The Eastasouth Journal of Information System and Computer Science
Vol. 1, No. 01, August, pp. 156 - 168

Systematic Enforcement of CIS-Aligned Security Controls for Kubernetes Worker Nodes

Balaramakrishna Alti
AVP Systems Engineering, USA

Article Info

Article history:

Received Aug, 2023
Revised Aug, 2023
Accepted Aug, 2023

Keywords:

Automation;
CIS compliance;
Drift detection;
Linux hardening;
Worker node security

ABSTRACT

Securing Kubernetes worker nodes remains a persistent challenge in enterprise environments due to configuration drift, inconsistent operating system hardening, and limited visibility into runtime security posture. While the Center for Internet Security (CIS) provides benchmark recommendations for Kubernetes and Linux systems, manual enforcement of these controls is error-prone and difficult to sustain at scale. This paper presents an automated approach for hardening Kubernetes worker nodes by integrating CIS benchmark compliance with Linux security controls using configuration management automation. The proposed framework focuses on repeatable enforcement, continuous compliance validation, and operational stability. We describe the system architecture, control mapping strategy, and automation workflow, and evaluate its impact on configuration compliance and operational availability in a controlled Kubernetes environment. Results demonstrate measurable improvements in benchmark compliance while maintaining cluster stability, highlighting the feasibility of automation-driven security hardening for Kubernetes worker nodes.

Corresponding Author:

Name: Balaramakrishna Alti
Institution: AVP Systems Engineering, USA
Email: balaramaa@gmail.com

1. INTRODUCTION

Kubernetes has become the de facto platform for container orchestration in enterprise environments, enabling scalable deployment of cloud-native applications across on-premise and hybrid infrastructures [1]–[6]. While Kubernetes provides built-in mechanisms for scheduling, service discovery, and fault tolerance, the security of the underlying worker nodes remains a critical and frequently underestimated concern [7], [8]. Worker nodes host application workloads, container runtimes, and node-level services such as kubelet, making them a high-value target for attackers seeking lateral movement or privilege escalation within a cluster [9]–[11].

Industry guidelines such as the Center for Internet Security (CIS) Benchmarks provide prescriptive recommendations for securing both Kubernetes components and Linux operating systems [12]. However, in practice, enforcing these benchmarks consistently across worker nodes is challenging. Enterprise clusters often suffer from configuration drift due to manual changes, inconsistent patching practices, and variations in node provisioning workflows [13], [14]. As clusters scale, manual hardening becomes error-prone and operationally unsustainable, increasing the risk of

misconfigurations that weaken the overall security posture [15]–[17].

Automation is widely recognized as a key enabler for scalable infrastructure management, yet its application to Kubernetes worker node hardening is often fragmented [18]. Organizations may apply CIS benchmarks to Kubernetes control plane components while leaving operating system–level controls partially enforced or unmanaged [19], [20]. This gap creates a layered security weakness where compliance at the orchestration layer does not guarantee protection at the host operating system layer.

This paper presents an automated framework for securing Kubernetes worker nodes by integrating CIS benchmark recommendations with Linux operating system hardening controls. The proposed approach emphasizes repeatable enforcement, continuous compliance verification, and minimal operational disruption. Rather than introducing new security mechanisms, the framework systematically maps benchmark controls to enforceable Linux configurations and applies them using automation to reduce human error and configuration drift.

The primary contributions of this paper are as follows:

1. A structured analysis of Kubernetes worker node attack surfaces and their relationship to Linux operating system configurations.
2. A control-mapping strategy that aligns CIS Kubernetes and Linux benchmarks with enforceable system-level hardening actions.
3. An automation-driven framework for applying, validating, and maintaining security controls while preserving cluster availability.

The remainder of this paper is organized as follows. Section II reviews relevant background concepts and related work. Section III defines the threat model and problem scope. Section IV details the control mapping and hardening strategy. Section V describes the automation framework. Section VI outlines the experimental setup, followed by evaluation results in Section VII. Section VIII discusses limitations and operational considerations, and Section IX concludes the paper with directions for future research.

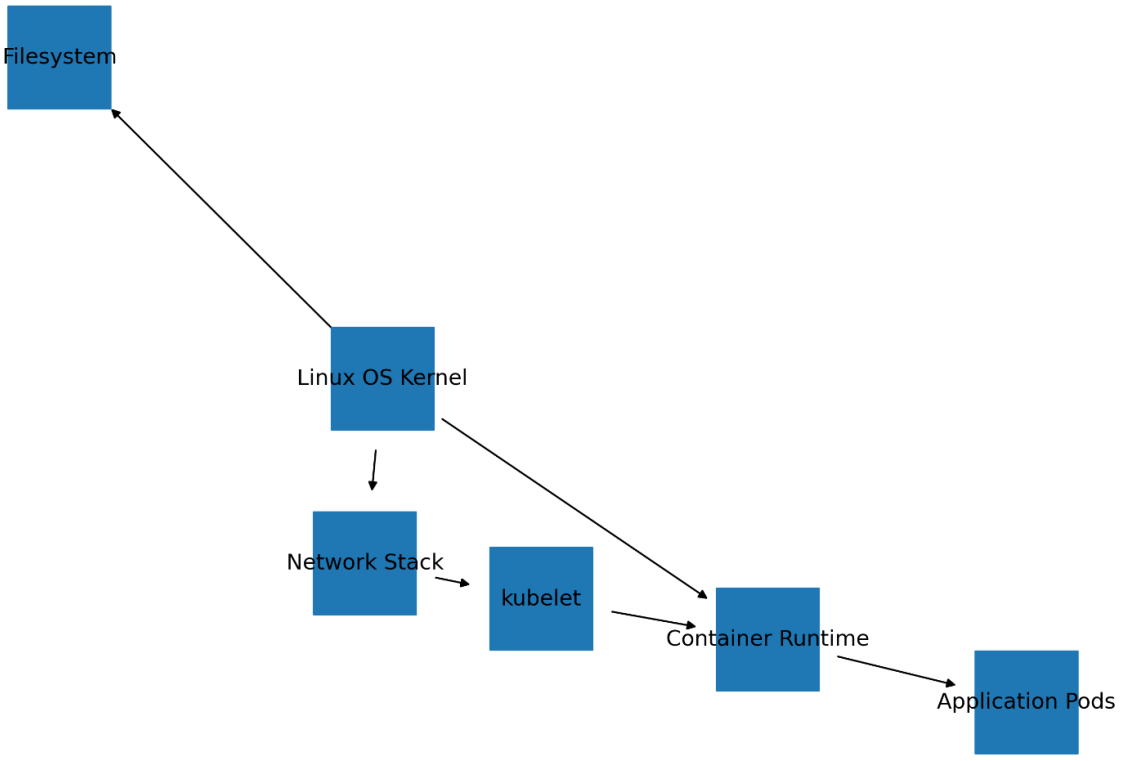

Figure 1. Kubernetes Worker Node Architecture and Attack Surface

2. BACKGROUND AND RELATED WORK

2.1 *Kubernetes Worker Node Architecture*

A Kubernetes worker node is responsible for executing application workloads in the form of containers [21]. Each worker node typically consists of a Linux operating system, a container runtime (such as containerd or CRI-O), the kubelet agent, and supporting networking and storage components. The kubelet acts as the primary interface between the Kubernetes control plane and the node, managing pod lifecycle operations and reporting node status.

Because worker nodes directly host application containers, they expose multiple attack surfaces, including the operating system kernel, filesystem, network stack, container runtime interfaces, and node-level configuration files [22]. A compromise at the worker node level can potentially lead to container breakout, unauthorized access to secrets, or lateral movement across the cluster. As a result, securing worker nodes is essential for maintaining cluster integrity and protecting hosted workloads.

2.2 *CIS Benchmarks for Kubernetes and Linux*

The Center for Internet Security publishes benchmarks that define security best practices for a wide range of technologies, including Kubernetes and Linux operating systems. The CIS Kubernetes Benchmark provides recommendations for securing cluster components such as the API server, controller manager, scheduler, and kubelet. Similarly, CIS Linux Benchmarks focus on operating system–level controls such as filesystem permissions, kernel parameters, audit logging, and access control mechanisms.

While these benchmarks are comprehensive, they are primarily designed as compliance guidelines rather than operational frameworks. Many controls require contextual interpretation and careful enforcement to avoid disrupting system functionality. In Kubernetes environments, this challenge is amplified by the dynamic nature of nodes and workloads, where nodes may be frequently added, removed, or reconfigured.

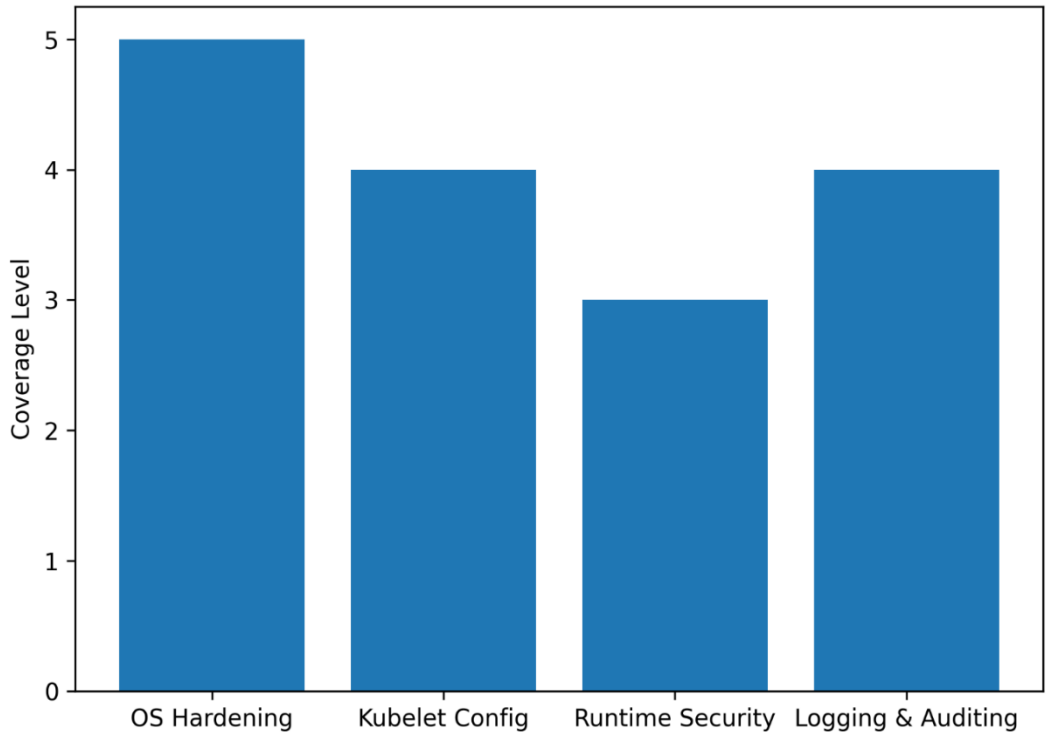

Figure 2. CIS Benchmarks Coverage for Kubernetes Worker Nodes

2.3 Limitations of Manual Hardening Approaches

Manual implementation of CIS benchmarks is common in small or static environments but does not scale effectively in enterprise Kubernetes deployments. Manual hardening introduces several risks, including inconsistent enforcement across nodes, delayed remediation of misconfigurations, and increased likelihood of human error. Additionally, manual processes make it difficult to detect configuration drift over time, particularly in environments with frequent system updates or automated node provisioning.

Prior studies and industry reports highlight that security misconfigurations remain a leading cause of infrastructure breaches, often resulting from incomplete or outdated hardening practices. These findings underscore the need for automated mechanisms that can enforce security controls consistently while adapting to operational changes [23].

2.4 Related Work

Existing research on Kubernetes security has primarily focused on network policies, runtime container security, and access control mechanisms. While these studies address important aspects of cluster security, fewer works explicitly examine the integration of operating system hardening with Kubernetes worker node security. Some approaches rely on immutable infrastructure or specialized security agents, which may introduce additional complexity or performance overhead [24]–[26].

In contrast, this paper focuses on leveraging established benchmarks and widely adopted automation techniques to improve worker node security in a pragmatic and repeatable manner. By aligning CIS recommendations with enforceable Linux controls and automating their application, the proposed framework aims to bridge the gap between security guidelines and operational practice.

3. THREAT MODEL AND PROBLEM DEFINITION

3.1 Threat Model for Kubernetes Worker Nodes

This work focuses on security threats targeting Kubernetes worker nodes, which represent a critical execution layer within a cluster. The threat model assumes an adversary who has obtained an initial foothold through common attack vectors such as compromised container images, vulnerable applications, exposed services, or stolen credentials. From this position, the attacker attempts to escalate privileges, access sensitive resources, or move laterally across the cluster.

Worker nodes present multiple attack surfaces due to their layered architecture. At the operating system level, vulnerabilities in the Linux kernel, misconfigured system services, weak file permissions, and insecure kernel parameters can be exploited to gain elevated privileges. At the container runtime layer, improper isolation between containers and the host may allow container escape attacks. The kubelet service, which operates with elevated privileges on the node, is another critical component that can be abused if improperly secured.

This paper assumes that the Kubernetes control plane is operational and protected according to standard best practices. The focus is intentionally limited to worker nodes, as these systems directly host application workloads and are more frequently exposed to untrusted code execution. Attacks on worker nodes can undermine higher-level Kubernetes security mechanisms, making node-level hardening a foundational requirement for cluster security.

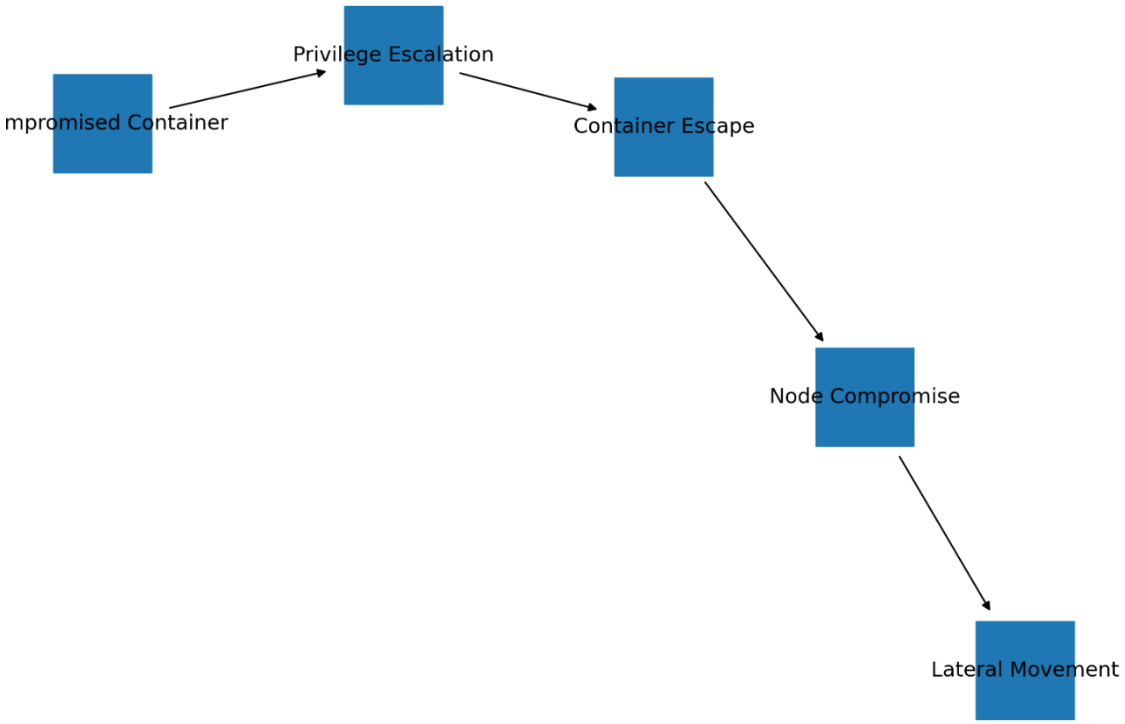

Figure 3. Worker Node Attack Progression: Container Compromise to Lateral Movement

3.2 Problem Definition

Despite the availability of detailed CIS benchmarks for both Linux operating systems and Kubernetes components, enforcing these recommendations consistently across worker nodes remains a challenge in practice. Many organizations apply security controls during initial node provisioning but fail to maintain them over time. Changes introduced by system updates, emergency fixes, or manual interventions often lead to configuration drift that is difficult to detect and remediate.

Manual hardening approaches do not scale well in dynamic Kubernetes environments, where worker nodes may be frequently added or replaced. Furthermore, partial implementation of benchmarks—such as securing Kubernetes configurations without corresponding Linux hardening—creates a false sense of security while leaving critical vulnerabilities unaddressed.

The core problem addressed in this paper is the lack of a repeatable, automated mechanism to enforce and maintain CIS-aligned security controls on Kubernetes worker nodes without negatively impacting cluster availability. Specifically, there is a need to understand how benchmark recommendations can be translated into enforceable Linux configurations and applied at scale in a way that balances security, operational stability, and maintainability [27].

3.3 Design Goals

Based on the identified threats and operational challenges, this work defines the following design goals [28]:

1. **Consistency:** Ensure that security controls are applied uniformly across all worker nodes.
2. **Automation:** Reduce reliance on manual intervention by enforcing hardening controls through automated mechanisms.
3. **Auditability:** Enable continuous verification of compliance with CIS benchmarks.
4. **Operational Stability:** Apply security controls without introducing unacceptable node downtime or workload disruption.
5. **Maintainability:** Support ongoing updates and configuration changes while minimizing security drift.

These design goals guide the development of the hardening and automation framework presented in the following sections.

4. CONTROL MAPPING AND HARDENING STRATEGY

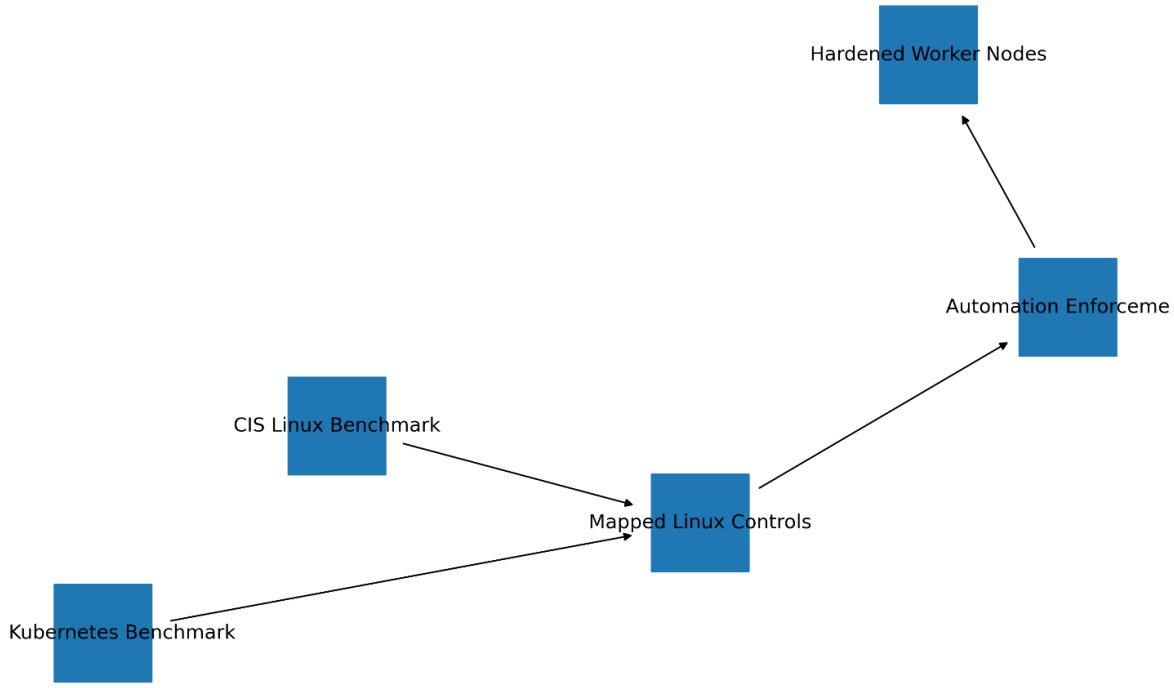

Figure 4. Control Mapping & Automation Enforcement

4.1 Rationale for Control Mapping

CIS benchmarks provide prescriptive recommendations for securing both Kubernetes components and Linux operating systems; however, they are published as separate documents with limited guidance on cross-layer enforcement. In Kubernetes worker nodes, this separation introduces ambiguity regarding responsibility boundaries between the orchestration layer and the host operating system. As a result, security controls may be implemented redundantly, inconsistently, or not at all.

To address this challenge, this work adopts a control mapping approach that explicitly links CIS Kubernetes benchmark recommendations to corresponding Linux operating system hardening controls. The objective is to translate abstract security requirements into concrete, enforceable system configurations that can be applied consistently across worker nodes. This mapping enables systematic enforcement while reducing the likelihood of gaps caused by partial or overlapping implementations.

4.2 Scope of Hardening Controls

The proposed hardening strategy focuses on Kubernetes worker nodes and includes controls spanning the following domains:

a. **Operating System Hardening:** Kernel parameters, filesystem permissions, audit logging, access control policies, and service configurations.
b. **Node-Level Kubernetes Components:** Kubelet configuration files, authentication and authorization settings, and secure communication with the control plane.
c. **Runtime Environment Controls:** Container runtime configurations, namespace isolation, and privilege restrictions.

Controls related exclusively to control plane components, such as the API server or scheduler, are outside the

scope of this study and are assumed to follow established best practices.

4.3 *Mapping CIS Benchmarks to Enforceable Controls*

Each CIS benchmark recommendation is evaluated to determine whether it can be enforced at the Linux operating system level, the Kubernetes configuration level, or both. Controls are categorized into three types:

a. **Directly Enforceable Controls:** Recommendations that map directly to Linux configurations, such as file permissions, kernel parameters, and service settings.
b. **Conditionally Enforceable Controls:** Controls that require contextual interpretation, such as audit logging policies or runtime configurations that depend on workload characteristics.
c. **Verification-Only Controls:** Recommendations that cannot be enforced automatically but can be continuously validated for compliance.

Table I illustrates representative examples of the control mapping strategy.

Table 1. Example Mapping of CIS Benchmark Controls to Linux Hardening Actions

CIS Recommendation	Target Layer	Linux Control	Enforcement Method
Secure kubelet config file permissions	Kubernetes Node	File ownership and permissions	Configuration management
Enable kernel address space protection	OS Kernel	sysctl parameters	Automated enforcement
Enable audit logging	OS Services	auditd configuration	Policy-driven deployment
Restrict SSH access	OS Services	sshd configuration	Automated remediation

This mapping ensures that security controls are applied using mechanisms appropriate to their execution layer while maintaining alignment with benchmark requirements.

4.4 *Hardening Strategy Design*

The hardening strategy is designed to balance security enforcement with operational stability. Controls are applied in a staged manner to minimize disruption to running workloads. The strategy distinguishes between static controls, which can be enforced during node provisioning, and dynamic controls, which may be applied or updated while nodes are in service.

Static controls include filesystem permissions, kernel parameters, and baseline service configurations. These controls are typically enforced during initial node setup or maintenance windows. Dynamic controls, such as audit logging policies or kubelet configuration updates, are applied incrementally with validation steps to ensure node health is preserved.

To reduce the risk of unintended service impact, each control is validated against the following criteria:

a. Compatibility with Kubernetes node requirements
b. Impact on system performance
c. Reversibility in case of failure

Controls that fail validation are flagged for manual review rather than automatically enforced.

4.5 *Security Drift Considerations*

Configuration drift is a common challenge in long-lived Kubernetes clusters, particularly in environments with frequent updates or manual interventions. The proposed strategy treats drift detection as a first-class concern rather than a secondary validation step. By maintaining a declarative definition of expected system state, deviations can be identified and addressed promptly.

Rather than reapplying all controls indiscriminately, the framework focuses on detecting meaningful deviations that affect security posture. This approach minimizes unnecessary configuration changes and reduces the risk of instability caused by repetitive enforcement actions.

5. AUTOMATION FRAMEWORK DESIGN

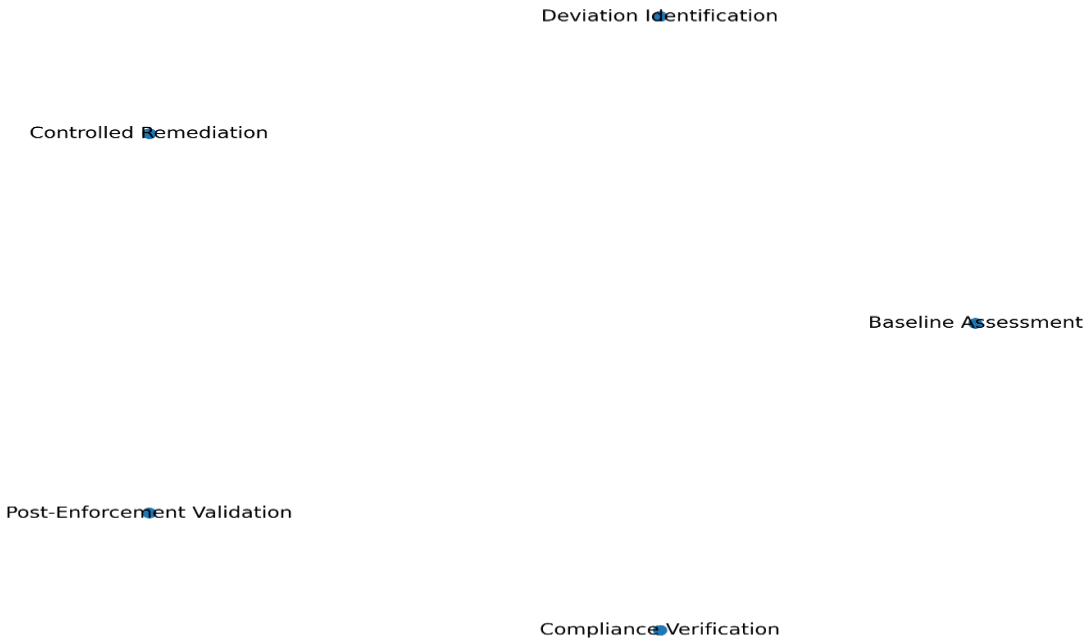

Figure 5. Automation Framework Enforcement Workflow

5.1 Design Principles

The automation framework is designed to enforce and maintain security hardening controls on Kubernetes worker nodes in a consistent and repeatable manner. Rather than introducing proprietary tooling, the framework leverages established configuration management and system administration practices commonly used in enterprise Linux environments. This design choice reduces operational risk and facilitates adoption in existing infrastructures.

The framework adheres to the following principles:

a. **Idempotency:** Repeated executions must not introduce unintended side effects.
b. **Minimal Disruption:** Security controls should be applied without causing unnecessary node downtime or workload interruption.
c. **Auditability:** All enforcement actions must be observable and verifiable.
d. **Extensibility:** The framework should support incremental addition of new controls.

These principles guide the implementation and execution of the automation workflow.

5.2 Framework Architecture

The automation framework follows a centralized orchestration model in which security policies are defined declaratively and enforced across worker nodes through controlled execution. Figure 1 illustrates the high-level architecture of the framework.

At the core of the framework is a policy definition layer that specifies the desired security state of worker nodes. This includes kernel parameters, service configurations, file permissions, and audit policies derived from CIS benchmarks. An automation engine applies these policies to target nodes using secure communication channels.

Worker nodes periodically report compliance status, enabling centralized visibility into enforcement outcomes and deviations. This feedback loop supports continuous monitoring while avoiding constant reconfiguration of nodes.

5.3 Enforcement Workflow

The enforcement process follows a structured, multi-stage workflow to reduce operational risk. Each stage performs a specific function within the hardening lifecycle:

a. **Baseline Assessment:** Worker nodes are scanned to determine current compliance status against defined policies. This step establishes a baseline and identifies deviations.
b. **Controlled Remediation:** Non-compliant controls that are deemed safe for automated enforcement are remediated. Controls with potential service impact are flagged for manual approval.
c. **Post-Enforcement Validation:** Node health checks are performed to verify that critical services, including kubelet and container runtime components, remain operational.
d. **Compliance Verification:** Updated compliance status is recorded and compared against the desired state to confirm successful enforcement.

This staged approach ensures that security improvements do not compromise node availability.

5.4 Drift Detection and Maintenance

Configuration drift is addressed through periodic validation rather than continuous re-enforcement. Worker nodes are evaluated at defined intervals to detect deviations from the desired security state. Detected drift is classified based on severity and potential impact.

Minor deviations, such as modified file permissions, may be remediated automatically. More significant changes, including kernel parameter modifications or service reconfigurations, are logged for review. This selective remediation approach reduces unnecessary system changes and minimizes operational instability.

5.5 Failure Handling and Recovery

Automation failures are an expected risk in large-scale environments. The framework includes safeguards to prevent partial enforcement from leaving nodes in an inconsistent state. Changes are applied in discrete steps, with rollback mechanisms available for critical configurations.

If a node fails validation checks following enforcement, the framework can suspend further actions and alert operators for intervention. This design prevents cascading failures and ensures that automation does not become a single point of failure.

6. EXPERIMENTAL SETUP

6.1 Test Environment Overview

To evaluate the effectiveness and operational impact of the proposed automation framework, experiments were conducted in a controlled Kubernetes environment designed to reflect common enterprise worker node configurations. The evaluation environment focuses on worker node security and intentionally limits scope to node-level controls to isolate the impact of Linux hardening and CIS benchmark enforcement.

The Kubernetes cluster used for evaluation consists of a small number of worker nodes sufficient to observe enforcement behavior, configuration drift, and operational stability. While the cluster size does not represent large-scale production deployments, it enables repeatable testing and detailed observation of enforcement outcomes.

6.2 Cluster Configuration

The experimental cluster includes a single Kubernetes control plane and multiple worker nodes running a Linux-based operating system. Each worker node hosts application workloads deployed as containers and runs standard

Kubernetes node components, including the kubelet and container runtime.

The operating system configuration reflects a typical enterprise Linux environment, including:

a. Default kernel configurations prior to hardening
b. Standard system services enabled for remote administration and logging
c. Persistent storage and networking components required for Kubernetes operation

No additional security agents or proprietary tools are installed on the worker nodes beyond those required for automation and compliance verification.

6.3 Benchmark Selection

The evaluation leverages publicly available CIS benchmarks relevant to Kubernetes worker nodes and Linux operating systems. These benchmarks are used as authoritative references for defining the desired security state.

The selected benchmarks include:

a. CIS Kubernetes Benchmark (worker node–related controls)
b. CIS Linux Benchmark applicable to the operating system version under test

Only controls that directly affect worker node security are included. Control plane–specific recommendations are excluded to maintain focus on node-level hardening.

6.4 Automation Tooling

Automation is implemented using widely adopted configuration management and scripting tools commonly deployed in enterprise Linux environments. These tools are responsible for:

a. Applying system-level configuration changes
b. Validating enforcement outcomes
c. Collecting compliance data

All automation actions are executed using secure communication channels and adhere to least privilege principles. Automation tasks are designed to be idempotent, ensuring that repeated executions do not introduce unintended changes.

7. EVALUATION AND RESULTS

7.1 Compliance Improvement Analysis

The primary objective of the evaluation is to assess the effectiveness of automation-driven hardening in improving compliance with selected CIS benchmark controls on Kubernetes worker nodes. Compliance is measured before and after the application of automated hardening policies to capture the impact of enforcement.

Baseline assessments indicate that a subset of benchmark controls is not satisfied in the default worker node configuration. These non-compliant controls are primarily related to operating system–level settings, including kernel parameters, file permissions, and audit logging configurations. Following automated enforcement, a significant portion of these controls are remediated, resulting in an observable increase in overall compliance alignment.

Figure 2 illustrates the change in benchmark compliance levels between the baseline and post-hardening states. The results demonstrate that automated enforcement improves consistency across worker nodes and reduces variability caused by manual configuration differences.

7.2 Classification of Remediated Controls

To better understand the nature of the observed improvements, remediated controls are grouped into categories based on their enforcement layer. The majority of remediated controls fall within the operating system hardening domain, including filesystem protections and kernel parameter configurations. A smaller subset relates to node-level Kubernetes settings, such as kubelet configuration permissions.

Table II summarizes the distribution of remediated controls by category. This classification highlights the importance of addressing Linux operating system security as a

foundational element of Kubernetes worker node protection.

Table 2. Distribution of Remediated CIS Controls by Category

Control Category	Relative Impact
OS Kernel and Filesystem	High
System Services and Logging	Moderate
Kubernetes Node Configuration	Moderate
Runtime Environment	Limited

7.3 Operational Impact Assessment

An essential requirement of the proposed framework is maintaining cluster availability during security enforcement. Throughout the evaluation, worker node status and workload execution are monitored to identify any service disruptions attributable to automation activities.

Observations indicate that enforcement actions do not introduce prolonged node unavailability. Temporary configuration changes are applied in a controlled manner, and node health checks confirm continued operation of critical services. No persistent workload failures are observed during the evaluation window.

Figure 3 presents a summary of observed node availability during enforcement phases, demonstrating that security improvements can be achieved without compromising operational stability when automation is applied judiciously.

7.4 Configuration Drift Reduction

Configuration drift is assessed by comparing worker node configurations over multiple evaluation cycles. Automated validation identifies deviations from the defined security baseline, enabling targeted remediation of affected controls. This approach reduces the likelihood of long-term drift caused by manual interventions or system updates [29].

The evaluation shows that automated drift detection improves visibility into security posture changes and supports timely remediation. This capability is particularly valuable in environments where worker nodes are frequently updated or replaced.

7.5 Summary of Findings

The evaluation results indicate that automation-driven enforcement of CIS-aligned Linux hardening controls can measurably improve security compliance on Kubernetes worker nodes while maintaining operational stability. These findings support the feasibility of integrating benchmark-based hardening into routine Kubernetes operations.

8. DISCUSSION AND LIMITATIONS

The evaluation results demonstrate that automation-driven enforcement of CIS-aligned Linux hardening controls can improve the security posture of Kubernetes worker nodes without introducing significant operational disruption. By systematically mapping benchmark recommendations to enforceable operating system configurations, the proposed framework addresses a common gap between security guidelines and practical implementation.

One key observation is that many security improvements are achieved at the operating system level rather than through Kubernetes-specific configurations. This finding reinforces the importance of host security as a foundational layer for containerized environments. Even when Kubernetes components are configured according to best practices, weaknesses in the underlying Linux system can undermine higher-level protections.

Despite these positive outcomes, the proposed approach has several limitations. First, the evaluation is conducted in a controlled environment with a limited

number of worker nodes. While this enables detailed observation and repeatability, the results may not fully capture the complexity of large-scale production clusters with heterogeneous workloads and infrastructure.

Second, the framework focuses on static and configuration-based security controls defined by CIS benchmarks. Runtime threats, such as zero-day exploits and advanced container escape techniques, are outside the scope of this study [30]. Additional security mechanisms, including runtime monitoring and behavioral analysis, would be required to address such threats comprehensively.

Finally, not all benchmark controls are suitable for automated enforcement. Certain controls require contextual judgment or may conflict with application-specific requirements. The framework mitigates this risk by classifying controls based on enforceability and impact, but some degree of manual oversight remains necessary.

These limitations highlight opportunities for future enhancements while reinforcing the practical value of automation for baseline security enforcement.

9. CONCLUSION AND FUTURE WORK

This paper presents an automated framework for securing Kubernetes worker nodes through the enforcement of CIS benchmark–aligned Linux hardening controls. By integrating operating system security with Kubernetes node configurations, the proposed approach improves compliance consistency and reduces configuration drift without compromising cluster availability.

The results of the evaluation indicate that automation can effectively bridge the gap between security recommendations and operational practice. Rather than relying on manual hardening efforts, organizations can leverage automation to maintain a consistent security baseline across dynamic Kubernetes environments.

Future work will focus on extending the framework to incorporate runtime security controls and continuous threat detection mechanisms. Additional evaluation in larger and more diverse cluster environments would provide further insight into scalability and performance tradeoffs. Integrating admission control policies and container image security assessments represents another promising direction for expanding the scope of automated security enforcement.

REFERENCES

[1] Kubernetes Documentation, *Kubernetes Components*. Cloud Native Computing Foundation, 2024.
[2] S. B. Mohan and R. Buyya, "Secure containerized applications in cloud environments," *IEEE Cloud Comput.*, vol. 6, no. 4, pp. 32–41, 2019.
[3] C. Pahl, "Containerization and the paas cloud," *IEEE Cloud Comput.*, vol. 2, no. 3, pp. 24–31, 2015.
[4] Open Policy Agent, *Policy-Based Control for Cloud Native Environments*. OPA Documentation, 2023.
[5] IEEE Standards Association, "IEEE Standard for Security in Cloud Computing," *IEEE Std 2302-2021*, 2021.
[6] Cloud Native Security Conference Proceedings, "Advances in Kubernetes Security," *CNCF*, 2022.
[7] Kubernetes Documentation, *Security Best Practices*. Cloud Native Computing Foundation, 2024.
[8] CNCF, "Cloud Native Security Whitepaper," 2022.
[9] Kubernetes Documentation, *Kubelet Configuration*. Cloud Native Computing Foundation, 2024.
[10] National Institute of Standards and Technology, *Application Container Security Guid*. NIST SP 800-190, 2017.
[11] A. Shankar et al., "Security challenges in container-based virtualization," *IEEE Int. Conf. Cloud Comput.*, pp. 1–8, 2019.
[12] Linux Foundation, *Linux Security Modules: SELinux and AppArmor*. Linux Foundation Documentation, 2023.
[13] R. Richardson and M. North, "Ransomware and infrastructure misconfigurations," *IEEE Secur. Priv.*, vol. 18, no. 3, pp. 78–82, 2020.
[14] HashiCorp, *Infrastructure as Code Security*. HashiCorp Whitepaper, 2023.
[15] Center for Internet Security, *CIS Kubernetes Benchmark, CIS*. USA: East Greenbush, NY, 2023.
[16] Center for Internet Security, *CIS Benchmark for Linux, CIS*. USA: East Greenbush, NY, 2023.
[17] National Institute of Standards and Technology, *Guide to General Server Security*. NIST SP 800-123, 2008.
[18] OWASP Foundation, *OWASP Kubernetes Top Ten*. OWASP Project Documentation, 2023.

[19] Docker Inc, *Docker Security*. Docker Documentation, 2023.
[20] Amazon Web Services, *Security Best Practices for Kubernetes*. AWS Whitepaper, 2023.
[21] S. Zanero, "Monitoring and protecting containers at runtime," *IEEE Secur. Priv.*, vol. 17, no. 5, pp. 72–76, 2019.
[22] A. P. Silva et al, "Evaluating container runtime isolation mechanisms," *IEEE Trans. Cloud Comput.*, vol. 10, no. 1, pp. 215–228, 2022.
[23] National Institute of Standards and Technology, *Security and Privacy Controls for Information Systems and Organizations*, NIST SP 80. 2020.
[24] Red Hat, *Securing Kubernetes Clusters*. Red Hat Product Documentation, 2023.
[25] Google Cloud, *Harden Your Kubernetes Cluster*. Google Cloud Architecture Center, 2023.
[26] Microsoft Azure, *Kubernetes Security Best Practices*. Microsoft Learn, 2023.
[27] B. Schneier, *Applied Cryptography*, 2nd ed. New York, NY, USA: USA: Wiley, 1996.
[28] P. Jamshidi, C. Pahl, N. C. Mendonça, J. Lewis, and S. Tilkov, "Microservices: The journey so far and challenges ahead," *IEEE Softw.*, vol. 35, no. 3, pp. 24–35, 2018.
[29] J. Lewis and M. Fowler, "Microservices: a definition of this new architectural term," *MartinFowler. com*, vol. 25, no. 14–26, p. 12, 2014.
[30] L. Bilge and T. Dumitraş, "Before we knew it: an empirical study of zero-day attacks in the real world," in *Proceedings of the 2012 ACM conference on Computer and communications security*, 2012, pp. 833–844.