Kubernetes Security Lab — Week 1

Insecure-by-Design K3s Cluster → Hardened Cluster

Michael O'Malley

Contributor: Thomas Cerruti

1 Abstract

This lab focused on deploying and hardening a lightweight K3s Kubernetes cluster configured with common insecure defaults. The environment simulated real-world misconfigurations including excessive RBAC permissions, unrestricted pod communication, exposed API endpoints, and insecure secret handling. Security posture was evaluated using kube-bench and manual enumeration techniques. Mitigations were implemented following CIS Kubernetes Benchmark guidance and modern cloud-native security practices.

2 Lab Objective and Threat Model

2.1 Objective

- Deploy a functional K3s cluster
- Identify insecure defaults and privilege risks
- Simulate realistic attacker paths
- Apply layered defensive controls

2.2 Threat Model

- External user with network visibility to exposed cluster services
- Authenticated low-privilege Kubernetes identity
- Compromised container attempting privilege escalation or lateral movement

3 Environment Setup

3.1 Cluster Configuration

- Kubernetes Distribution: K3s
- Node Role: Single-node control plane
- Container Runtime: containerd (embedded with K3s)
- Networking: Flannel CNI (default)
- Host Platform: ARM-based lab environment

3.2 Intentional Misconfigurations

- API server exposed on all interfaces (*:6443)
- Cluster-admin permissions assigned to default user
- Containers running as root with unrestricted capabilities

- No network policy enforcement
- Secrets stored without encryption at rest

4 Attack Surface Analysis and Findings

4.1 Exposed Kubernetes API

The Kubernetes API server was initially configured to listen on all interfaces. This expanded the external attack surface and allowed remote probing attempts. Binding was later restricted to a specific internal interface through the K3s configuration file.

4.2 RBAC Misconfigurations

Privilege enumeration revealed unrestricted access:

```
kubectl auth can-i --list
```

The primary identity possessed cluster-admin privileges, allowing unrestricted creation, deletion, and modification of resources.

4.3 Container Privilege Issues

Pods were configured to run as root and could install additional tooling during runtime. This created a viable privilege escalation path if a workload was compromised.

4.4 Secrets Management

Secrets were stored in base64 format without encryption at rest. Any user with `get secrets` permission could decode credentials and pivot further into the environment.

5 Security Assessment Tools

5.1 kube-bench

Used to evaluate CIS benchmark compliance. Several controls were flagged due to insecure kubelet settings and missing authorization restrictions.

5.2 kubectl auth can-i

Used for privilege auditing and access mapping across namespaces.

5.3 Manual Network Testing

Pod-to-pod connectivity confirmed unrestricted east-west communication, demonstrating a lack of segmentation.

6 Impact Analysis

- Cluster-wide administrative control from compromised credentials
- Unauthorized retrieval of sensitive secrets
- Lateral movement between workloads
- Persistent attacker foothold via privileged containers

7 Mitigations and Hardening

7.1 API Server Hardening

- Restricted API bind address to internal interface
- Disabled anonymous authentication

7.2 RBAC Least Privilege

- Created dedicated namespace for controlled workloads
- Implemented read-only and limited-permission users
- Audited RoleBindings for unnecessary cluster-admin assignments

7.3 Pod Security Controls

- Enforced non-root execution
- Disabled privilege escalation
- Applied default seccomp profiles
- Restricted Linux capabilities
- Enabled read-only root filesystem

7.4 Network Segmentation

- Implemented NetworkPolicies
- Default deny traffic model
- Microsegmentation between namespaces
- Zero-trust internal communication model

7.5 Secrets Management Improvements

- Enabled AES-CBC encryption for secrets
- Restricted access through RoleBindings
- Reduced secret exposure scope

7.6 CIS Benchmark Remediation

Key remediations included:

- Disabled anonymous kubelet authentication
- Enabled certificate rotation
- Enforced TLS for kubelet communications
- Disabled insecure read-only ports
- Enabled seccomp defaults
- Limited pod process counts

Note: Certain kube-bench checks produced warnings due to K3s embedding the kubelet inside the server binary.

8 Key Takeaways

- Default Kubernetes configurations prioritize functionality over security
- RBAC misconfiguration presents one of the fastest paths to compromise
- Network policies are essential for preventing lateral movement
- Runtime controls significantly reduce exploit impact

9 Conclusion

This lab demonstrated the security risks introduced by insecure Kubernetes configurations and highlighted the effectiveness of layered defensive controls. Through RBAC enforcement, pod hardening, network segmentation, secrets protection, and CIS-aligned configuration changes, the cluster transitioned from an insecure baseline to a significantly more resilient architecture.

10 References

- CIS Kubernetes Benchmark
- Kubernetes Official Documentation
- Rancher K3s Documentation
- kube-bench Project Documentation