# Containerized Security for ICS/SCADA Systems: From PLC Simulation to Kubernetes-Based Containment

**Janne HALL, PhD**
Division of Information Technology, Jackson State University
Jackson, MS 39217, USA

**April L. TANNER, PhD**
Department of Electrical & Computer Engineering and Computer Science, Jackson State University
Jackson, MS 39217, USA

## ABSTRACT

Industrial Control Systems (ICS) and Supervisory Control and Data Acquisition (SCADA) networks continue to face increasing cybersecurity risks due to their historically flat architectures and limited isolation capabilities. This paper presents a container-native containment validation framework for modern ICS/SCADA environments, leveraging Docker and Kubernetes to simulate adversarial behaviors, including privilege escalation attempts and unauthorized API access, within a segmented, microservice-based infrastructure. Through controlled experiments, we demonstrate how containerization confines potential attacker activity to isolated namespaces, enforces Role-Based Access Control (RBAC), and limits lateral movement, even under permissive runtime conditions. Unlike prior work, this study uniquely integrates Kubernetes-native RBAC enforcement, namespace isolation, and adversarial pod injection with TensorFlow-based telemetry filtering to empirically validate containment strategies in ICS/SCADA workloads. These simulations demonstrate that unlike traditional ICS systems where initial compromise often escalates, container-native architectures offer measurable resilience by preserving operational boundaries and reducing lateral exposure.

**Keywords:** Industrial Control Systems (ICS), Supervisory Control and Data Acquisition (SCADA), Containerization, Kubernetes Security, Privilege Escalation, Containment Validation, Cybersecurity Simulation, Adversarial Workload, Role-Based Access Control (RBAC), Infrastructure Resilience

## 1. INTRODUCTION

ICS and SCADA systems form the operational backbone of critical infrastructure sectors such as energy, water, and manufacturing. Originally engineered for air-gapped environments, these systems now face escalating cybersecurity risks due to increased connectivity, legacy communication protocols, and insufficient authentication mechanisms [1-2]. The convergence of operational technology (OT) and information technology (IT) has significantly expanded the attack surface, exposing ICS/SCADA environments to a broader range of cyber threats, including ransomware, insider attacks, and nation-state campaigns [3].

Advanced Persistent Threats (APTs) have demonstrated the capability to infiltrate ICS networks, manipulate programmable logic controllers (PLCs), and disrupt physical processes using custom-built malware and reconnaissance tools [3]. These threats are no longer hypothetical, real-world incidents such as Triton, Industroyer, and the Oldsmar water treatment breach have underscored the urgent need for proactive, architecture-aware defense strategies [1], [4]. Traditional perimeter-based security approaches, including firewalls and intrusion detection systems, often fail to contain adversarial activity once initial access is gained [5]. This paper addresses that gap by exploring a container-native containment framework that leverages Kubernetes and Docker to simulate, detect, and restrict malicious behavior within ICS/SCADA environments.

### 1.1 Industry Relevance

While academic research has made strides in ICS/SCADA cybersecurity, practical validation environments that reflect container-native deployment models remain scarce [6-7]. Traditional physical testbeds often lack the scalability, reproducibility, and modularity required to emulate sophisticated cyberattack scenarios and containment mechanisms. As a result, security researchers and engineers are limited in their ability to evaluate adversarial resilience in industrial contexts [6].

To address this, both industry and academia are increasingly adopting containerization technologies, such as Docker and Kubernetes to simulate ICS environments with realistic network behavior, system orchestration, and security policy enforcement [8-10]. Despite this shift, few

ISBN: 978-1-950492-85-5
ISSN: 2771-0947



https://doi.org/10.54808/WMSCI2025.01.73

studies offer systematic validation of containment strategies or explore the integration of orchestration-layer security primitives like RBAC, namespace isolation, and runtime confinement. This research addresses that gap by constructing a reproducible, container-native framework for evaluating the containment properties of Kubernetes-based ICS architectures under simulated adversarial conditions.

### 1.2 Threat Model and Testbed Rationale

A key threat vector in containerized environments is the malicious pod—a containerized workload deployed within a Kubernetes cluster to perform unauthorized or harmful actions. These pods may be introduced by attackers who gain initial access through misconfigured APIs, leaked credentials, or compromised CI/CD pipelines [11–12]. Once deployed, a malicious pod can:

- Exfiltrate secrets or environment variables [11], [13]
- Scan internal networks and enumerate services [11], [14]
- Escalate privileges via misconfigured RBAC policies [15]
- Attempt lateral movement or disrupt ICS telemetry [11], [14]

In the context of ICS/SCADA, such a pod could access sensitive telemetry data, interfere with control logic, or issue unauthorized commands to PLCs—posing a direct threat to physical safety and operational continuity [13–14].

This paper addresses a critical research gap: while containerization and orchestration are widely adopted in IT infrastructure [8], their application in ICS/SCADA cybersecurity research remains limited and often lacks empirical validation [16], [17]. To address this, we present a container-native framework that:

- Simulates adversarial workloads (e.g., unauthorized API access, privilege escalation)
- Evaluates Kubernetes' built-in containment mechanisms under realistic ICS conditions
- Enables reproducible testing of RBAC enforcement, namespace isolation, and telemetry integrity
- Supports comparative analysis of containerized vs. traditional ICS architectures in terms of exposure surface and containment guarantees [9], [13], [16–18]

By emphasizing adversarial emulation, policy-driven isolation, and standards-aware design, this work contributes a scalable platform for ICS/SCADA cybersecurity experimentation—one that reflects the persistent vulnerabilities of legacy deployments, including unauthenticated protocols [19], flat network topologies [20], remote third-party access [21], and limited monitoring [22].

## 2. CONTAINER-NATIVE SECURITY FOUNDATION

The adoption of containerized microservices enables industrial systems to emulate, test, and harden ICS workloads in isolated, reproducible environments. Docker provides lightweight packaging of applications and their dependencies, while Kubernetes orchestrates deployment, scaling, and fault recovery across distributed nodes [22-23].

Recent research highlights Kubernetes-native controls, such as RBAC, pod security policies, and namespace-based segmentation as critical mechanisms for enforcing workload isolation and minimizing lateral movement in control system environments [24]. Sanne emphasizes the importance of least-privilege RBAC and supply chain integrity in securing Kubernetes clusters [22], while Elhachmi et al. and Ben Yaala et al. outline practical implementations of image scanning, admission control, TLS encryption, and runtime monitoring to defend against container-level threats [23-24].

These capabilities form the foundation for the containment validation framework presented in this paper, enabling adversarial simulation and policy enforcement in a reproducible, standards-aligned ICS testbed.

This layered approach also facilitates testbed simulation for ICS scenarios. By leveraging lightweight Kubernetes control-plane environments, such as Minikube researchers can construct scalable, self-contained replicas of SCADA systems to observe how detection and containment mechanisms behave under controlled adversarial conditions or stress events.

### 2.1 Evolving ICS Security Models and the Role of Containerization

Although container technologies are widely adopted in cloud-native and enterprise environments, their application to ICS/SCADA cybersecurity simulation and validation remains underrepresented in the literature. Existing ICS simulation platforms often replicate OT traffic and control logic using monolithic virtual machines or static topologies, which limits experimentation, scalability, and dynamic security testing [21].

This paper contributes a Docker and Kubernetes based ICS/SCADA testbed, leveraging Minikube to deploy a multi-tiered, containerized architecture that supports: • Secure, namespace-scoped deployments • Malicious pod simulations to evaluate lateral movement and privilege escalation.

Table 1 Comparative Analysis of ICS/SCADA Security Architectures

| **Approach** | **Advantages** | **Limitations** |
|---|---|---|
| Traditional ICS Security | Strong network segmentation | Lacks automation & scalability |
| IDS-Based Security | Detects anomalies | High false-positive rates |
| Containerized Security (Docker/ Kubernetes) | Scalable, automated, isolated environments | Requires industry adoption |

Table 1 provides a high-level synthesis of how traditional and modern ICS/SCADA security strategies differ in terms of operational strengths and systemic limitations. It highlights the shift from static perimeter defenses to dynamic, containerized architectures that emphasize automation, scalability, and workload isolation.

Recent guidance from NIST and industry analysts emphasizes that while traditional methods remain foundational, they must be augmented with orchestration-layer controls, such as RBAC, runtime policy enforcement, and container isolation to address today's evolving threat landscape [25-28].

## 3. SYSTEM ARCHITECTURE

### 3.1 Setup Overview

The proposed framework integrates Docker, Kubernetes, and ICS/SCADA protocols to construct a scalable, modular industrial security environment. This architecture is purpose-built to simulate and evaluate targeted cyberattack scenarios, specifically, unauthorized API access attempts and malicious pod deployments within a containerized infrastructure.

By leveraging containerization and orchestration, the framework enables researchers to observe how modern ICS networks respond to compromised workloads and privilege abuse. The controlled deployment environment supports dynamic testing without impacting physical devices or production systems.

The core architectural components include:

- Docker Containers – Encapsulate and isolate ICS/SCADA application components (e.g., HMI, PLC simulators, telemetry filters), limiting the blast radius of compromise during API misuse or pod-level privilege escalation.
- Kubernetes Orchestration – Manages deployment, scaling, and service resilience across namespaces. It also facilitates the simulation of unauthorized workloads (e.g., malicious pods) and enforces RBAC policies.
- ICS/SCADA Integration – Preserves communication fidelity with legacy protocols such as Modbus and MQTT, ensuring realism and compatibility with industrial telemetry sources.

### 3.2 System Topology Overview

To construct the containerized ICS/SCADA testbed, we designed a modular system topology that reflects both the architectural hierarchy and network segmentation required for secure industrial deployment. The topology models the interaction between ICS control logic components, Kubernetes orchestration primitives, and simulated adversarial vectors all managed within a containerized environment [22], [24].

At its core, the setup includes:

- Namespace-segmented services such as telemetry filters, PLC simulators, and control interfaces deployed as Docker containers and orchestrated by Kubernetes for isolation and fault tolerance [23-24].
- An Ingress controller manages API routing and external communication, allowing simulation of unauthorized access to exposed endpoints [22].
- A dedicated malicious pod injection pathway enables the controlled deployment of adversarial workloads into isolated namespaces, supporting privilege escalation and lateral movement simulation [9], [18].
- Native support for legacy industrial protocols (e.g., Modbus, MQTT) ensures fidelity in telemetry exchange between containerized field devices and supervisory components [6], [19].

This layered environment enables scalable experimentation, real-time monitoring, and precise evaluation of Kubernetes-native containment strategies under adversarial conditions. The system topology, shown in Figure 1, illustrates a modular ICS/SCADA testbed built on Docker and Kubernetes. It features namespace-segmented services, an Ingress controller for API access modeling, containerized SCADA components (e.g., HMI, PLC, RTU), and support for legacy protocols such as Modbus and MQTT. This architecture enables adversarial simulation, RBAC enforcement, and real-time telemetry analysis in a reproducible, isolated environment [6], [9], [22-24]. Table 2 displays the layer, components, and roles played in the testbed.

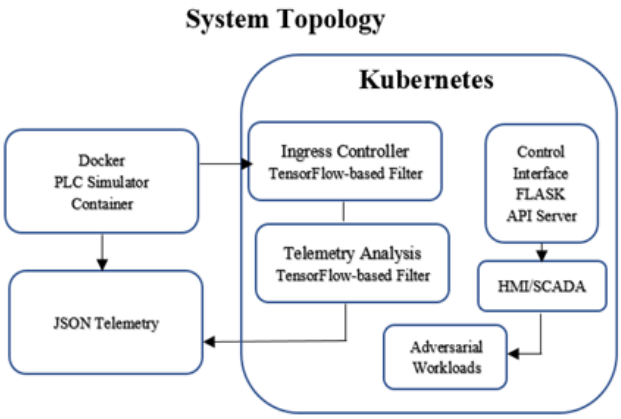


Figure 1. Containerized ICS/SCADA Testbed Topology

Table 2. Testbed Components

| Layer | Component | Role |
|---|---|---|
| **Field Device Simulation** | PLC Simulator (Docker container) | Emits control logic as JSON-formatted telemetry to mimic industrial process states |
| **Telemetry Analysis** | TensorFlow-based filter container | Parses JSON telemetry and performs real-time anomaly scoring via embedded ML model |
| **Control Interface** | Flask API server (accessed via PowerShell/terminal) | Provides RESTful endpoints for interacting with simulated SCADA logic through command-line queries |
| **Protocol Emulation** | JSON-formatted telemetry stream | Emulates industrial communication patterns (e.g., Modbus/MQTT) to preserve operational realism |
| **Orchestration Layer** | Kubernetes (Minikube) | Orchestrates service deployment, namespace segmentation, RBAC enforcement, and container isolation |
| **Ingress Controller** | NGINX or Traefik (Kubernetes Ingress) | Manages external access to services and simulates exposed API pathways for adversarial testing |
| **Adversarial Workloads** | Malicious pods (deployed via YAML manifests) | Evaluate lateral movement, privilege escalation, and namespace containment in post-compromise setups |
| **Monitoring & Logs** | `kubectl logs`, audit logs, Kubernetes Dashboard | Collect runtime logs and container status for post-simulation analysis and validation metrics |

These scenarios reflect common adversarial tactics observed in containerized environments and are modeled to test the containment boundaries of Kubernetes-native security primitives.

## 4. EXPERIMENTAL SETUP

### 4.1 Objective

The objective of this experiment is to assess the effectiveness of containerized security in protecting ICS/SCADA environments. Specifically, the testbed is designed to evaluate how well containerized deployments (Docker + Kubernetes) handle attack resilience, scalability, and isolation under three targeted cyberattack scenarios:

- **Unauthorized API Access Attempts** – Simulating adversarial requests to Kubernetes APIs to test RBAC enforcement and audit logging.
- **Malicious Pod Deployment** – Introducing unauthorized workloads to observe namespace isolation, network segmentation, and runtime behavior.
- **Privilege Escalation via RBAC Misconfiguration** – Evaluating how over-permissive roles or service account bindings can be exploited to gain elevated access within the cluster [6], [9], [24].

### 4.2 Setup Details

#### 4.2.1 Test Environment

- Dockerized ICS/SCADA Stack Simulated control system components, including PLCs, RTUs, HMIs, and a telemetry filtering layer are containerized using Docker to ensure modularity and isolation [36], [38].
- Kubernetes Orchestration (via Minikube)
  - Manages container lifecycle and service discovery
  - Enforces security primitives such as namespaces, RBAC, and Network Policies [16], [17], [20]
  - Deploys and scales the alarm filtration module and SCADA telemetry services
- Ingress Controller + API Gateway
  - Routes traffic to designated API endpoints for filtration and telemetry ingestion
  - Serves as a chokepoint to simulate unauthorized access attempts and enforce ingress-layer policies [26], [28]
- Attack Injection Simulation
  - External scripts issue malformed or unauthorized API POST requests to test RBAC enforcement and audit logging
  - A secondary malicious pod is intentionally deployed into a namespace with minimal isolation controls to

simulate insecure configurations and evaluate the testbed's containment resilience under permissive security settings [9], [46], [48]

Repeatability & Scalability

- Minikube enables isolated, reproducible cluster creation for each test cycle [17]
- YAML manifests define all deployments, services, and attack vectors, ensuring version-controlled reproducibility [8], [9]
- Telemetry and logs are collected using kubectl logs and custom instrumentation for post-simulation analysis [6], [10]

**4.3 Metrics Evaluated** This evaluation focuses on how effectively the containerized ICS/SCADA testbed handles security events, workload stress, and privilege containment under simulated adversarial conditions. The metrics were selected to reflect core performance and security attributes of Kubernetes-native architectures in industrial environments. These metrics were also selected to reflect core security principles, such as isolation integrity, RBAC enforcement, and policy consistency under adversarial workloads and repeatable attack scenarios. Table 3 summarizes the key evaluation criteria used to assess the containment and observability capabilities of the Kubernetes-based ICS/SCADA testbed.

**Table 3. Evaluated Parameters**

| Metric | Description |
|---|---|
| Containment Success Rate | Frequency with which adversarial pods were blocked from privilege escalation or lateral movement |
| RBAC Enforcement Accuracy | Degree to which access was denied for unauthorized service accounts or API requests |
| Namespace Isolation Integrity | Whether workloads in isolated namespaces were protected from cross-namespace traffic or role inheritance |
| Telemetry Coverage | Visibility into attack attempts via Kubernetes logs and instrumentation |
| Policy Enforcement Consistency | Observed consistency in application of Network Policies and admission controllers across test iterations |
| Testbed Reproducibility | Success in recreating attack conditions, results, and cluster behavior across redeployments |

**4.4 Attack Scenarios**

**4.4.1 Deploying a Malicious Pod Attack** To emulate a post-compromise attack scenario within the Kubernetes environment, we introduced a deliberately misconfigured workload via a manually created pod file: malicious-pod.yaml. This manifest simulates adversarial behavior following unauthorized access to the cluster. Figure 2 shows a deployed alarm-filter pod visualized in Kubernetes Dashboard prior to deploying the malicious-pod.

The pod executes a TensorFlow-based anomaly detection pipeline, processing JSON-formatted telemetry emitted by the simulated PLC container and Figure 3 is a display of the partial YAML manifest.

The YAML file provisions a single BusyBox container that performs enumeration tasks commonly observed in real-world Kubernetes attacks. Upon deployment, the container attempts to:

- Enumerate environment variables to extract sensitive data
- Access mounted service account tokens typically used to authenticate with the Kubernetes API
- Maintain persistence using a passive sleep loop

In addition, the pod is configured with elevated privileges, running as the root user with privileged: true enabled, to assess the system's response to over-privileged containers. This configuration enables the study of lateral movement risks and privilege escalation within namespace boundaries.

Deploying this pod within the testbed allows us to evaluate the effectiveness of RBAC policies, namespace isolation, and pod-level security controls in containing unauthorized workloads and limiting blast radius.

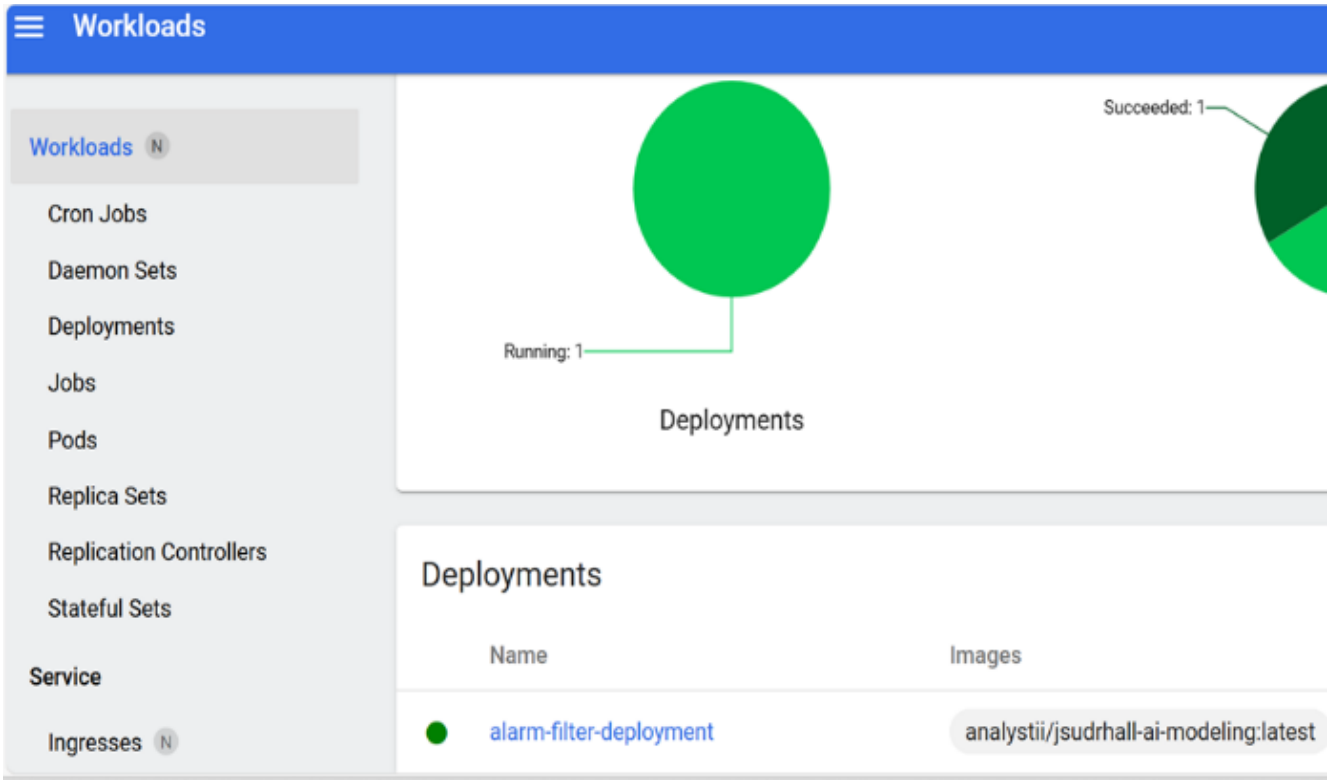


Figure 2. Alarm-Filter Image Deployed in the Kubernetes Dashboard (via Minikube)

```
apiVersion: v1
kind: Pod
metadata:
  name: malicious-pod
  namespace: default
  labels:
    app: suspicious-container
spec:
  containers:
    - name: busybox
      image: busybox
      command:
```

Figure 3. Malicious-Pod Yaml Partial Manifest

The container image used in this simulation is based on BusyBox, a minimalist Linux utility suite commonly referred to as the "Swiss Army knife" of embedded systems [30]. BusyBox consolidates numerous standard UNIX utilities such as sh, env, cat, and ls into a single lightweight binary, making it ideal for rapid deployment in constrained environments or adversarial simulations. In our setup, BusyBox serves as a compact yet functional platform for emulating attacker behavior, including environment enumeration and token harvesting. Its small footprint (typically under 1 MB) allows it to be pulled and executed quickly within Kubernetes clusters without requiring custom image builds [30]. This makes it a popular choice for both legitimate system diagnostics and malicious post-compromise activity in containerized environments [30-32].

**4.4.2 Deployment Confirmation** Terminal output confirms the successful deployment of the malicious-pod using the kubectl apply command. This step validates the application of the declarative YAML manifest and signals that the testbed is ready for adversarial emulation scenarios within the targeted namespace. This marks the entry point of the simulated attack, representing the initial post-compromise action within the Kubernetes environment.

The Kubernetes Dashboard view of the active malicious-pod within the designated namespace is shown in Figure 4. Metadata fields such as image source (BusyBox), labels, and runtime status (Running) reflect the pod's operational presence and its ability to blend into typical workload activity, a characteristic often leveraged in real-world containerized attacks is shown in Figure 5 *(note-IP address was redacted)*.

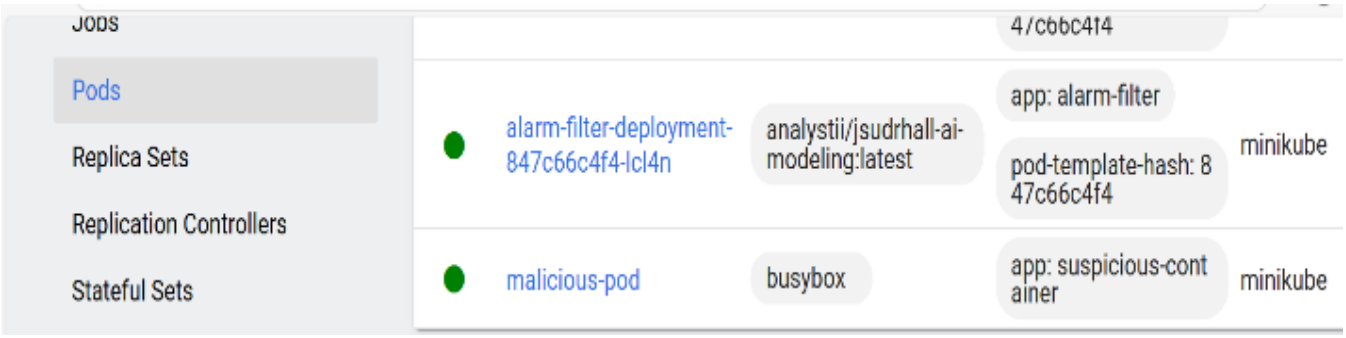


Figure 4. Malicious-Pod Deployed in Kubernetes (via Minikube)

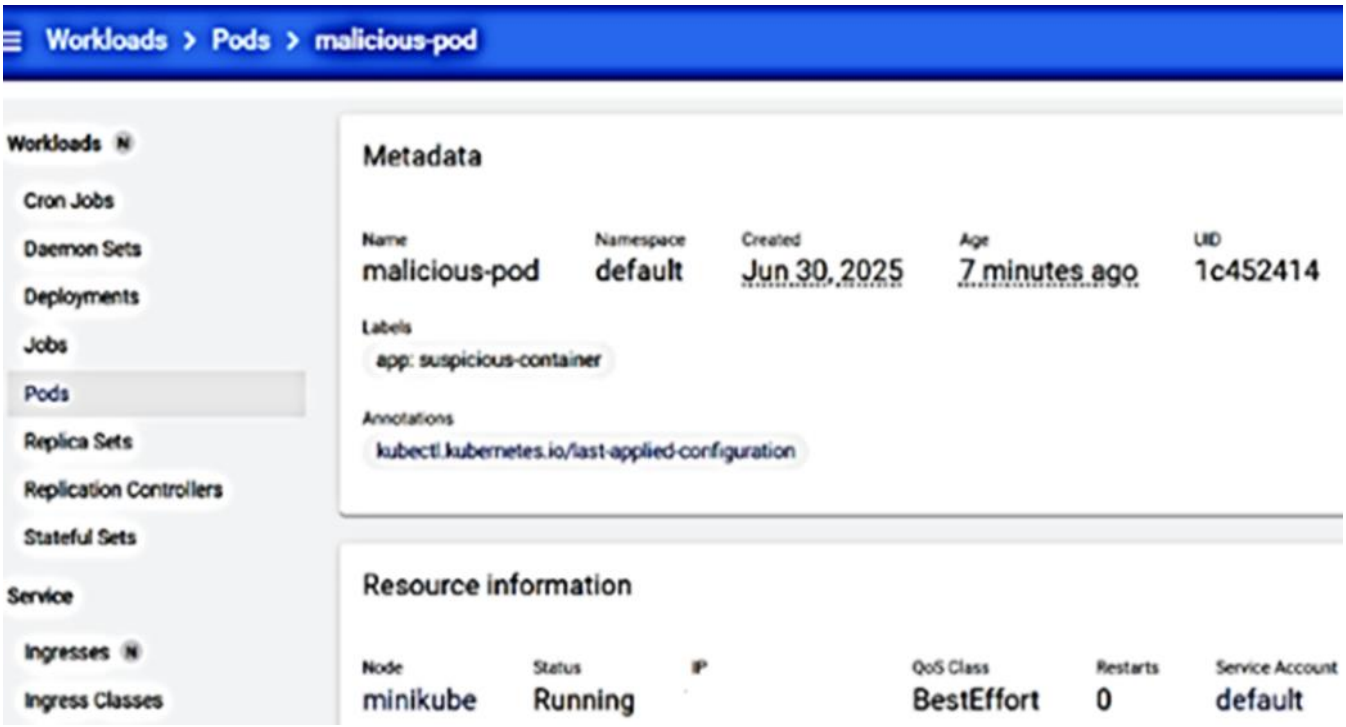


Figure 5. Malicious-Pod Metadata Fields

The log output from the malicious pod showing post-compromise enumeration behavior is shown in Figure 6 *(note-Port numbers and IP addresses were redacted).* Although the pod lacks direct control over ICS components like alarm-filter-deployment, its access to environment metadata and service information demonstrates how minimal privileges can reveal cluster topology.

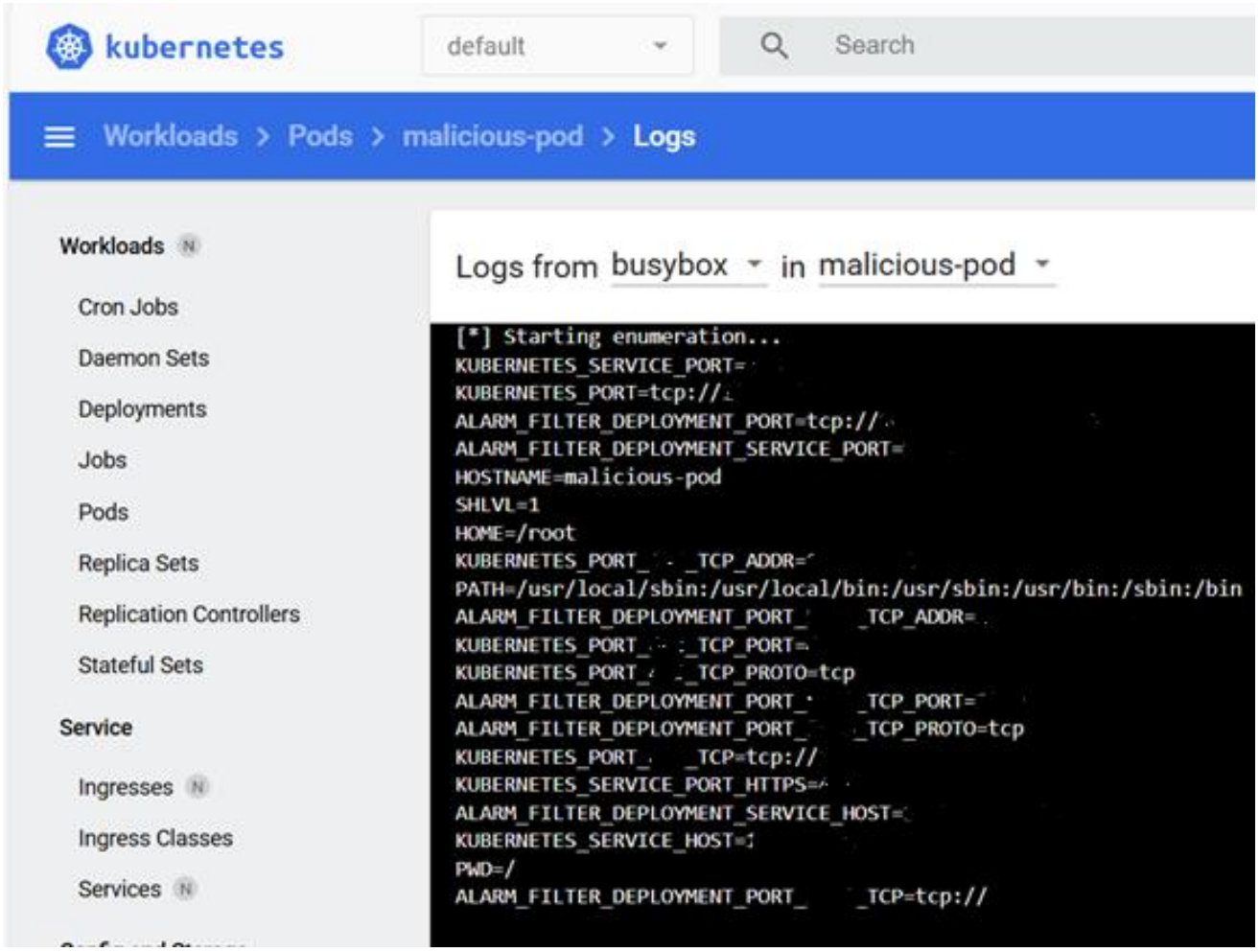


Figure 6. Logs from Busybox in Malicious-Pod

**4.4.4 Privilege Escalation** Following the enumeration performed by the malicious-pod, an adversary gains visibility into environment variables and potentially exposed service account tokens.
This reconnaissance phase yields critical metadata that can be repurposed in a subsequent privilege escalation attempt, mirroring how attackers chain access to deepen compromise.

In the following scenario, we explore how this exposed configuration data informs a misconfigured pod deployment intended to elevate privileges within the container runtime. To simulate this escalation attempt, we deploy a new pod named escalation-contained using a modified YAML manifest, shown in Figure 7. We used the command *kubectl apply -f escalation-contained.yml* for deployment, which triggered the creation of the escalation-contained pod within the designated namespace. This action validated the configuration of our containment scenario and served as the initiating step in the privilege escalation test sequence. The pod is configured with elevated runtime parameters (runAsUser: 0, privileged: true) and attempts to mount the host system's /etc directory. While the container successfully runs with root privileges and simulates host-level access, it remains confined within Kubernetes security boundaries, highlighting the limitations of such privilege escalation efforts in a properly isolated containerized environment.

```
1  apiVersion: v1
2  kind: Pod
3  metadata:
4    name: escalation-contained
5    namespace: malicious-pod
6  spec:
7    containers:
8      - name: escalate
9        image: busybox
10       command:
```

Figure 7. Escalation-Contained Yaml File

Figure 8 shows a partial log output from escalation-contained simulating privilege escalation through a hostPath volume mount and root user execution. The attacker successfully lists 636 files under /host/etc, suggesting local escalation. However, this visibility does not translate into actionable control beyond the container boundary, highlighting how Kubernetes-native isolation mechanisms contain compromise even under permissive runtime conditions.

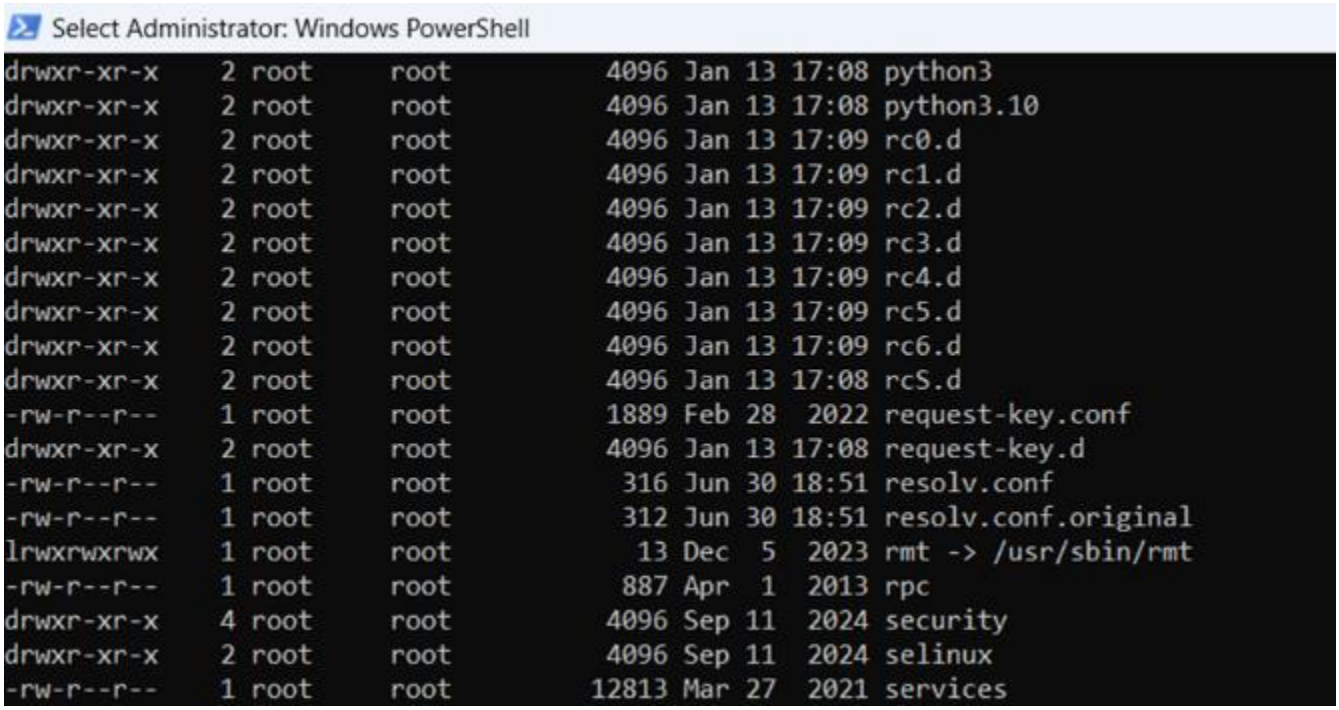


Figure 8. Kubectl Logs Escalation-Contained -n Malicious-Pod

We used the command *kubectl delete pod escalation-contained -n malicious-pod* to terminate the simulated privilege escalation workload. The successful execution of this command confirmed its removal, which was immediately reflected in the Kubernetes Dashboard where the escalation-contained pod no longer appeared. This action illustrates how malicious workloads can be surgically removed without disrupting neighboring ICS services or compromising overall system stability.

Despite deliberate misconfiguration to simulate privilege escalation, the malicious pod was constrained within its operational scope. ICS workloads remained unaffected, highlighting container-native controls as effective mechanisms against lateral movement or cross-service compromise.

**4.4.5 Unauthorized API Access Attack** In Kubernetes-based ICS/SCADA environments, the Kubernetes API server represents a critical control plane component. If improperly secured, it can become a high-value target for adversaries seeking to enumerate cluster resources, escalate privileges, or manipulate workloads. In traditional ICS systems, attackers often exploit flat network topologies and shared credentials to gain unrestricted access to control logic and device configurations. By contrast, Kubernetes introduces RBAC, namespace isolation, and tightly scoped service accounts to limit unauthorized API interactions.

This scenario simulates a malicious pod attempting to access the Kubernetes API using a default-mounted service account token. The goal is to evaluate whether the containerized environment enforces access boundaries that prevent unauthorized enumeration or manipulation of cluster resources. The outcome of this simulation reinforces the importance of default-deny RBAC policies and the principle of least privilege in container-native ICS deployments.

Recent studies have highlighted the risks of exposing the Kubernetes API server without proper authentication and authorization mechanisms [31-32]. Moreover, systematized security practices recommend disabling anonymous access, enforcing RBAC, and auditing API interactions to prevent lateral movement and privilege escalation within the cluster [33].

The partial Kubernetes manifest (unauthorized-api-pod.yaml) shown in Figure 9, defines the api-probe-pod in the malicious-pod namespace. The pod uses the curlimages/curl:latest container to simulate an adversarial workload attempting unauthorized access to the Kubernetes API server. The image provides a minimal, secure environment with the curl binary and trusted root certificates, enabling HTTPS-based API probing without additional dependencies [34-36].

```
apiVersion: v1
kind: Pod
metadata:
  name: api-probe-pod
  namespace: malicious-pod
spec:
  containers:
    - name: probe
      image: curlimages/curl:latest
      imagePullPolicy: IfNotPresent
      command:
```

Figure 9. Partial Manifest Unauthorized-api-pod.yaml

The api-probe-pod was deployed in the malicious-pod namespace using a declarative YAML manifest. This container simulates a non-privileged adversary attempting to enumerate resources exposed by the Kubernetes API server, forming a key component of the early reconnaissance phase in the testbed's threat simulation workflow.

Log output from the api-probe-pod reveals an attempted GET request to /api/v1/namespaces, which is denied by Kubernetes with a 403 Forbidden error. This response confirms that the service account associated with the pod lacks the necessary cluster-scoped read permissions, validating the effectiveness of the defined RBAC policies.

The unauthorized API access simulation successfully demonstrated the enforcement of Kubernetes-native access boundaries within the containerized ICS security framework. Upon deployment, the api-probe-pod attempted to retrieve namespace data using the service account token mounted by default.

As expected, the Kubernetes API server returned a 403 Forbidden response, indicating that the requestor lacked the necessary RBAC permissions to perform cluster-scoped enumeration. This outcome validates the effectiveness of default RBAC configurations and namespace isolation in preventing adversarial workloads from discovering or manipulating system-wide resources. Unlike traditional ICS architectures that often lack comparable access controls, the containerized environment offers measurable resistance to lateral movement and privilege reconnaissance.

## 5. RESULTS

The simulated adversarial scenarios confirmed the enforcement of Kubernetes-native boundaries under realistic attack conditions. The api-probe-pod was unable to enumerate cluster namespaces due to RBAC constraints, while the escalation-contained pod demonstrated that even a container running as root with privileged: true remained confined within its designated namespace. These findings reinforce earlier studies comparing container security frameworks and demonstrate reproducible, policy-driven isolation.

Several evaluated metrics from Table 3 directly align with these observed behaviors. The high Containment Success Rate was reflected in the inability of adversarial pods to escalate privileges or move laterally. RBAC Enforcement Accuracy was demonstrated by the denial of unauthorized API requests from the api-probe-pod, while Namespace Isolation Integrity was validated by the strict confinement of the escalation-contained pod.

These results underscore the practical containment of privilege escalation in ICS contexts demonstrating that Kubernetes-native RBAC and namespace isolation can enforce security boundaries even under adversarial conditions.

## 6. COMPARISON ANALYSIS: TRADITIONAL ICS VS. CONTAINERIZED ARCHITECTURE

Traditional ICS/SCADA systems frequently rely on monolithic architectures, minimal runtime isolation, and static access control mechanisms. These designs often leave entire networks exposed once a single component is compromised. By contrast, the containerized framework explored in this study introduces namespace isolation, RBAC enforcement, and fine-grained resource controls, which greatly reduce the attacker's operational freedom and lateral movement capabilities [37-38]. Table 4 shows the comparison analysis between traditional ICS vs. containerized architecture.

These results underscore the architectural advantages of Kubernetes when adapted for ICS use cases. The automated denial of unauthorized API access, for example, provides an actionable baseline for containment validation that is rarely feasible in legacy infrastructure [39], [41].

Table 4. Comparison Analysis: Traditional ICS vs. Containerized Architecture

| **Feature** | **Traditional ICS Architecture** | **Kubernetes-Based Containerized ICS** |
|---|---|---|
| Network Segmentation | Often flat or VLAN-based, with limited micro segmentation [43] | Enforced via namespaces and Network Policies [40] |
| Access Control | Role-based, but often coarse-grained and hardcoded into devices [43] | Fine-grained RBAC with scoped service accounts [39] |
| Runtime Isolation | Minimal; processes often share OS and memory space [43] | Strong isolation via Linux namespaces, cgroups, and container boundaries [41] |
| Attack Surface | Broad; compromise of one node often exposes entire system [43] | Narrow: per-pod and per-namespace exposure limits lateral movement [39-40] |
| Response to Unauthorized Access | Often undetected or delayed due to lack of centralized logging [43] | Immediate denial with audit logging and event tracking [40] |
| Reproducibility of Tests | Difficult due to hardware dependencies and vendor-specific firmware [43] | High, using declarative YAML manifests and containerized testbeds [42] |

## 7. CONCLUSION

This research validates that container-native mechanisms, particularly RBAC enforcement, namespace partitioning, and pod-level runtime controls, provide actionable containment guarantees in ICS/SCADA environments [39-40]. The experiments show that even when simulated adversarial pods are launched with elevated privileges or token access, Kubernetes' security primitives curtail their ability to enumerate, traverse, or disrupt unrelated workloads. This level of isolation is rarely achievable in traditional ICS infrastructure, where control logic and network privileges are often shared broadly [37]. By building a reproducible and extensible containment framework, this work contributes a reproducible and extensible testbed that supports ICS adversarial emulation, facilitates automated security auditing, and enables rigorous policy validation in orchestrated cyberattack scenarios.

## 8. FUTURE WORK

To expand the practical relevance and research value of this containment framework, future work will focus on the following dimensions:

- Integration of Behavior-Based Telemetry Filtering: Building on the current alarm-filter module, we will implement behavior-based anomaly detection pipelines that adapt to evolving adversarial patterns. This includes dynamic model retraining using real-time telemetry and adversarial injection data, supporting predictive classification across diverse ICS workloads.

- Live Incident Emulation and Blue Team Validation: Future iterations will incorporate blue team defense scenarios such as intrusion alerts, response workflows, and forensic evidence collection. This bidirectional model aims to validate not only attacker containment, but also defender agility under orchestrated attack conditions.

- Framework Interoperability with NIST SP 800-82r3 and IEC 62443: By aligning containment metrics with the newly updated NIST SP 800-82 Rev. 3 and IEC 62443-4-2 component-level controls, future versions will support regulatory compliance and security maturity scoring.

- Attack Replay and Scenario Benchmarking: We will expand the YAML manifest library to include repeatable attack templates modeled on MITRE ATT&CK for ICS. Each attack will be benchmarked for containment performance, telemetry fidelity, and detection latency, enabling standardized evaluation across research environments.

## 9. REFERENCES

[1] IEEE Public Safety Technology, "Cybersecurity of Critical Infrastructure with ICS/SCADA Systems," *IEEE*, 2021. [Online]. Available: https://publicsafety.ieee.org/topics/cybersecurity-of-critical-infrastructure-with-ics-scada-systems/

[2] The Claroty Team, "ICS/SCADA Vulnerability Management: Proactive Strategies for Cyber Resilience," *Claroty*, Mar. 24, 2024. https://claroty.com/blog/ics-scada-vulnerability-management-proactive-strategies-for-cyber-resilience

[3] Cybersecurity and Infrastructure Security Agency (CISA), "APT Cyber Tools Targeting ICS/SCADA Devices," *CISA*, 2022. [Online]. Available: https://www.cisa.gov/news-events/cybersecurity-advisories/aa22-103a

[4] A. Hahn, A. Ashok, S. Sridhar, and M. Govindarasu, "Cyber-Physical Security Testbeds: Architecture, Application, and Evaluation for Smart Grid," *IEEE Trans. Smart Grid*, vol. 4, no. 2, pp. 847–855, 2013. doi: 10.1109/TSG.2012.2226919

[5] A. S. Syed, M. A. Khan, and M. A. Jan, "Security Challenges in Kubernetes: A Survey," *IEEE Access*, vol. 9, pp. 140–157, 2021. doi: 10.1109/ACCESS.2021.3051234

[6] E. Sitnikova, E. Foo, and R. B. Vaughn, "The Power of Hands-On Exercises in SCADA Cyber Security Education," Information Assurance and Security Education and Training, pp. 83–94, 2013, doi: https://doi.org/10.1007/978-3-642-39377-8_9.

[7] "ICS/SCADA Cybersecurity," EC-Council. https://www.eccouncil.org/train-certify/ics-scada-cybersecurity/ (accessed Jun. 27, 2025).

[8] G. Shekhar, "The Role of Containers and Orchestration in Scalable System Design," ESP Journal of Engineering & Technology Advancements, vol. 3, no. 4, Dec. 2023, doi: https://doi.org/10.56472/25832646/JETA-V3I8P107.

[9] A. P. Bhuyan, "Kubernetes vs Docker Swarm: Ultimate Container Orchestration Comparison," DEV Community, Feb. 23, 2025. https://dev.to/adityabhuyan/kubernetes-vs-docker-swarm-ultimate-container-orchestration-comparison-19m3 (accessed Jun. 27, 2025).

[10] "Kubernetes vs Docker: Understand the key differences, use cases, and how they work together to streamline container management and deployment.," Lucidity.cloud, 2025. https://www.lucidity.cloud/blog/kubernetes-vs-docker (accessed Jun. 27, 2025).

[11] "Escape to Host, Technique T1611 - Enterprise | MITRE ATT&CK®," *attack.mitre.org*. https://attack.mitre.org/techniques/T1611/ (accessed Jun. 27, 2025).

[12] Conner Buell , "Beyond the Top 5: ICS/SCADA IT/OT Security," *Redbot Security*, May 06, 2024. https://redbotsecurity.com/beyond-the-top-5-ics-scada-it-ot-security/ (accessed Jun. 27, 2025).

[13] Georgia Tech & Sandia National Labs. "SCAPHY: Detecting Modern ICS Attacks by Correlating Behaviors in SCADA and Physical Layers." *arXiv:2211.14642*, 2022. [Online]. Available: https://arxiv.org/pdf/2211.14642

[14] M. Maheswari, "CYBER THREAT ANALYTICS OF ICS/SCADA SYSTEMS USING QATD ALGORITHM," *IJARCCE International Journal of Advanced Research in Computer and Communication Engineering Impact Factor 8.102 ⁄Peer-reviewed & Refereed journal ⁄Vol*, vol. 14, no. 5, 2025, doi: https://doi.org/10.17148/IJARCCE.2025.14539.

[15] "KodeKloud Notes," *KodeKloud Notes*, 2025. https://notes.kodekloud.com/docs/Kubernetes-and-Cloud-Native-Security-Associate-KCSA/Kubernetes-Threat-Model/Compromised-Applications-in-Containers (accessed Jun. 27, 2025).

[16] ICSSIM Project, "Simulating Industrial Control System Testbed for Cybersecurity Experiments," GitHub, 2025. https://github.com/AlirezaDehlaghi/ICSSIM

[17] C. Lo et al., "TRIST: Towards a Container-Based ICS Testbed for Cyber Threat Simulation and Anomaly Detection," Cyber Science 2024, Springer, Apr. 2025.

[18] KodeKloud, "Compromised Applications in Containers – Kubernetes Threat Model," KCSA Training Portal, 2025.

[19] ISA, "Understanding and Applying the ANSI/ISA-18.2 Alarm Management Standard," *ISA.org*, 2009. https://www.isa.org/getmedia/55b4210e-6cb2-4de4-89f8-2b5b6b46d954/PAS-Understanding-ISA-18-2.pdf

[20] Siemens, "Setting a New Standard in Alarm Management," *White Paper*, 2021. https://assets.new.siemens.com/siemens/assets/api/uuid:234f1026-298f-49dc-8961-0c5223c38588/white-paper-alarm-management-2021-final.pdf

[21] D. Kostadinov, "ICS/SCADA Intrusion Detection & Prevention," *Infosec Institute*, Apr. 28, 2020.

[22] M. Gaiceanu et al., "Intrusion Detection on ICS and SCADA Networks," *Studies in Systems, Decision and Control*, vol. 255, 2019.

[23] S. Ali, "Security in SCADA System: A Technical Report on Cyber Attacks and Risk Assessment Methodologies," *Lecture Notes in Networks and Systems*, vol. 935, 2024.

[24] S. H. V. Sanne, "Techniques for Securing and Hardening Kubernetes Clusters," *International Journal of Core Engineering & Management*, vol. 6, no. 12, 2021.

[25] J. Elhachmi, S. Laaouamir, and I. Ziad, "Securing Containerized Workloads in Kubernetes," *SSRN*, Mar. 2025.

[26] S. Ben Yaala et al., "Enhancing Kubernetes Security," *Advanced Information Networking and Applications*, Apr. 2025.

[27] "What Are the Differences Between OT, ICS, & SCADA Security?," *Palo Alto Networks*. https://www.paloaltonetworks.com/cyberpedia/ot-vs-ics-vs-scada-security

[28] K. A. Stouffer, V. Y. Pillitteri, S. Lightman, M. Abrams, and A. Hahn, "Guide to Industrial Control Systems (ICS) Security," *NIST*, Jun. 2015, Available: https://www.nist.gov/publications/guide-industrial-control-systems-ics-security

[29] "ICS/SCADA security overview | Infosec," *Infosecinstitute.com*, 2019. https://www.infosecinstitute.com/resources/scada-ics-security/ics-scada-security-overview/ (accessed Jun. 29, 2025).

[30] BusyBox Project, "BusyBox: The Swiss Army Knife of Embedded Linux," [Online]. Available: https://www.busybox.net/BusyBox.html

[31] Docker Inc., "How to Use the BusyBox Docker Official Image," *Docker Blog*, 2023. [Online]. Available: https://www.docker.com/blog/use-cases-and-tips-for-using-the-busybox-docker-official-image/

[32] S. Kamani, "Using the BusyBox Docker Image for Building Applications," *Soham Kamani Blog*, 2022.

[Online]. Available: https://www.sohamkamani.com/docker/busybox-guide/
[33] M. S. I. Shamim, F. A. Bhuiyan, and A. Rahman, "XI Commandments of Kubernetes Security: A Systemization of Knowledge Related to Kubernetes Security Practices," *IEEE SecDev*, 2020. [Online]. Available: http://secdev.ieee.org/wp-content/uploads/2020/11/s4-02-shamim.pdf
[34] A. S. Syed, M. A. Khan, and M. A. Jan, "Security Challenges in Kubernetes: A Survey," *IEEE Access*, vol. 9, pp. 140–157, 2021. doi: 10.1109/ACCESS.2021.3051234
[35] "Enhancing Kubernetes Security: Comprehensive Strategies for a Robust Cluster," in *Lecture Notes in Computer Science*, Springer, 2023. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-031-87784-1_20
[36] curlimages, "Official curl Docker Image," GitHub, 2024. [Online]. Available: https://github.com/curl/curl-docker
[37] Docker Hub, "curlimages/curl," Docker Official Image Registry, 2024. [Online]. Available: https://hub.docker.com/r/curlimages/curl
[38] J. Fuller, "curl/curl-container: Infrastructure for Official curl Docker Images," GitHub, 2024. [Online]. Available: https://github.com/curl/curl-container
[39] M. S. I. Shamim, F. A. Bhuiyan, and A. Rahman, "XI Commandments of Kubernetes Security: A Systemization of Knowledge Related to Kubernetes Security Practices," *IEEE SecDev*, 2020. [Online]. Available: http://secdev.ieee.org/wp-content/uploads/2020/11/s4-02-shamim.pdf
[40] A. S. Syed, M. A. Khan, and M. A. Jan, "Security Challenges in Kubernetes: A Survey," *IEEE Access*, vol. 9, pp. 140–157, 2021. doi: 10.1109/ACCESS.2021.3051234
[41] A. V. Karpov et al., "A Comparison of Kubernetes and Kubernetes-Compatible Platforms," *2021 11th IEEE IDAACS*, pp. 1–6, 2021. doi: 10.1109/IDAACS53288.2021.9660392
[42] G. Kambala, "Cloud-Native Architectures: A Comparative Analysis of Kubernetes and Serverless Computing," *JETIR*, vol. 10, no. 4, pp. 1–10, 2023. [Online]. Available: https://www.jetir.org/papers/JETIR2304F29.pdf
[43] A. Hahn, A. Ashok, S. Sridhar, and M. Govindarasu, "Cyber-Physical Security Testbeds: Architecture, Application, and Evaluation for Smart Grid," *IEEE Transactions on Smart Grid*, vol. 4, no. 2, pp. 847–855, 2013. doi: 10.1109/TSG.2012.2226919