = 12 candidate plans:

$$S(\pi) = \Sigma\, r(\hat{G}, a) \;-\; \lambda_v\, \Sigma\, v(\hat{G}, a) \;-\; \lambda_e\, \Sigma\, e(\hat{G}, a)$$

progress reward − hard-violation − escalation cost

Defaults: $H = 6$, $\lambda_v = 1.0$, $\lambda_e = 0.25$

Constraint trap (Def. 1)

A state where ALL admissible H-step plans eventually violate policy.

If every candidate scores a hard violation → escalate with a trap explanation.

Drift detection → replan when reality diverges from prediction

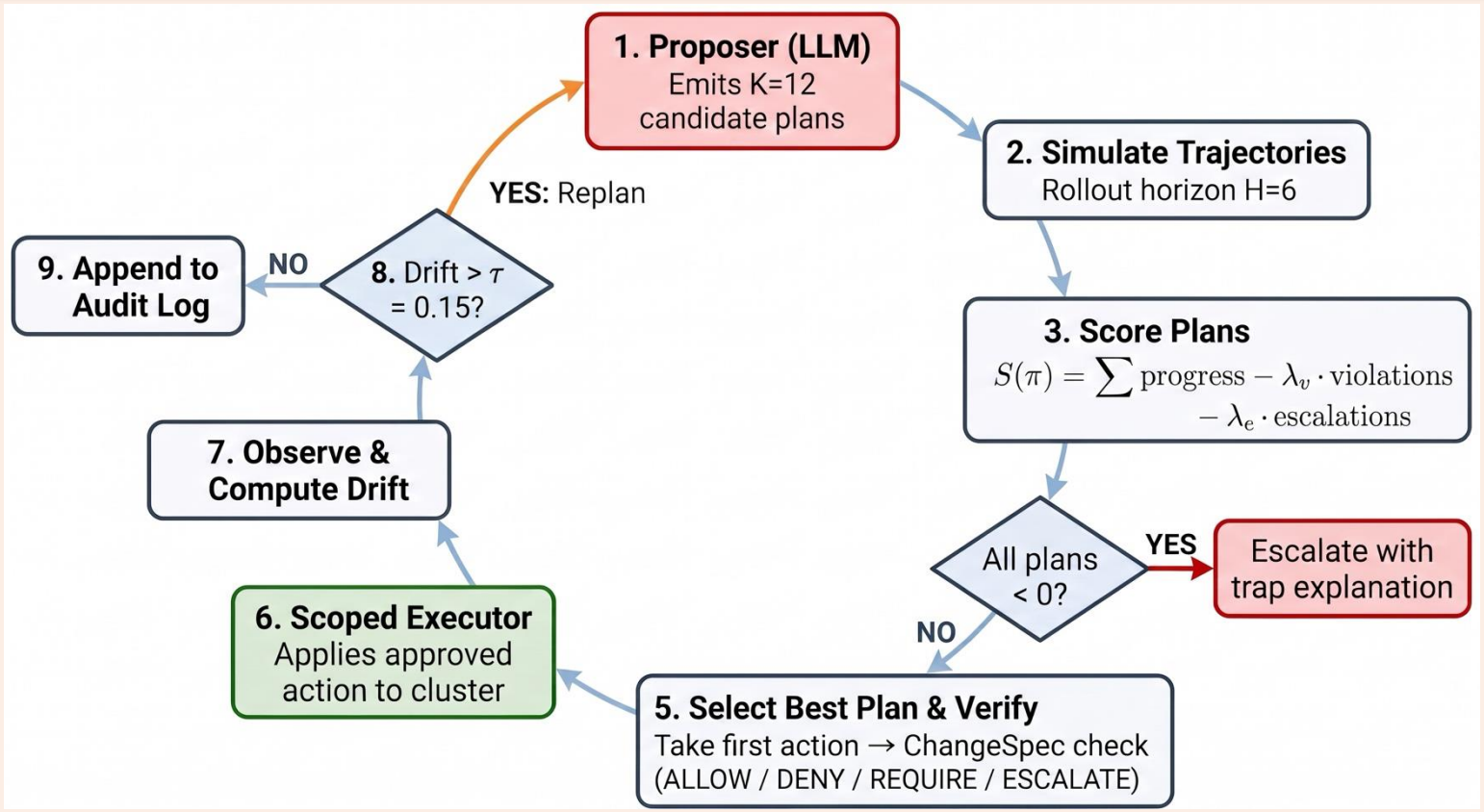
$$\text{Drift}(G_t, \hat{G}_t) = 1 - (|V_t \cap \hat{V}_t| + |E_t \cap \hat{E}_t|) \,/\, (|V_t \cup \hat{V}_t| + |E_t \cup \hat{E}_t|) \qquad \text{replan when Drift} > \tau = 0.15$$

λ_v is set larger than any single-step reward — a predicted hard violation can never outrank a compliant plan.

The propose–simulate–act loop in one step

Algorithm 2 — the same policy engine runs on current and predicted graphs

FIGURE D3 · Decision loop (Algorithm 2)



Trap avoided

Canary Gating:

LLM-Agent attempts full rollout without approval → blocked → rollback.

PatchWeaver predicts the dead end at h = 3, requests approval earlier → completes, no violation.

Evaluation setup

200 episodes · 7 baselines · 10 attack categories · seeded for replay

4 workloads (50 episodes each)

- ImageRebuild — patch CVE-2024-21626 (runc)
- RBAC Minimization — 12 ServiceAccounts
- Canary Gating — ≤ 20% unavail. + staging gate
- PolicyDrift Recovery — mid-rollout change

Platform & proposer

- 12-node K8s v1.28.4 · AWS EC2 m5.2xlarge
- Gatekeeper 3.14 · Argo 1.6.4 · Cosign 2.2.2
- Proposer: GPT-4-turbo (temp 0.0), 10 req/s
- ~8,400 LoC Go + 1,200 LoC policy DSL

7 baselines

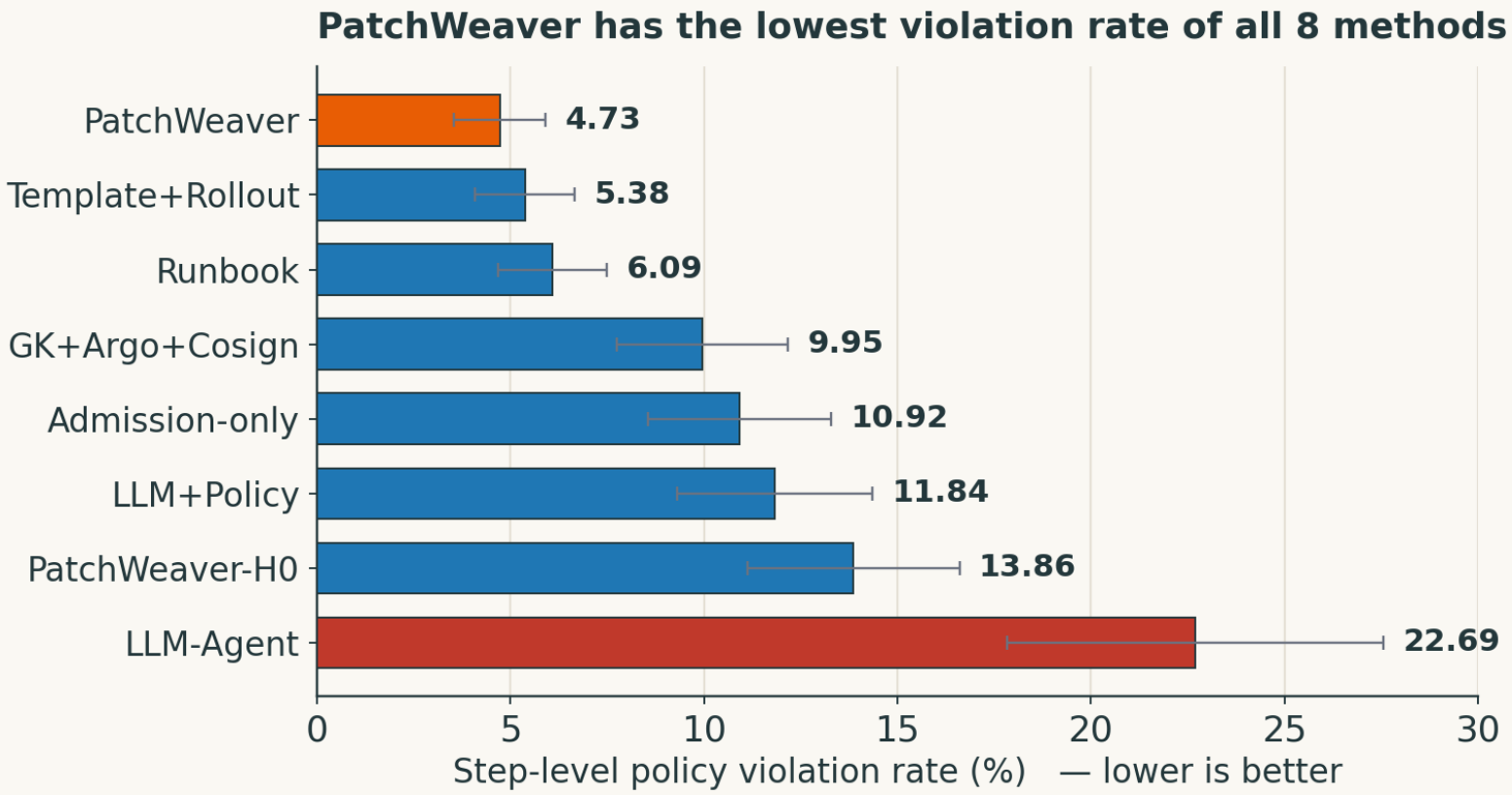
- Admission-only · GK+Argo+Cosign toolchain
- Runbook (static) · LLM-Agent (ReAct, no policy)
- PatchWeaver-H0 (no lookahead) · LLM+Policy
- Template+Rollout (det. proposer + rollouts)

Metrics

- Step- & episode-level violation rate
- Time-to-remediate · escalation · rollback
- Attacks: block / escalate / miss + dwell
- Paired Wilcoxon · Holm · 95% bootstrap CIs

Plan-level reasoning cuts violations in half — and more

Step-level policy violation rate, aggregated over 4 workloads



−52.5%

step violations vs. strongest static baseline (GK+Argo+Cosign)

−79.1%

step violations vs. LLM-Agent

3.1×

lower worst-case (p99) episode violations vs. LLM-Agent

It’s the tail that causes incidents

PatchWeaver’s violation distribution is shifted left with a tighter tail

16.8%

PatchWeaver p99 episode violation rate

51.3%

LLM-Agent p99 (= 3.1× worse)

4.24%

PatchWeaver median (vs 26.1% LLM-Agent)

4.08%

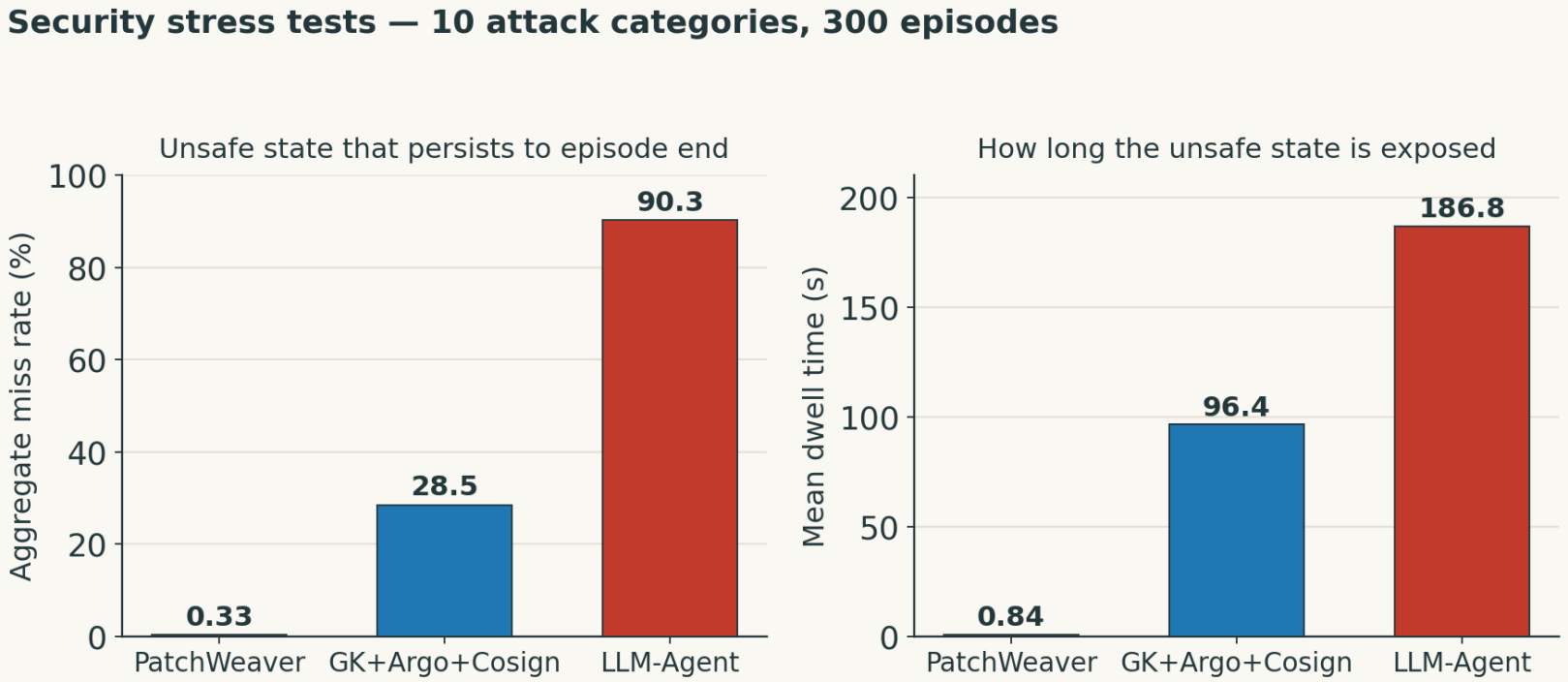
PatchWeaver IQR (vs 12.5% LLM-Agent)

Why the distribution matters

90% of PatchWeaver episodes stay below an 11.4% violation rate — vs 21.5% (GK+Argo+Cosign) and 42.4% (LLM-Agent). Rare bad episodes drive rollbacks and incidents.

Security stress tests: adversarial, multi-step attacks

10 attack categories from real CVEs & incidents · 300 episodes



0.33%

aggregate miss rate
(GK 28.5% · LLM 90.3%)

0.84 s

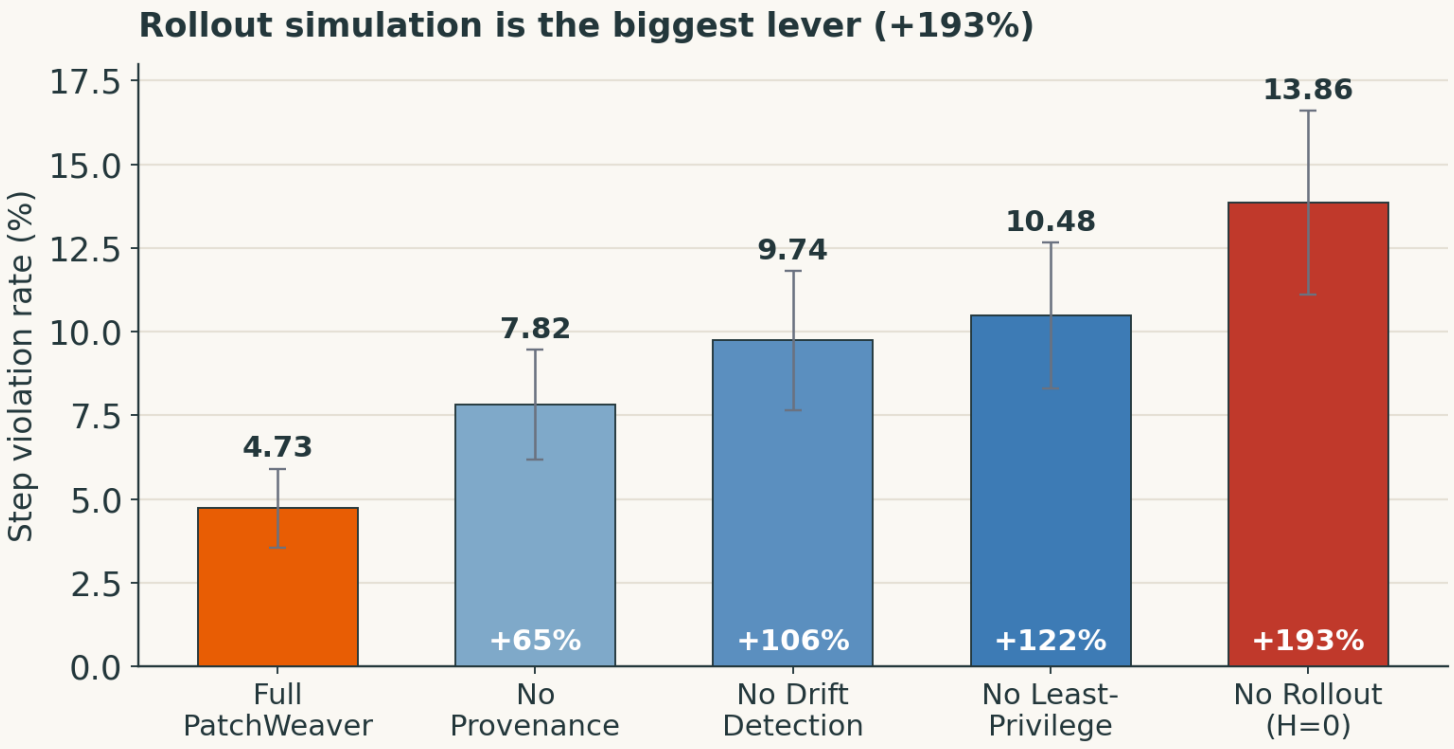
mean dwell time
(GK 96.4 s · LLM 186.8 s)

88.7 / 11.0

% blocked / escalated
escalation is a deliberate valve

Every component earns its place — and it's cheap to run

Ablation (left) and control-plane overhead (right)



Control-plane overhead

7.4 / 31 ms

admission latency p50 / p99
at 32 predicates (30 s timeout)

0.86 cores · 928 MB

total resources
(LLM API client dominates)

Decision latency 1.2–3.8 s (LLM-bound); p99 admission < 50 ms even at 48 predicates.

What PatchWeaver does — and does not — guarantee

Fail-closed by design; escalation is a feature, not a bug

Guarantees

- Accepted actions pass policy on current + simulated near-future states
- Otherwise: deny, wait for evidence, or escalate to a human
- Least-privilege execution via scoped connectors
- Every decision logged with policy version + evidence + explanation

Does NOT guarantee

- That your policies are complete or correct (enforced as written)
- Remediation success when a compliant plan needs missing evidence
- Not a replacement for OPA/Rego or Cedar
- Fail-closed can over-escalate (7.82%) → alert-fatigue tuning
- Evaluated on a single 12-node cluster; federation left to future work

Policy-bounded autonomy: simulate first, act second

Treat the LLM as an untrusted proposer; bound it with executable governance

Constraint traps — locally admissible actions that dead-end — are invisible to point-in-time checks.

Short-horizon rollout scoring over an executable policy layer catches them before execution.

Practical & deployable — Kubernetes-native, sub-50 ms admission, under 1 CPU core.

PatchWeaver — headline numbers

−52.5%	−79.1%	0.33%	0.84 s	3.1×
step viol. vs toolchain	step viol. vs LLM-Agent	attack miss rate	attack dwell time	lower p99 tail

Artifact available — Zenodo DOI 10.5281/zenodo.20477170