controlplane

CLOUD NATIVE COMPUTING FOUNDATION

Kyverno End User Threat Model and Hardening Guide

April 2026

Table of Contents

Executive Summary

This guide presents a threat-model-driven approach to hardening Kyverno and its supporting components within Kubernetes environments. It is intended as a practical reference for platform engineers and security teams, with a focus on architectural risks, operational failure modes, and configuration issues that may not be fully identified through application-focused penetration testing alone.

Kyverno operates as a dynamic admission controller implemented via Kubernetes admission webhooks. It participates in the validation and mutation of resource requests that match its configured scope. As a result, misconfigurations in Kyverno policies, webhook settings, or associated role-based access control (RBAC) can significantly weaken or bypass intended policy enforcement. This is particularly relevant in environments where Kyverno is relied upon as a primary or central control for workload governance.

A threat modeling exercise identified a range of risks affecting Kyverno deployments across both single-cluster and multi-cluster (hub-and-spoke) topologies. These risks span multiple domains, including admission control behavior, policy integrity and supply chain security, RBAC, network isolation, image verification, and external integrations such as registries and service endpoints.

In practice, the most significant risks often arise from misconfigurations rather than underlying software vulnerabilities. For Kyverno, these commonly include webhook availability and failure policy settings that may allow requests to bypass enforcement, supply chain risks affecting policy distribution, and RBAC misconfigurations that permit unauthorized modification of policies or creation of permissive PolicyException resources. In centralized multi-cluster models, compromise of a hub cluster or its associated signing and distribution mechanisms can impact policy enforcement across multiple downstream clusters.

This guide provides hardening recommendations focused on configuration and architecture, including policy enforcement strategies, supply chain protections, access control boundaries, and observability. For selected high-impact scenarios, attack trees illustrate how combinations of misconfigurations can be chained to bypass controls or achieve key attacker objectives, and how layered mitigations reduce these risks.

This document is intended for platform engineers and security teams seeking to deploy Kyverno securely in production, multi-tenant Kubernetes environments - including single or multi-cluster deployments. It complements Kyverno's native security model by addressing risks that arise in real-world operational deployments.

Andrew Martin
CEO, ControlPlane

Preface from Kyverno maintainers

Kyverno helps teams enforce security, governance, and compliance in Kubernetes through policy-as-code. This threat model and hardening guide is intended to be practical and actionable. It focuses on helping teams reason clearly about trust boundaries, common misconfiguration paths, and security tradeoffs when deploying Kyverno across different environments. The goal is to support stronger day-to-day security decisions across admission control, policy lifecycle, exception management, and multi-cluster operations.

A key theme throughout this guide is that policy outcomes depend on platform posture. Strong policies alone are not enough if webhook behavior is fail-open, RBAC is overly broad, exception handling is weak, or promotion workflows are not reviewed and auditable. Kyverno should be treated as a core security control-plane component, with hardening applied accordingly.

This guide provides a strong and practical foundation for hardening Kyverno deployments. Because organizations operate under different regulatory, threat, and operational constraints, teams should apply the guidance in a way that fits their environment. We hope it helps teams identify meaningful improvements and translate them into measurable controls and repeatable operational practices.

As always for a security project, if you believe you have found a vulnerability, please follow Kyverno's responsible disclosure process so issues can be investigated and addressed quickly and safely.

Kyverno is shaped by its community. If you see unclear guidance, missing scenarios, or hardening recommendations that should be added, please open an issue or contribute a pull request. Your feedback directly improves this guide and strengthens security outcomes for everyone using Kyverno.

Introduction

Kyverno is an open-source policy engine designed natively for Kubernetes. It runs as a dynamic admission controller and enables platform and security teams to validate, mutate, generate, and clean up Kubernetes resources using policies defined as Kubernetes custom resources. Policies are authored in YAML and applied to clusters using standard Kubernetes tooling.

Kyverno supports image verification via Sigstore and Notary, policy-based resource generation and cleanup, and produces structured compliance reports through its `PolicyReport` and `ClusterPolicyReport` resources. It is widely adopted in regulated industries including financial services, healthcare, and telecommunications, where consistent enforcement of security, governance, and compliance requirements across Kubernetes environments is critical.

A threat-model-driven analysis of Kyverno is presented alongside practical hardening guidance for production deployments.

Scope

In scope

Threat-model-driven security considerations for deploying Kyverno in production Kubernetes environments, targeting version v1.17.1. The analysis covers the following core open source Kyverno controllers and components:

- Admission controller: the webhook that validates and mutates resources at admission time
- Background controller: scans existing cluster resources against configured policies
- Reports controller: generates `PolicyReport` and `ClusterPolicyReport` resources
- Cleanup controller: deletes resources based on policy-defined criteria and schedules
- Image verification: Cosign and Sigstore integration for supply chain security at admission time

Security considerations focused on deployment architecture, configuration, and operational practices that influence the security posture of Kyverno installations. In particular, we examine:

- Kubernetes Role-Based Access Control (RBAC) considerations for Kyverno components and policy resources
- Policy supply chain integrity, including signing and verification of policy bundles
- Network isolation and multi-tenant security considerations
- Security considerations related to `PolicyException` management and lifecycle
- Image verification configuration and enforcement

The goal is to identify common misconfigurations or systemic architectural risks and provide actionable recommendations for securely deploying and operating Kyverno in production environments.

Out of scope

This document does not constitute a vulnerability assessment, penetration test, or formal security audit of Kyverno or any of its associated components. The guidance provided should not be interpreted as an evaluation of the security posture of specific deployments.

The following sub-projects and other areas are explicitly out of scope:

- Security assessments of custom or bespoke Kyverno deployments
- Kyverno Chainsaw: an end-to-end testing framework for Kubernetes operators and Kyverno policies
- Kyverno JSON: applies Kyverno policies outside Kubernetes to JSON payloads
- The Kyverno Authz Server: external authorisation server sub-project
- Policy Reporter: third-party reporting UI, referenced in example architectures but not assessed
- Any out-of-tree controllers or third-party integrations built on the Kyverno policy engine

A comprehensive threat model of the Kyverno software supply chain is out of scope. However, downstream risks arising from a compromised policy distribution pipeline or tampered policy bundle are considered in scope and addressed where relevant.

Broader systemic threats, such as malicious and intentional source code injections by maintainers, are treated as inherent risks within the open-source ecosystem. Kyverno achieved **CNCF Graduated status** in March 2026, which mandates rigorous governance, community oversight, and transparency requirements.

Additionally, this guide does not address:

- Threats associated with advanced persistent threat (APT) actors, for example highly targeted ransomware campaigns or state level attackers
- Risks that apply only to unsupported or end-of-life versions of Kyverno
- Security considerations specific to managed Kubernetes distributions where Kyverno behaviour may differ from upstream standards

Threat Landscape

Policy engines, such as Kyverno, operate in a highly privileged position within Kubernetes clusters. Specifically, admission controllers sitting directly between all workload deployments and the API server, form a critical enforcement boundary. Thus, misconfigurations in Kyverno, its policies, or associated RBAC are rarely isolated weaknesses; they can degrade or entirely bypass the enforcement layer that other security controls rely on.

CIA impact assessment

Risk severity is evaluated using the Confidentiality, Integrity, and Availability (CIA) triad, mapping key Kyverno components to their real-world operational impact.

Typical risk ratings

Kyverno-related threats are classified using CIA-based risk profiles that reflect their potential impact across production and pre-production environments. These ratings provide a practical basis for prioritising remediation efforts and aligning mitigation strategies with the severity of enforcement failure.

Confidentiality	High	Exfiltration of cosign private keys used to sign Kyverno policy bundles, enabling undetected policy tampering across all consuming clusters Compromise of spoke API server credentials stored in the hub cluster's `policy-reporter` namespace, granting read access to all spoke cluster API servers
	Medium	Exposure of `PolicyReport` or `ClusterPolicyReport` contents, revealing detailed information about which policies are failing on which workloads and which could provide useful reconnaissance for privilege escalation Leakage of Kyverno controller logs containing admission request details, resource metadata, or policy evaluation results
	Low	Disclosure of `ClusterPolicy` or `Policy` resource definitions without write access, which would reveal enforcement posture but does not directly enable policy bypass Exposure of non-sensitive Kyverno `ConfigMap` configuration objects that do not grant policy modification capability

Integrity	High	Unauthorised modification of `ClusterPolicy` or `PolicyException` resources, silently weakening or disabling enforcement for production workloads Compromise of the policy signing key enabling arbitrary unsigned policy bundles to be accepted as trusted, with persistent enforcement impact across all clusters pulling from the external registry Tampering with `ValidatingWebhookConfiguration` or `MutatingWebhookConfiguration` resources, redirecting or disabling admission control entirely
	Medium	Modification of `PolicyException` resources to broaden scope beyond intended workloads, silently exempting sensitive workloads from policy enforcement Manipulation of image verification policy rules to accept unsigned or tampered images at admission time Drift between policy definitions in the external registry and CRDs applied to cluster, creating undetected inconsistency between intended and actual enforcement posture
	Low	Non-critical configuration drift in Kyverno CRs without resulting enforcement bypass Misconfiguration of background scan intervals or `CleanupPolicy` schedules without immediate security impact
Availability	High	External service call failure during policy evaluation. For example, if `ClusterPolicy` rules reference external data sources via HTTP service calls and those services become unavailable, policy evaluation may fail or be skipped at admission time, causing either unexpected workload blocking or silent enforcement gaps depending on `failurePolicy` configuration. Kyverno admission controller failure combined with `failurePolicy: Fail` preventing all resource creation cluster-wide, causing production outage Disruption of `ValidatingWebhookConfiguration` or `MutatingWebhookConfiguration` blocking all admission requests to the API server Loss of policy signing infrastructure preventing policy updates from being distributed to spoke clusters

	Medium	Background controller failure causing existing policy violations to go undetected for extended periods Reports controller failure causing `PolicyReport` and `ClusterPolicyReport` data to become stale, degrading compliance visibility Policy Reporter misconfiguration or failure causing compliance blind spots across multiple clusters without alerting
	Low	Cleanup controller failure causing stale resources to accumulate without immediate security impact Short-term disruption of `PolicyReport` generation in non-production environments Delayed policy propagation to spoke clusters due to external registry or pipeline availability issues

Threat actors

Each relevant threat actor is described in terms of their typical capabilities, motivations, and representative attack patterns, and classified as either in scope or out of scope. In-scope actors are those whose actions are addressed by the threats and recommendations in this guide, while out-of-scope actors operate beyond the practical limits of end user hardening.

In scope

Threat Actor	Capability	Personal Motivation	Sample Attacks
Cluster Administrator	Full control over all cluster resources including CRDs, RBAC, Secrets, admission webhooks, and Kyverno configuration	Financial gain, coercion, insider threat, or operational misuse.	Modify or delete `ClusterPolicy` resources to silently disable enforcement Tamper with `ValidatingWebhookConfiguration` or `MutatingWebhookConfiguration` resources to redirect or disable admission control Create broad `PolicyException` resources exempting sensitive workloads from all policy enforcement Extract cosign private keys used to sign policy bundles, enabling undetected policy tampering
Namespace / Tenant Administrator	Administrative control within a specific tenant namespace, including permissions to manage namespace-scoped `Policy`, `PolicyException`, and `PolicyReport` resources.	Privilege escalation, lateral movement, tenant isolation bypass, financial gain.	Create namespace-scoped `PolicyException` resources to exempt workloads from cluster-wide enforcement Abuse misconfigured RBAC to read `PolicyReport` contents across namespaces, identifying enforcement gaps for exploitation Deploy non-compliant workloads by exploiting `failurePolicy: Ignore` during Kyverno unavailability
Internal Platform Engineer	Authenticated Kubernetes user with restricted cluster-level permissions, typically including read access	Privilege escalation or unauthorized policy modification.	Exploit overly permissive RBAC to modify `ClusterPolicy` or `PolicyException` resources Push unsigned or tampered policy

	to Kyverno resources and pipeline access for policy distribution.		bundles to the external registry Abuse pipeline credentials with cluster write access to apply unauthorised policies directly via `kubectl`
Internal Developer / Workload Owner	Ability to deploy applications and request certificates within specific assigned namespaces.	Accidental misuse, privilege escalation or financial gain	Deploy insecure workloads, for example using privileged containers or excessive capabilities, by exploiting presence of audit-only policies or broad `PolicyException` resources Read `PolicyReport` objects across namespaces to identify policy gaps and plan bypasses Attempt to deploy unsigned images by exploiting missing or misconfigured image verification policies
External vandal (script kiddie, tresspasser)	Exploit publicly exposed services using common tools and known vulnerabilities	Curiosity or notoriety through defacement and service disruption	Trigger denial-of-service of Kyverno admission controller pods, exploiting `failurePolicy: Ignore` to bypass admission controls during the outage Exploit publicly exposed Policy Reporter UI to enumerate policy configurations and identify enforcement gaps

Out of scope

Threat Actor	Capability	Personal Motivation	Sample Attacks
Supply Chain Attackers (Kyverno maintainers and contributors)	Ability to introduce malicious code through open-source contributions or compromised dependencies	Political sabotage or financial gain	Deliberately introduce malicious logic into Kyverno admission controller or policy evaluation engine Accidentally approve or merge malicious or insecure changes from contributors or compromised third-party dependencies. For example, changes could weaken image signature verification or introduce policy evaluation bypasses
Advanced Persistent Threat (APT) / Organized Crime Group	Sophisticated attacker with resources for targeted attacks and long-term persistence including custom exploits and coercion	Data theft, espionage, financial fraud or ransom.	Compromise policy signing infrastructure to distribute tampered `ClusterPolicy` resources cluster-wide, persistently weakening enforcement across all consuming clusters Leverage Kyverno misconfigurations or supply chain flaws to disable image verification, enabling deployment of backdoored workloads Exploit spoke API server credentials stored in the hub cluster to gain persistent read access across all cluster API servers in a multi-cluster deployment

Data

To effectively secure a policy-driven admission control pipeline, organizations must first classify the sensitivity of the artifacts and credentials that govern its operation. We provide a comprehensive risk assessment of the Kyverno ecosystem, evaluating key assets against the Confidentiality, Integrity, and Availability (CIA) triad.

Contextual Factors
All assigned risk ratings are context-sensitive and may require adjustment based on deployment architecture, trust boundaries, regulatory requirements, and operational dependency on policy-based admission control. For instance, a loss of integrity in a production cluster poses a more severe impact than a similar compromise in a pre-production cluster. The ratings provided throughout this section assume:

- Moderate operational criticality (does not apply to safety-critical or national security systems)
- Standard Kubernetes RBAC and NetworkPolicy practices to protect relevant resources
- Formal hardware-backed external secrets management via HSM
- Default Kyverno architecture (without custom policy plugins)

Priority legend

The following colour codings are used to indicate the risk levels associated with each data type.

Colour	Risk
Red	High
Amber	Medium
Green	Low

Data dictionary

Kyverno leverages standard Kubernetes manifests, which can be stored in a number of locations. The following table enumerates each data type, providing notes on how they are typically used and stored, alongside colour coded typical confidentiality, integrity, and availability requirements for quick reference.

Data type	Notes	Confidentiality	Integrity	Availability
Spoke Cluster API Server Credentials	Credentials stored in the hub cluster's `policy-reporter` namespace, used by Policy Reporter to authenticate against spoke cluster API servers for `PolicyReport` and `ClusterPolicyReport` aggregation	High	High	High
Policy Signing Keys	Cosign private keys used to sign Kyverno policy bundles before storage in the external registry. Typically stored in an external HSM or Vault instance, referenced from the signing namespace	High	High	High
Kyverno controller TLS certificates and secrets	TLS certificates and keys automatically generated by Kyverno for securing webhook communication with the Kubernetes API server. Stored as Kubernetes `Secrets` in the `kyverno` namespace	High	High	High
`ValidatingWebhookConfiguration` and `MutatingWebhookConfiguration` CRs	Native Kubernetes resources in the `admissionregistration.k8s.io` API group, created and managed by Kyverno, that define which API requests are intercepted and forwarded to Kyverno's admission controller for policy evaluation They control the scope of interception (resource types, operations), the webhook endpoint, and crucially the `failurePolicy` behaviour if Kyverno is unreachable. These resources cannot be protected by Kyverno's own admission policies and must be protected via Kubernetes RBAC. Tampering can redirect, narrow, or entirely disable admission control	Low	High	High

`ClusterPolicy` and `Policy` resources	Policy definitions governing validation, mutation, generation, image verification, and cleanup behaviour across the cluster or within specific namespaces	Low	High	High
`PolicyException` resources	Resources that exempt specific workloads from matching policy rules. Overly broad or unreviewed exceptions silently weaken enforcement posture without modifying policies themselves	Low	High	Medium
`ImageValidatingPolicy` resources	CEL-based policy resources introduced in v1.17 governing image signature and attestation verification at admission time. Modification can silently disable supply chain controls	Low	High	High
Kyverno policy bundles	Signed artifacts containing `ClusterPolicy` and related resources, stored in the external registry and retrieved by platform tooling for distribution to clusters. Integrity depends on signing and verification controls in the distribution pipeline	Low	High	Medium
`PolicyReport` and `ClusterPolicyReport` resources	Structured compliance report objects generated by the reports controller. Contain detailed pass/fail results for all evaluated policies across namespaced and cluster-scoped resources. Useful reconnaissance material if exposed to unauthorised principals	Medium	Medium	Medium
Controller logs and metrics	Operational logs, events, and Prometheus metrics generated by Kyverno controller pods. May contain admission request details, resource metadata, policy evaluation results, and image reference information	Medium	Low	Low
`ConfigMap` configuration objects	Kyverno runtime configuration stored in `ConfigMaps` within `kyverno`, including resource filters, webhook settings, and controller behaviour flags.	Low	Medium	Medium

Example Architectures

This section presents two representative reference architectures for Kyverno deployment, designed to frame the threat modelling and hardening guidance that follows.

1. The Single-cluster architecture is presented first as a simple and widely-applicable baseline
2. The Multi-cluster hub-and-spoke architecture follows, representing a mature enterprise deployment pattern suitable for organisations with environmental separation or multi-region operational requirements

Both architectures incorporate intentional misconfigurations designed to reflect realistic operational decisions rather than obviously insecure configurations. Neither should be treated as a production blueprint. Significant flaws are included throughout to motivate the subsequent threat model and following recommendations.

External components

Both architectures utilise the following components:

External registry

The external registry is deployed outside Kubernetes entirely, representing a typical enterprise deployment using a product such as Harbor, Artifactory, a managed cloud registry service, or a combination of a Git repository and container registry. This external registry typically stores Kyverno policy bundles, application container images, cosign signatures, and attestations.

Depending on the registry technology in use, policy distribution may follow different mechanisms. An OCI-compatible registry such as Harbor or ECR supports cosign signature storage alongside artifacts natively. A Git-based policy store such as GitLab requires a pipeline-mediated distribution approach, with policies pulled as YAML and applied to clusters via `kubectl`, rather than as signed OCI artifacts. In either case, since signing and verification is typically not enabled by default, thus explicit enforcement of security controls which ensure the integrity of both container images and Kyverno policies is required.

Potential misconfigurations

The usage of a single shared registry for both policy bundles and application images introduces a meaningful supply chain risk. Specifically, a principal with registry write access could tamper with both the images being verified and the policies performing that verification. Repository-level RBAC within the registry should therefore separate the principals authorised to write policy bundles from those authorised to push application images.

External HSM

Cosign private keys used to sign Kyverno policy bundles are stored in an external HSM rather than as Kubernetes `Secrets`. The `platform-ops` namespace holds a signing key reference used to authenticate against the HSM at signing time. Organisations without HSM infrastructure commonly store signing keys as Kubernetes `Secrets`, which would require encrypting these secrets at rest and careful least privilege RBAC controls to reduce the risk of key exfiltration.

Actors

The following actors are referenced throughout both example architectures and the subsequent threat model.

Security / platform SRE team: Highly privileged administrators responsible for Kyverno policy authoring, signing, and distribution. In the multi-cluster architecture, this is the only team granted access to the hub cluster. In the single-cluster architecture, they operate within the `platform-ops` namespace.

Application developers: Authenticated Kubernetes users with namespace-scoped access to one or more tenant namespaces. Access is granted via `RoleBinding` and does not extend to cluster-scoped resources or the `kyverno` namespace.

External application users: Unauthenticated or externally authenticated users accessing application workloads via ingress. They have no direct Kubernetes API access.

CI pipeline principal: Non-human identities used by CI/CD pipelines to push application images, cosign signatures, and attestations to the external registry. These principals have registry write access but should not have direct Kubernetes API access. In architectures where policy distribution is pipeline-mediated, a separate pipeline principal with cluster write access applies `ClusterPolicy` resources to spoke clusters.

Single-cluster architecture

Overview

A single Kubernetes cluster hosting both application workloads and Kyverno policy infrastructure represents a common starting point for Kyverno adoption. This type of deployment typically reflects the experience of practitioners who have installed Kyverno via the default Helm chart and are working toward a hardened posture.

Our example single-cluster architecture uses a soft multi-tenant model in which each application development team is assigned a dedicated namespace, with sufficient access scoped via `RoleBinding` resources. A combination of namespaced `Policy` resources and cluster-wide `ClusterPolicy` resources, both authored and managed by the platform operations team, are used to apply Kyverno policy enforcement across tenant namespaces.

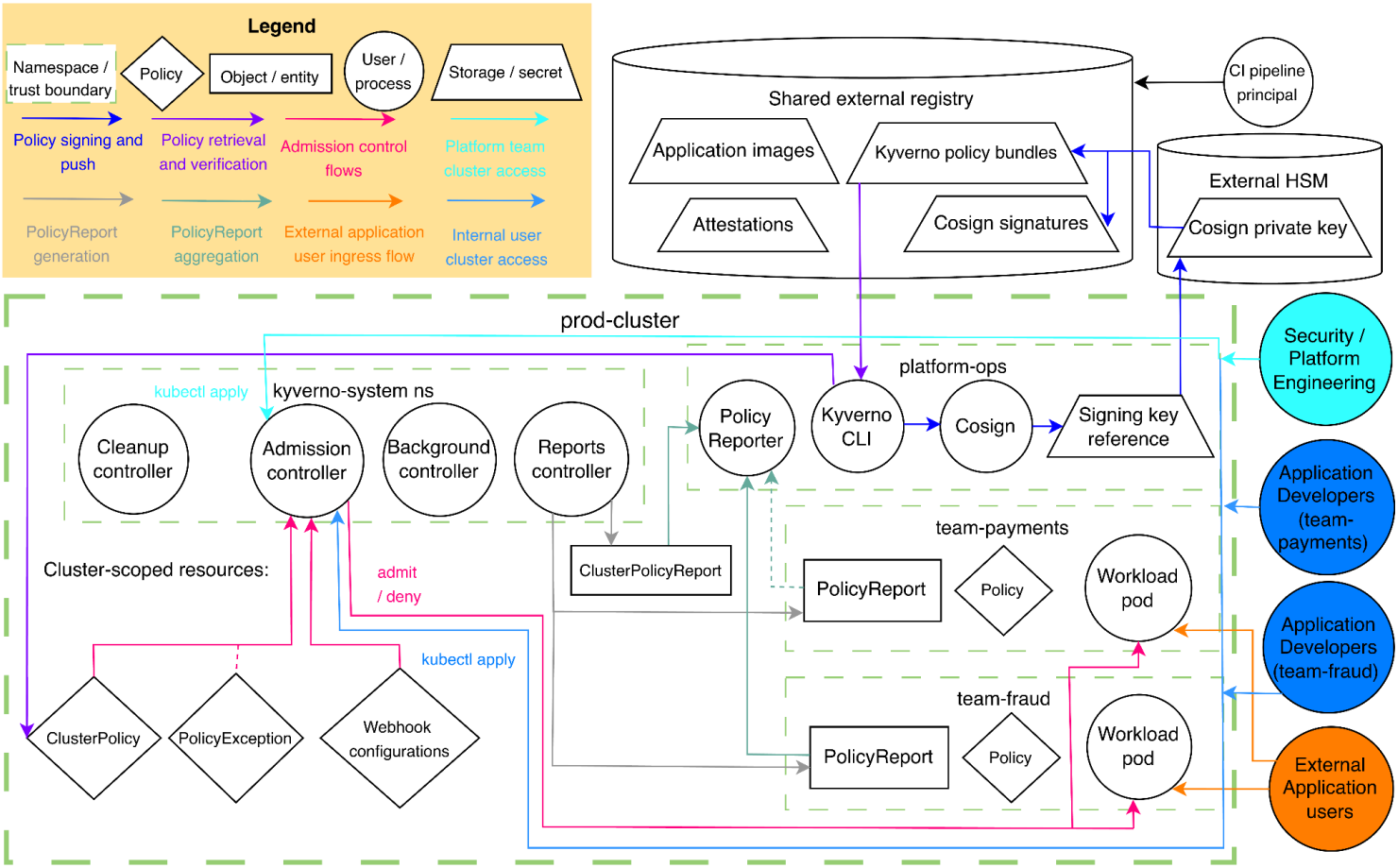
Benefits

- Simplified operational model (one control plane, one Kyverno deployment, one set of policies and webhook configurations etc to manage)
- Policy Reporter watches the local API server directly, removing spoke registration complexity
- Lower infrastructure cost (no cross-cluster networking, credential management, or Policy Reporter multi-cluster configuration required)

Drawbacks

- No environmental isolation at cluster level, meaning a misconfigured `ClusterPolicy` or `PolicyException` affects all tenants simultaneously
- More significant blast radius associated with a Kyverno-related compromise, for example a single webhook or `ClusterPolicy` misconfiguration has immediate cluster-wide impact
- No natural separation between policy authoring and policy enforcement environments, as the platform operations team works in the same cluster as application workloads
- Policy authoring tooling co-located with application workloads in the `platform-ops` namespace, meaning a compromised workload with excessive RBAC could potentially interact with signing or distribution tooling

Data flow diagram



Cluster-scoped resources

The following resources are cluster-scoped and represent the primary policy enforcement configuration. They are among the highest-value targets for unauthorised modification.

`ValidatingWebhookConfiguration` and `MutatingWebhookConfiguration` resources are registered and managed dynamically by Kyverno based on the installed policies. With no policies installed, the webhook configurations are empty and no admission requests are forwarded to Kyverno. Critically, these resources cannot be protected by Kyverno's own admission policies, native Kubernetes RBAC is the primary security control against unauthorised webhook modification.

`ClusterPolicy` resources define validation, mutation, generation, image verification, and cleanup rules applied across all namespaces. The `failureAction` field on each policy rule controls whether violations block the admission request (`Enforce`) or are recorded without blocking (`Audit`). The newer CEL-based policy types introduced in v1.17, `ValidatingPolicy`, `MutatingPolicy`, `GeneratingPolicy`, and `ImageValidatingPolicy`, provide equivalent functionality. The latter provides CEL-based image signature and attestation verification as a dedicated resource, rather than a rule within a `ClusterPolicy`. It is evaluated by the admission controller at admission time, requiring outbound connectivity to the external registry to retrieve and verify cosign signatures or attestations.

`PolicyException` resources allow specific workloads to bypass matching policy rules without modifying the policies themselves. In a well-governed deployment, exceptions are narrowly scoped, time-limited, and subject to regular review. In practice they tend to accumulate, broaden in scope, and persist indefinitely.

Potential misconfigurations

- `ClusterPolicy` resources configured with `failureAction: Audit` cluster-wide, no workloads are blocked regardless of policy violations. A further misconfiguration uses `failureActionOverrides` to audit specific tenant namespaces while enforcing others, reflecting a gradual rollout that was never completed
- `PolicyException` resources with overly broad scope, silently exempting sensitive workloads from enforcement. Insufficient RBAC on `PolicyException` resources means application developers can create exceptions without platform team review
- `ValidatingWebhookConfiguration` and `MutatingWebhookConfiguration` insufficiently protected by Kubernetes RBAC, allowing principals beyond the platform operations team to modify webhook routing

`kyverno` namespace

A default Kyverno installation includes deployments for each of the four controllers within the `kyverno` namespace:

- **Admission controller**: handles webhook callbacks from the Kubernetes API server, evaluates resources against configured policies, and performs image verification via cosign or Sigstore at admission time
- **Background controller**: periodically scans existing cluster resources against configured policies, generating evaluation results for resources that predate a policy or were not subject to admission control
- **Reports controller**: generates and maintains `PolicyReport` (namespace-scoped) and `ClusterPolicyReport` (cluster-scoped) resources based on policy evaluation results from both admission and background scanning
- **Cleanup controller**: deletes resources on a schedule based on criteria defined in `CleanupPolicy` resources

However, by default each controller is only deployed using a single replica, though both horizontal and vertical scaling can be implemented to improve availability.

Potential misconfigurations

- No `NetworkPolicy` resources are installed by default, leaving the `kyverno` namespace unprotected from lateral movement originating in tenant namespaces
- `failurePolicy: Ignore` on `ValidatingWebhookConfiguration`, therefore if Kyverno is unavailable or unresponsive, all admission requests are permitted without evaluation. This makes Kyverno availability a security control
- Insufficient `NetworkPolicy` resources isolating `kyverno`, thus a compromised workload pod could communicate directly with Kyverno controllers
- Overly permissive RBAC on Kyverno service accounts and sensitive resources including `ValidatingWebhookConfiguration`, `ClusterPolicy`, and `PolicyException`
- Image verification not configured, meaning unsigned or tampered container images could be admitted without verification

platform-ops namespace

The `platform-ops` namespace consolidates policy authoring, signing, and reporting tooling. It contains:

- **Cosign**: used to sign policy bundles prior to storage in the external registry
- **Signing key reference**: authenticates against the external HSM to retrieve private key material at signing time

- **Kyverno CLI**: used by platform engineers to retrieve policy bundles from the external registry and apply them to the cluster via `kubectl apply`
- **Policy Reporter**: aggregates `PolicyReport` and `ClusterPolicyReport` resources from the local cluster's API server, providing compliance visibility across namespaces

Intentional misconfigurations

Insufficient RBAC on `platform-ops` could provide application developers broader access to platform-level resources than intended, potentially enabling malicious interactions with signing or distribution tooling.

Tenant namespaces

The `team-payments` and `team-fraud` namespaces each contain application workload pods and a `PolicyReport` resource generated by the reports controller. Developer access is scoped to each team's own namespace via `RoleBinding` resources. Namespace-scoped `Policy` resources may be authored by teams to supplement cluster-wide enforcement. These policies are evaluated independently alongside ClusterPolicy resources (applied additively) and are limited to their namespace scope. They cannot directly override enforcing `ClusterPolicy` rules but may influence outcomes depending on policy configuration and evaluation context.

Multi-cluster hub-and-spoke architecture

Overview

The following architecture represents a mature enterprise deployment pattern for Kyverno in multi-cluster environments, suitable for organisations with strict requirements around environmental separation, data residency compliance, or blast radius reduction. It introduces a dedicated hub cluster for policy authoring and distribution, with multiple spoke clusters hosting application workloads. In a typical deployment, spoke clusters are provisioned per region and environment (for example `eu-dev`, `eu-prod`, `us-dev`, `us-prod`)), each operating an independent Kubernetes control plane while centralising policy distribution and compliance reporting through the hub.

This hub-and-spoke model is intended as a demonstrative example rather than a prescriptive reference architecture. Many organisations deploy Kyverno without multi-cluster configurations, and those that do often adopt different topologies. It has been selected to illustrate the breadth of flexibility Kyverno can offer at scale, and to surface the security risks that this flexibility can introduce.

Each spoke runs its own autonomous Kyverno deployment. Policy enforcement is entirely local, there is no central Kyverno component that governs spoke behaviour at runtime. The hub's role is limited to policy authoring, signing, distribution, and aggregation of compliance reports from registered spokes.

Benefits

- Environmental isolation at cluster level, meaning a compromise or misconfiguration in one spoke does not directly affect other spokes or the hub
- Reduced blast radius, as `ClusterPolicy` and `PolicyException` misconfigurations are contained within a single cluster
- Clean separation between policy authoring and enforcement, as the hub cluster is inaccessible to application teams and dedicated to platform operations
- Supports realistic illustration of the full policy promotion workflow (from authoring on the hub, through signed storage in the external registry, to retrieval and enforcement on the spoke clusters)
- Maps naturally to organisations with data residency requirements, regulatory mandates for environment separation, or multi-region Kubernetes deployments
- Centralised compliance visibility via Policy Reporter aggregating `PolicyReport` and `ClusterPolicyReport` resources across all registered spokes

Drawbacks

- Significantly greater operational complexity (multiple control planes, per-spoke Kyverno deployments, cross-cluster networking, and Policy Reporter spoke registration to manage)
- The `policy-reporter` namespace on the hub concentrates spoke API server credentials in a single location, meaning hub compromise yields authenticated read access to all spoke API servers simultaneously
- Policy drift between autonomous spoke deployments can accumulate over time without centralised enforcement of Kyverno configuration itself
- `PolicyException` sprawl is harder to audit fleet-wide without deliberate aggregation
- Policies developed and tested against a misconfigured lower environment may carry incorrect assumptions when promoted to production, and without explicit review gates this is a realistic source of enforcement gaps
- Higher infrastructure cost, requiring dedicated control plane nodes per spoke cluster

Topology

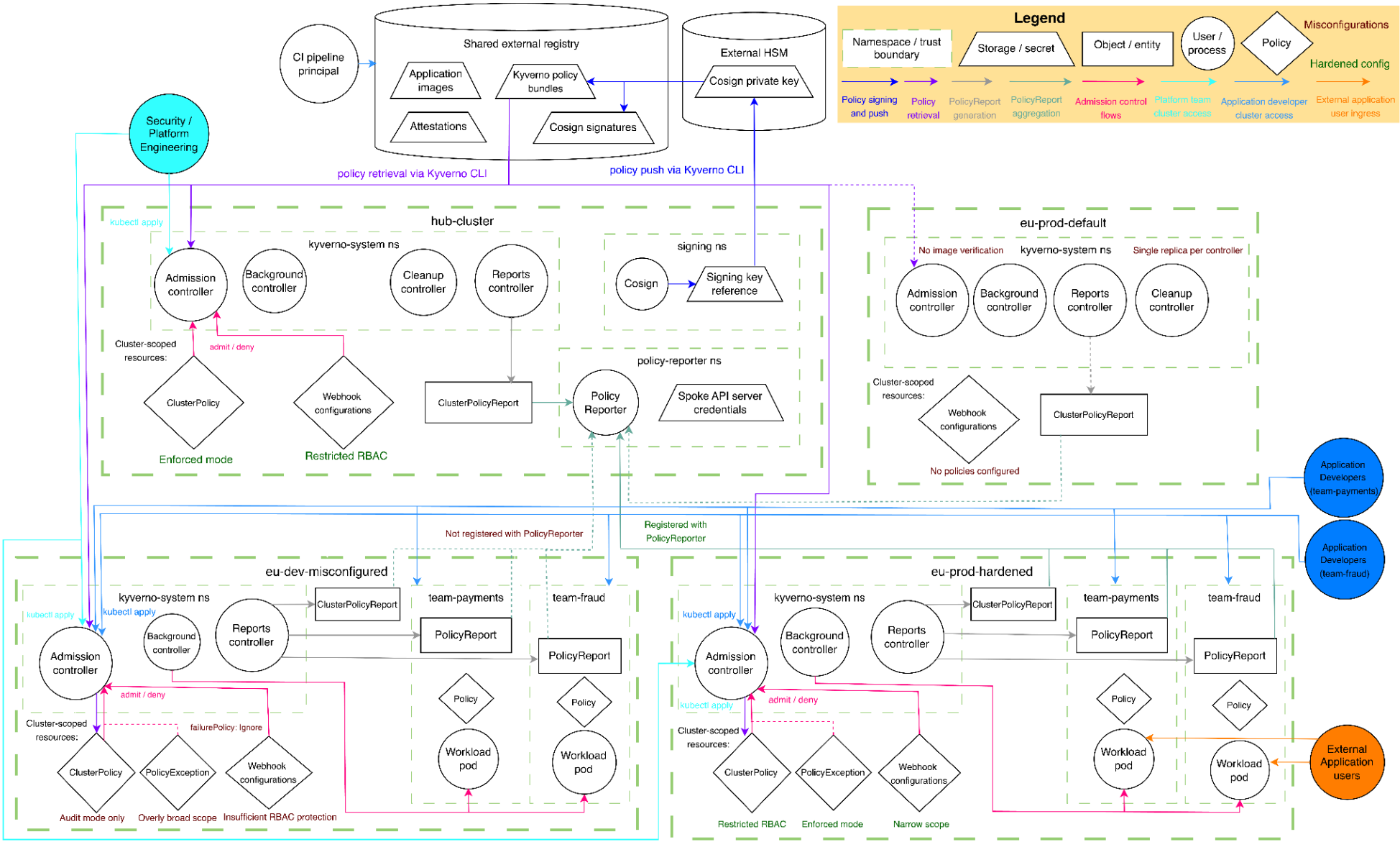
The example multi-cluster architecture consists of:

- One hub cluster, accessed exclusively by the platform SRE team
- Three spoke clusters: `eu-prod-hardened`, `eu-dev-misconfigured`, and `eu-prod-default`

- One shared external registry and external HSM, as described in the previous Shared external components section

Each spoke uses the same soft multi-tenant namespace model as the single-cluster architecture, with `team-payments` and `team-fraud` namespaces representing the same application teams across environments.

Data flow diagram



Hub cluster

The hub cluster is a highly restricted administrative zone, accessed exclusively by platform SRE team members. It is intended for platform-level operations including Kyverno policy authoring, testing, signing, and report aggregation, and should not be used to host application workloads.

Cluster-scoped resources

- `ValidatingWebhookConfiguration` and `MutatingWebhookConfiguration` present and actively enforcing admission policies on hub workloads
- `ClusterPolicy` resources configured with `failureAction: Enforce`
- `PolicyException` resources narrowly scoped and subject to regular review
- Restricted RBAC on all cluster-scoped resources, preventing modification by non-SRE principals

`kyverno` namespace

- All four Kyverno controllers deployed
- The hub is not exempt from Kyverno enforcement — the policy authoring environment itself should be subject to the same controls applied to spoke clusters

`signing/tooling` namespace

- Cosign for signing policy bundles prior to storage in the external registry
- Signing key reference authenticating against the external HSM to retrieve private key material at signing time
- Kyverno CLI for retrieving policy bundles from the external registry and applying them to spoke clusters via authenticated API server access

`policy-reporter` namespace

- Policy Reporter configured to watch `PolicyReport` and `ClusterPolicyReport` resources across all registered spoke clusters
- Spoke API server credentials stored as Kubernetes Secrets, used to authenticate against each spoke's API server

Intentional misconfigurations:

- Insufficient RBAC on the `policy-reporter` namespace, compromise of this namespace yields authenticated read access to every registered spoke cluster's API server simultaneously
- Policy bundles are pushed to the external registry without cosign signatures, and spoke clusters are not configured to verify bundle integrity on retrieval

- Insufficient review gates before promoting policies developed on lower environments to production spokes

`eu-prod-hardened` spoke

Production cluster with all hardening recommendations applied, representing the target posture this guide is designed to achieve.

Cluster-scoped resources

- `ValidatingWebhookConfiguration` and `MutatingWebhookConfiguration` configured with `failurePolicy: Fail`, ensuring admission requests are denied rather than permitted if Kyverno is unavailable
- `ClusterPolicy` resources configured with `failureAction: Enforce`
- `PolicyException` resources narrowly scoped and subject to regular review
- RBAC on all cluster-scoped resources restricted to platform SRE team members, preventing modification by application developers or CI pipeline principals
- `ClusterPolicyReport` registered with hub Policy Reporter for centralised compliance visibility

`kyverno` namespace

- All four controllers deployed with three replicas each for high availability
- `NetworkPolicy` resources isolating `kyverno` from tenant namespaces
- Restricted RBAC on Kyverno service accounts, scoped to the minimum permissions required per controller
- Image verification configured via `ImageValidatingPolicy`, requiring valid cosign signatures on all workload images at admission time
- Reports controller generating `PolicyReport` resources in each tenant namespace and `ClusterPolicyReport` at cluster scope, all registered with hub Policy Reporter

Tenant namespaces (`team-payments`, `team-fraud`)

- Application workload pods subject to cluster-wide `ClusterPolicy` enforcement
- `PolicyReport` resources generated by the reports controller and registered with hub Policy Reporter
- Developer access scoped to each namespace via `RoleBinding`

`eu-dev-misconfigured` spoke cluster

Development environment with accumulated realistic misconfigurations. Structurally identical to the hardened spoke but affected by several demonstrative Kyverno misconfigurations.

Cluster-scoped resources

- `ClusterPolicy` resources configured with `failureAction: Audit` cluster-wide, meaning no workloads are blocked regardless of policy violations
- Insufficient RBAC on `ValidatingWebhookConfiguration`, `ClusterPolicy`, and `PolicyException` resources, meaning principals beyond the platform SRE team can modify enforcement configuration
- `ClusterPolicyReport` not registered with hub Policy Reporter, making this cluster invisible to centralised compliance monitoring
- `ValidatingWebhookConfiguration` and `MutatingWebhookConfiguration` configured with `failurePolicy: Ignore`, meaning Kyverno unavailability results in all admission requests being permitted without evaluation
- `PolicyException` resources with overly broad scope, silently exempting sensitive workloads from enforcement

`kyverno` namespace

- Cleanup controller not deployed, meaning stale resources accumulate without policy-driven removal
- Single replica deployed for the other three Kyverno controllers, exposing these critical components to potential disruptions in availability
- Policies retrieved from the external registry without signature verification, meaning a tampered or substituted bundle would be applied without detection
- No `NetworkPolicy` isolating `kyverno` from tenant namespaces
- Overly permissive RBAC on Kyverno service accounts, granting excess cluster-level permissions beyond those required for each controller's function

Tenant namespaces (`team-payments`, `team-fraud`)

- Application workload pods subject to audit-only `ClusterPolicy` enforcement, meaning non-compliant workloads are admitted without restriction
- `PolicyReport` resources generated locally but not registered with hub Policy Reporter, meaning per-namespace compliance data is not visible centrally

An additional key threat illustrated by this cluster is the promotion of policies developed and tested against its lax enforcement posture to the hardened production cluster. Without explicit review gates in the distribution pipeline, this can be a realistic source of production enforcement gaps.

`eu-prod-default` spoke cluster

Default Kyverno installation using the standard Helm chart values with no additional configuration, representative of organisations that have installed Kyverno but not yet begun the hardening process.

Cluster-scoped resources

- `ValidatingWebhookConfiguration` and `MutatingWebhookConfiguration` present but dynamically managed by Kyverno — with no policies installed these remain empty and no admission requests are evaluated by default
- No `ClusterPolicy` resources installed — the Kyverno engine is deployed but not enforcing anything
- No `PolicyException` resources
- Default RBAC on cluster-scoped resources is considered overly permissive by the community, with Kyverno ClusterRoles aggregated to standard Kubernetes user-facing roles by default
- `ClusterPolicyReport` generated by the reports controller but not registered with hub Policy Reporter by default — explicit configuration of spoke API server credentials on the hub is required

`kyverno` namespace

- All four controllers deployed with a single replica each, providing no high availability
- `kyverno` excluded from webhook processing by default, meaning Kyverno does not enforce policies on its own namespace
- No `NetworkPolicy` resources isolating `kyverno` from tenant namespaces
- No image verification configured
- TLS certificates for webhook communication are self-signed and auto-generated, expiring after one year
- Default RBAC on Kyverno service accounts is considered overly permissive by the community, particularly for the background controller which holds broad view access to namespaced resources across the cluster
- If `--autoUpdateWebhooks=false` is set, Kyverno creates webhook configurations with `failurePolicy: Ignore` for all resources by default

If the optional `kyverno-policies` Helm chart is installed alongside the core chart, a set of Pod Security Standards baseline policies becomes available. These default to `failureAction: Audit`, providing violation visibility without blocking non-compliant workloads.

Tenant namespaces

No tenant namespaces are provisioned as part of the default Helm installation. Application teams are responsible for creating their own namespaces, and no Kyverno policies are applied to workloads deployed within them until policies are explicitly authored and installed.

Key Threats and Recommendations

ControlPlane applied its threat modeling methodology to both example architectures, identifying 13 common threats that could compromise production Kyverno deployments. All findings are categorized by the associated operational domain and assigned standardized severity ratings. To provide industry-standard context, adversarial TTPs are mapped to the MITRE ATT&CK framework, while all remediation guidance is aligned with NIST SP 800-53 security controls.

Severity

The severity associated with each threat has been labelled throughout as follows:

Colour	Severity
Red	High
Amber	Medium
Green	Low

All severity ratings are context-dependent and reflect typical production environments. Thus, they should always be customised to a specific environment's regulatory requirements, data classifications, deployment architecture, trust boundaries, and reliance on policy-based admission controls, to ensure that remediation actions and mitigation priorities align with the actual risk exposure and impact in each environment.

Threat categorisation

- **Secrets and key management:** Threats related to the confidentiality, integrity, or availability of sensitive data used by Kyverno, including policy signing keys, spoke API server credentials, and controller TLS certificates
- **Policy integrity and supply chain:** Threats arising from the unauthorised modification, substitution, or unsigned distribution of Kyverno policy bundles, `ClusterPolicy` resources, or `PolicyException` resources - including risks introduced by the policy authoring and distribution pipeline
- **Admission control:** Threats arising from insecure or suboptimal configuration of Kyverno's admission controller, webhook configurations, or `failurePolicy` settings, which could result in policy enforcement being bypassed or disabled
- **Policy configuration:** Threats arising from misconfigured `ClusterPolicy`, `PolicyException`, or `ImageValidatingPolicy` resources, including overly permissive exceptions, audit-only enforcement, and missing image verification controls

- **Reporting and observability:** Threats arising from misconfigured or absent `PolicyReport` and `ClusterPolicyReport` aggregation, resulting in compliance blind spots or undetected policy violations
- **RBAC:** Threats stemming from overly permissive Role-Based Access Control assignments on Kyverno service accounts, cluster-scoped policy resources, or webhook configurations
- **Network:** Threats related to insufficient network isolation of the `kyverno` namespace, uncontrolled communication between workload pods and Kyverno controllers, or exposure of Kyverno components to other workloads
- **Kubernetes:** Threats associated with underlying Kubernetes platform or API misconfigurations that affect Kyverno security, including `ValidatingWebhookConfiguration` tampering and namespace exclusion behaviour
- **Default configuration:** Threats arising from default Kyverno behaviour as configured in the default Helm chart values. These values are designed for ease of installation and broad compatibility rather than production-grade security. The Kyverno documentation provides hardening guidance and recommended configuration values which should be applied where possible to align with organisational security standards

Findings

Each finding identified during the threat modelling process was assigned a unique ID and includes a description of the observed misconfiguration, along with potential mitigation strategies and recommendations. Interactions between several of these threats are illustrated using Attack Trees in a later section.

T-01: Admission control bypass via `failurePolicy: Ignore`

Affected architectures: Both

Severity	Relevant categories	MITRE ATT&CK mappings	NIST SP 800-53 mappings
High	Admission control	T1562.001: Impair Defenses: Disable or Modify Tools T1499.004: Endpoint Denial of Service	CM-6: Configuration Settings CM-7: Least Functionality

Impact: When `failurePolicy: Ignore` is configured on Kubernetes `ValidatingWebhookConfiguration` or `MutatingWebhookConfiguration` resources, any failure of the Kyverno admission controller to respond within the configured webhook timeout causes the Kubernetes API server to admit the request without policy evaluation. Kyverno availability therefore becomes a direct security control. An attacker who can disrupt Kyverno controller pods through resource exhaustion, denial of service, or a crash bug could bypass all admission enforcement for the duration of the outage. Workload creation requests during this window are admitted without validation, mutation, or image verification regardless of the configured policies - potentially resulting in unsigned or tampered container images being deployed to production.

Description: The `failurePolicy` field on `ValidatingWebhookConfiguration` and `MutatingWebhookConfiguration` resources controls Kubernetes API server behaviour when a webhook call fails to complete. This could occur when the webhook endpoint is unreachable, returns an error, or exceeds the configured timeout. When `failurePolicy` is set to `Ignore`, the API server admits the request as if the webhook were not configured. When set to `Fail`, the API server rejects the request.

The `failurePolicy: Ignore` configuration is frequently used in development environments or during initial Kyverno rollout to avoid cluster-wide admission failures caused by Kyverno downtime. In production environments where Kyverno is the primary enforcement mechanism, however, this setting creates a direct relationship between Kyverno availability and the cluster' security posture.

Two additional configuration paths can introduce this setting silently:

- Setting `--autoUpdateWebhooks=false` causes Kyverno to create webhook configurations with `failurePolicy: Ignore` for all resources by default
- Setting `--forceFailurePolicyIgnore` forces `Ignore` mode globally across all webhook configurations

Individual `ClusterPolicy` resources can also override this setting via `spec.webhookConfiguration.failurePolicy`. However, per-policy configuration does not protect against the global bypass paths described above.

The attack surface for triggering this bypass does not require compromise of Kyverno components directly. A workload in a tenant namespace with sufficient permissions to exhaust resources in `kyverno`, or an indirect failure of a Kyverno dependency, is sufficient to trigger the enforcement gap.

Recommendations:

- Set `failurePolicy: Fail` on all `ValidatingWebhookConfiguration` and `MutatingWebhookConfiguration` resources in production environments
- Deploy Kyverno with three replicas per controller to reduce the likelihood of complete admission controller unavailability
- Do not set `--forceFailurePolicyIgnore` in production environments
- Avoid setting `--autoUpdateWebhooks=false` unless the implications for `failurePolicy` defaults are explicitly understood and compensated for
- Review Kyverno's global `webhookTimeoutSeconds` (or `spec.webhookConfiguration.timeoutSeconds` per-policy) to ensure timeouts are not configured so low that typical operational latency triggers `failurePolicy: Ignore` behaviour
- Apply `NetworkPolicy` resources to isolate `kyverno` from tenant namespaces, reducing the risk of resource exhaustion or lateral movement from compromised workloads (see T-09: kyverno namespace exposure due to absent NetworkPolicy and lack of mutual authentication)
- Configure alerting on Kyverno controller pod availability to ensure enforcement gaps are detected promptly
- Periodically review all `ValidatingWebhookConfiguration` resources to verify `failurePolicy` settings have not drifted from the intended configuration

References:

- Kyverno documentation: Customization
- Kyverno documentation: High Availability
- Kyverno documentation: Installation Customization
- Kubernetes documentation: Webhook Configuration

T-02: Unsigned or unverified image admission due to missing or misconfigured image verification policy

Affected architectures: Both

Severity	Relevant categories	MITRE ATT&CK mapping	NIST SP 800-53 mapping
High	Policy configuration Default configuration	T1525: Implant Internal Image T1195.002: Supply Chain Compromise	SA-10: Developer Configuration Management SI-7: Software, Firmware, and Information Integrity

Impact: Where image verification policies are absent or configured in audit mode only, unsigned or tampered container images can be admitted without detection. An attacker who has compromised a container registry or CI pipeline can substitute a malicious image for a legitimate one, and it will be scheduled and run without any cryptographic verification of its provenance.

Description: Kyverno supports image signature and attestation verification through two mechanisms: `verifyImages` rules within a `ClusterPolicy`, and the dedicated `ImageValidatingPolicy` resource. Both verify that container images have been signed by a trusted key before admission and can enforce digest pinning to prevent tag-based substitution. Neither is configured by default.

This threat can arise when any of the following misconfiguration patterns, which result in gaps in image verification, are present:

- Where no image verification policies are enforced, images are admitted without any signature checks. Specifically, if no `verifyImages` rules or `ImageValidatingPolicy` resources cover a given image reference, then images from that source are admitted without a signature check. Gaps in `imageReferences` patterns, such as covering only a specific registry while leaving others uncovered, are a common source of incomplete coverage
- When policies containing `verifyImages` rules are configured with `failureAction: Audit`, this can result in the admission of non-compliant images. Teams that deploy image verification policies in audit mode during rollout and do not transition to `Enforce` may believe verification is active while only logging and no blocking is occurring
- When `failurePolicy: Ignore` is set on a policy containing `verifyImages` rules, failures to reach the signature registry are silently ignored and pods are admitted regardless. This is distinct from T-01: Admission control bypass via failurePolicy: Ignore, since even if Kyverno itself is healthy, verification can be skipped if the signature registry is unreachable
- When `required: false` is set on `verifyImages` rules, images with no signature are

admitted without error. This is sometimes used during initial rollout to development environments and left in place inadvertently
- The `skipImageReferences` field excludes matched image patterns from verification entirely. Patterns that are too broad can inadvertently exempt production images from verification

Recommendations:
- Deploy `verifyImages` rules or `ImageValidatingPolicy` resources covering all production registries and repositories, configure `failureAction: Enforce` on all policies intended to block unsigned images
- Set `required: true` on all `verifyImages` rules where possible, to ensure images without signatures are blocked rather than silently admitted
- Set `failurePolicy: Fail` on all policies containing `verifyImages` rules so that signature registry unavailability results in admission denial rather than silent bypass
- Review `imageReferences` and `skipImageReferences` patterns across all image verification policies to confirm no production image sources are inadvertently excluded
- Confirm `mutateDigest: true` is set on all `verifyImages` rules to enforce digest pinning at admission time and mitigate the related risk of tag mutability attacks
- Periodically audit image verification policy coverage against the full set of image sources in use, as new registries and repositories are introduced over time
- Consider migrating from `verifyImages` rules in `ClusterPolicy` to dedicated `ImageValidatingPolicy` resources for more granular policy management

References:
- Kyverno documentation - Verify Image

T-03: Policy supply chain compromise via unsigned policy distribution

Affected architectures: Both

Severity	Relevant categories	MITRE ATT&CK mappings	NIST SP 800-53 mappings
High	Policy integrity and supply chain Secrets and key management	T1195.002: Supply Chain Compromise: Compromise Software Supply Chain T1553.002: Subvert Trust Controls: Code Signing	SA-10: Developer Configuration Management SC-12: Cryptographic Key Establishment and Management

Impact: If Kyverno policy bundles are stored in an external registry without cosign signatures, or if clusters are not configured to verify signatures on retrieval, an attacker with registry write access can substitute or modify policy bundles that are subsequently applied to all consuming clusters. Attackers with no access to the cluster could tamper with policies and weaken enforcement rules, introduce permissive `PolicyException` entries, disable image verification, or set `failureAction` to `Audit` across the fleet. In a multi-cluster deployment without signature verification, a single malicious registry write operation targeting stored Kyverno policy bundles could affect all spoke clusters simultaneously.

Description: The security of a Kyverno policy distribution pipeline depends on two controls operating together: signing of policy bundles before storage, and verification of signatures before application to clusters. Without both controls in place, there is no assurance that policies applied to clusters haven't been tampered with.

In the reference architectures described in this document, policy bundles are signed using cosign with private keys stored in an external HSM, then pushed to the shared external registry. Clusters retrieve policy bundles and verify signatures before applying them. If the signing step is omitted, policy bundles are stored unsigned and the verification step provides no protection against substitution. Similarly, signing provides no value unless each cluster verifies bundles before applying them.

A related risk arises where a single registry is used for both policy bundles and application images. This is detailed in T-08: Conflated registry principals enabling simultaneous image and policy tampering.

Recommendations:

- Sign all policy bundles with cosign before storage in the external registry, using private keys stored in an HSM or equivalent hardware-backed key store
- Configure all clusters to verify policy bundle signatures using `cosign verify` before applying retrieved bundles, and reject unsigned or unverified bundles
- Establish a secure process for distributing the cosign public key to all consuming clusters, treating the public key itself as a trust anchor requiring integrity protection
- Rotate policy signing keys on a defined schedule and maintain audit logs of all key usage operations via the HSM
- Implement explicit review gates in the policy distribution pipeline requiring approval before policies are promoted from development to production environments
- Monitor the external registry for unexpected writes to policy bundle repositories and alert on unsigned artifact pushes

References:

- Kyverno documentation: Security
- Sigstore cosign documentation
- SLSA Supply Chain Levels for Software Artifacts

Related findings:

- T-08: Conflated registry principals enabling simultaneous image and policy tampering

T-04: Unauthorised modification of Kyverno webhook or policy resources

Affected architectures: Both

Severity	Relevant categories	MITRE ATT&CK mappings	NIST SP 800-53 mappings
High	RBAC Default configuration	T1562.001: Impair Defenses: Disable or Modify Tools T1098: Account Manipulation	AC-6: Least Privilege AU-2: Event Logging

Impact: Principals with overly permissive access to cluster-scoped Kyverno resources can compromise the deployment in several ways, for example:

- **Webhook configuration tampering:** A principal with write access to `ValidatingWebhookConfiguration` or `MutatingWebhookConfiguration` resources can modify the scope of admission interception, redirect webhook traffic to an attacker-controlled endpoint, reduce the configured timeout to trigger `failurePolicy: Ignore` behaviour (see T-01: Admission control bypass via failurePolicy: Ignore), or delete the configurations entirely. In each case, malicious users configured with overly privileged RBAC could disable admission enforcement without touching any Kyverno-managed resources
- **Policy resource modification:** A principal with write access to `ClusterPolicy` resources can weaken or remove policy rules, change `failureAction` from `Enforce` to `Audit`, or add exclusions that silently exempt sensitive workloads from enforcement. Changes of this kind may not surface in policy evaluation results immediately, making them difficult to detect without audit logging in place
- **Scheduled resource deletion:** A principal with write access to `ClusterCleanupPolicy` or `CleanupPolicy` resources can schedule deletion of arbitrary cluster resources on a defined interval. Unlike the above paths, this is not an admission bypass, it is a destructive availability threat that can silently remove workloads, configuration, or policy resources on a recurring basis

These attack paths do not require cluster-admin privileges, targeted write access to the relevant cluster-scoped resources is sufficient.

Description: `ValidatingWebhookConfiguration` and `MutatingWebhookConfiguration` are native Kubernetes resources in the `admissionregistration.k8s.io` API group, created and managed by Kyverno. These resources cannot be protected by Kyverno's own admission policies: Kubernetes does not forward webhook configuration changes to admission controllers, creating an inherent protection gap. Though Kubernetes-native `ValidatingAdmissionPolicy` resources can provide a

compensating layer, carefully configured least privilege RBAC is the most reliable control to mitigate this threat.

Kyverno uses an aggregated `ClusterRole` model, with each controller holding its own set of `ClusterRoles`. Granting Kyverno controllers additional permissions via this aggregation mechanism, or granting application teams with `edit` or `admin` `ClusterRoles` binding them to Kyverno-related roles may result in unintended write access to `ClusterPolicy` or `PolicyException` resources. Furthermore, modification of these resources may not be immediately visible in policy evaluation results, making changes difficult to detect without robust audit logging in place.

Kyverno's default aggregated `ClusterRoles` can be extended by creating new `ClusterRoles` with labels such as `rbac.kyverno.io/aggregate-to-admission-controller: "true"` or `rbac.kyverno.io/aggregate-to-background-controller: "true"`. Any `ClusterRole` carrying these labels automatically extends the corresponding controller's permissions, making it essential to audit which principals can create `ClusterRoles` with these labels, not just who holds existing Kyverno roles.

Recommendations:

- Restrict write access to `ValidatingWebhookConfiguration`, `MutatingWebhookConfiguration`, `ClusterPolicy`, `PolicyException`, `ImageValidatingPolicy`, `ClusterCleanupPolicy`, and `CleanupPolicy` resources to the platform SRE team only, via explicit RBAC
- Restrict `ClusterRole` create and update permissions to prevent privilege escalation via the aggregation mechanism. Audit all `ClusterRoles` carrying Kyverno aggregation labels (such as `rbac.kyverno.io/aggregate-to-admission-controller: "true"`) to verify no unintended rules are being pulled into Kyverno controller permissions
- Enable Kubernetes API server audit logging and configure alerts on create, update, and delete operations against webhook configuration, policy resources, `ClusterRole` resources, and cleanup policy resources. For `ClusterCleanupPolicy` and `CleanupPolicy` resources specifically, alert on any create or update events outside of approved platform team activity, as scheduled deletion rules may not trigger observable activity until the deletion interval fires
- Regularly review RBAC bindings for CI pipeline principals and application developer service accounts to confirm no unintended access to cluster-scoped policy resources exists
- Implement `ValidatingAdmissionPolicy` at the Kubernetes level, independent of Kyverno, to provide an additional enforcement layer protecting critical resource modifications

References:

- Kyverno documentation: Security
- Kubernetes documentation: Using Admission Controllers
- Kubernetes documentation: Audit Logging

Related threats:

- T-01 - Admission control bypass via failurePolicy: Ignore
- T-05: Enforcement bypass via misconfigured PolicyException resources

T-05: Enforcement bypass via misconfigured `PolicyException` resources

Affected architectures: Both

Severity	Relevant categories	MITRE ATT&CK mapping	NIST SP 800-53 mapping
High	Policy configuration RBAC	T1562.001: Impair Defenses: Disable or Modify Tools T1610: Deploy Container	AC-3: Access Enforcement AC-6: Least Privilege

Impact: An attacker with write access to `PolicyException` resources, or one exploiting an overly permissive exception already present in the cluster, can deploy non-compliant workloads without generating policy violation records. This includes workloads with privileged containers, host path mounts, or unverified images. When a `PolicyException` matches, Kyverno records a skip result in `PolicyReport` rather than a fail. If `reportResult: pass` is set, the exempted resource will appear as passing rather than simply absent from violation reports. In either case, exempted resources do not surface as violations in compliance monitoring.

Description: `PolicyException` is a namespace-scoped resource that provides declarative exemptions from policy rules without modifying the policies themselves. It is disabled by default and must be explicitly enabled via the `enablePolicyException` flag. When enabling `PolicyExceptions`, `exceptionNamespace` must also be set to restrict exception creation to a single controlled namespace. If omitted, `PolicyException` resources from all namespaces will be evaluated, significantly widening the attack surface.

Where `exceptionNamespace` is not tightly scoped, namespace administrators or developers with edit rights in permitted namespaces may be able to create exceptions without platform team review. Exceptions can be defined to match by resource kind, namespace, name, or label selector, and the scope of existing exceptions tends to expand incrementally as teams add new resources without reviewing what is already exempted.

By default, `PolicyException` resources also apply during background scanning. When an exception applies to background scanning, either by explicitly setting `spec.background: true` or leaving the field unset, subsequent scan runs produce a skip result rather than a fail for matching resources in `PolicyReport` output. This means a newly created exception can cause previously reported violations to disappear from compliance reports on the next scan cycle, without any change to the underlying resource.

CVE-2024-48921 demonstrated that without namespace scoping, a user with exception creation rights in any namespace could create a `PolicyException` that bypassed `ClusterPolicies`

in namespaces they did not own, enabling privilege escalation. This motivates the requirement to always set `exceptionNamespace` explicitly when enabling this feature.

Mitigating factors: The `PolicyException` feature must be explicitly enabled via the `enablePolicyException` flag on the Kyverno admission controller. If this flag is not set, any `PolicyException` resources present in the cluster have no effect, and organisations that have not enabled this feature are not exposed to this threat.

Recommendations:

- Restrict write access to `PolicyException` resources to the platform SRE team via explicit RBAC. Namespace administrators and developers should not be able to create exceptions independently
- Set `exceptionNamespace` to a single tightly controlled namespace when enabling PolicyExceptions, ensuring exceptions cannot be created outside platform team oversight
- Scope all `PolicyException` resources as narrowly as possible, for example to specific resource names, namespaces, and policy rules rather than broad selectors
- Implement a mandatory review and approval process for all `PolicyException` creation and modification before exceptions take effect
- Implement time-limited exceptions where possible, with a defined expiry and regular review cadence to remove stale exceptions
- Monitor `PolicyException` resource creation and modification via Kubernetes audit logs and alert on unexpected changes
- Periodically audit all `PolicyException` resources in each cluster for scope creep and continued justification

References:

- Kyverno documentation: Policy
- Exceptions Kyverno documentation: Security

T-06: Hub cluster compromise enabling fleet-wide policy distribution takeover

Affected architectures: Multi-cluster only

Severity	Relevant categories	MITRE ATT&CK mapping	NIST SP 800-53 mapping
High	Policy integrity Secrets management	T1195.002: Supply Chain Compromise: Compromise Software Supply Chain T1552.007: Unsecured Credentials: Container API	SC-12: Cryptographic Key Establishment and Management AC-6: Least Privilege

Impact: In the multi-cluster hub-and-spoke architecture, the hub cluster concentrates three high-value capabilities: policy signing infrastructure with access to HSM-backed cosign keys, spoke API server credentials, and the policy distribution pipeline for the entire fleet. A sufficiently privileged hub compromise therefore yields simultaneous control over policy distribution, read access to all spoke API servers, and the ability to produce legitimately signed tampered bundles that spoke clusters will trust. This represents a higher impact than compromising any individual spoke.

Description: The signing key reference in the `signing/tooling` namespace authenticates against an external HSM. A principal with sufficient access to the hub can invoke the signing pipeline to produce valid signed policy bundles containing arbitrary content, without extracting the underlying private key from the HSM. Key exfiltration and key misuse are distinct threats: HSM storage prevents the former but does not prevent the latter where the signing environment itself is compromised.

The `policy-reporter` namespace stores Kubernetes Secret objects containing credentials for every registered spoke cluster's API server. These credentials are stored in etcd and are accessible to any principal with Secret read access in that namespace. Compromise of this namespace does not require HSM interaction and directly yields authenticated API server access to all registered spokes.

If registry administrative credentials are stored on the hub, a hub compromise may also yield control over the policy distribution store itself.

Mitigating factors:

- Whether `etcd` encryption at rest is enabled on the hub cluster significantly affects the risk of direct credential exfiltration. By default, Kubernetes `Secrets` are stored as Base64-encoded plaintext in `etcd` and are not encrypted unless explicitly configured
- Where spoke clusters independently verify cosign signatures on retrieved policy bundles against a known public key, a hub compromise that stops short of the signing environment yields distribution access but not the ability to deploy trusted tampered bundles to correctly configured spokes

Recommendations:

- Treat the hub cluster as critical infrastructure and apply the most stringent access controls of any cluster in the fleet, including MFA on all human access, comprehensive audit logging, and regular access reviews
- Implement separate signing key references for development and production policy distribution pipelines, so a hub compromise affecting development infrastructure cannot be used to sign production-destined bundles
- Store spoke API server credentials in the `policy-reporter` namespace as short-lived tokens rather than long-lived static credentials where possible. Regularly rotate these credentials and audit which principals have Secret read access in that namespace
- Ensure `etcd` encryption at rest is enabled on the hub cluster, given the sensitivity of the spoke credentials stored there
- Ensure registry administrative credentials are not stored on the hub cluster. Registry operations should be independently operable from cluster access
- Monitor for anomalous invocations of the policy signing pipeline, including unexpected signing events or signing by principals other than the designated platform pipeline identity

References:

- Kyverno documentation: Security
- Kubernetes documentation: Secrets
- SLSA Supply Chain Levels for Software Artifacts

T-07: Insecure policy promotion from misconfigured environments

Affected architectures: Both

Severity	Relevant categories	MITRE ATT&CK mapping	NIST SP 800-53 mapping
Medium	Policy integrity and supply chain Policy configuration	T1195.002: Supply Chain Compromise: Compromise Software Supply Chain T1562.001: Impair Defenses: Disable or Modify Tools	SA-10: Developer Configuration Management, CM-3: Configuration Change Control

Impact: Policies authored and validated in a development environment running `failureAction: Audit` with broad `PolicyException` resources may contain assumptions that break in a production cluster running `failureAction: Enforce`. A policy that passes testing in audit mode may cause unexpected admission denials in production, for example where edge cases were not encountered during development. Conversely, a `PolicyException` created in development to unblock a team may be inadvertently included in a policy bundle promoted to production, silently weakening enforcement. In both cases the impact is deferred and may not be detected until a production incident or compliance review.

Description: The policy promotion workflow, from authoring in development through storage in the external registry to deployment on production clusters, introduces a trust chain between environments. If the enforcement posture of the development environment differs significantly from production, for example allowing permissive for audit-only policies and broad exceptions in development versus enforcing strict policies that actively block malicious deployments in production, then testing in development provides limited assurance about production behaviour.

This risk increases significantly if the pipeline lacks explicit review gates between environments. Without a mandatory review step that compares policy content and associated exceptions against the target environment's enforcement requirements, promoted policies may contain assumptions or exceptions that are appropriate for development but unsuitable for production.

A related organisational risk arises where development teams are granted permissions to create or request `PolicyException` resources for workloads they do not own or operate. In environments where exception creation is decentralise (for example where namespace administrators can raise exceptions directly without platform team review) exceptions may be scoped to resources outside the requesting team's remit, or created to resolve an immediate deployment blocker without adequate consideration of the security implications. This is compounded where exception requests are processed informally, without a documented approval workflow or time-bound scope, as exceptions created under time pressure during incidents can often persist long after the original justification has lapsed.

The `kyverno apply --cluster` command can be used to validate promoted policies against live cluster resources before switching enforcement from `Audit` to `Enforce`, surfacing admission denials against existing workloads ahead of time.

Recommendations:

- Test policies in a staging environment that mirrors the enforcement posture of production, with `failureAction: Enforce` and no broad exceptions, before promoting to production
- Implement explicit review and approval gates in the policy distribution pipeline before promotion to production. Maintain separate enforcement profiles for each environment and ensure policy bundles are audited for environment-specific `PolicyException` resources before promotion
- Ensure that `PolicyException` requests are raised and approved through a documented process that requires the requesting team to demonstrate ownership of the affected resources. Exceptions covering resources outside the requester's namespace or ownership scope should require explicit platform team sign-off and a defined expiry
- Use `kyverno apply --cluster` to validate promoted policies against live cluster resources before switching enforcement from `Audit` to `Enforce`, surfacing admission denials against existing workloads ahead of time
- Use `kyverno test` as part of the distribution pipeline to validate policy behaviour against known resources before promotion
- Periodically audit `PolicyException` resources in each environment for continued relevance and scope creep, independent of the promotion review process
- Consider using separate signing keys for development and production policy bundles to enforce a deliberate promotion step with human review

References:

- Kyverno documentation: Testing Policies
- Kyverno CLI documentation: kyverno test

Related findings:

- T-03: Policy supply chain compromise via unsigned policy distribution

T-08: Conflated registry principals enabling simultaneous image and policy tampering

Affected architectures: Both

Severity	Relevant categories	MITRE ATT&CK mapping	NIST SP 800-53 mapping
Medium	Policy integrity and supply chain Secrets and key management	T1195.002: Supply Chain Compromise: Compromise Software Supply Chain T1553.002: Subvert Trust Controls: Code Signing	AC-6: Least Privilege SA-10: Developer Configuration Management

Impact: Where a single registry principal has write access to both application image repositories and policy bundle repositories, a compromised CI pipeline can push a malicious container image and simultaneously modify the Kyverno policy that would have blocked it. The enforcement mechanism is neutralised at the same time the malicious image is deployed, with no detection gap between the two actions.

This threat severity rating assumes policy bundle signing and verification are in place, as detailed in T-03: Policy supply chain compromise via unsigned policy distribution. Where those controls are absent, the combined impact is more accurately rated as high severity.

Description: OCI registries such as Harbor and Artifactory support repository-level access control, but this separation requires deliberate configuration and is not enforced by default. In organisations using a single shared registry for application images and Kyverno policy bundles, the CI pipeline principal may have write access to policy bundle repositories as an unintentional side effect of being granted access to image repositories.

The risk arises from the circular trust relationship in this architecture: Kyverno uses policies stored in the registry to verify images also stored in the registry. If the same principal can write to both, the verification chain can be broken at the registry level without any cluster access. This attack requires only registry write access, which may be achievable by compromising CI pipeline credentials rather than Kubernetes credentials.

Mitigating factors: Where policy bundles are signed with cosign and clusters verify signatures before applying retrieved bundles, a compromised CI pipeline principal can overwrite policy bundle content but cannot produce a valid signature without access to the signing key. Signature verification therefore significantly limits the impact of this threat even where principal separation is incomplete.

Recommendations:

- Enforce repository-level RBAC within the shared registry, ensuring CI pipeline principals cannot write to policy bundle repositories
- Implement policy bundle signing and signature verification, as detailed in T-03: Policy supply chain compromise via unsigned policy distribution, ensuring that even if registry write access is compromised, tampered bundles are rejected on retrieval
- Use separate registry credentials for CI pipeline image pushes and platform team policy bundle pushes, with each scoped to its respective repository set
- Audit registry access control configuration regularly, verifying no single credential set has write access to both image and policy bundle repositories
- Consider storing policy bundles in a separate registry instance from application images if the registry does not support sufficiently granular repository-level access control
- Implement registry-side alerts on writes to policy bundle repositories from any principal other than the designated platform signing pipeline

References:

- Sigstore cosign documentation
- SLSA Supply Chain Levels for Software Artifacts
- Harbor documentation: Project Access Control

Related findings:

- T-03: Policy supply chain compromise via unsigned policy distribution

T-09: `kyverno` namespace exposure due to absent `NetworkPolicy` and lack of mutual authentication

Affected architectures: Both

Severity	Relevant categories	MITRE ATT&CK mapping	NIST SP 800-53 mapping
Medium	Network Default configuration	T1046: Network Service Discovery T1210: Exploitation of Remote Services	AC-4: Information Flow Enforcement SC-7: Boundary Protection

Impact: Without `NetworkPolicy` isolating `kyverno`, a compromised workload pod in a tenant namespace can reach Kyverno controller pods directly potentially accessing their unauthenticated APIs. Direct network access increases the attack surface available to a compromised workload and reduces the effort required to exploit any future vulnerability in controller endpoints.

Description: The default Kyverno Helm installation does not include `NetworkPolicy` resources for `kyverno`. Kyverno controllers expose webhook endpoints (such as `kyverno-svc.kyverno.svc.cluster.local`) and metrics endpoints on port 8000. By default, both of these services are reachable from tenant namespaces without network controls in place.

Kyverno controllers expose unauthenticated webhook handlers which are secured using TLS; however this only provides assurance of the identity of the Kyvero server not the client calling it, which could be the Kube API server or an attacker.

The network attack surface of these controllers is fairly limited, and mitigated by the presence of any network policies restricting access to the `kyverno` namespace. However, network connectivity to the admission controller could allow an attacker to bypass Kubernetes RBAC controls and auditing by querying Kyverno directly. This could be used to extract secrets injected into mutations or leak policy information without direct permissions to view these policies.

The absence of `NetworkPolicy` does not directly enable policy bypass. However, it means a compromised tenant workload has broader network access to the policy enforcement infrastructure than necessary.

Mitigating factors: In clusters where a CNI plugin capable of enforcing `NetworkPolicy` is not installed, this control is not available regardless of whether the resources are created. The threat is also reduced where Pod Security Admission controls or equivalent controls are enforced on tenant namespaces, limiting the capabilities available to workload containers.

Recommendations:

- Deploy `NetworkPolicy` resources in `kyverno` to restrict ingress to only the Kubernetes API server for webhook callbacks and authorised monitoring systems for metrics scraping on port 8000
- Restrict egress from `kyverno` to only the Kubernetes API server, any configured external service endpoints, and where image verification is configured, the external OCI registry
- Apply equivalent network isolation to any namespace containing policy authoring or distribution tooling, such as `platform-ops` in the single-cluster architecture
- Validate `NetworkPolicy` effectiveness using network policy testing tools to confirm that tenant workloads cannot establish connections to Kyverno controller endpoints

References:

- Kubernetes documentation: Network Policies
- Kyverno documentation: Security

T-10: Unauthorised access to kyverno namespace resources

Affected architectures: Both

Severity	Relevant categories	MITRE ATT&CK mapping	NIST SP 800-53 mapping
Medium	RBAC Default configuration	T1078: Valid Accounts T1098: Account Manipulation	AC-6: Least Privilege AU-2: Event Logging

Impact: A principal with create, update or delete permissions on resources within `kyverno` can manipulate Kyverno controller behaviour outside of any Kyverno policy evaluation, since the namespace is excluded from webhook processing by default (see T-13: Default kyverno namespace exclusion from webhook processing). This includes modifying controller configuration via `ConfigMap` resources, manipulating Kyverno service account tokens to escalate privileges, or deploying malicious pods into the namespace. None of these actions require cluster-admin privileges or modification of cluster-scoped policy resources.

Description: Access to resources within `kyverno` is governed entirely by Kubernetes RBAC, as Kyverno's own admission policies do not apply to this namespace by default. The resources of highest value to an attacker are `ConfigMap` resources containing controller configuration, service account tokens with cluster-wide read permissions, and pod resources that could be used to execute arbitrary code within the policy enforcement environment.

Kyverno's default Helm installation does not create `RoleBinding` resources explicitly restricting access to `kyverno` beyond what Kubernetes provides by default. In clusters where developers or CI pipeline service accounts hold broad `ClusterRole` bindings such as `cluster-admin` or `edit`, these may grant unintended access to `kyverno` resources without any Kyverno-specific RBAC review catching it.

Mitigating factors:

- Organisations that have set `config.excludeKyvernoNamespace: false` reduce this risk, as Kyverno admission policies will apply to resources created in `kyverno` at admission time
- Pod Security Admission enforcement on `kyverno` provides a Kyverno-independent admission layer regardless of the namespace exclusion setting

Recommendations:

- Audit all `RoleBinding` and `ClusterRoleBinding` resources that grant permissions on `kyverno` resources, ensuring only Kyverno service accounts and platform SRE team members hold create, update, patch or delete permissions
- Deploy Pod Security Admission enforcement on `kyverno` using the `enforce` mode label, providing an admission control layer that operates independently of Kyverno
- Apply `NetworkPolicy` resources to `kyverno` to limit the blast radius of any compromise of resources within the namespace
- Enable Kubernetes API server audit logging on `kyverno` resource operations and alert on unexpected create, update or delete events within the namespace

References:

- Kyverno documentation - security

Related threats:

- T-13: Default kyverno namespace exclusion from webhook processing

T-11: Availability and confidentiality risks from external service calls in policy rules

Affected architectures: Both

Severity	Relevant categories	MITRE ATT&CK mapping	NIST SP 800-53 mapping
Medium	Policy configuration Network	T1499.004: Endpoint Denial of Service T1041: Exfiltration Over C2 Channel	SC-8: Transmission Confidentiality and Integrity SC-7: Boundary Protection

Impact: `ClusterPolicy` rules that reference external HTTP endpoints via `apiCall.service`, or CEL policies using the HTTP library, introduce a runtime dependency on the availability of that endpoint during admission evaluation. If the external service is unavailable when a policy rule is evaluated, the outcome depends on `failurePolicy` configuration: with `failurePolicy: Fail` the admission request is blocked, potentially denying legitimate workloads; with `failurePolicy: Ignore` enforcement is silently skipped. Either outcome represents a security or availability impact. Additionally, because Kyverno sends admission request data to the external endpoint as part of the call, any sensitive fields present in the request, including resource metadata, labels, annotations, or spec fields, may be transmitted to services outside the cluster's trust boundary. This risk applies equally to both `apiCall.service` context entries in Kyverno policies and CEL policies using the HTTP library.

Description: The `apiCall.service` context type allows `ClusterPolicy` rules to make HTTP requests to arbitrary endpoints, including both in-cluster services and external URLs, during admission evaluation. The response is stored as a context variable and used in policy rule logic. This makes the availability and integrity of the external service a direct dependency of admission control for any resource matched by the policy.

Service calls do not natively support authentication beyond custom HTTP headers, meaning the external endpoint would typically either be network-accessible without credentials or rely on header-based token authentication. Where the endpoint is accessible without authentication, any party able to reach it can interact with it, and there is no mutual authentication between Kyverno and the service.

A further risk arises where a principal with write access to `ClusterPolicy` resources can craft a policy containing a malicious `apiCall.service` URL. On every subsequent admission request matching that policy, the Kyverno admission controller will make an outbound HTTP request to the attacker-specified endpoint. This can be used to probe internal cluster services, reach cloud instance metadata endpoints, or exfiltrate admission request data to an

attacker-controlled host (effectively using the admission controller to exploit server-side request forgery). This attack requires only `ClusterPolicy` write access and does not require any compromise of Kyverno components directly.

Admission requests contain the full object being admitted, including all fields submitted by the requesting principal. Where a policy uses `apiCall.service` with a `POST` method and includes fields from `request.object` in the request body, sensitive data, (such as environment variable values, secret references, or user-supplied annotations) may be transmitted to the external endpoint. If that endpoint is operated by a third party or logs request bodies, this represents a confidentiality risk.

For `GlobalContextEntry` using `apiCall`, if the external endpoint becomes unavailable or returns an error, any policy referencing that GCE may fail at runtime evaluation when the cached data cannot be refreshed. Rather than a single admission request being affected, all requests evaluated against policies that depend on that GCE are at risk until the dependency recovers, widening the availability impact compared to a per-rule `apiCall.service` failure. This failure may not be immediately visible in policy status, making it harder to detect than a condition that explicitly marks a policy as unhealthy.

Mitigating factors:

- Where external service calls are made only to in-cluster services within a trusted namespace, the network exposure and availability risk is significantly reduced compared to calls to external URLs
- Policies using `apiCall.service` with `GET` requests and no request body do not transmit admission request data to the external endpoint, limiting the confidentiality risk to the response path only
- As of V1.17.0 Kyverno includes a default block list which prevent access to loopback of cloud metadata services via service calls

Recommendations:

- Avoid referencing external URLs in `apiCall.service` configurations or CEL HTTP library calls where in-cluster services can fulfil the same function, keeping the dependency within the cluster's trust boundary
- Where external service calls are necessary, ensure the endpoint is served over TLS and configure `caBundle` to verify the server certificate, preventing interception of transmitted admission request data
- Review all `ClusterPolicy` rules using `apiCall.service` with `POST` methods to identify which fields from `request.object` are included in the request body, and confirm these do not include sensitive data that should not leave the cluster
- Similarly, review all CEL policies using the HTTP library to identify which data is included in outbound requests, and confirm sensitive admission request fields are not transmitted to external endpoints

- Monitor external service endpoint availability and configure alerting to detect degradation before it affects admission evaluation
- Where `GlobalContextEntry` resources reference external endpoints, ensure the refresh interval and retry configuration are appropriate for the availability characteristics of the endpoint, and document the admission impact of `GlobalContextEntry` unavailability
- Apply `NetworkPolicy` egress controls on `kyverno` to enforce an explicit allowlist of permitted egress destinations. This limits the impact of both misconfigured and malicious `apiCall.service` URLs, and mitigates the risk of the admission controller being used as an SSRF primitive via either `apiCall.service` or CEL HTTP library calls to reach internal services or cloud metadata endpoints
- Utilize the `--httpBlocklist` and `--httpAllowlist` flags to limit which URLs can be reached using service calls

References:

- Kyverno documentation - external data sources
- Kyverno documentation - CEL libraries
- secure HTTP calls #15789

Related threats:

- T-01: Admission control bypass via failurePolicy: Ignore
- T-09: kyverno namespace exposure due to absent NetworkPolicy

T-12: Audit-only policy configuration as a compliance blind spot

Affected architectures: Both

Severity	Relevant categories	MITRE ATT&CK mapping	NIST SP 800-53 mapping
Medium	Policy configuration Reporting and observability	T1562.001: Impair Defenses: Disable or Modify Tools	CM-6: Configuration Settings AU-2: Event Logging

Impact: `ClusterPolicy` rules configured with `failureAction: Audit` record violations but do not block non-compliant workloads at admission time. While audit results may surface through `PolicyReport` resources, Kubernetes events, or Prometheus metrics, if none of these are actively monitored and acted upon, policies in audit mode provide no effective enforcement. Thus, non-compliant workloads - including those with privileged containers, excessive permissions, or unverified images are admitted without restriction. Organisations that have deployed Kyverno with policies in audit mode may believe enforcement is in place when no admission blocking is occurring. For organisations in regulated environments such as financial services, this gap may result in non-compliant workloads running undetected for extended periods.

Description: `failureAction` defaults to `Audit` in Kyverno's policy settings, meaning policies that do not explicitly set `failureAction: Enforce` will record violations without blocking admission. The Kyverno sample policy library also ships with most policies set to `Audit` mode deliberately, to avoid disruption for new users. Organisations that deploy sample policies without reviewing `failureAction` may therefore have no effective enforcement in place despite having `ClusterPolicy` resources installed and generating `PolicyReport` output.

The risk is compounded by the structure of `PolicyReport` resources themselves. Policy report results are spread across multiple namespaces and combine pass and fail results from multiple policies into a single report per namespace, making it difficult to identify new violations or track policy compliance trends without dedicated tooling. Without active monitoring, via Policy Reporter, Prometheus metrics, or equivalent, `PolicyReport` failures accumulate silently and the compliance picture they represent is often never acted upon.

A further consideration arises when transitioning policies from `Audit` to `Enforce` mode. For pre-existing resources that violate a newly enforced policy, Kyverno permits subsequent updates to those resources even if they continue to violate the policy. Existing workloads are not immediately impacted. This means non-compliant workloads already running will not be blocked until they are recreated or until an update brings them into compliance. Once a resource has been made compliant, any subsequent update that would reintroduce a violation is blocked.This behaviour can be disabled by setting `validate.allowExistingViolations: false` on

enforce mode validate rules, causing updates to pre-existing violating resources to be blocked immediately.

Mitigating factors:

- Organisations that have configured `failureAction: Enforce` on all production policies are not exposed to this threat for those policies
- Background scanning with `spec.background: true` ensures that violations from resources created before a policy was installed are captured in `PolicyReport` output, providing visibility even for pre-existing non-compliant resources

Recommendations:

- Review all installed `ClusterPolicy` resources and confirm `failureAction` is explicitly set to `Enforce` for any policy intended to provide admission enforcement in production environments
- Establish a process for promoting policies from `failureAction: Audit` to `Enforce` after a defined observation period, rather than leaving policies in audit mode indefinitely.
- Consider setting `validate.allowExistingViolations: false` on enforce mode policies in production environments where pre-existing violations should not be permitted to persist after a policy is applied
- Deploy active `PolicyReport` monitoring via Policy Reporter or equivalent tooling, with alerting on new `FAIL` results to ensure audit-mode violations are detected and remediated
- Do not treat the presence of `ClusterPolicy` resources or `PolicyReport` output as evidence of enforcement, verify `failureAction` settings explicitly as part of compliance reviews
- When deploying policies from the Kyverno sample library, review and explicitly set `failureAction` on each policy before deploying to production, rather than accepting the sample defaults
- After switching a policy from `Audit` to `Enforce`, audit existing `PolicyReport` failures for that policy and remediate non-compliant workloads, as they will not be automatically blocked until they are next updated or recreated

References:

- Kyverno documentation - Validate - Failure Action
- Kyverno documentation - Policy Reports

Related threats:

- T-07: Insecure policy promotion from misconfigured environments
- T-08: Conflated registry principals enabling simultaneous image and policy tampering

T-13: Default `kyverno` namespace exclusion from webhook processing

Affected architectures: Both

Severity	Relevant categories	MITRE ATT&CK mapping	NIST SP 800-53 mapping
Medium	Default configuration RBAC Kubernetes	T1562.001: Impair Defenses: Disable or Modify Tools	CM-6: Configuration Settings CM-7: Least Functionality

Impact: By default, `kyverno` is excluded from Kyverno's admission webhook processing, meaning no `ClusterPolicy` rules apply to resources created in that namespace at admission time. Any principal with permissions to create or modify resources in `kyverno` can do so without Kyverno policy evaluation, relying solely on Kubernetes-native controls.

Description: The exclusion is controlled by `config.excludeKyvernoNamespace`, which defaults to `true`. When enabled, Kyverno adds `kyverno` to the `namespaceSelector` exclusion list on all webhook configurations. The exclusion is intentional: without it, a Kyverno outage combined with `failurePolicy: Fail` could prevent Kyverno from recovering by blocking its own pod creation. Setting `config.excludeKyvernoNamespace: false` provides a stronger admission-time security posture, though it requires manual intervention to recover Kyverno in outage scenarios where `failurePolicy: Fail` is configured.

The exclusion applies only to admission-time requests. Background scanning via the reports controller is not subject to this exclusion, meaning `PolicyReport` resources will still be generated for `kyverno` resources, providing some compensating visibility.

Mitigating factors:

- Restrict write access to `kyverno` resources to Kyverno service accounts and platform SRE team members only, enforced via explicit `RoleBinding` and `ClusterRoleBinding` review
- Deploy Kubernetes-native Pod Security Admission enforcement on `kyverno` using the `enforce` mode label, providing an admission control layer that operates independently of Kyverno
- Apply `NetworkPolicy` resources to `kyverno` to compensate for the absence of Kyverno admission policy protection on this namespace
- Consider setting `config.excludeKyvernoNamespace: false` in production environments where the recovery implications are understood and compensated for via three-replica deployment and documented recovery runbooks

Recommendations:

- Deploy Pod Security Admission enforcement on `kyverno` using the `enforce` mode label as a Kyverno-independent admission control layer
- Apply `NetworkPolicy` resources to `kyverno` to limit network access to controller pods regardless of admission policy coverage
- Consider setting `config.excludeKyvernoNamespace: false` in production environments where the recovery implications are understood and compensated for via three-replica deployment and documented recovery runbooks

References:

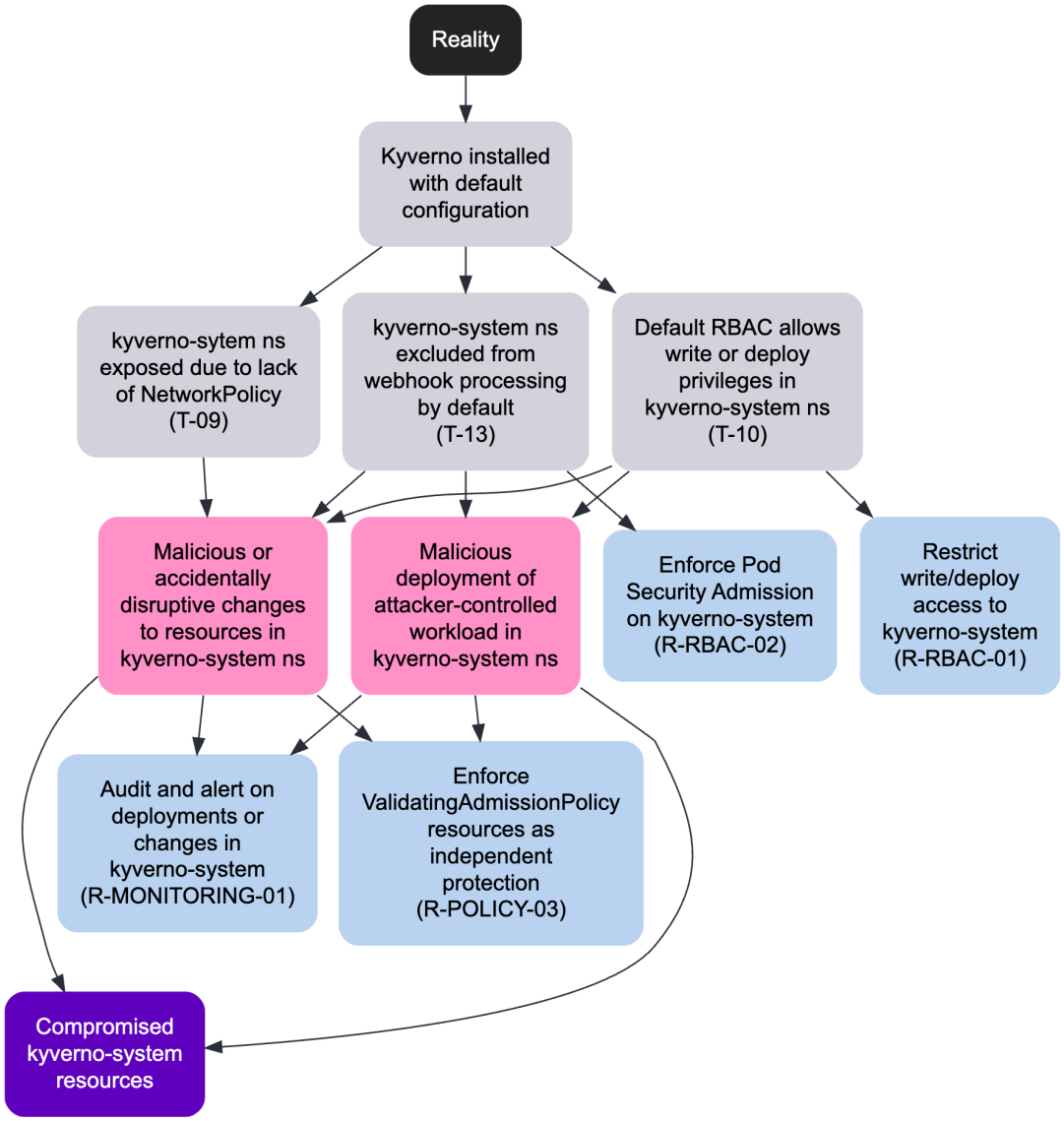
- Kyverno documentation: Security
- Kubernetes documentation: Pod Security Admission

Attack Trees

This section illustrates the potential pathways an attacker could exploit within the example Kyverno deployment. These attack tree diagrams break down the steps an adversary might take to achieve a specific goal and the security controls that could be used to mitigate each threat.

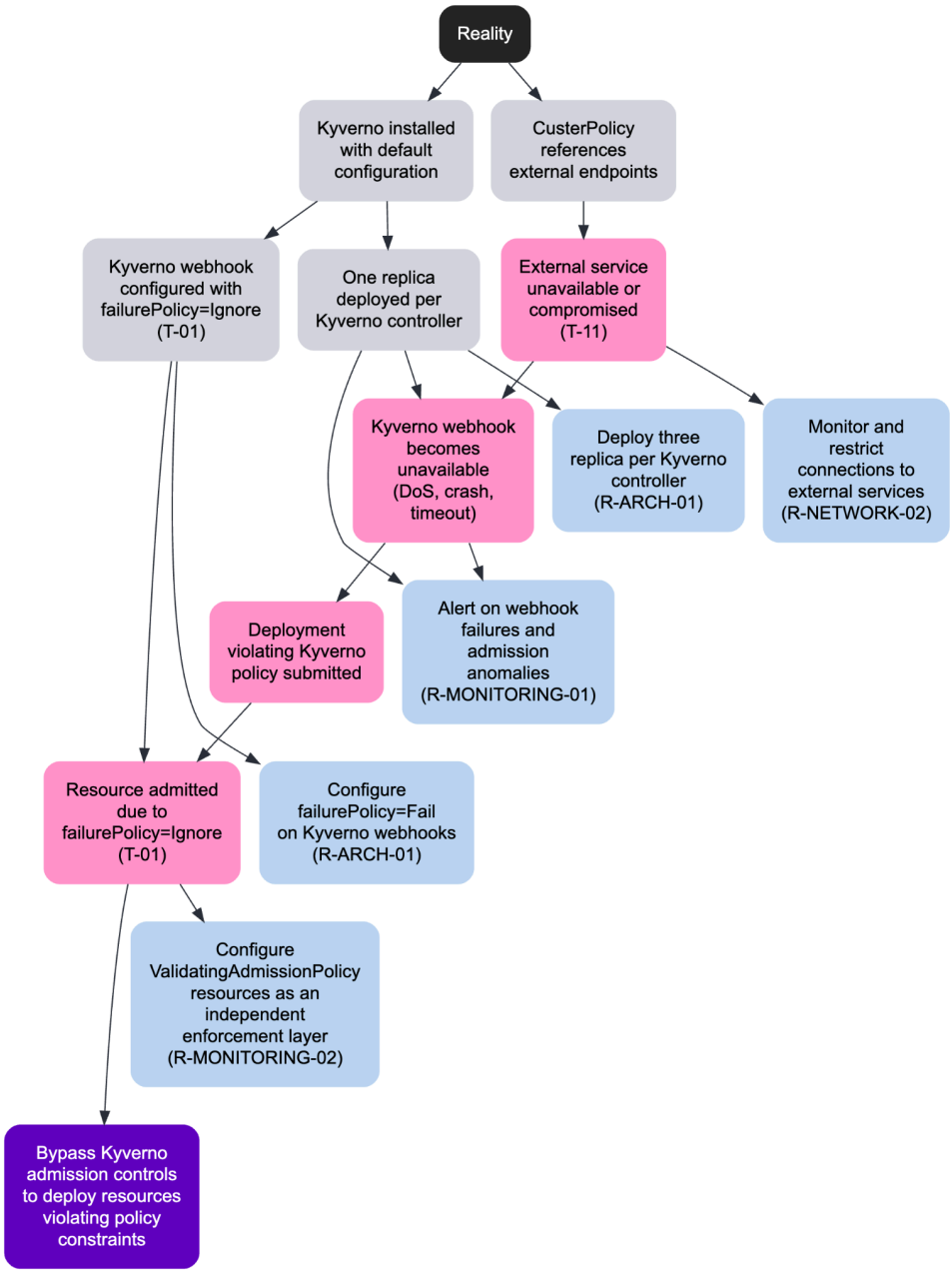
Threats to the kyverno namespace

Because `kyverno` is excluded from Kyverno's own admission webhook processing by default, it sits outside the enforcement boundary Kyverno creates for the rest of the cluster. This tree illustrates how insufficient access controls and absent network isolation on this namespace create a path for an attacker to manipulate Kyverno's configuration or deploy malicious workloads in an environment where policy evaluation does not apply.



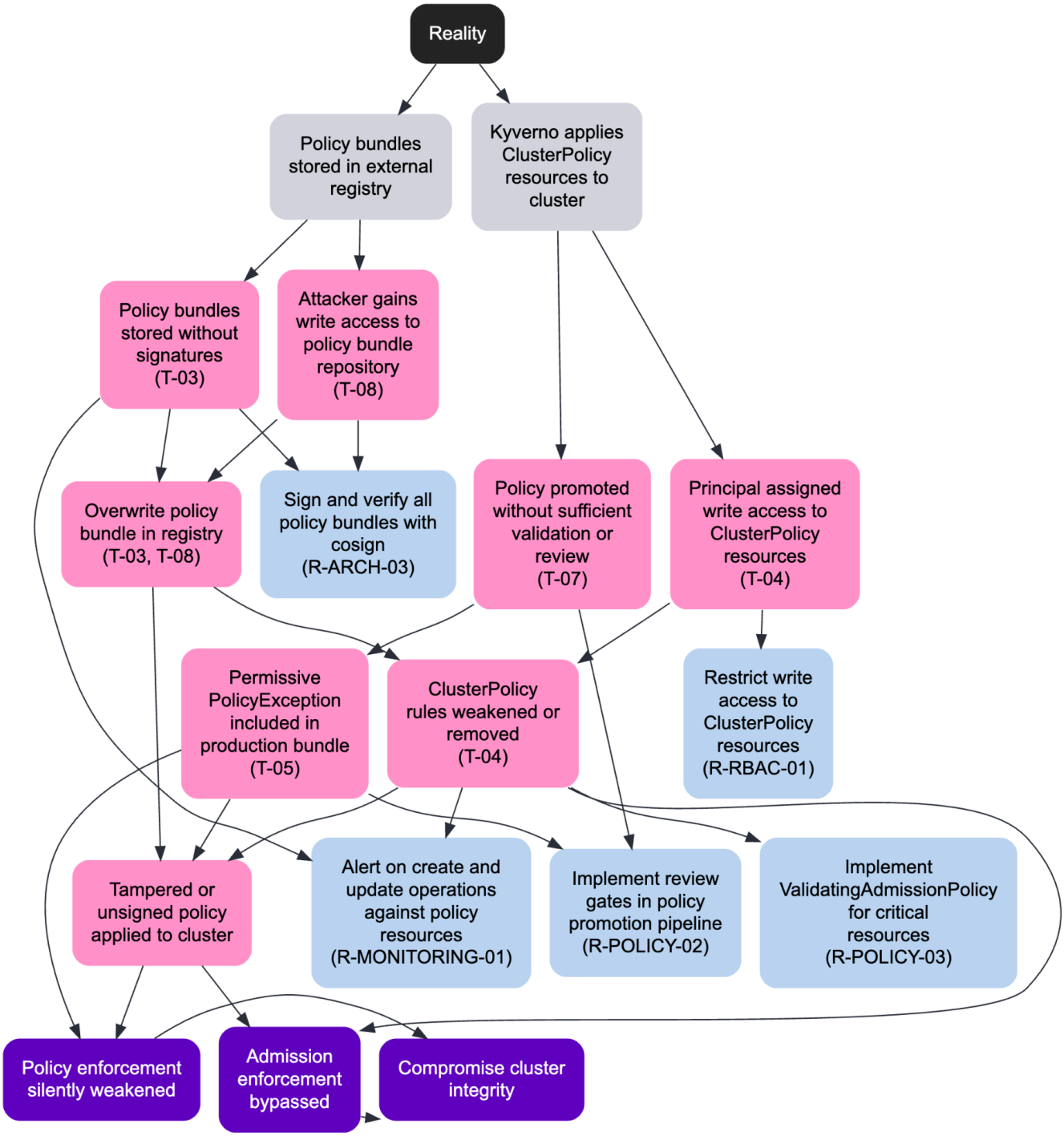
Unavailability of Kyverno webhook resulting in admission enforcement bypass

Kyverno's security guarantees are only as strong as its availability. This tree illustrates how an attacker can exploit the relationship between Kyverno's operational state and `failurePolicy` configuration. For example, disrupting the admission controller through resource exhaustion or denial of service to create a window during which all admission requests are processed without policy evaluation.



Threats to cluster-scoped policy resources

The policies Kyverno enforces are themselves often targeted by attackers. This tree illustrates



Weaknesses in configured policies

Even a correctly installed Kyverno deployment provides no meaningful enforcement if its policies are misconfigured. This tree illustrates how gaps in image verification coverage, policies left in audit mode, and insufficiently reviewed `PolicyException` resources can individually or in combination allow non-compliant and unsigned workloads to be admitted - even while compliance reports suggest everything is operating normally.

End User Hardening

Building upon the identified threats, ControlPlane provides 13 hardening recommendations designed to secure the Kyverno ecosystem against real-world attack vectors. These recommendations are categorized by their technical domain and prioritized by their impact on the cluster's overall security posture. To ensure alignment with federal and enterprise standards, each recommendation is mapped to the relevant security controls within the NIST SP 800-53 Revision 5 framework

Priority

Each recommendation has been evaluated using the following scoring system:

Colour	Priority
Red	High
Amber	Medium
Green	Low

Categories

- **ARCH:** Recommendations focused on the overall design and deployment architecture of Kyverno, including cluster topology, policy distribution mechanisms, and separation of policy authoring from enforcement environments
- **RBAC:** Recommendations for managing and restricting Kubernetes permissions associated with Kyverno service accounts, controllers, and sensitive cluster-scoped resources including `ValidatingWebhookConfiguration`, `ClusterPolicy`, and `PolicyException` resources
- **NETWORK:** Recommendations addressing network-level controls for Kyverno workloads, including isolation of the `kyverno` namespace and restriction of egress from Kyverno controllers.

- **CONFIG:** Recommendations focused on Kyverno-specific configuration best practices, including `failurePolicy` settings, `failureAction` enforcement mode, image verification configuration, `PolicyException` lifecycle management, and high availability settings
- **MONITORING:** Recommendations for visibility, auditing, and incident response, including `PolicyReport` and `ClusterPolicyReport` aggregation, Policy Reporter configuration, and alerting on policy violations or Kyverno component availability

Recommendations

ID: Title	Priority	Mitigated threats	Recommendation details	NIST SP 800-53 800-53 Revision 5
R-ARCH-01: Webhook Failure Policy Hardening	High	T-01: Admission control bypass via failurePolicy: Ignore T-09: kyverno namespace exposure due to absent NetworkPolicy and lack of mutual authentication	• Configure `failurePolicy: Fail` on all `ValidatingWebhookConfiguration` and `MutatingWebhookConfiguration` resources in production environments • Do not set `--forceFailurePolicyIgnore` or `--autoUpdateWebhooks=false` unless implications are explicitly understood • Deploy Kyverno with three replicas per controller to reduce the likelihood of complete admission controller unavailability triggering the bypass • Review `webhookTimeoutSeconds` to ensure timeouts are not configured so low that operational latency triggers `failurePolicy: Ignore` behaviour	CM-6: Configuration Settings CM-7: Least Functionality
R-ARCH-02: Image Verification Policy Enforcement	High	T-02: Unsigned or unverified image admission due to missing or misconfigured image verification policy T-12: Audit-only policy configuration as a compliance blind spot	• Deploy `verifyImages` rules or `ImageValidatingPolicy` resources covering all production registries and repositories with `failureAction: Enforce` • Set `required: true` on all `verifyImages` rules to block images without signatures • Set `failurePolicy: Fail` on all policies containing `verifyImages` rules to prevent silent bypass when the signature registry is unreachable • Enable `mutateDigest: true` to enforce digest pinning at admission time • Periodically audit image verification policy coverage against the full set of image sources in use	SA-10: Developer Configuration Management SI-7: Software, Firmware, and Information Integrity

			• Prefer `ImageValidatingPolicy` over `verifyImages` rules in `ClusterPolicy` for cleaner separation of concerns	
R-ARCH-03: Policy Bundle Signing and Verification	High	T-03: Policy supply chain compromise via unsigned policy distribution T-06: Hub cluster compromise enabling fleet-wide policy distribution takeover T-08: Conflated registry principals enabling simultaneous image and policy tampering	• Sign all policy bundles with cosign before storage in an external registry, using private keys stored in an HSM or equivalent hardware-backed key store • Configure all clusters to verify policy bundle signatures using `cosign verify` before applying retrieved bundles, and reject unsigned or unverified bundles • Establish a secure process for distributing the cosign public key to all consuming clusters, treating it as a trust anchor requiring integrity protection • Use separate signing keys for development and production policy distribution pipelines • Enforce repository-level RBAC within the registry to ensure CI pipeline principals cannot write to policy bundle repositories. • Monitor the registry for unexpected writes to policy bundle repositories	SA-10: Developer Configuration Management, SC-12: Cryptographic Key Establishment and Management
R-RBAC-01: Least Privilege Access to Kyverno Cluster-Scoped Resources	High	T-04: Unauthorised modification of Kyverno webhook or policy resources T-05: Enforcement bypass via misconfigured PolicyException resources T-10: Unauthorised access to kyverno namespace resources	• Restrict create, update, patch and delete permissions on `ValidatingWebhookConfiguration`, `MutatingWebhookConfiguration`, `ClusterPolicy`, `PolicyException`, `ImageValidatingPolicy`, `ClusterCleanupPolicy`, and `CleanupPolicy` resources to the platform SRE team only • Restrict `ClusterRole` create and update permissions to prevent privilege escalation via the Kyverno aggregation mechanism • Audit all `ClusterRoles` carrying Kyverno aggregation labels (such as `rbac.kyverno.io/aggregate-to-admission-controller: "true"`) to verify no unintended rules are being pulled into Kyverno controller permissions	AC-3: Access Enforcement, AC-6: Least Privilege

			• Regularly review RBAC bindings for CI pipeline principals and application developer service accounts	
R-RBAC-02: kyverno Namespace Access Controls	High	T-10: Unauthorised access to kyverno namespace resources T-13: Default kyverno namespace exclusion from webhook processing	• Audit all `RoleBinding` and `ClusterRoleBinding` resources granting permissions on `kyverno` resources, ensuring only Kyverno service accounts and platform SRE team members hold create, update, patch or delete permissions • Deploy Pod Security Admission enforcement on `kyverno` using the `enforce` mode label, providing an admission control layer that operates independently of Kyverno • Apply `NetworkPolicy` resources to `kyverno` to limit network access to controller pods • Consider denying all external access for resources within `kyverno` unless external API calls are being used. • Consider setting `config.excludeKyvernoNamespace: false` in production environments where the recovery implications are understood and compensated for via three-replica deployment and documented recovery runbooks	AC-6: Least Privilege CM-6: Configuration Settings
R-RBAC-03: PolicyException Governance	High	T-05: Enforcement bypass via misconfigured PolicyException resources T-12: Audit-only policy configuration as a compliance blind spot	• Ideally, restrict write access to `PolicyException` resources to only the platform SRE team via explicit RBAC • Scope all `PolicyException` resources as narrowly as possible to specific resource names, namespaces, and policy rules. For example, set `exceptionNamespace` to a single namespace • Implement a mandatory review and approval process for all `PolicyException` creation and modification • Ensure `PolicyException` requests are raised through a documented process, requiring the requesting team to demonstrate ownership of the affected resources. Exceptions	AC-3: Access Enforcement, AC-6: Least Privilege

			covering resources outside the requester's namespace should require explicit platform team sign-off and a defined expiry. • Monitor `PolicyException` creation and modification via Kubernetes API server audit logs	
R-MONITORING-01: Audit Logging and Alerting on Kyverno Resources	High	T-04: Unauthorised modification of Kyverno webhook or policy resources T-06: Hub cluster compromise enabling fleet-wide policy distribution takeover T-10: Unauthorised access to kyverno namespace resources T-12: Audit-only policy configuration as a compliance blind spot	• Enable Kubernetes API server audit logging and configure alerts on create, update, and delete operations against webhook configurations, policy resources, `ClusterRole` resources, and cleanup policy resources • For `ClusterCleanupPolicy` and `CleanupPolicy` specifically, alert on any create or update events outside approved platform team activity • Configure alerting on Kyverno controller pod availability to detect enforcement gaps promptly • Deploy active `PolicyReport` monitoring via Policy Reporter or equivalent tooling with alerting on new `FAIL` results	AU-2: Event Logging AU-12: Audit Record Generation
R-ARCH-04: Hub Cluster Hardening (multi-cluster only)	High	T-01: Admission control bypass via failurePolicy: Ignore	• Treat the hub cluster as critical infrastructure and apply the most stringent access controls of any cluster in the fleet, including MFA on all human access, comprehensive audit logging, and regular access reviews • Implement separate signing key references for development and production policy distribution pipelines • Store spoke API server credentials in the `policy-reporter` namespace as short-lived tokens rather than long-lived static credentials where supported • Rotate spoke API server credentials regularly and audit which principals have `Secret` read access in that namespace • Ensure `etcd` encryption at rest is enabled on the hub cluster given the sensitivity of spoke credentials stored there. Ensure	SC-12: Cryptographic Key Establishment and Management AC-6: Least Privilege

			registry administrative credentials are not stored on the hub cluster	
R-POLICY-01: Enforcement Mode Governance	Medium	T-07: Insecure policy promotion from misconfigured environments T-12: Audit-only policy configuration as a compliance blind spot	• Review all installed `ClusterPolicy` resources and confirm `failureAction` is explicitly set to `Enforce` for all policies intended to provide admission enforcement in production • Establish a process for promoting policies from `failureAction: Audit` to `Enforce` after a defined observation period • Use `kyverno apply --cluster` to validate promoted policies against live cluster resources before switching enforcement mode • After switching a policy from `Audit` to `Enforce`, audit existing `PolicyReport` failures and remediate non-compliant workloads, as pre-existing violating resources will not be blocked until recreated or updated • Consider setting `validate.allowExistingViolations: false` on enforce mode policies in production where pre-existing violations should not be permitted to persist	CM-6: Configuration Settings CM-3: Configuration Change Control
R-POLICY-02: Policy Promotion Pipeline Controls	Medium	T-07: Insecure policy promotion from misconfigured environments T-08: Conflated registry principals enabling simultaneous image and policy tampering	• Implement explicit review and approval gates in the policy distribution pipeline before promotion to production • Maintain separate enforcement profiles for each environment and ensure policy bundles are audited for environment-specific `PolicyException` resources before promotion • Test policies in a staging environment that mirrors the production enforcement posture with `failureAction: Enforce` and no broad exceptions • Use `kyverno test` as part of the distribution pipeline to validate policy behaviour against known resources before promotion	SA-10: Developer Configuration Management CM-3: Configuration Change Control

			• Periodically audit `PolicyException` resources in each environment for continued relevance and scope creep, independent of the promotion review process	
R-NETWORK-01: Network Isolation of Kyverno Components	Medium	T-09: kyverno namespace exposure due to absent NetworkPolicy and lack of mutual authentication T-11: Availability and confidentiality risks from external service calls in policy rules	• Deploy `NetworkPolicy` resources in `kyverno` to restrict ingress to only the Kubernetes API server for webhook callbacks and authorised monitoring systems for metrics scraping on port 8000 • Restrict egress from `kyverno` to only the Kubernetes API server, any configured external service endpoints, and where image verification is configured, the external OCI registry • Apply equivalent network isolation to any namespace containing policy authoring or distribution tooling • For `apiCall.service` configurations, enforce an explicit egress allowlist to limit the impact of misconfigured or malicious service call targets	AC-4: Information Flow Enforcement SC-7: Boundary Protection
R-NETWORK-02: External Service Call Controls	Medium	T-11: Availability and confidentiality risks from external service calls in policy rules	• Avoid referencing external URLs in `apiCall.service` configurations where in-cluster services can fulfil the same function • Where external service calls are necessary, ensure the endpoint is served over TLS and configure `caBundle` to verify the server certificate • Review all `ClusterPolicy` rules using `apiCall.service` with POST methods to confirm no sensitive fields from `request.object` are transmitted to external endpoints • Monitor external service endpoint availability and configure alerting to detect degradation before it affects admission evaluation • Where `GlobalContextEntry` resources reference external endpoints, ensure the refresh interval and retry configuration are appropriate for the availability characteristics of the endpoint	SC-8: Transmission Confidentiality and Integrity SC-7: Boundary Protection

			• Utilize the `--httpBlocklist` and `--httpAllowlist` flags to limit which URLs can be reached using service calls	
R-POLICY-03: ValidatingAdmissionPolicy as Independent Enforcement Layer	Medium	T-04: Unauthorised modification of Kyverno webhook or policy resources	• Implement `ValidatingAdmissionPolicy` at the Kubernetes level, independent of Kyverno, to provide an independent enforcement layer protecting critical resource modifications, including webhook configurations and cluster-scoped policy resources • Ensure the above or other controls remain effective even where Kyverno itself has been tampered with or is unavailable. Particularly if Kyverno admission policies are not enabled on the `kyverno` namespace	CM-6: Configuration Settings SI-3: Malicious Code Protection

About

ControlPlane is a global cloud native and open source cybersecurity consultancy operating in London, New York, and Auckland. We have industry-leading expertise in the architecture, audit, and implementation of zero trust infrastructure for regulated industries. With a deep understanding of secure-by-design and secure-by-default cloud, Kubernetes, and supply chain security we conduct threat modeling, penetration testing, and cloud native security training to the highest standard.

ControlPlane is trusted as the partner of choice in securing: multinational banks; major public clouds; international financial institutions; critical national infrastructure programs; multinational oil and gas companies, healthcare and insurance providers; and global media firms.

https://control-plane.io/

Team

Samuel Holmes
Tom Cope

Reviewers

Shuting Zhao
Jim Bugwadia
Andrew Martin
Tom Holtby
Mike Bryant