



■ WHITEPAPER · SEPTEMBER 2026

The Kyverno Operator's Playbook

Hardening and managing Kyverno at scale — policy testing, CVE response, and safe upgrades.

For SecOps and platform teams running Kyverno in production.

Version risk matrix

Find the release you run

5-question assessment

Score your deployment

3 commands

Check it yourself in a minute

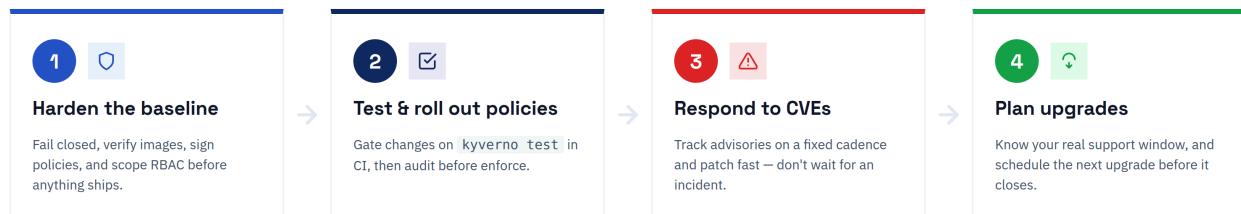
Executive summary

Kyverno sits directly in your admission-control path — every request to create or modify a Kubernetes resource passes through it. That makes it one of the highest-leverage components in a cluster, for defenders and attackers alike. In 2026 alone, four critical-to-high severity Kyverno CVEs (CVSS 8.8–10.0) were disclosed within roughly six months, and every one of them hit exactly the surface area teams tend to treat as “set and forget”: webhook failure behavior, the `apiCall` context, and image verification. Running Kyverno well isn't a one-time install — it's a lifecycle.

This whitepaper lays out that lifecycle as four practical stages: harden the baseline, build a policy testing and rollout discipline, respond fast when a CVE lands, and plan upgrades against a real support window. It builds on the findings of the [CNCF, ControlPlane, and Kyverno maintainer Threat Model and Hardening Guide](#), which modeled 13 threats and mapped 17 hardening recommendations to MITRE ATT&CK and NIST 800-53.

■ THE KYVERNO LIFECYCLE

Four stages, repeated every release



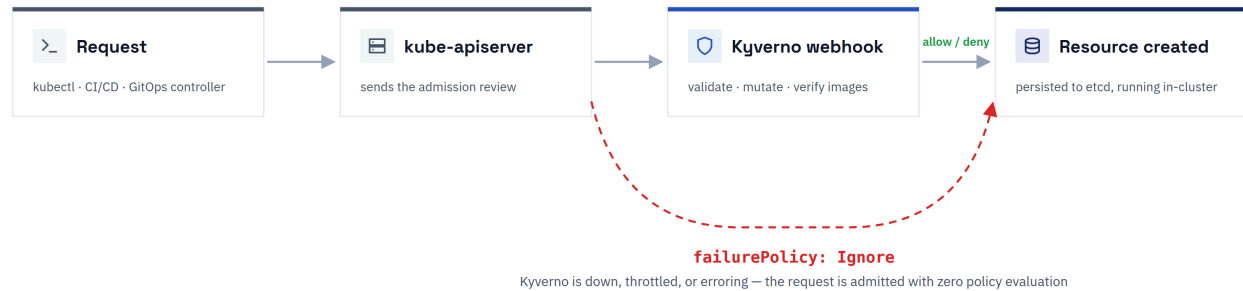
🔄 Every new Kyverno release, and every new CVE, starts the loop again

1. Harden the baseline

Most Kyverno incidents don't start with a novel exploit — they start with a default that was never revisited. The threat model guide's own findings point to the same handful of gaps repeatedly:

- **Fail-open webhooks.** A `failurePolicy: Ignore` setting (or `--forceFailurePolicyIgnore`) means any Kyverno outage — a crash, resource exhaustion, a bad rollout — admits workloads with zero policy evaluation.
- **Unverified images.** No `verifyImages` rule, or one left in audit mode, lets unsigned or tampered images reach production unchecked.
- **Unsigned policy distribution.** Policy bundles pushed without cosign signing — and verification on every cluster — can be swapped for tampered ones in transit.
- **RBAC & PolicyException sprawl.** Broad write access to policy and exception resources lets anyone quietly exempt workloads from enforcement.

Where Kyverno sits — and how it gets bypassed



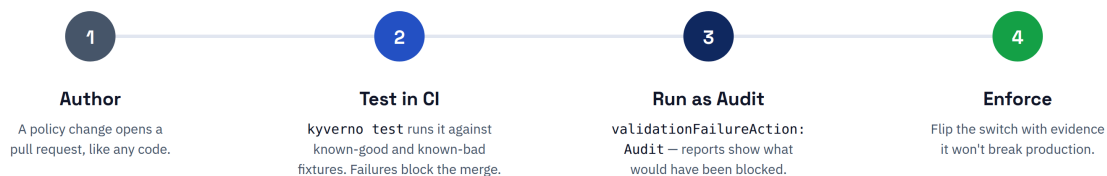
Secure by default. Nirmata Enterprise Kyverno ships with these settings already aligned to the guide's 17 hardening recommendations, so a hardened baseline doesn't depend on every team re-deriving it — and re-applying it — cluster by cluster.

2. Build a policy testing and rollout discipline

Policies are code, and they deserve the same lifecycle discipline: test before merge, roll out gradually, and know when a change is deprecated out from under you. The Kyverno CLI's `test` and `apply` commands support exactly that — running policies against known-good and known-bad resource fixtures in CI, and gating a pull request on the result rather than discovering a regression in production. Pair that with a staged rollout — `validationFailureAction: Audit` before `Enforce` — so a new rule proves itself against real traffic before it can block anything.

That discipline is about to be tested at scale: Kyverno 1.20 removes the legacy `ClusterPolicy`, `Policy`, `CleanupPolicy`, and legacy `PolicyException` types entirely, in favor of CEL-based policy resources. Every fleet running the older types needs a migration plan before that release lands — not after.

From pull request to Enforce



⚠️ **Put the CEL migration through this same pipeline.** Kyverno 1.20 removes `ClusterPolicy`, `Policy`, `CleanupPolicy` and legacy `PolicyException` entirely — every fleet on the older types needs a plan before that release lands.

Nirmata Enterprise Kyverno's policy reporting surfaces audit-mode findings before you flip to enforce, and Nirmata's Kyverno experts can plan the CEL migration across a whole fleet rather than cluster by cluster.

3. Respond fast when a CVE lands

Vulnerability discovery is accelerating across open source, not just for Kyverno: Palo Alto Networks' NOVA research system surfaced 14,090 vulnerabilities across 3,915 open source projects in two months of automated, AI-assisted scanning, and monthly CVE disclosures overall have jumped 145% over the last two years (Dark Reading). Kyverno and other open source projects are seeing similar trends in CVE findings — the table below is one data point: four critical-to-high severity CVEs, disclosed within roughly six months of 2026.

CVE	CVSS	Issue	Fixed in	Also backported by Nirmata
CVE-2026-22039	10.0	Authorization bypass via apiCall context — namespace-scoped requests could reach privileged resources.	1.13.6	1.14.4
CVE-2026-41323	8.8	ServiceAccount token leaked to external servers via an apiCall service URL, enabling cluster compromise.	1.16.4	—
CVE-2026-4789	9.8	SSRF via the CEL http.Lib library — policies could be coerced into calling internal/metadata endpoints.	1.18.0	1.13.6, 1.17.0
CVE-2026-54523	9.6	Boundary gap let namespace-scoped users create resources cluster-wide from a single misconfigurable policy field.	1.18.2	1.17.2

Each of those fixes landed in a different minor release — 1.13.6, 1.16.4, 1.18.0, and 1.18.2. That's the practical problem: upstream Kyverno supports only its latest minor release, for roughly three months of community patches, until the next minor ships and support moves on. A team that pins a version for even one release cycle can already be outside the support window by the time the next critical CVE lands — with no official patch coming for the version they're actually running.

As of September 2026, only 1.19 sits inside the upstream community patch window, and only until 1.20 ships (projected November 2026). Find the version you actually run:

You're running	Upstream community patches	Nirmata Enterprise Kyverno
1.18.x	Ended when 1.19 shipped	Supported through Feb 2028
1.17.x	Ended	Supported through Aug 2027
1.16.x	Ended	Supported through Jul 2027
1.15.x	Ended	Supported through Apr 2027
1.14.x	Ended	Supported through Jan 2027
1.13.x	Ended	Ended Aug 2026

If your row says “Ended” in the middle column, the next critical Kyverno CVE has no community patch coming for the version you run. Your options are an unplanned upgrade, or a vendor that still patches that branch.

The backport gap. This is what Nirmata Enterprise Kyverno is built to close. Nirmata backports these fixes to Kyverno branches upstream has already marked end-of-life — coverage that unpatched community images, including Chainguard's Kyverno images, do not carry. Teams aren't forced to choose between an emergency upgrade and running an unpatched, EOL Kyverno. See the [Nirmata Enterprise Kyverno release notes](#) for the current backport history.

4. Plan upgrades against a real support window

Upgrading Kyverno isn't always a drop-in patch. Kyverno 1.10, 1.13, and the upcoming 1.20 all carry breaking changes or major deprecations, and the project's own guidance is explicit: if you skip a minor version, read the release notes for every version in between before you upgrade. At four minor releases a year, staying inside the community patch window means a review, test, and staged fleet rollout every quarter — an upgrade project that never ends, running purely to stay supported. For teams on regulated, air-gapped, or slow-moving change-control clusters, that math rarely works.

The practical fix is decoupling your upgrade cadence from upstream's EOL clock. Nirmata Enterprise Kyverno's [LTS and compatibility matrix](#) extends support for each release to roughly 18–24 months, mapped explicitly to the Kubernetes versions it runs on — so a version you deployed a year ago is still a supported, patched configuration, not a liability you're racing to replace. Upgrade timing becomes a planning decision your team makes on its own schedule, with direct access to Nirmata's Kyverno maintainers when you make it.

Score your deployment

Five questions, one point each. Answer honestly — the score is the point.

- ☐ Is `failurePolicy: Fail` set in production, with at least three replicas of each controller?
- ☐ Is image signature verification actually enforcing, not sitting in audit mode?
- ☐ Are policy bundles signed, and verified on every cluster that consumes them?
- ☐ Is every policy change gated on `kyverno test` in CI and rolled out audit-before-enforce?
- ☐ Do you know the support end date for the version you run, with the next upgrade already scheduled?

0–2 — a single Kyverno outage or one unpatched CVE becomes a fleet-wide bypass. Start with stages 1 and 3.

3–4 — hardened, but the clock is your weak point. Stage 4 is where you're exposed.

5 — you're operating Kyverno like a pro. The work now is holding that cadence as the fleet grows.

Three commands to check your own answers, in about a minute:

```
# 1. Any fail-open webhooks?
kubectl get validatingwebhookconfigurations \
  -o jsonpath='{range .items[*]}{.metadata.name}{"\t"}{.webhooks[*].failurePolicy}{"\n"}{end}'

# 2. Which Kyverno version are you actually running?
kubectl get deploy -n kyverno \
  -o jsonpath='{range .items[*]}{.spec.template.spec.containers[*].image}{"\n"}{end}'

# 3. Everything else in one command - RBAC exposure, network policies, replica
# counts, PDBs, restarts, HPA, probes, reports offload - each graded
# healthy / warning / critical, with one overall verdict
nctl kyverno health
```

Beyond that health check, **nctl** scans Kubernetes manifests, Helm charts, Terraform plans, Dockerfiles, and GitHub Actions workflows against your policies, runs compliance scans against named standards with evidence auditors accept, and with `--apply` creates the network policies the health check reports as missing. Through `nctl ai` it also generates Kyverno policies from plain-language prompts, writes their tests, and recommends fixes for failing workloads.

Get started

Scan your own cluster — `nctl` runs offline, with no policy engine required — nirmata.com/scanners

Check your version's support window — [Nirmata Enterprise Kyverno compatibility & LTS matrix](#)

Talk to the team that created Kyverno — nirmata.com

Further reading: the [CNCF and ControlPlane Kyverno threat model and hardening guide](#), and the [Nirmata Enterprise Kyverno release notes](#) for current backport history.



About Nirmata. Nirmata provides an **AI Security and Governance Platform powered by Kyverno**. As a leading contributor to the Kyverno project and the maintainer of Nirmata Enterprise Kyverno — a hardened, enterprise-supported distribution used by SecOps and platform teams to run policy-as-code in production with confidence — Nirmata extends that same policy engine to secure AI workloads and agents running on Kubernetes.

Sources

- [Kyverno release cadence and support policy](#) — the three-month community patch window
- [Upgrading Kyverno](#) — breaking changes and skip-version guidance
- [Kyverno 1.18 release announcement](#) — SSRF and token-scoping fixes
- [GHSA-f9g8-6ppc-pqq4](#) — CVE-2026-41323 advisory
- [Testing Policies](#) — Kyverno CLI test and CI integration