

CONTAINER & KUBERNETES SECURITY.

Kubernetes ships permissive by default, not secure by default — **flat pod networking, unscoped RBAC, and unencrypted secrets are the starting point, not a misconfiguration someone introduced later.**

- IMAGE SCANNING & SIGNING
- LEAST-PRIVILEGE RBAC
- NETWORK POLICY SEGMENTATION
- RUNTIME THREAT DETECTION

01 THE MECHANISM



Shared Kernel, Not a Full Isolation Boundary

Containers on the same host share the underlying operating system kernel, unlike a full virtual machine — **a kernel vulnerability or a privileged container can break out of its namespace isolation** and reach the host or neighboring containers directly.



Flat Pod Networking by Default

Kubernetes lets any pod reach any other pod across the cluster unless a NetworkPolicy explicitly restricts it — **without default-deny rules, a single compromised pod can reach everything else** on the network.



Secrets Are Encoded, Not Encrypted, By Default

Kubernetes Secrets are base64-encoded for storage, not cryptographically encrypted, unless encryption at rest is explicitly configured — **anyone with read access to etcd or the API can trivially decode them.**

02 THE FLOW

The Hardened Cluster

CONTAINED

1

Every image is scanned for known vulnerabilities and signed before it's allowed into the registry, and an admission controller rejects any workload that doesn't meet policy at deploy time.

2

Pods run under Restricted Pod Security Standards with least-privilege RBAC and default-deny NetworkPolicies, so **a compromised pod can't escalate privileges or reach unrelated workloads.**

3

Runtime monitoring watches for anomalous process execution or network activity inside running containers, catching **what static scanning at build time could never see.**

The Unhardened Cluster

EXPOSED

1

Images are pulled from public registries and deployed without scanning or signature verification, so a compromised or vulnerable base image reaches production unnoticed.

2

Pods run as root with broad RBAC permissions and no NetworkPolicies, so **a single compromised container can escalate privileges and move laterally** across the entire flat network.

3

Secrets sit as base64-encoded values with no encryption at rest, so **anyone who reaches the etcd store or API can decode credentials directly.**

03 CORE PILLARS & COVERAGE

PILLAR	WHAT IT COVERS
Image Scanning & Signing	Scans container images for known vulnerabilities before deployment and cryptographically signs them so provenance can be verified, closing the gap a public registry alone leaves open.
Least-Privilege RBAC & Pod Security Standards	Scopes service accounts and roles to only what a workload needs, and enforces the Restricted Pod Security Standard to block privileged or root containers.
Network Policies & Segmentation	Replaces Kubernetes' default flat pod networking with explicit default-deny rules, so lateral movement requires its own separate authorization.
Runtime Threat Detection	Monitors running containers for anomalous process execution, file access, or network activity that static image scanning at build time can't catch.

04 DETECTION APPROACHES COMPARED

APPROACH	SUFFICIENT ALONE?	WHAT IT MISSES
Image Scanning Alone, No Runtime Monitoring	No	Catches known vulnerabilities at build time but misses zero-days and attacks that only manifest once a container is actually running.
Default RBAC and Flat Pod Networking	No	Leaves every pod able to reach every other pod, and every service account with whatever broad permissions it was granted by default.
Secrets Stored Base64-Encoded, No Encryption at Rest	No	Base64 is an encoding, not encryption; anyone with etcd or API read access can trivially decode stored credentials.
Privileged Containers for Convenience	No	Running containers as root or in privileged mode removes the isolation boundary that's supposed to contain a compromise.
Image Signing + Least-Privilege RBAC + Network Policies + Runtime Detection	Yes	Covers the build, deploy, and runtime phases together, so no single stage becomes the weak link.

CONTAINER & KUBERNETES SECURITY.

SCAN · ISOLATE · MONITOR

05 PROGRAM OPERATIONS

Containers isolate less than a virtual machine by design. They share the host kernel, so a kernel-level or privileged-mode escape reaches the host directly, not just a hypervisor boundary.

Kubernetes ships permissive by default, not secure by default. Flat pod networking and unscoped RBAC are the out-of-the-box starting point, not an exception someone misconfigured.

PodSecurityPolicy is gone; Pod Security Standards replaced it. Kubernetes 1.25 removed PSP entirely, so clusters still referencing it are running on a deprecated model that no longer exists.

Supply chain risk starts at the base image, not the deployment. A compromised or typosquatted image pulled into the build pipeline carries its risk into every container built from it.

06 CONTROL MAPPING

MIST SP 800-190

The Application Container Security Guide, covering image, registry, orchestrator, and host-level container risks.

CIS KUBERNETES BENCHMARK

Prescriptive configuration guidance for hardening the Kubernetes control plane, worker nodes, and policies.

KUBERNETES POD SECURITY STANDARDS

The built-in Privileged, Baseline, and Restricted policy levels that replaced the deprecated PodSecurityPolicy.

SLSA (SUPPLY-CHAIN LEVELS FOR SOFTWARE ARTIFACTS)

A framework for build and provenance integrity, addressing how a compromised pipeline can inject risk before an image is ever deployed.

07 WHAT THIS DOES NOT SOLVE

Application-Layer Vulnerabilities

Hardening the container and cluster doesn't patch a vulnerability in the application code running inside it.

Host Operating System Security

Container and Kubernetes controls don't replace hardening the underlying node's own operating system, which remains a distinct attack surface.

Cloud Provider Misconfiguration

A hardened cluster can still be undermined by misconfigured cloud IAM, storage, or networking around it, outside Kubernetes' own policy layer.

CI/CD Pipeline Compromise

Signing and scanning images doesn't stop an attacker who compromises the pipeline itself before the image is ever built or signed.

08 MYTHS VS. REALITY

MYTH	REALITY
"Containers provide the same isolation as a virtual machine."	Containers share the host kernel — a kernel vulnerability or privileged-mode misconfiguration can break out of container isolation in a way a VM's hypervisor boundary generally prevents.
MYTH	REALITY
"Kubernetes Secrets are encrypted by default."	Secrets are base64-encoded, not encrypted , unless encryption at rest is explicitly configured — base64 is trivially reversible.
MYTH	REALITY
"Scanning container images before deployment is sufficient security."	Image scanning catches known vulnerabilities at build time , but misses zero-days and attacks that only appear in a container's actual runtime behavior.
MYTH	REALITY
"Kubernetes is secure by default out of the box."	Default settings include flat pod networking and permissive RBAC — meaningful hardening requires deliberately applying NetworkPolicies, Pod Security Standards, and scoped roles.
MYTH	REALITY
"PodSecurityPolicy is still the way to enforce pod security in Kubernetes."	PodSecurityPolicy was removed in Kubernetes 1.25 , replaced by the built-in Pod Security Standards (Privileged, Baseline, Restricted).

09 SPEAK THE PROGRAM — KEY TERMS

CONTAINER BREAKOUT

An attack that **escapes a container's namespace and cgroup isolation** to reach the underlying host or neighboring containers.

POD SECURITY STANDARDS (PSS)

Kubernetes' built-in **Privileged, Baseline, and Restricted policy levels** for pod configuration, replacing the deprecated PodSecurityPolicy.

NETWORKPOLICY

A Kubernetes resource that **defines allowed pod-to-pod traffic**, replacing the default flat, fully-open pod network with explicit rules.

ADMISSION CONTROLLER

A Kubernetes component that **enforces policy at deployment time**, rejecting workloads that don't meet security requirements before they ever run.

SBOM (SOFTWARE BILL OF MATERIALS)

A detailed inventory of **every component and dependency inside a container image**, used to assess supply-chain risk and known vulnerabilities.

RUNTIME SECURITY

Monitoring for **anomalous behavior inside already-running containers**, catching what static image scanning at build time cannot.