What is Kubernetes penetration testing?

Kubernetes penetration testing is the practice of attacking a Kubernetes cluster the way a real attacker would, to find what is actually exploitable. Kubernetes adds an entire control plane (the API server, scheduler, controller manager, kubelet on each node) on top of the underlying VMs or cloud accounts. A pentest covers the cluster API itself, the role-based access control (RBAC) wired to it, the pod identity that applications use to read secrets and call AWS / Azure / GCP, and the network paths a compromised container uses to reach the rest of the cluster.

HOW IT WORKS

01 What are the most common findings on a Kubernetes pentest?

From the engagements we ran in the last year:

- Over-permissive RBAC. Service accounts holding cluster-wide read or write on secrets, or roles like `cluster-admin` attached where they were not needed.
- Pods running as root or with hostPath mounts. Containers that can write to the host filesystem or use the host network. Most pod-escape techniques start here.
- Exposed kubelet API. On older or self-managed clusters, the kubelet exposes an API on the worker node that lets anyone with network reach execute commands in running pods.
- Pod-to-cloud metadata reachable. Pods on EKS / AKS / GKE reach the node's cloud metadata endpoint by default and inherit the node's credentials. Fixable with a hop-count limit or a network policy.
- Default-deny network policy missing. No NetworkPolicy means every pod can reach every other pod, every namespace, every internal service.
- Etcd reachable from worker nodes. The cluster state store reachable from places it should not be.
- Secrets stored unencrypted in etcd. Default on older clusters. Modern clusters can encrypt at rest with a KMS key, but the setting must be enabled.

SOURCES

[1] CIS Kubernetes Benchmark

[2] Kubernetes Security Best Practices

[3] NIST SP 800-190 Application Container Security Guide

[4] MITRE ATT&CK for Containers

02 How does an attacker pivot inside a cluster?

The canonical chain we reproduce on engagements:

1. Application running in a pod has a vulnerability (SQL injection, SSRF, RCE in a dependency). 2. Attacker uses the vulnerability to execute code inside the container. 3. Inside the container, attacker reads the pod's service account token (a JWT mounted at a known path). 4. With the token, attacker calls the Kubernetes API server. If RBAC is over-permissive, they list secrets, list other pods, read configmaps with database credentials inside. 5. Attacker reaches the node's cloud metadata service (if not blocked) and reads the node's cloud credentials. 6. With cloud credentials, attacker reaches resources well outside the cluster.

Each hop is a place defense can break the chain. Most production clusters break it at zero or one hop.

03 How does SecureLayer7 test a Kubernetes cluster?

Four phases.

Phase 1, external recon. Reachable cluster endpoints, exposed dashboard, exposed admission webhooks, kubelet exposure. Sometimes finds the engagement before the rest of the testing starts.

Phase 2, in-cluster recon. Stand up a pod with the access a typical workload would have. Enumerate RBAC, secrets, configmaps, services, network policies. Identify the privilege boundaries.

Phase 3, escalation chains. From the in-cluster vantage, attempt every documented pivot: RBAC abuse, secret theft, pod-to-cloud metadata, pod escape via hostPath / privileged container, etcd reach.

Phase 4, cloud-impact. When a pivot reaches cloud credentials, document the realistic blast radius the same way an AWS / Azure / GCP pentest would.

Deliverable maps findings to the CIS Kubernetes Benchmark and includes the specific manifest / RBAC change to close each chain.

Test your Kubernetes environment end to end.

securelayer7.net/learn/cloud-security/k8s-pentesting

Open online