

USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

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

Rui Li and Shuang Cao, *Dawn Computing*

https://www.usenix.org/conference/usenixsecurity26/presentation/li-rui

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 35th USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 35th USENIX Security Symposium.

August 12–14, 2026 • Baltimore, MD, USA

ISBN 978-1-939133-58-8

Open access to the Artifact Appendices to the Proceedings of the 35th USENIX Security Symposium is sponsored by



جامعة الملك عبدالله للعلوم والتقنية

King Abdullah University of Science and Technology



USENIX Security '26 Artifact Appendix: PatchWeaver

Rui Li
Dawn Computing
mark.li@dawncomputing.com

Shuang Cao
Dawn Computing
scao@dawncomputing.com

A Artifact Appendix

A.1 Abstract

This artifact contains the PatchWeaver control-plane implementation, the ChangeSpec policy interface, deterministic remediation workloads, adversarial stress tests, ablation and sensitivity evaluators, overhead micro-benchmarks, raw result files, and figure-generation scripts. The artifact has received the *Artifacts Available*, *Artifacts Functional*, and *Results Reproduced* badges. The artifact exercises the real PatchWeaver state graph, ChangeSpec policy engine, rollout scorer, action vocabulary, and transition model in a deterministic discrete-event evaluation. The evaluation reruns the experiments, regenerates result JSON files and figures, and validates the paper's main claims: trajectory-level planning reduces policy violations, multi-step attack scenarios are blocked or escalated, ablations identify the key contributors, and policy-evaluation overhead remains practical.

A.2 Description & Requirements

The archive includes Rust crates for PatchWeaver, ChangeSpec, benchmarks, and evaluation utilities; workload and attack specifications; default policies and configuration; generated JSON results; paper-result figures; and scripts for building, running, and plotting the artifact. The artifact supports the paper's claims about policy-bounded remediation by executing the same policy and trajectory-scoring logic on controlled workloads.

A.2.1 Security, privacy, and ethical concerns

The artifact uses synthetic Kubernetes-style workloads and simulated attack scenarios. By default it does not connect to a live Kubernetes cluster, cloud account, ticketing system, registry, LLM provider, or evaluator data source. The attack experiments are simulations that write result files under the artifact directory; they do not launch privileged containers, modify host security settings, or perform destructive operations. No private user data is included.

A.2.2 How to access

The permanent artifact archive is available from Zenodo through the concept DOI `https://doi.org/10.5281/zenodo.20477169`, which resolves to the latest published version while retaining access to earlier versions. Download the archive, unpack it, and enter the artifact directory containing `Cargo.toml` and `scripts/`.

A.2.3 Hardware dependencies

None. A standard CPU-based environment is sufficient. We recommend at least 4 CPU cores, 8 GB RAM, and 10 GB free disk space for Rust build artifacts and generated results. No GPU, trusted execution environment, FPGA, custom hardware, Kubernetes cluster, or GPT/API credential is required for the Phase-2 artifact evaluation.

A.2.4 Software dependencies

The artifact has been prepared for common Unix-like environments such as Ubuntu 22.04 LTS or macOS. Required software is Rust 1.75 or newer with Cargo and Python 3.8 or newer with `venv` support. The helper script `scripts/setup.sh` creates an isolated `.venv`, installs the versions of `numpy` and `matplotlib` declared in `requirements.txt`, and builds the benchmark runner. It does not use Ubuntu's system `python3-matplotlib` package.

A.2.5 Benchmarks

The benchmark suite contains four remediation workloads: ImageRebuild, RBAC Minimization, Canary Gating, and PolicyDrift Recovery. It also contains ten Kubernetes security stress-test categories, an ablation study, horizon and penalty

sensitivity sweeps, and a policy-evaluation latency benchmark.

A.3 Set-up

A.3.1 Installation

After downloading and unpacking the Zenodo archive:

```
cd <artifact-directory>
bash scripts/setup.sh
```

The setup script builds the main PatchWeaver binary and the benchmark runner. If Rust is already installed, the core build step can also be run directly:

```
cargo build --release -p benchmarks
```

A.3.2 Basic Test

Run a small workload to verify that the evaluator, policy engine, rollout scorer, and result writer are working:

```
mkdir -p /tmp/patchweaver-smoke
./target/release/benchmarks evaluate \
  --workload imagerebuild \
  --num-episodes 2 \
  --seed 42 \
  --output /tmp/patchweaver-smoke \
  --baselines
ls /tmp/patchweaver-smoke
```

Expected outcome: the command completes without errors and writes JSON result files in `/tmp/patchweaver-smoke`. Results are deterministic for the specified seed.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): PatchWeaver's artifact includes the main implementation components needed to exercise policy-bounded remediation: typed state graph, ChangeSpec policy predicates, plan proposer interface, rollout scorer, executor stubs, and audit logging. This is checked by the build and basic test.

(C2): In the artifact evaluation, trajectory-level planning (PatchWeaver and Template+Rollout) produces lower policy-violation rates than myopic, partial-policy, and no-policy baselines. This reproduces the main Section 7 claim that trajectory-level compliance checking is the primary driver of safer autonomous remediation.

(C3): The security stress tests show that explicit policies plus trajectory scoring block or escalate multi-step adversarial scenarios that evade point-in-time checks.

(C4): The ablation, sensitivity, and overhead experiments reproduce the paper's component-level claims: removing rollout simulation or drift detection substantially increases violations, while policy-evaluation latency remains practical for admission-style use.

A.4.2 Experiments

(E1): Main workload evaluation [5 human-minutes + 10–30 compute-minutes]:
Preparation: Build the benchmark runner using the Installation command above.
Execution: Run:
`export SEED=42`
`bash scripts/run_experiments.sh`
Results: Inspect `baseline_comparison.json` under `results/workloads/`. PatchWeaver and Template+Rollout should remain among the lowest-violation methods, reproducing the paper's main comparative claim that trajectory-level planning is safer than myopic, partial-policy, and no-policy baselines.

(E2): Security stress tests [included in E1, or 5–10 compute-minutes when run separately]:

```
./target/release/benchmarks attack \
  --category all \
  --num-episodes 30 \
  --seed 42 \
  --output results/attacks
```

Expected output is `attack_results.json` under `results/attacks/`, summarizing block, escalation, miss, and dwell-time behavior for ten attack categories. The output shows that multi-step attack scenarios are rejected or escalated at high rates, reproducing the paper's attack-resistance claim.

(E3): Ablation, sensitivity, overhead, and figures [included in E1]:

```
./target/release/benchmarks ablation \
  --num-episodes 50 \
  --seed 42 \
  --output results/ablation
./target/release/benchmarks overhead \
  --num-predicates 32 \
  --iterations 1000 \
  --output results/overhead
./.venv/bin/python \
  scripts/generate_figures.py \
  --input results/ \
  --output figures/
```

Expected output includes JSON summaries under `results/` and regenerated PDF figures under `figures/`. The ablation output should show large violation increases when rollout simulation or drift detection is removed, and the overhead output should show practical policy-evaluation latency.

A.4.3 Output-to-paper mapping and scope

The archive's `AE_OUTPUT_MAPPING.md` provides the complete mapping. In summary, `baseline_comparison.json` supports Table 2 and Figure 3; the four workload JSON files support Figure 4; `attack_results.json` supports Table 3 and the standalone artifact-mode Figure 5; `ablation_results.json` supports Table 4; and `overhead_results.json` supports the artifact-mode Figure 6.

The standalone Figure 5 reports PatchWeaver attack dwell times and aggregate outcomes. Baseline dwell-time streams from the original cluster evaluation are not included in the standalone JSON and are therefore not regenerated. The Figure 6 artifact measures local in-process policy-engine latency; webhook round-trip and cluster CPU/memory values require the original cluster and are not reported as standalone measurements.

A.5 Notes on Reusability

Researchers can add workloads under `data/workloads/`, attack scenarios under `data/attacks/`, and policy rules under `data/policies/default_policy.dsl`. The benchmark crate exposes the workload, baseline, ablation, sensitivity, and overhead drivers used by the submitted artifact.

A.6 Version

Based on the LaTeX template for Artifact Evaluation V20231005. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/usenixsec2026/.