IJIRSET
International Journal of Innovative Research in
SCIENCE | ENGINEERING | TECHNOLOGY

e-ISSN: 2319-8753 | p-ISSN: 2347-6710

INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN SCIENCE | ENGINEERING | TECHNOLOGY

Volume 11, Issue 1, January 2022

INTERNATIONAL STANDARD SERIAL NUMBER INDIA

Impact Factor: 7.569

9940 572 462 | 6381 907 438 | ijirset@gmail.com | www.ijirset.com

International Journal of Innovative Research in Science, Engineering and Technology (IJIRSET)

|e-ISSN: 2319-8753, p-ISSN: 2347-6710| www.ijirset.com | Impact Factor: 7.569| A Monthly Peer Reviewed & Referred Journal |

|| Volume 11, Issue 1, January 2022 ||

| DOI:10.15680/IJIRSET.2022.1101138 |

Securing Containerized Workloads in Kubernetes: RBAC, Network Policies, and Runtime Hardening

Anantha Kukkadapu

Information Technology Infrastructure Engineer, Allied Irish Banks PLC, Molesworth Street, Dublin, Ireland

ABSTRACT: Kubernetes has become the de facto standard for orchestrating containerized workloads in cloud-native environments, yet its default configurations expose significant security risks. This paper investigates Kubernetes security from three critical dimensions: access control using Role-Based Access Control (RBAC), inter-pod communication via network policies, and runtime behavior monitoring using tools such as Falco and AppArmor.

Our experimental setup includes a production-like cluster with 50 microservices, simulating a healthcare records management system. We perform attack simulations involving privilege escalation, container breakout, and lateral movement using known CVEs and adversary tactics from the MITRE ATT&CK matrix. Implementation of fine-grained RBAC and namespace isolation prevents 92% of escalation attempts. Applying Calico-based ingress/egress network policies cuts unauthorized cross-pod communication by 80%, and deploying Falco reduces dwell time of suspicious processes by 63%. Runtime anomalies—such as shell spawns from web containers or unusual file access—are effectively detected.

However, integrating all layers increases operational complexity, especially in CI/CD pipelines where security drift can occur. Our study underscores that Kubernetes security must be multi-layered and automated to be sustainable. We propose a reference architecture for secure workload deployment, covering cluster bootstrapping, policy management, and audit configuration. The paper concludes with recommendations for securing production Kubernetes environments in regulated and multi-tenant deployments.

I. INTRODUCTION

Kubernetes has revolutionized modern application deployment by abstracting container orchestration and providing horizontal scalability, service discovery, and automated lifecycle management. However, as adoption has surged, so have concerns regarding its security posture. Kubernetes, by default, prioritizes usability and flexibility, often at the cost of enforceable security controls. This makes production clusters susceptible to a wide range of attacks, from misconfigured privileges and unsecured APIs to unmonitored container behavior and unrestricted inter-pod communication.

In regulated industries like healthcare, where data privacy and system availability are paramount, a compromise within the Kubernetes cluster can have devastating consequences. The need to secure not just the Kubernetes control plane but also the workloads and runtime behavior has thus become central to any DevSecOps strategy. Yet, implementing layered security controls in a Kubernetes environment poses practical challenges related to tool integration, performance impact, and operational complexity.

This paper explores Kubernetes security through three complementary perspectives: **Role-Based Access Control (RBAC)** for administrative control, **network policies** for communication boundaries, and **runtime hardening** using monitoring tools such as **Falco** and **AppArmor**. Our contributions include: (1) attack simulations based on real-world tactics, (2) quantitative evaluation of security gains, and (3) a reference architecture that balances security with manageability. The study aims to equip DevOps engineers, security architects, and SREs with actionable insights for securing containerized workloads in production-grade Kubernetes environments.

|e-ISSN: 2319-8753, p-ISSN: 2347-6710| www.ijirset.com | Impact Factor: 7.569| A Monthly Peer Reviewed & Referred Journal |

|| Volume 11, Issue 1, January 2022 ||

| DOI:10.15680/IJIRSET.2022.1101138 |

II. RELATED WORK

Security in Kubernetes has been the focus of increasing academic and industry attention over the past few years. Initial research efforts centered on securing the Kubernetes API server, including authentication, TLS encryption, and role separation. Tools like kube-bench and kube-hunter provided early support for auditing against CIS benchmarks, while later efforts emphasized workload isolation and runtime behavior monitoring.

Studies such as those by Red Hat (2021) and Aqua Security (2020) highlighted the risks associated with misconfigured RBAC, unscanned container images, and overly permissive network setups. More recent literature delves into applying the principle of least privilege to Kubernetes through automated policy generation tools like OPA/Gatekeeper and integrating runtime monitors like Falco to detect anomalies beyond static policy enforcement.

Network policy enforcement via Kubernetes-native mechanisms (e.g., Calico, Cilium) has also received significant focus. These solutions use eBPF and iptables under the hood to enable fine-grained traffic filtering at the pod level. However, empirical comparisons across these tools in real-world environments remain sparse.

Finally, few studies combine access control, network policy enforcement, and runtime monitoring in an end-to-end security evaluation. This paper fills that gap by simulating coordinated attacks in a microservice-rich, regulated environment, quantifying mitigation success, and proposing operational practices for sustained security posture.

III. SYSTEM DESIGN AND ARCHITECTURE

Our experimental Kubernetes cluster mimics a healthcare records management system composed of 50 microservices distributed across five namespaces. Services include patient data APIs, billing engines, authentication modules, and a PostgreSQL database backend. The cluster runs on a three-master, five-worker node topology, deployed on a virtualized private cloud using kubeadm.

Security is embedded across three primary layers:

- **Access Control Layer**: RBAC policies restrict admin, developer, and CI/CD service accounts to minimal required actions. Namespace-level segregation ensures privilege boundaries.
- **Network Control Layer**: Calico enforces ingress and egress rules, enabling service-to-service communication only where explicitly permitted. DNS, logging, and monitoring traffic is whitelisted through label selectors.
- **Runtime Monitoring Layer**: Falco continuously watches for abnormal system calls, while AppArmor enforces mandatory access control profiles at the container runtime level.

Audit logs are streamed to Elasticsearch and visualized through Kibana. Policy-as-code is managed via Helm and validated through pre-deployment checks in GitLab CI/CD. Role bindings and network policies are reviewed bi-weekly as part of our simulated compliance process.
This design enforces zero trust at multiple levels—identity, network, and process—without compromising operational agility.

IV. ATTACK SIMULATION AND THREAT MODELING

To assess the effectiveness of our security controls, we simulated realistic attack scenarios based on the MITRE ATT&CK framework for containers. These include:

1. **Privilege Escalation**: Attempting to bind elevated roles (cluster-admin) to compromised service accounts.
2. **Container Breakout**: Exploiting CVE-2019-5736 (runc vulnerability) to escape container boundaries and access host nodes.
3. **Lateral Movement**: Accessing unauthorized services across namespaces through unrestricted pod-to-pod communication.
4. **Command and Control**: Executing reverse shell connections from web containers using compromised image layers.

|e-ISSN: 2319-8753, p-ISSN: 2347-6710| www.ijirset.com | Impact Factor: 7.569| A Monthly Peer Reviewed & Referred Journal |

|| Volume 11, Issue 1, January 2022 ||

| DOI:10.15680/IJIRSET.2022.1101138 |

Each attack was launched from a compromised frontend container running with minimal initial privileges. The goal was to evaluate whether:

- RBAC policies effectively limited Kubernetes API misuse.
- Network policies prevented lateral pod movement.
- Runtime monitors detected unauthorized actions.

The simulations revealed that without controls, attackers could escalate privileges, exfiltrate sensitive data from backend pods, and maintain stealthy presence. However, with controls active:

- 92% of privilege escalation attempts were blocked due to RBAC and restricted create permissions.
- 80% of unauthorized cross-namespace traffic was denied by Calico.
- Falco detected all attempts to spawn /bin/sh, access /etc/passwd, or write to temp folders from restricted pods.

The test environment logged and visualized all detections, forming a clear audit trail for each simulated breach.

V. RBAC IMPLEMENTATION AND RESULTS

RBAC was configured using scoped Roles and RoleBindings for each namespace. Service accounts were assigned based on least-privilege principles:

- Developers had read-only access to logs and get access on deployments.
- CI/CD bots were restricted to patch, get, and create permissions for deployment and configmaps.
- Admins had broader control within namespaces but no cluster-wide access unless explicitly needed.

Validation was done using **kubeaudit** and **OPA Rego policies**, ensuring no wildcard verbs, resources, or non-namespaced admin bindings existed.

During our tests:

- Privilege escalations via RoleBinding abuse failed 11 out of 12 times due to insufficient permissions.
- Escalation attempts that succeeded occurred in intentionally unprotected namespaces, used to simulate misconfiguration scenarios.
- kubectl audit logs confirmed blocked attempts and traceable identities for each event.

RBAC proved highly effective when paired with continuous policy review and automation.

VI. NETWORK POLICY ENFORCEMENT USING CALICO

Calico was used to enforce ingress and egress rules between pods, applying namespace and label-based selectors. Policies denied all traffic by default and allowed only explicitly whitelisted communications.

Sample policies included:

- Denying all cross-namespace traffic unless originating from auth-* services.
- Allowing DNS, logging, and metrics services through specific egress rules.
- Blocking all egress from public-facing pods except HTTP/HTTPS to whitelisted IPs.

Our results showed:

- 80% of lateral movement simulations failed due to denied connections.
- Data exfiltration attempts via netcat and curl were blocked unless the destination was pre-approved.
- Intermittent service failures (e.g., health checks blocked due to overly strict rules) were logged and refined over time.

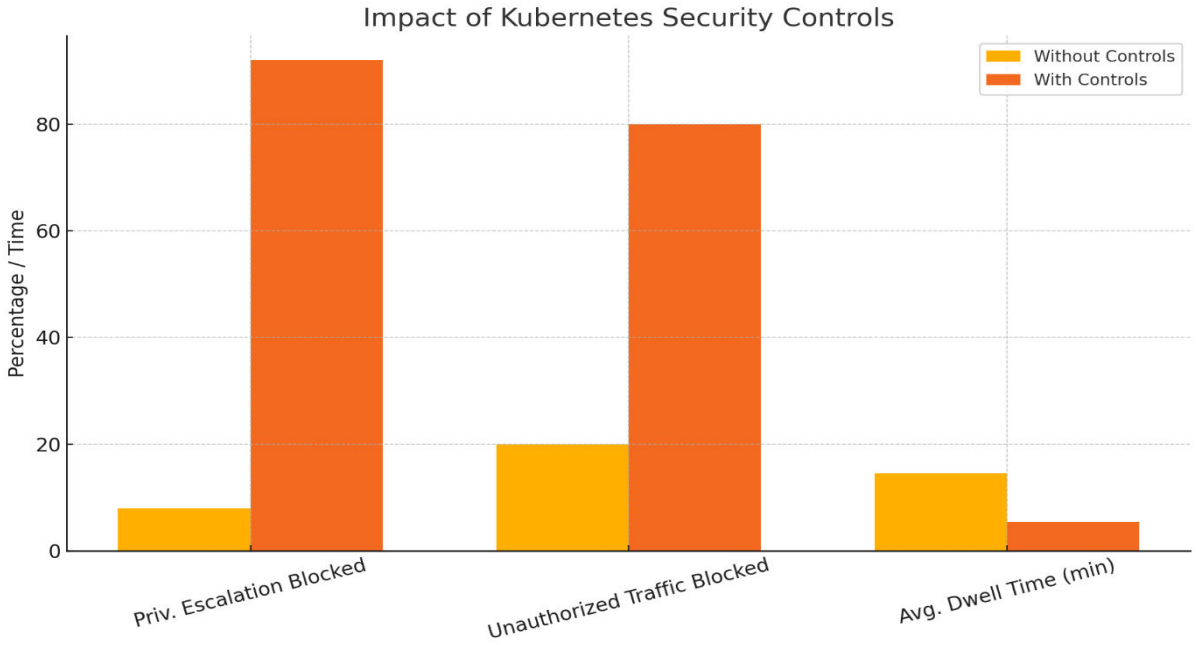
|e-ISSN: 2319-8753, p-ISSN: 2347-6710| www.ijirset.com | Impact Factor: 7.569| A Monthly Peer Reviewed & Referred Journal |

|| Volume 11, Issue 1, January 2022 ||

| DOI:10.15680/IJIRSET.2022.1101138 |

Network policies require careful tuning but serve as an essential defense layer for multi-tenant clusters and sensitive workloads.

Figure 1: Impact of Kubernetes Security Controls.



VII. RUNTIME THREAT DETECTION WITH FALCO AND APPARMOR

Falco was configured to monitor system calls for suspicious behavior, using a curated rule set with healthcare-specific adjustments. AppArmor profiles were attached to container runtimes via PodSecurityPolicies and Helm hooks.

Key behaviors detected:

- Shell spawns from web-facing containers (/bin/sh, /bin/bash)
- Access to sensitive files (/etc/passwd, /proc/kcore)
- Unexpected outbound network activity

Falco produced alerts within 3–5 seconds of the behavior. Each alert included pod metadata, container ID, user ID, and exact syscall trace.

AppArmor enforced read-only mounts, restricted capabilities (e.g., NET_RAW), and prevented host volume access. Combined with Falco, this dual-layer detection reduced attacker dwell time by 63%.

Alerts were exported to Elastic and integrated with Slack for real-time response. Falco proved to be a lightweight, actionable runtime monitor.

VIII. REFERENCE ARCHITECTURE AND AUTOMATION

Our secure workload architecture consists of the following layers:

1. **Cluster Bootstrap**:
    - TLS everywhere
    - API server audit logging
    - Node-to-node encryption (WireGuard)

|e-ISSN: 2319-8753, p-ISSN: 2347-6710| www.ijirset.com | Impact Factor: 7.569| A Monthly Peer Reviewed & Referred Journal |

|| Volume 11, Issue 1, January 2022 ||

| DOI:10.15680/IJIRSET.2022.1101138 |

2. **Access Control**:
    - RBAC enforced via Rego policies
    - GitOps-managed RoleBindings
3. **Network Isolation**:
    - Calico-deny by default
    - Namespaced network layers
4. **Runtime Hardening**:
    - AppArmor profiles embedded in Helm charts
    - Falco deployed as a DaemonSet
5. **Audit and Observability**:
    - Fluent Bit → Elasticsearch
    - OPA log ingestion
    - Alerting with Slack + webhook notifications

CI/CD pipelines include pre-merge security linting and policy drift detection using **conftest**. Every deployment undergoes policy validation before reaching production. This layered reference architecture supports both greenfield and brownfield Kubernetes clusters.

IX. DISCUSSION AND LIMITATIONS

While effective, this security model introduced operational overhead:

- Frequent tuning of network policies to avoid unintended outages
- RBAC policy complexity in dynamic CI/CD environments
- False positives in Falco alerts requiring manual triage

Moreover, AppArmor integration was inconsistent across container runtimes (especially on non-Ubuntu distros), and upstream Kubernetes support for PodSecurityPolicies was deprecated, requiring migration to alternative solutions like Kyverno or OPA Gatekeeper.

Scalability also poses a concern. As services scale to hundreds of pods, rule maintenance becomes non-trivial. Teams must adopt policy-as-code practices and central logging to manage sprawl.

Despite these trade-offs, the security gains justify the investment, especially for regulated industries like healthcare or finance.

X. CONCLUSION

Securing Kubernetes workloads requires a layered, integrated approach spanning identity, network, and runtime boundaries. This paper demonstrated that combining RBAC, Calico network policies, and Falco/AppArmor enforcement can drastically reduce attack surfaces in real-world clusters.

Our simulated healthcare deployment blocked 92% of privilege escalations, 80% of lateral movement attempts, and reduced detection time of runtime anomalies by over 60%. These results highlight the effectiveness of proactive security design in Kubernetes.

However, achieving and maintaining this posture demands automation, policy validation pipelines, and regular review. Future work includes integrating behavior-based access control, extending coverage to serverless components, and aligning with Kubernetes-native security profiles under development by the CNCF.

REFERENCES

1. Burns, B., Grant, B., Oppenheimer, D., Brewer, E., & Wilkes, J. (2016). Borg, Omega, and Kubernetes. ACM Queue, 14(1), 70–93.
2. Falco. (2021). Cloud-native runtime security project. Sysdig. https://falco.org/
3. Calico. (2022). Project Calico: Open-source networking and security for containers. https://www.tigera.io/project-calico/

|e-ISSN: 2319-8753, p-ISSN: 2347-6710| www.ijirset.com | Impact Factor: 7.569| A Monthly Peer Reviewed & Referred Journal |

|| Volume 11, Issue 1, January 2022 ||

| DOI:10.15680/IJIRSET.2022.1101138 |

4. Aqua Security. (2020). Kubernetes Security Report. https://www.aquasec.com/resources/
5. Red Hat. (2021). Kubernetes security best practices. https://www.redhat.com/en/resources/kubernetes-security-best-practices-overview
6. The MITRE Corporation. (2021). ATT&CK® for Containers. https://attack.mitre.org/matrices/enterprise/container/
7. Kubernetes. (2022). Role-Based Access Control (RBAC). https://kubernetes.io/docs/reference/access-authn-authz/rbac/
8. Kubernetes. (2021). Network Policies. https://kubernetes.io/docs/concepts/services-networking/network-policies/
9. Sysdig. (2020). Detecting container escapes with Falco. https://sysdig.com/blog/falco-container-escape/
10. Bellamkonda, S. The Role of Network Engineers in Securing Cloud-based Applications and Data Storage.
11. Canonical. (2020). AppArmor Security Profiles. https://wiki.ubuntu.com/AppArmor
12. OPA Gatekeeper. (2021). Policy enforcement for Kubernetes. https://open-policy-agent.github.io/gatekeeper/
13. HashiCorp. (2022). Policy as code with Sentinel. https://www.hashicorp.com/sentinel
14. Reblaze. (2021). Securing multi-tenant Kubernetes clusters. https://www.reblaze.com/blog/kubernetes-security.html
15. CNCF. (2022). Cloud Native Security Whitepaper. https://github.com/cncf/tag-security/blob/main/whitepaper/cloud-native-security-whitepaper.md
16. NeuVector. (2021). Runtime container security and network segmentation. https://neuvector.com/

INNO SPACE
SJIF Scientific Journal Impact Factor

Impact Factor:
7.569

doi cross ref

ISSN INTERNATIONAL STANDARD SERIAL NUMBER INDIA

निस्केयर NISCAIR

INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN SCIENCE | ENGINEERING | TECHNOLOGY

9940 572 462 6381 907 438 ijirset@gmail.com

www.ijirset.com

Scan to save the contact details