Using LSMs to Strengthen Access Control of Applications that use hostPath in Kubernetes Clusters

Ronaldo Medeiros[1], **Lília Sampaio**[1], **Raphael Agra**[1], **Reinaldo Gomes**[1,2]

[1]Laboratório de Sistemas Distribuídos
Universidade Federal de Campina Grande (UFCG)
Campina Grande, PB – Brazil

[2]Rede Nacional de Ensino e Pesquisa (RNP)
Rio de Janeiro, RJ – Brazil

{ronaldomedeiros,liliars,raphael.agra,reinaldo}@lsd.ufcg.edu.br

Abstract. *This paper presents a methodology to enhance the security of Kubernetes clusters using hostPath volumes, a feature that introduces considerable risks due to direct host access. Our approach leverages Linux Security Modules (LSMs) to enforce access control between workloads and host nodes, aligning with Zero Trust principles. The proposed solution is demonstrated through integration with the SPIFFE CSI Driver within the SPIRE framework. Experimental results show that, among various performance metrics, only the SPIRE Agent's identity synchronization latency increased notably, around* 29.68%*. While this overhead appears significant, it may be acceptable in less time-sensitive scenarios, particularly when balanced against the improved security posture achieved.*

1. Introduction

The proliferation of cloud computing, particularly in microservice architectures, has led to widespread adoption of containerization and Kubernetes (k8s) for orchestration [Deng et al. 2024]. While Kubernetes offers numerous advantages, certain applications, such as the SPIFFE Runtime Environment (SPIRE) [SPIFFE 2025], require the use of hostPath volumes, a k8s volume abstraction that directly maps a directory or file from the host node's file system into a pod [Gunathilake and Ekanayake 2024]. This presents a significant security challenge, as hostPath volumes can expose the underlying host system to potential privilege escalation attacks.

Attackers could exploit this access to compromise the entire Kubernetes cluster by gaining access to sensitive data or manipulating critical system components [Islam Shamim et al. 2020]. Therefore, securing k8s environments that rely on hostPath volumes is crucial. Attackers could potentially read or write to the etcd store, create unauthorized pods, or obtain access tokens, leading to a complete system compromise. The inherent risks associated with hostPath volumes require the exploration of alternative solutions and mitigation strategies to minimize the attack surface and improve the overall security posture of Kubernetes deployments [Perera et al. 2022].

Despite security concerns, eliminating hostPath volumes is not always feasible due to application dependencies and architectural constraints. Disabling this volume is not an available option for many production environments that rely on this application.

For example, while the official k8s documentation advises against using hostPath volumes, SPIRE would require a significant architectural overhaul to function without them. One of the primary alternatives to it is the Container Storage Interface (CSI), specifically the SPIFFE CSI Driver, which provides a more secure mechanism for managing storage access in k8s SPIRE environments [Vasilenko and Mahesh 2019].

This driver introduces an intermediary layer that centralizes and secures access to host resources by delegating the mounting of hostPath volumes to a privileged CSI container sidecar, rather than having each workload directly access host directories. This design significantly reduces the number of pods with direct access to hostPath volumes, limiting the potential for privilege escalation [SPIFFE 2023]. However, it does not eliminate the underlying risk since the CSI driver itself must run with elevated privileges and any compromise of this component could still lead to full node access [Mirantis 2022].

In light of these persistent security gaps, we propose the integration of KubeArmor [KubeArmor 2025], a runtime enforcement framework built upon Linux Security Modules (LSMs), such as AppArmor and SELinux, to apply fine-grained Mandatory Access Control (MAC) policies to workloads utilizing hostPath volumes. KubeArmor allows the definition and enforcement of Zero Trust principles within the k8s environment, ensuring that even privileged components such as the SPIFFE CSI Driver operate within strictly confined access boundaries.

Our approach leverages KubeArmor to implement a least-privilege model that restricts access to only the necessary file system paths and operations for the SPIFFE CSI Driver. This prevents unauthorized file system access, mitigates the risk of lateral movement, and significantly narrows the attack surface in the event of container compromise. Furthermore, because KubeArmor policies are applied at the runtime level, they can adapt to changes in deployment patterns and provide continuous monitoring and enforcement, a crucial feature in dynamic, multi-tenant Kubernetes environments [Findlay et al. 2020, van Vugt and Malik 2023].

Experimental results showed that, for most SPIRE performance metrics evaluated, particularly those related to the latency of common operations, the proposed solution exhibited no significant differences when compared to a default SPIRE Kubernetes environment. The only exception was the *SPIRE Agent Entries Latency* metric, which increased by approximately 29.68% when the proposed methodology was applied.

The remainder of this paper is organized as follows: Section 2 provides background information relevant to understanding the proposed methodology. Section 3 reviews related work that inspired or influenced this study. Section 4 details the proposed methodology and the architecture of a corresponding implementation, referred to as the solution. Section 5 introduces a threat model for the SPIFFE CSI Driver. Section 6 presents the key results from the experimental evaluation. Finally, Section 7 summarizes the conclusions and outlines directions for future work.

2. Background

2.1. Cloud Native Environments

In the early days of computational infrastructure, all data and operations had to be physically located alongside the hardware. When additional memory or processing power

was required, users had to purchase new machines and allocate more physical space to accommodate them. The advent of Cloud Computing and Cloud Native environments revolutionized this model, allowing users to access computing resources from thousands of kilometers away. These environments eliminate the need for acquiring new hardware by leveraging virtualized, isolated systems that scale efficiently to meet demand [Deng et al. 2024].

Over time, virtual environments evolved from requiring a full Virtual Machine (VM) with an entire operating system to using containers. Unlike VMs, containers allow applications to run with only the necessary components, eliminating the need for a complete system [Pereira Ferreira and Sinnott 2019]. This flexibility enables applications to run anywhere with minimal overhead. However, just as production systems once had to coordinate a vast number of physical machines, they now face the challenge of managing numerous containers. This need gave rise to orchestration tools like Kubernetes and Red Hat OpenShift, designed to manage and coordinate large-scale containerized environments efficiently.

As a consequence of being widely used in extensive production environments, some concerns started to emerge about how *k8s* manages the persistence of data, since in case of a crash, all data in the container is lost due to its ephemeral nature. Additionally, it is not practical to manually assemble a file-share system in a large-scale context outside the context of *k8s*. To solve these issues, the concept of volumes in Kubernetes was created, which is an abstraction of filesystems. Ephemeral Volumes have the lifetime of a pod, meaning that if a pod is deleted its volume is deleted as well. Meanwhile, Persistent Volumes (PV) outlive pods, and they will be preserved even in case of crashes or restarts.

That way, many functionalities are abstracted by the Persistent Volume API when it comes to handling persistent data management. In the case of a user with the necessity for other types of disk configurations or different sizes of volumes, Persistent Volume Claims (PVC) help to manage this need. It binds with a PV that matches its claims to its attributes and grants the requested volume.

2.2. SPIRE and the hostPath Volume Issue

The hostPath is a type of Persistent Volume in *k8s* that enables users to mount files and directories from the Kubernetes host filesystem. Depending on the intended use, it can be configured in various modes, such as read-only or read/write, which adapts to the necessity of the user. However, its characteristics can pose a security risk by increasing the likelihood of privilege escalation attacks. One common attack vector involves a workload mounting sensitive host directories, breaking the isolation between the containerized environment and the host system. This breach can lead to the compromise of both the Kubernetes cluster and the underlying node. Despite these risks, many applications (such as SPIRE) rely on hostPath volumes to operate correctly.

While SPIRE has implemented mechanisms to reduce the risks associated with hostPath volumes, it does not eliminate their use entirely. One such mechanism is the SPIFFE CSI Driver, a container storage interface that allows workloads running in the cluster to access the SPIRE Agent Workload API without directly mounting sockets from the host filesystem. Working in coordination with the SPIRE Server and Agent, the driver retrieves the SPIFFE identity for the pod and mounts it as a volume. However, hostPath

volumes remain necessary for the CSI driver to interact with the Kubernetes *kubelet* daemon. Additionally, the SPIFFE CSI Driver must run in privileged mode, which increases the potential impact if compromised and makes privilege escalation an even more serious issue.

2.3. Linux Security Modules (LSMs)

In 2001, the National Security Agency of the United States introduced the foundation for what is now known as the Linux Security Module (LSM) [Smalley et al. 2001]. LSM is a framework that adds an extra layer of security to the operating system by using modules. These modules enable the implementation of specific security features without requiring direct modifications to the Linux Kernel.

LSMs introduced an innovative approach to managing file and directory access, named Mandatory Access Control (MAC). MAC enforces access restrictions using policies, or rules, to prevent unauthorized users from interacting with system resources. Within this framework, files and directories are treated as objects, while processes are regarded as subjects.

This concept was subsequently integrated into the Linux kernel by Linus Torvalds and has been included by default in major Linux distributions since 2003. Among the most prominent implementations of Linux Security Modules (LSMs) is AppArmor [Zhu and Gehrmann 2022], which enforces access control through the use of profiles. Each profile defines a set of policies that specify the permissible actions and accessible resources for a given process. AppArmor supports two operational modes: Complaining Mode, which logs policy violations without enforcing restrictions, and Enforcing Mode, which actively denies unauthorized actions while logging them.

2.4. KubeArmor

KubeArmor is a security tool that leverages LSMs to enhance security in Cloud Native environments, particularly in Kubernetes clusters, containerized applications, and VMs [KubeArmor 2025, Lee et al. 2022]. Although LSMs have been in use for over two decades, benefiting from continuous updates, refinements, and redesigns, they remain challenging for many users. Their complexity and lack of user-friendly configuration often make them impractical for widespread adoption.

To address these challenges, KubeArmor emerged as an alternative, combining the power of LSMs with the simplicity of higher-level configuration files known as Security Policies. These policies are easier to understand, write, and maintain, making it more practical to leverage LSMs in Cloud Native environments, as we can see in the SE-RAN 5G project [SE-RAN-5G 2025]. It uses KubeArmor to protect networking with policies and keep it supervised. Through this approach, KubeArmor has simplified the use of LSMs, enabling enhanced security with greater accessibility.

3. Related Work

Security in Cloud Native environments has been the subject of intense research, with an emphasis on identifying and mitigating vulnerabilities across multiple layers of the infrastructure. Zeng et al [Zeng et al. 2023] conducted a comprehensive analysis of vulnerabilities on Cloud Native platforms, categorizing them according to the types of attacks

and affected components, such as Docker, Kubernetes, and Istio. The study also compares several open source security tools, highlighting gaps in the coverage of these tools for complex contexts such as k8s, especially regarding the detection of misconfiguration problems and malicious activities.

Complementing this analysis, Martijn van Vugt and Malik [van Vugt and Malik 2023] conducted an effectiveness evaluation of open source security tools in k8s microservices environments, focusing on observability and response to simulated attacks. These studies reinforce the importance of robust observation and response systems for effective protection of workloads in k8s environments, highlighting that many tools still have gaps in detecting and responding to complex threats.

The security risks posed by the use of hostPath volumes in k8s have been extensively documented in both academic literature and industry analyses. Garfinkel and Rosenblum [Garfinkel and Rosenblum 2003] were among the first to describe the dangers of breaking isolation in virtualized environments, which directly parallels the risks introduced by hostPath volumes, as it grants containers direct access to k8s host/node file systems. More recently, Gülcü et al [Gülcü et al. 2021] provided a systematic taxonomy of security challenges in k8s, identifying hostPath volumes as a critical vulnerability that allows for escalation of privileges, unauthorized access to secrets, and lateral movement between nodes.

The CNCF whitepaper by Peck et al [Peck et al. 2019] further elaborates on real-world exploitation scenarios, including how attackers can use access to the k8s node/host file system to manipulate etcd data, deploy rogue pods, or steal service account tokens. Despite Kubernetes documentation discouraging the use of hostPath in most cases, its usage persists due to its simplicity and necessity in certain operational workflows, particularly when exposing system-level resources like sockets and certificates.

Some mitigation strategies have been proposed to address these risks. The SPIFFE CSI Driver represents a community-led effort to isolate hostPath volume usage within a privileged container, thereby reducing direct exposure to k8s node/host resources [SPIFFE 2023]. However, this solution retains the underlying reliance on privileged operations, which remains a significant attack surface.

CNCF introduced KubeArmor [KubeArmor 2025], a runtime security enforcement tool based on Linux Security Modules that enables the implementation of fine-grained access policies for containers, including those interacting with k8s node/host mounted volumes. These tools, while not entirely eliminating the need for hostPath volumes, offer practical mechanisms for reducing their associated risks, particularly when applied in defense-in-depth architectures or in conjunction with Zero Trust security models. Despite these advancements, a fully secure and scalable alternative to hostPath remains an open challenge, especially in systems requiring close integration with underlying host services, such as SPIRE-based identity frameworks.

In the context of Zero Trust security, D'Silva et al [D'Silva and Ambawade 2021] examine the application of the Zero Trust model within Kubernetes environments to enhance network security through rigorous authentication and authorization mechanisms. Their study highlights the model's effectiveness in isolating and safeguarding systems, a particularly important strategy in multi-tenant k8s deployments. Similarly, Rodigari et al

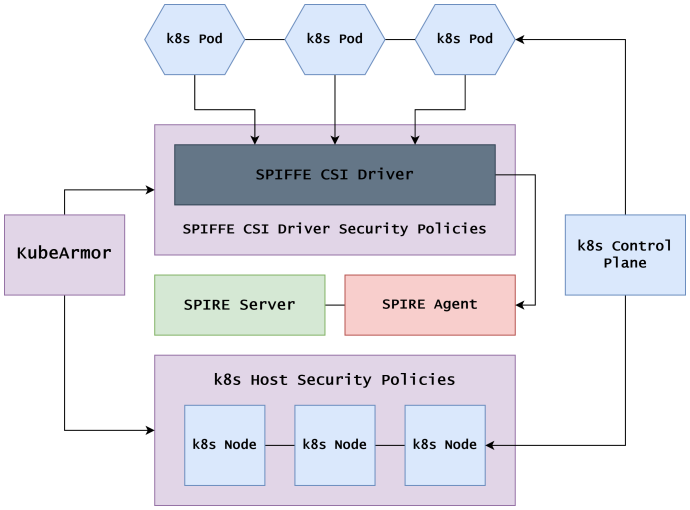

Figure 1. An architectural overview of the proposed solution running on a k8s cluster.

[Rodigari et al. 2021] assess the impact of adopting Zero Trust principles in multi-cloud architectures using Istio, focusing on performance metrics and resource consumption, and providing practical insights into the operational trade-offs of this security model in Cloud Native environments.

In terms of reducing the attack surface, Ghavamnia et al [Ghavamnia et al. 2020] introduce the Confine tool, which automatically generates system call policies for containers through static analysis. This solution addresses Linux Kernel protection against exploits from malicious containers, a crucial strategy for k8s environments where complete isolation between containers and the host may not be fully guaranteed.

4. Solution Architecture

Figure 1 illustrates the architecture of the proposed solution, which integrates KubeArmor's [KubeArmor 2025] runtime enforcement with the SPIFFE identity framework and the SPIRE implementation. To illustrate the proposed methodology, we use the SPIFFE CSI Driver, though the underlying policy model is also generalizable to other applications relying on hostPath volumes.

This design can be structured into three invisible layers that operate within a Kubernetes environment: the Workload Layer, the Identity and Policy Enforcement Layer, and the Infrastructure Layer.

4.1. Workload Layer

At the top of the architecture are the k8s pods, which host application containers running in the cluster. These pods rely on the Kubernetes Control Plane for orchestration, scheduling, and service discovery. When a pod starts, it requests an identity to be used for secure communication within the system.

The SPIFFE CSI Driver, in its turn, is responsible for mounting the SPIFFE Verifiable Identity Document (SVID) into the pod's file system. Each identity is then used by the workloads to assert their trustworthiness to other components or services in the mesh.

The communication between pods and the SPIFFE CSI Driver occurs transparently during pod initialization.

4.2. Identity and Policy Enforcement Layer

The SPIFFE CSI Driver acts as the intermediary between pods and the SPIRE infrastructure. Once a pod is scheduled on a node, the CSI Driver initiates a connection with the local SPIRE Agent to retrieve the appropriate SVID. These operations are governed by security policies defined to control access to the driver and to ensure it functions within secure parameters.

Next, we have the SPIRE Agent and Server. As expected, the SPIRE Agent handles workload attestation and identity issuance, validating that the requesting workload matches an existing registration entry and communicates with the SPIRE Server to retrieve the corresponding identity. The SPIRE Server, then, manages all trust bundles, registration entries, and configuration data.

KubeArmor contributes at this layer by enforcing security policies related to both the SPIFFE CSI Driver and the SPIRE components. Specifically, it applies mandatory access control (MAC) rules using AppArmor or another LSM to prevent unauthorized interactions with the SPIRE Agent or attempts to subvert the identity provisioning process.

4.3. Infrastructure Layer

At the bottom of the stack are the Kubernetes Nodes that form the physical or virtual infrastructure for the cluster. The SPIRE Server and Agent run on these nodes, as do the containers and kernel-level components used by KubeArmor.

KubeArmor enforces node-level policies designed to restrict the behavior of workloads and system components. These Host Security Policies limit the system calls, file access, and network connections that individual workloads or even system services can initiate. Such restrictions are especially important for protecting the SPIRE Agent, which performs sensitive cryptographic operations and exposes a local API endpoint.

We can then define that KubeArmor applies two classes of policies:

- **SPIFFE CSI Driver Security Policies:** These control the operations the driver can perform, limiting its access to only necessary resources and interfaces.
- **Host-Level Security Policies:** These protect the node, SPIRE components, and control plane agents from malicious actions or privilege escalations.

It is important to note that the integration of KubeArmor is non-intrusive to SPIRE. It does not modify the SPIRE codebase or disrupt its control flow. Instead, it monitors and controls system-level interactions through policy enforcement at runtime.

4.4. Security Policy Implementation

To implement the proposed methodology, we leverage KubeArmor's ability to enforce fine-grained access control policies that govern interactions within Kubernetes clusters. In our solution, KubeArmor is used to protect the SPIFFE CSI Driver and sensitive paths in the Kubernetes host filesystem, focusing primarily on the hostPath volumes that facilitate communication between the CSI Driver and the underlying node infrastructure.

The security mechanism is structured around two complementary sets of policies, as defined in Section 4.3, the **SPIFFE CSI Driver Security Policies**, and the **Kubernetes Host Security Policies**. By enforcing these policies, KubeArmor brings key regions of the system under the control of LSM-based protections, adding a runtime layer of defense rooted in the kernel.

Both policy sets are implemented as YAML definitions following KubeArmor's specification and are designed to operate transparently alongside existing Kubernetes and SPIFFE/SPIRE components. Although our focus is the SPIFFE CSI Driver, the same policy-based approach is generalizable to other workloads that use hostPath volumes or interact with sensitive host resources.

4.4.1. SPIFFE CSI Driver Security Policies

Within the pod where the SPIFFE CSI Driver runs, the policy created was developed with a **Zero Trust** strategy [Rose et al. 2020] as its motivation, restricting file operations to only those explicitly required. The policy prohibits the creation, modification, deletion, or execution of any file outside a predefined allowlist. This allowlist is constructed through an analysis of the CSI Driver's source code and expected behavior. In the event of a pod compromise, any malicious attempt to escalate privileges or tamper with the driver's functionality would be blocked by AppArmor's enforcement.

4.4.2. Kubernetes Host Security Policies

For the Kubernetes nodes themselves, a restrictive host-level policy is applied designed to isolate access to critical directories used by the Kubernetes control components and the SPIFFE/SPIRE stack. Unlike the pod-level policy, which is tailored for a specific workload, host policies must address a more complex environment involving multiple interacting services.

The primary goal is to prevent unauthorized processes from accessing or manipulating host resources required for the operation of the SPIFFE CSI Driver and SPIRE. To that end, only trusted components, such as KubeArmor, the Kubernetes Control Plane, and selected daemons are granted access to these paths.

Specifically, the policy enforces access control on:

- `/run/spire/agent-sockets/`: which holds the UNIX socket through which the SPIRE Agent exposes its Workload API to the CSI Driver.
- `/var/lib/kubelet` and its subdirectories: which are critical to the Kubernetes runtime and include metadata and mount points required by the CSI subsystem.

These restrictions are essential to contain the blast radius of a potential node compromise, minimizing the risk of lateral movement and identity spoofing within the cluster.

4.5. Execution Flow Example

To better illustrate the behavior of the architecture, we describe the typical execution flow when a new k8s pod is deployed:

1. A developer or automation tool schedules a new pod on a Kubernetes node. The Control Plane handles scheduling and resource allocation.
2. Upon initialization, the SPIFFE CSI Driver is invoked to mount the SVID into the pod. The driver makes a request to the local SPIRE Agent.
3. The SPIRE Agent attests the workload using node and workload selectors and verifies its eligibility based on registration entries.
4. If attestation is successful, the Agent contacts the SPIRE Server to retrieve or generate the appropriate SVID for the workload.
5. The SPIRE Server issues the SVID and sends it back to the Agent, which delivers it to the CSI Driver.
6. The CSI Driver mounts the SVID into the pod's file system, making it available for mutual TLS communication or other secure interactions.
7. Meanwhile, KubeArmor monitors the system and enforces the defined security policies. It ensures that the SPIFFE CSI pod cannot access unauthorized resources, execute commands that are not explicitly allowed or perform privileged operations that have not been previously specified.
8. On the k8s host side, sensitive directories created by using hostPath volumes have their access control managed by LSMs, preventing access to these directories by unauthorized elements.

This flow highlights the coordination between Kubernetes scheduling, SPIFFE identity issuance, and KubeArmor's runtime enforcement, all contributing to a secure workload onboarding process.

5. Threat Model

The primary goal of the SPIFFE CSI Driver is to minimize the use of hostPath volumes between workloads and the SPIRE Agent Workload API. However, their use cannot be entirely eliminated, as certain hostPath volumes are still required between the SPIFFE CSI Driver containers and the *kubelet* daemon that manages the nodes in the Kubernetes cluster.

To build a Threat Model for SPIFFE CSI Driver focusing on the hostPath problem, we used the STRIDE methodology to conduct some investigations [Shostack 2014]. It is necessary to identify potential security threats inside the context of the solution architecture presented in Section 4. With this goal in mind, we adopt the following assumption regarding the deployment of the SPIFFE CSI Driver in Kubernetes clusters: the SPIFFE CSI Driver pod is the only workload in the cluster permitted to create hostPath volumes.

5.1. Spoofing

An attacker can change the build variables while compiling the SPIFFE CSI Driver binary or building its container image. This may compromise the generated binary and its distribution. This compromised workload could attempt to impersonate a legitimate SPIFFE CSI Driver on the cluster and thereby compromise the entire system.

Some ways to deal with this are to validate and authenticate Git Tags and Commits before building using digital signatures (Git Commit Signing). All SPIFFE CSI Driver instances must be regularly listed, and their identifiers periodically verified.

5.2. Tampering

Similar to the Spoofing threat category, an attacker may manipulate downloaded dependencies during the execution of package management tools to inject malicious code into the build process, resulting in a compromised binary. Mitigation strategies include the use of vulnerability scanners, hash verification of generated software artifacts, and more advanced integrity assurance mechanisms such as software attestation.

It is also crucial to configure trusted repositories for package management software, as they are increasingly subject to supply chain attacks [Ohm et al. 2020].

5.3. Repudiation

Pod logs may be manipulated or may not contain enough information for auditing. This can impact a lack of traceability of the activities carried out in the Kubernetes cluster. Some ways to deal with this are the use of centralized logging with tools like Fluentd or CloudWatch and using labels or annotations to track pod activity.

It is also possible that an attacker with access to the cluster could deny involvement in malicious actions involving the SPIFFE CSI Driver. This highlights the importance of implementing logging and monitoring with proper timestamps and identity verification to CSI and gRPC interfaces. Digital signatures and secure audit trails can also help in tracking actions.

5.4. Information Disclosure

Sensitive data, such as Kubernetes secrets or authentication tokens for cloud platforms, may be exposed within a pod, potentially leading to data leakage. To mitigate this risk, it is recommended to mount secrets using Kubernetes Secrets Volumes rather than injecting them through environment variables. Additionally, implementing periodic secret rotation enhances security. When available, cloud-native secret management solutions, such as AWS Key Management Service (KMS) or Google Cloud KMS, should be used to further safeguard sensitive credentials.

5.5. Denial of Service

The SPIFFE CSI Driver pod may be overloaded with excessive traffic or allocated to a host with insufficient resources, causing outages and service interruption. To deal with it, it is necessary to configure resource request limits on the pod for CPU and Memory and Auto Scaling for pods and nodes in the Kubernetes cluster. Horizontal Pod Autoscaler and Cluster Autoscaler can also be used, as well as third-party tools that implement rate limiting, such as Istio and AWS API Gateway.

If the attacker is unable to mount a privileged directory using a malicious or legit SPIFFE CSI Driver, the *kubelet* daemon can be exploited to cause a denial of service attack by creating a large number of CSI volume mount requests. Mitigation demands enforcing a limit on the number of instances of SPIFFE CSI Driver in the cluster.

5.6. Elevation of Privilege

The SPIFFE CSI Driver and other containers utilizing SPIRE may obtain superuser privileges on the host node or access Kubernetes APIs beyond their designated namespace,

potentially resulting in full cluster compromise. To mitigate these risks, containers should be configured to run as non-root users and, where feasible, use a read-only filesystem. Furthermore, access to Kubernetes APIs should be tightly controlled using Role-Based Access Control (RBAC). Additional safeguards include enforcing strict namespace isolation and limiting Kubernetes Service Account permissions through fine-grained access control policies.

6. Experimental Evaluation

This section presents the experimental evaluation of the solution introduced in Section 4, detailing the evaluation environment, selected scenarios, performance metrics, and key experimental results.

6.1. Objectives

The objective of this experiment is to evaluate the performance impact of the proposed solution, which integrates KubeArmor into a SPIRE-enabled Kubernetes environment. The analysis focuses primarily on latency-related performance metrics associated with the SPIRE Server and SPIRE Agent, including the time required for workload attestation, SVID retrieval and synchronization, and operations involving the datastore and trust bundles. These metrics provide insight into the potential overhead introduced by the runtime enforcement mechanism, particularly in scenarios with dynamic workload registration and identity issuance.

It is worth noting that this experimental evaluation does not aim to empirically verify whether the proposed solution mitigates the threats outlined in the threat model, such as reduction of the attack surface or prevention of privilege escalation. These aspects are considered out of scope. Our focus in this set of experiments is on assessing the performance impact of the solution. We assume that the use of LSM by KubeArmor contributes to enforcing fine-grained access control and mitigating privilege escalation threats, for instance, even though this is not empirically verified in our experiments.

6.2. Setup and Scenarios

The Kubernetes cluster used in the experiment consists of a Control Plane node of the *general.medium* flavor and four *general.large* Worker nodes, all deployed via Kubeadm on virtual machines within a private OpenStack cloud environment. Experiment data was collected using the Prometheus monitoring tool through its integration with SPIRE for telemetry. Table 1 provides additional details on the experimental setup and hardware configuration.

The experiment was conducted under two distinct scenarios. In the first, referred to as the *baseline*, the cluster ran only SPIRE. In the second, referred to as the *solution* scenario, the cluster included both SPIRE and the solution described in Section 4. In each scenario, 100 different workloads were executed, each registering with the SPIRE Server and subsequently requesting its corresponding SVID from the SPIRE Agent through the SPIFFE CSI Driver installed on the cluster.

The workloads used in the experiment are created in the form of k8s pods using a minimal base image, containing only the commands for registering with the SPIRE Server and requesting their respective SVID.

Quantity	Type	VM Flavor	vCPUS	RAM	Disk
04	Worker	*general.large*	4	8 GB	20 GB
01	Control Plane	*general.medium*	2	4 GB	20 GB

Table 1. Virtual Machines configuration of the k8s cluster used in the experiment for both selected scenarios (baseline and solution).

6.3. Metrics Evaluated

In this Subsection we present the metrics considered in this experiment. We chose to evaluate some performance metrics of the SPIRE Server and SPIRE Agent, focusing on the latency of some common operations in the SPIRE life cycle to provide identity to workloads present in the Kubernetes cluster:

- **SPIRE Agent Workload API Latency (wapi)**: is one of the main metrics exposed by the SPIRE Agent for monitoring the latency of the workload attestation process in milliseconds. It measures the time required for the SPIRE Agent to complete the attestation of a workload through the Workload API. That is the time between the start of the identity request by the workload and the delivery of an SVID.
- **SPIRE Agent Entries Latency (entries)**: measures the latency of the SVIDs synchronization process on the SPIRE Agent in milliseconds. This synchronization happens periodically to ensure that valid SVIDs are available for workloads authenticated through the Workload API Socket.
- **SPIRE Server Bundle Retrieval Latency (bundle)**: refers to the time it takes for the SPIRE Server to retrieve its trusted bundle in milliseconds. A Trust Bundle is essentially a set of trusted certificates used to validate identities between different domains or between nodes within the same Trust Domain;
- **SPIRE Server Datastore Latency (datastore)**: measures the time required in milliseconds for operations on the Datastore, which is the component responsible for persistently storing data such as SVIDs, Trust Bundles, workload logs, and other important information for the operation of SPIRE;
- **SPIRE Server Synchronization Latency (syncro)**: indicates the time in milliseconds it takes for the SPIRE Server to synchronize data from the identity store with its internal subsystems, including workload data, federated entries, and Trust Bundles.

6.4. Results and Discussion

In this Subsection, we report and interpret the results of our experimental evaluation, comparing the baseline and solution scenarios in terms of performance metrics, followed by a discussion of their implications with respect to the proposed solution.

Figure 2 provides a general comparative analysis of the selected SPIRE performance metrics across the investigated scenarios. Upon initial examination, it is evident that the baseline and solution scenarios exhibit similar performance for most metrics, with only minor differences. The metric displaying the most significant variation between the two scenarios was the SPIRE Agent Entries Latency *(entries)*.

This behavior is within expectations since introducing a new component in a given system can increase the latency of operations performed on it. For this metric, the median

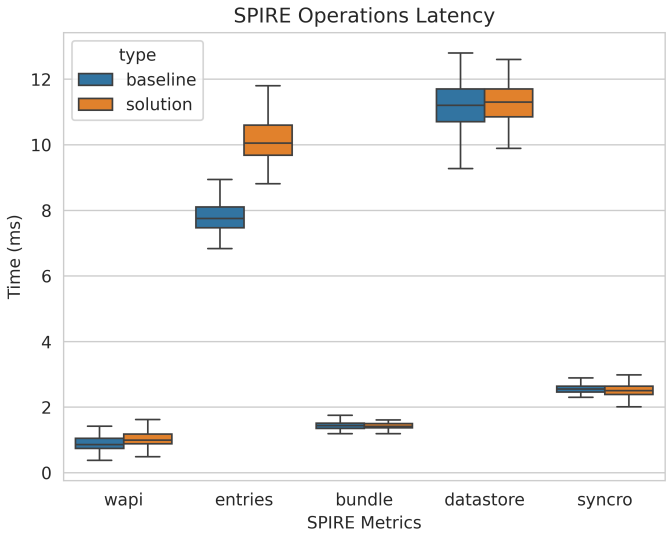

Figure 2. Comparison of the investigated scenarios (baseline and solution) regarding the selected SPIRE performance metrics.

values for the baseline scenario were $7.75ms$, while for the solution scenario, it was $10.05ms$, indicating a considerable increase of approximately 29.68%. Figure 3 shows a detailed temporal comparison of the *entries* metric, more specifically when we compare the latency measurements individually for both scenarios of interest.

The high initial peaks observed in Figure 3 (from time instants $t5$ to $t8$) suggest an initialization overhead that eventually stabilizes. One potential explanation for this is the impact of the security policies enforced by KubeArmor, which directly affect the SPIFFE CSI Driver. This influence may arise during the handshake process between the SPIFFE CSI Driver and the SPIRE Agent Workload API via gRPC interfaces, or due to the syscall inspection performed by the AppArmor LSM. Additionally, the SPIRE plugin-based model requires plugins to be loaded or restarted at certain points, and the security policies implemented in the solution scenario could potentially interfere with this process.

Considering these aspects, the observed overhead of approximately 29% should be interpreted in light of the cluster's operational priorities. In environments where performance is critical, such an increase may represent a relevant trade-off. However, in scenarios where security takes precedence over latency, this level of overhead may be entirely acceptable. It is also important to note that direct comparisons with other studies are not straightforward, as variations in cluster configurations, workload profiles, enforcement mechanisms, and evaluation methodologies can significantly affect performance outcomes. We leave this assessment to the reader, acknowledging that the actual impact of the proposed solution will vary depending on workload characteristics, cluster size, and specific configuration choices.

Next, when we compare the previous results with a similar metric, this time for the SPIRE Server, as is the case of the SPIRE Server Synchronization Latency *(syncro)*, we can see in Figure 4 that most of the time the solution scenario had relatively similar results or even lower in some cases, except for some peaks (between time instants $t1$ and $t6$). A smaller difference between the baseline and solution scenarios for this metric was

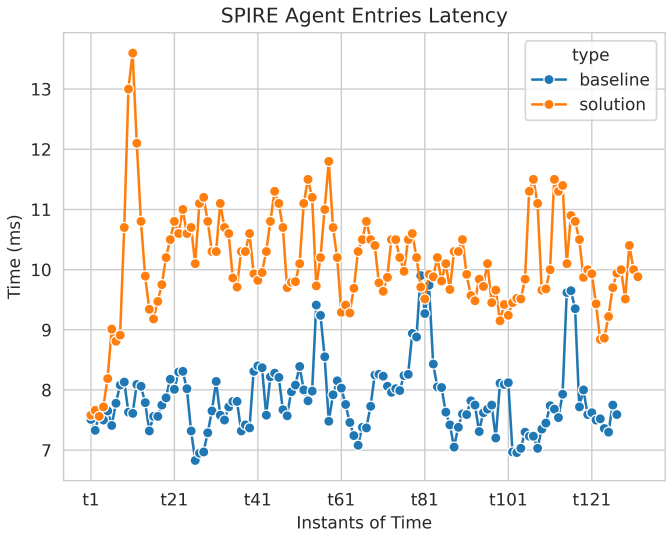

Figure 3. Temporal comparison of the SPIRE Agent Entries Latency (entries) metric for the baseline and solution scenarios.

within the expected results since SPIRE Server metrics tend to be less affected by the SPIFFE CSI Driver, a result that was also observed in the other metrics related to it, as can be seen in Figures 2 and 5.

In Figure 5, we can see the remaining metrics related to the SPIRE Server. We can see that, with the exception of a few spikes, the behavior remains similar, which is within what was expected for the SPIRE Server metrics, since it does not communicate directly with the KubeArmor components. This work is handled by the SPIRE Agent and, more specifically, the SPIFFE CSI Driver, where access control by the LSM is more intense, since we adopted a security policy for the CSI Driver inspired by the principles of Zero Trust. One of the possible factors for the spikes observed in the metrics could be fluctuations in the use of the OpenStack cloud.

7. Conclusion and Future Work

The primary contribution of this work is the proposal of a methodology aimed at enhancing the security of applications utilizing hostPath volumes in Kubernetes clusters. This methodology focuses on strengthening access control at the interception regions between Kubernetes pods and host file systems, specifically during their interaction. It achieves this by integrating LSMs into the Kubernetes cluster ecosystem.

We implemented the proposed methodology using KubeArmor [KubeArmor 2025], a tool that enables the creation of restrictive security policies based on Zero Trust principles, facilitated by the AppArmor LSM for Kubernetes nodes and pods. The solution was tested within a SPIRE environment, incorporating the SPIFFE CSI Driver to enhance the security of the hostPath volume mounted between the driver and the Kubernetes kubelet control daemon.

In the experimental evaluation, we observed minimal performance differences between the baseline and solution scenarios for most SPIRE performance metrics, indicating no significant overhead. The only exception was the SPIRE Agent Entries Latency (en-

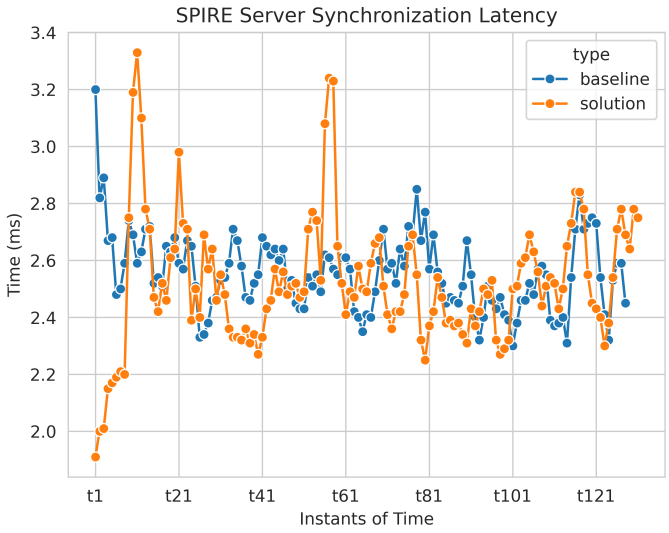

Figure 4. Temporal comparison of the SPIRE Server Synchronization Latency (syncro) metric for the baseline and solution scenarios.

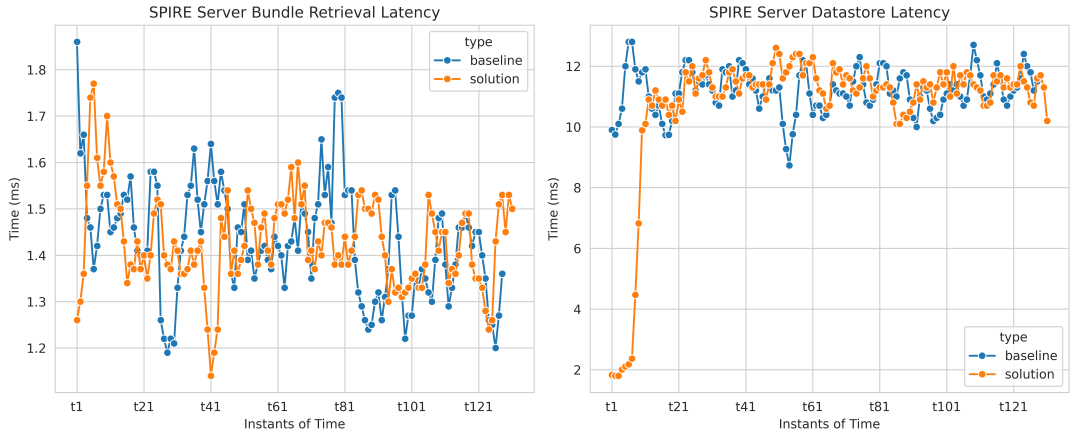

Figure 5. Temporal comparison of the SPIRE Server Bundle Retrieval Latency (bundle) and SPIRE Server Datastore Latency (datastore) metrics for the baseline and solution scenarios.

tries), which increased by approximately 29.68% in the solution scenario compared to the baseline. While this increase is notable, it can be considered acceptable, particularly in environments where security is prioritized over performance, as the solution scenario provides significant security benefits.

Future work should focus on a more comprehensive evaluation of the proposed methodology, incorporating a wider range of metrics. Additionally, varying the experimental setup, such as using Kubernetes clusters of different sizes or exploring SPIRE environments with federation, would offer deeper insights into the methodology's impact. Expanding the scope by applying the proposed methodology to different applications utilizing hostPath volumes and experimenting with various LSMs could further contribute to the understanding and enhancement of security in Kubernetes environments.

Acknowledgments

This work has been financed through the *"SPIRE Scalability and Interoperability for Cloud Native Environments"* (S4CN) project, a collaboration between Hewlett Packard Enterprise (HPE) - Brazil and the EMBRAPII unit UFCG-CEEI (Universidade Federal de Campina Grande) with the incentive of the Informatics Law (Law 8.248 from October 23rd, 1991).

References

Deng, S., Zhao, H., Huang, B., Zhang, C., Chen, F., Deng, Y., Yin, J., Dustdar, S., and Zomaya, A. Y. (2024). Cloud-native computing: A survey from the perspective of services. *Proceedings of the IEEE*, 112(1):12–46.

D'Silva, D. and Ambawade, D. D. (2021). Building a zero trust architecture using kubernetes. In *2021 6th International Conference for Convergence in Technology (I2CT)*, pages 1–8.

Findlay, W., Somayaji, A., and Barrera, D. (2020). bpfbox: Simple precise process confinement with ebpf. In *Proceedings of the 2020 ACM SIGSAC Conference on Cloud Computing Security Workshop*, CCSW'20, page 91–103.

Garfinkel, T. and Rosenblum, M. (2003). A virtual machine introspection based architecture for intrusion detection. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*.

Ghavamnia, S., Palit, T., Benameur, A., and Polychronakis, M. (2020). Confine: Automated system call policy generation for container attack surface reduction. In *23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020)*, pages 443–458, San Sebastian. USENIX Association.

Gunathilake, K. and Ekanayake, I. (2024). K8s pro sentinel: Extend secret security in kubernetes cluster. In *2024 9th International Conference on Information Technology Research (ICITR)*, pages 1–5.

Gülcü, B., Karaaslan, E., and Kantarcioglu, M. (2021). Security challenges in kubernetes. *ACM Transactions on Privacy and Security (TOPS)*, 24(3):1–27.

Islam Shamim, M. S., Ahamed Bhuiyan, F., and Rahman, A. (2020). Xi commandments of kubernetes security: A systematization of knowledge related to kubernetes security practices. In *2020 IEEE Secure Development (SecDev)*, pages 58–64.

KubeArmor (2025). Kubearmor: Runtime security enforcement engine based on linux security modules. `https://kubearmor.io/`. Accessed: 2025-04-07.

Lee, J., Kim, H., and Cho, Y. (2022). Kubearmor: Runtime security enforcement with lsms in kubernetes. In *IEEE Symposium on Security and Privacy Workshops (SPW)*.

Mirantis (2022). Securing kubernetes csi drivers: Challenges and recommendations. `https://www.mirantis.com/blog/securing-kubernetes-csi-drivers/`.

Ohm, M., Plate, H., Sykosch, A., and Meier, M. (2020). Backstabber's knife collection: A review of open source software supply chain attacks. In *Detection of Intrusions and*

Malware, and Vulnerability Assessment: 17th International Conference, DIMVA 2020, Lisbon, Portugal, June 24–26, 2020, Proceedings 17, pages 23–43. Springer.

Peck, M., Morag, A., and Sergent, R. (2019). Hacking kubernetes: Threat matrix and mitigation strategies. `https://www.cncf.io/blog/2019/11/14/hacking-kubernetes-threat-matrix-and-mitigation-strategies`. CNCF Whitepaper.

Pereira Ferreira, A. and Sinnott, R. (2019). A performance evaluation of containers running on managed kubernetes services. In *2019 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 199–208.

Perera, H. P. D. S., Reza, B., De Silva, H. S. T., Karunarathne, A. D. H. U., Ganegoda, B., and Senarathne, A. (2022). Docker container security orchestration and posture management tool. In *2022 13th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, pages 1–6.

Rodigari, S., O'Shea, D., McCarthy, P., McCarry, M., and McSweeney, S. (2021). Performance analysis of zero-trust multi-cloud. In *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*, pages 730–732.

Rose, S., Borchert, O., Mitchell, S., and Connelly, S. (2020). Zero trust architecture. Technical Report SP 800-207, National Institute of Standards and Technology (NIST).

SE-RAN-5G (2025). 5g-kubearmor: critical security extensions to sd-ran's nrt-ric. `https://www.5gsec.com/tech/tech-5g-kubearmor`. Accessed: 2025-04-07.

Shostack, A. (2014). *Threat modeling: Designing for security*. John wiley & sons.

Smalley, S., Fraser, T., and Vance, C. (2001). Linux security modules: General security hooks for linux.

SPIFFE (2023). Spiffe csi driver documentation. `https://github.com/spiffe/spiffe-csi`. Accessed: 2025-04-07.

SPIFFE (2025). Spiffe: Secure production identity framework for everyone. `https://spiffe.io/`. Accessed: 2025-04-07.

van Vugt, T. M. and Malik, T. (2023). A practical analysis of open-source security tools in microservice kubernetes environments. In *2023 Cyber Research Conference - Ireland (Cyber-RCI)*, pages 1–8.

Vasilenko, D. and Mahesh, K. (2019). Dynamic tenant provisioning and service orchestration in hybrid cloud. *International Journal on Cloud Computing: Services and Architecture*, 9:1.

Zeng, Q., Kavousi, M., Luo, Y., Jin, L., and Chen, Y. (2023). Full-stack vulnerability analysis of the cloud-native platform. *Computers & Security*, 129:103173.

Zhu, H. and Gehrmann, C. (2022). Kub-sec, an automatic kubernetes cluster apparmor profile generation engine. In *2022 14th International Conference on COMmunication Systems & NETworkS (COMSNETS)*, pages 129–137.