National College of Ireland

Project Submission Sheet

Student Name: Abhishek Murari Sharma

Student ID: X23406461

Programme: MSCCLOUDJAN25AO **Year:** 2025

Module: MSc Research Project

Lecturer: Prof. Shreyas Setlur Arun

Submission Due Date: 11/12/2025

Project Title: Securing Multi-tenant Kubernetes Cluster Using eBPF

Word Count:

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the references section. Students are encouraged to use the Harvard Referencing Standard supplied by the Library. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action. Students may be required to undergo a viva (oral examination) if there is suspicion about the validity of their submitted work.

Signature: Abhishek Murari Sharma

Date: 11/12/2025

PLEASE READ THE FOLLOWING INSTRUCTIONS:

1. Please attach a completed copy of this sheet to each project (including multiple copies).
2. Projects should be submitted to your Programme Coordinator.
3. **You must ensure that you retain a HARD COPY of ALL projects**, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer. Please do not bind projects or place in covers unless specifically requested.
4. You must ensure that all projects are submitted to your Programme Coordinator on or before the required submission date. **Late submissions will incur penalties.**
5. All projects must be submitted and passed in order to successfully complete the year. **Any project/assignment not submitted will be marked as a fail.**
6. **Please check that you read AI and Academic Integrity Acknowledgement Supplments in this document**

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

AI Acknowledgement Supplement

MSc Research Project

Securing Multi tenant Kubernetes Cluster using eBPF

Your Name/Student Number	Course	Date

This section is a supplement to the main assignment, to be used if AI was used in any capacity in the creation of your assignment; if you have queries about how to do this, please contact your lecturer. For an example of how to fill these sections out, please click here.

AI Acknowledgment

This section acknowledges the AI tools that were utilized in the process of completing this assignment.

Tool Name	Brief Description	Link to tool

Description of AI Usage

This section provides a more detailed description of how the AI tools were used in the assignment. It includes information about the prompts given to the AI tool, the responses received, and how these responses were utilized or modified in the assignment. **One table should be used for each tool used**.

[Insert Tool Name]	
[Insert Description of use]	
[Insert Sample prompt]	[Insert Sample response]

Evidence of AI Usage

This section includes evidence of significant prompts and responses used or generated through the AI tool. It should provide a clear understanding of the extent to which the AI tool was used in the assignment. Evidence may be attached via screenshots or text.

Additional Evidence:

[Place evidence here]

Additional Evidence:

[Place evidence here]

National College of Ireland

Securing Multi-Tenant Kubernetes Clusters Using eBPF

MSc Research Project
Cloud Computing

Abhishek Murari Sharma
Student ID: 23406461

School of Computing
National College of Ireland

Supervisor: Prof: Shreyas Setlur Arun

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Abhisehk Murari Sharma
Student ID:	23406461
Programme:	MSc Cloud Computing
Year:	2025
Module:	MSc Research Project
Supervisor:	Prof. A Setlur
Submission Due Date:	11/12/2025
Project Title:	Securing Multi-Tenant Kubernetes Clusters Using eBPF
Word Count:	XXX
Page Count:	22

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Abhishek Murari Sharma
Date:	11th December 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Securing Multi-Tenant Kubernetes Clusters Using eBPF

Abhishek Murari Sharma
23406461

Abstract

In the cloud-native environment, multi-tenancy is on the rise to better optimize resource usage and lower the operation expenses, making kubernetes the de facto standard of container orchestration. Nonetheless, the conventional security tools, including iptables-based NetworkPolicy, fixed firewall settings, and user space intrusion detection tools, have a hard time supporting the dynamic security needs of contemporary multi-tenant deployments. These approaches have limited granular visibility, high performance overheads at scale, and have low levels of contextual awareness of application identity or behaviour. This Project explores the usage of eBPF-based security applications using Cilium for better tenant isolation and threat detection in a muliti-tenant kubernetes clusters. Through comprehensive experimental evaluation using a reproducible testbed with kind, Cilium and Kubernetes RBAC, this study compares the performance indicators and monitoring capabilities, and the enforcement functions of unauthorized communication between tenants using the eBPF based security mechanisms with the traditional approaches. The findings endorse that eBPF method facilitates inbuilt, real-time visibility and security policies with 100% insignificant resource oversight, accomplishing 24,086 Mbps throughput relative to 23,245 Mbps of conventional NetworkPolicy, and offers Layer 7 oversight capabilities. This work provides a viable direction toward building dynamic, efficient, and identity-sensitive security models of multi-tenant Kubernetes systems.

Keywords: eBPF, Kubernetes security, multi-tenant isolation, Cilium, container security, cloud-native security

1 Introduction

Kubernetes is becoming the industry standard of container orchestration in cloud-native systems that have radically changed the deployment, management, and scalability of distributed applications by modern organizations. The capabilities offered by the platform to automate the container lifecycle management, offer self-healing capabilities and support horizontal scaling have made it an indispensable platform for organizations ranging from startups to large enterprises. With the growing adoption of cloud-native, modern organizations are using multi-tenant architectures where multiple teams/departments/customer use a similar Kubernetes cluster to increase operational efficiency, as well as for efficient utilization of resources and cost savings from infrastructure. This method of consolidation enables the organizations to make the most out of their Kubernetes infrastructure

and retain the flexibility and agility to support the modern application development and deployment practices.

However, this consolidation presents great challenges in the areas of security, compliance, and reliability of operations that are not addressed sufficiently well by traditional security mechanisms. Cross-tenant data exposure, privilege escalation issues, inconsistent use of policies, and inadequate observability are all important topics in the shared cluster space where the security boundary between tenants needs to be carefully enforced. Traditional security mechanisms in Kubernetes (for example where network policies based on iptables are used, firewall configurations are static, and intrusion detection systems are implemented in user space) are not well suited to handle the needs of modern distributed workloads running in dynamic, large-scale multi-tenant deployments. These approaches have poor granular visibility into application layer behaviors, performance overheads at scale, and operate with poor contextual knowledge of application identity or behavior patterns, which often means that they are not well suited for the security needs of modern cloud-native systems.

In this study, we examine how eBPF-based security primitives, deployed with Cilium, can help to improve the isolation of tenants and threat detection in multi-tenant Kubernetes environments. eBPF offers a radically new policy enforcement and monitoring system and network security design using kernel-level observability and programmable datapaths. The Extended Berkeley Packet Filter (eBPF) technology provides safe and high performance kernel-level programmability, which is required to implement both security policies as well as monitoring capabilities at the kernel level without having to modify the kernel source code or kernel system calls. This research is aimed at addressing a gap in the academic literature by providing quantitative evidence of the effectiveness of eBPF for securing multi-tenant Kubernetes clusters through comprehensive experimental evaluation, performance benchmarking and security analysis which should be useful to production deployments and provide a critical guidance for deployment in multi-tenant Kubernetes clusters.

1.1 Problem Statement

There are several essential issues with securing multi-tenant Kubernetes clusters that the available solutions fail to resolve:

1. **Limited Granularity in Network Policies**: There is Limited granularity in the network policies as traditionally systems use L3 and L4 security layer where an attacker can act to send the post api request instead of get api and get the access to the endpoint api of the server.

2. **Performance Overhead**: Iptables based policies have huge CPU and latency overhead which not scale very well as pods, services and rules increases. Performance of rule-matching declines exponentially with chain length and hence is a bottleneck in high throughput production environments.

3. **Insufficient Real-Time Observability**: There is insufficient real-time obervability in the current multi-tenant kubernetes technology

4. **Static, Non-Adaptive Policy Models**: static Policy Management Policies that are defined manually cannot dynamically respond to the changing workload behavior or to new threats.

5. **Cross-Tenant Isolation Complexity**: Complex Multi-Layer Security To accomplish full tenant isolation requires RBAC, namespaces, network policies and resource quotas all to be coordinated. Misconfigurations at any of the layers can allow for lateral movement, data exfiltration or privilege escalation resulting in a hard to manage security surface with a vast amount of possible gaps.

These inter-related challenges will require dynamic, efficient, identity-aware security framework that can deliver the granularity, performance and observability in today's multi-tenant Kubernetes environments.

1.2 Motivation

The increased application of multi-tenant Kubernetes clusters has posed severe security dilemmas that require immediate concern. A single security breach or misconfiguration by one organization can impact over tenants simultaneously when more than one organization is on the same infrastructure which is incredibly damaging compared to the limited destruction of single-tenant systems. The recent security breach incidents involving the cloud have demonstrated how a failure in the shared infrastructure can result in massive data breaches across several organizations at a time, so it is not only a technical issue but a significant business and regulatory one. Meanwhile, businesses require that their applications perform well and in a timely fashion to satisfy the expectations of users and to fulfill the demands of the services, but most traditional security tools impose serious performance bottlenecks and slow down the performance of the applications, particularly when these clusters increase in size, with more pods and security rules. Strict regulations, such as a strong tenant isolation requirement, detailed audit logs, and provable security controls, pressure industries such as finance, healthcare, government, and others to comply with any of the standards, such as GDPR, HIPAA, and PCI-DSS. Such organizations require security implementations that perform efficiently in addition to being able to demonstrate compliance in terms of thorough logging and policy checking. Although eBPF technology has become a widely used technology in industry and larger cloud providers have begun to use it as a security and networking technology, systems such as Cilium have demonstrated promising performance in real production environments, there has not been sufficient scholarly research to systematically test the effectiveness of these eBPF-based solutions in the context of multi-tenant Kubernetes. The accessible majority of knowledge is of an industry report and vendor documentation instead of academic analysis through controlled experiments and quantitative measurements. The study fills such a gap, as it offers systematized assessment, controlled testing, and evidentiary outcomes regarding the level of security performance and effective use of eBPF-based solutions in comparison with conventional ones, enabling both researchers and practitioners to make informed choices regarding the implementation of these technologies in genuine multi-tenant settings.

1.3 Research Question

What does the security mechanisms that are implemented via eBPF and enforced by Cilium offer versus the traditional counterparts in terms of tenant isolation and threat detection in multi-tenant Kubernetes clusters and what are the performance characteristics of these implementations? Effectiveness and efficiency are the two key aspects of security of multi-tenant Kubernetes that will

be considered in this research question. The former one looks at the statement that security mechanisms operating on eBPF, specifically through Cilium, can provide superior tenant isolation and threat-detection functionality to iptables-based NetworkPolicy mechanisms. The investigation is necessary because a multi-tenant environment must have adequate sound isolation facilities that can prevent cross-tenant data breach, unauthorized access and lateral movement attacks. The second factor is that of considering the performance implication, which is the realization that security techniques must be not only efficient and effective, but also efficient enough to be put into production without incurring unacceptable overhead. The question acknowledges that the eBPF technology has the possible advantages in terms of being programmable in the kernel space and identity-aware policies, but must be empirically demonstrated whether such theoretical advantages would create practical advantages in the real-world situation of multi-tenant deployments. The research provides quantitative evidence by comparing eBPF-based solutions and traditional solutions when making technology choices and deployments in the organization that has multi-tenant Kubernetes architectures.

1.4 Research Approach

To evaluate the effectiveness of eBPF-based security mechanisms in multi-tenant Kubernetes environments, the study uses the evaluation approach of evidence through experimental analysis, which is systematic, reproducible, and quantitative. The strategy comprises four unique stages that progressively develop to offer overall assessment of security performances and effectiveness.

Phase 1: Testbed Development Involves developing testbeds, consisting of building a reproducible Kubernetes testbed, using kind (Kubernetes in Docker), which can be used to build multi-node clusters that are reproducible and provide an accurate simulation of production environments.

Phase 2: Baseline Establishment Provides the baseline measurements by assessing default connectivity patterns, network latency characteristics and resource utilization measurements under security policies.

Phase 3: Security Policy Implementation and Testing Executes and tests security policy, and in this step, the implementation and enforcement of traditional NetworkPolicy is carried out as the baseline with which how eBPF-based models can be compared. NetworkPolicy offers L3/L4 enforcement via iptables which is the current industry practice and offers a benchmark against which to measure the benefits provided by eBPF-based solutions.

Phase 4: Performance Benchmarking Performs the enormous performance benchmarking with the standardized performance testing tools such as the iperf3 network through testing, curl-based latencies testing, resource consumption testing with kubectl top and Prometheus resources metrics collection.

1.5 Research Challenges

The process of securing multi-tenant Kubernetes clusters does have a number of challenges that are interconnected to make the process more difficult, both to implement and to evaluate. Isolating tenants fully needs a close sense of coordination between several security layers such as RBAC, namespaces, and network policies such that a failure to configure each security layer correctly can lead to attack vectors that are hard to find.

In distributed systems, assessing the performance effects of security mechanisms is also problematic because of the varying background conditions that make it difficult to cause the actual cost of certain security controls. Also, since clusters can accept more tenants, the number of policies that have to be connected to them can be in the hundreds and it becomes even harder to study policy interactions, debug problems and ensure consistent security postures across the whole system.

1.6 Structure of the Thesis Report

The rest of this dissertation has been structured into eight chapters which are systematic presentation of the research problem, methodology, implementation, evaluation and conclusion.

Chapter 2 Related Work: presents critical review of the existing academic and industrial research in the areas of Kubernetes security, multi-tenancy frameworks, eBPF networking technologies, and identity-sensitive policy models.

Chapter 3 Methodology: gives the research methodology in detail such as research questions, hypotheses, experimental design, testbed architecture, security policy implementation strategies, test structure design, and measures of evaluation and respective formulas and statistical techniques.

Chapter 4 Design Specification: This chapter describes the decision making process behind architectural decisions, technical requirements, and design compromises that influenced the decision to implement the testbed.

Chapter 5 Implementation: This chapter outlines how the proposed solution would be implemented giving the final step to the implementation and output generated, computational tools and languages as well as the real difficulties that occurred during the implementation process.

Chapter 6 Evaluation This Chapter provides extensive experimental testing and performance results are given and the analysis of security efficacy, performance overhead, scaling properties, and observability enhancements in question are detailed. The chapter contains experimental findings of baseline, Network Policy and CiliumNetwork Policy, considering both the performance and security aspects with statistical analysis, comparative evaluation and discussion of the results.

Chapter 7 Conclusion and Future Work: Concludes by discussing and explaining the key findings within the context of the past researches, implications on academic knowledge and practical implementation, limitations of the research design and findings, and recommendations on the possible changes and improvements that may improve the future research. The findings are put in the context of cloud-native security research and critically analyzed in this discussion.Theodoropoulos et al. (2023)

2 Related Work

This chapter conducts literature research on container security, multi-tenant Kubernetes frameworks, and eBPF technology where gaps are found that prompt this research. We methodically analyze every area and form the basis of analyzing eBPF-based security systems in multi-tenant Kubernetes setting.

2.1 Container Security and Multi-Tenancy

Sultan et al. (2019) define the basic container security with four applications: defending against internal application, inter-container defense, host container protection against malicious hosts, and protection against containers, respectively. They name Linux kernel features (namespaces, cgroups, capabilities, seccomp) and LSMs. Primary defenses will be (AppArmor, SELinux). Nonetheless, they have an earlier analysis, adopted and not runtime dynamism consideration which are vital in multi-tenant kubernetes conditions under which policies have to respond dynamically to temporary workloads. Thompson and Kyer (2025) discuss the idea of CI/CD pipeline security with the help of automation. SAST, SBOM generation, and vulnerability Incorporated into a framework with Dockerfile linting and scanning. Although they are suitable in build-time security, their approach is in runtime, enforcement mechanisms. German and Ponomareva (2023) make a comparison between container security, vendors (Aqua, NeuVector, Sysdig Secure, Prisma Cloud, Falco), which displays that the commercial solutions are well-rounded features that need substantial investment, even though detection is available using open-source tools such as Falco, which do not offer any form of prevention. Importantly, the eBPF integration is not discussed in both works. The idea of resource allocation in a multi-tenant proposed by Nguyen and Nguyen and Kim (2022) is dynamic. Kubernetes, cutting down the pod creation time by 40 percent and being fair. However, the security monitoring is not part of their focus on resource management. Bringhenti et al. (2023) The automation of network policy generation in multi-cluster operational configurations is reduced (2023)dangers on misconfiguration, but no validation on run-time.Stanišić et al. (2025) detect generic Kubernetes attacks (images that have not been verified, Kubelet attacks, poor RBAC) and promote Zero Trust architecture, but they are not analyzed qualitatively. eBPF evaluation. All these studies have one crucial gap in common: lack of integrated. The runtime security solutions are based on resource isolation, policy enforcement, and threat. detection.

2.2 eBPF Technology in Cloud-Native Security

In Feng et al. (2024), the security of eBPF is demonstrated using the UHIDS provided by UCloud, universal <1% overhead reaction to threats less than a second. Nevertheless, they are incorrectly narrowed at the host level. is silent on challenges of Kubernetes. The paper by Parola et al. (2022) constructs disaggregated. eBPF network functions with 60-80 Gbps in one-node networks which is on par with Cilium, performance and enhancing maintainability. Although there are some architectural benefits, security integration is yet to be investigated.

Liu et al. (2020) suggest protocol-independent eBPF observability processing 10M+. <1% overhead (TCP: 0.3%, UDP/HTTP: 0.5–0.6%). Chou et al. (2024)combine eBPF monitoring with Prometheus/Grafana to get rid of sidecar, monitoring of requirements by

the use of kernel kprobes. They both dwell upon observability as opposed to. enforcement, which indicates this is repeated gap-enforced policy: eBPF is visible whilst it is not a secure policy enforcement research.

Kim et al. (2025) deeply examine CNI security and discover Cilium that is built on eBPF that has 8.9K Mbps with 1,000 L3/L4 and iptables based CNIs fail 60-70 percent. degradation at 200 pods. The L7 processing however, brings the Cilium throughput down to 94.Mbps vs 6.6K Mbps of Antrea which has performance-security trade-offs. Ragavan and Nadig (2025) and use eBPF to schedule the network noticing latency by 52.66%.

Jin et al. (2024) gives detailed study on enhancing eBPF verification through hybrid aproach of interpretation and symbolic execuion which challenges the Linux verifier limitations. However, their approach on individual programs on a basis per check in contrast to multi-tenancy put-together onto a security systems. The integration of eBPF such as rate limiting, socket is based on the evidence of Behl et al. (2023) by filtering acceleration (50%+ throughput gains), and filtering. While provingintegration security features, the security features incorporation has not been studied, so far.

2.3 Network Policies and Performance Evaluation

NPV suggested by Dong et al. (2024) is applicable in case of quick network policy testing, with 139-651x. run faster than currently used frameworks (Kano, Verikube). NPV verifies 200,000-pod clusters in less memory of 65% less memory in less time of <160ms. Verification however is concerned with being right.

Mazaheri et al. (2016) compare bare-metal system, Docker, and KVM systems based on benchmarking. Containers Container Achieving close to bare-metal performance (<2% degradation) with containers. VMs have high overheads (18 -27has baseline performance but does not have security centric measurements. Mart et al. (2020) address monitoring Prometheus with adaptive anomaly detection using forecasting. thresholds, but no security indicators are evaluated (policy violations, unauthorized access) Rostami (2023) focused on the emphasis on least-privilege access control. Nevertheless, the analysis of RBAC-network polices integration is nonexistent, especially in the case of eBPF-based CNIs where identity-aware enforcement generates a natural possibilities of integration.

2.4 Research Gaps and Contributions

The three essential findings in the literature are as follows: (1) studies on container security concentrate on the host/static level mechanisms without considering the business of multi-tenant Kubernetes dynamism, (2) multi-tenant studies focus on resource management and policy automation without considering the capability of monitoring/enforcement of their runtime, and (3) eBPF research studies the capabilities of security, networking, and monitoring separately. The study controlled these gaps by: (1) assessing the usefulness of eBPF in terms of multi-tenancy security, (2) comparing the performance of eBPF with those of traditional solutions

Author / Source / Year	Purpose / Aims	Results / Findings	Key Themes	Strengths / Limits	Similarities / Differences	Notes / Comments
Sultan et al. (2019) Container Security Taxonomy	Survey container security; classify mechanisms	Four use cases identified; systematic threat model	Isolation primitives; attack taxonomy	Comprehensive but pre-K8s era; static focus	Foundation for later works	Establishes baseline security framework
Thompson & Kyer (2025) CI/CD Pipeline Security	Automate build-time vulnerability detection	Build-time checks alone insufficient	Pipeline security; automation	Strong detection; no runtime enforcement	Complements Sultan; highlights gaps	Reinforces need for runtime eBPF
Nguyen & Kim (2022) Multi-Tenant Resource Mgmt	Dynamic resource allocation in K8s	Improved utilization in multi-tenant envs	Autoscaling; resource fairness	No security monitoring integration	Contrasts security-focused works	Demonstrates multi-tenant complexity
Bringhenti et al. (2023) Multi-Cluster Policy	Automate network policy generation	Reduces misconfiguration across clusters	Policy automation; multi-cluster	No runtime enforcement validation	Complements runtime tools	Pairs well with Cilium/ Hubble
Kazenas & Ponomareva (2023) Security Tools	Compare K8s security platforms	Tool capability overview provided	Security tooling landscape	Qualitative only; no eBPF depth	Context for tool selection	Useful for platform comparison
Feng et al. (2024) eBPF Security Apps	eBPF-based host intrusion detection	Sub-second reaction; minimal overhead	Kernel observability; detection	Host-level; not K8s-specific	Aligns with Cilium capabilities	Strong eBPF security justification
Parola et al. (2022) eBPF Network Functions	Disaggregated eBPF services for K8s	Modular architecture; high performance	eBPF modularity; datapaths	No security integration	Supports eBPF performance claims	Architecture foundation for CNIs
Liu et al. (2020) eBPF Observability	Protocol independent network monitoring	<1%overhead; 10M+ events/ sec	High performance tracing	Observability only; no enforcement	Establishes eBPF efficiency	Baseline for overhead comparison
Chou et al. (2024) eBPF Monitoring Platform	K8s monitoring with Prometheus/ Grafana	Practical eBPF-K8s integration	Observability metrics	Monitoring focus; no security	Shows eBPF-K8s feasibility	Integration pattern reference
Kim et al. (2025) CNI Security Analysis	Evaluate CNI security comprehensively	Quantitative performance metrics	CNI comparison; benchmarking	Performance focus; no attack eval	Complements Parola et al.	CNI selection criteria
Ragavan & Nadig (2025) Network-Aware Scheduling	eBPF-based K8s scheduler optimization	52.66% latency reduction achieved	Performance optimization	No security evaluation	Shows eBPF scheduler potential	eBPF for K8s optimization
Mazaheri et al. (2016) Container Benchmarking	Compare bare-metal vs containers vs VMs	Containers near-native performance	Performance baseline	Not security-focused	Context for overhead analysis	Reference for performance impact
Stanišić et al. (2025) K8s Security Overview	Survey K8s security threats	Comprehensive threat analysis provided	Security landscape; threats	Qualitative; no eBPF evaluation	Contextualizes security needs	Threat modeling reference
Behl et al. (2023) eBPF CNI Extensions	Evolutionary eBPF integration strategy	Incremental adoption feasible	eBPF adoption; migration	Performance focus; no security	Practical deployment approach	Adoption strategy reference

Table 1: Literature Review Comparative Table

3 Methodology

3.1 Research Design and Approach

3.1.1 Research Design and Strategy

This research adopts an experimental evaluation methodology for systematically evaluate the effectiveness of eBPF based security mechanisms as compared to traditional approaches. The methodology is based on empirical research in computer science principles, focusing on: (1) Reproducibility: All testbed configurations, scripts, and procedures are version controlled and automated, (2) Quantitative Analysis: Cannot include standardized metrics and statistical methods to evaluate task objectively (3) Comparative Evaluation Direct comparison done between eBPF based and traditional Armeis invested in physical and virtual segmentation (4) iptables-based approaches, and (4) Comprehensive Coverage: Security, performance, and dimensions of observability evaluated in a holistic way. The experimental design is a controlled experiment design, in which independent variables (security mechanism type, policy configuration); systematically manipulated under conditions of extraneous factors (cluster configuration)

3.2 Testbed Architecture

3.2.1 Cluster Architecture

The testbed uses a kind (Kubernetes in Docker) cluster configuration which provides a local testing environment which can be reproduced. The architecture of clusters is to simulate a multi-tenant production environment but not losing the ability to experiment. Architecture Components: (1) Control Plane Node Single control plane node hosting Kubernetes API server, etcd, controller manager and scheduler, (2) Worker Nodes: 4 worker nodes for workload distribution with high availability. (3) CNI Plugin: Cilium (eBPF based) instead of default CNI, (4) Observability Stack: Hubble (Relay + UI) Network flow analysis.

Table 1: Cluster Architecture Specifications

Component	**Specification**	**Rationale**
Platform	kind	Reproducible local testing
Node Count	1 control + 4 workers	Multi-tenant testing capability
Kubernetes Version	v1.28+	Latest stable version
CNI Plugin	Cilium (eBPF)	Primary technology under evaluation
Network Model	IPv4, ClusterIP	Standard Kubernetes networking

3.2.2 Multi-Tenant Environment Configuration

The testbed deploys a multi-tenant architecture with two namespaces of tenant, each having complete isolation mechanisms.

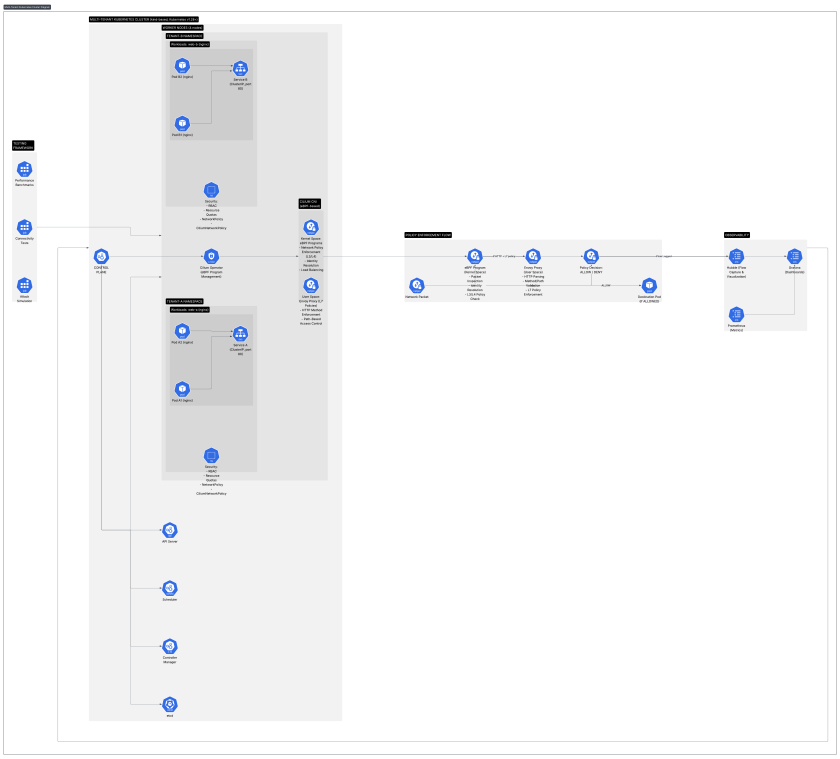

Figure 1: High Level Architecture Diagram of the System

3.3 Security Policy Implementation

3.3.1 Traditional NetworkPolicy Configuration

Traditional Kubernetes NetworkPolicy offers L3/L4 network isolation through pod. selectors, namespace selectors and IP blocks. The testbed is used to implement a baseline. Denying cross-namespace traffic NetworkPolicy. Characteristics of the policies: (1) The level of enforcement L3 (IP) and L4 (port/protocol), (2) Scope: Namespace- wide, (3) Ingress Rule: Only allow traffic which is ingested by. same namespace, (4) Egress Rule: And permit traffic to same namespace and DNS, (5) Implementation: iptables-based

Table 2: NetworkPolicy Configuration Summary

Element	**Configuration**	**Purpose**
Policy Type	Ingress + Egress	Bidirectional traffic control
Pod Selector	{} (all pods)	Apply to all pods in namespace
Ingress Source	namespaceSelector: tenant-a	Allow only same-namespace
Egress Dest	namespaceSelector: tenant-a + DNS	Allow egress + DNS
Enforcement	L3/L4 only	IP and port-based filtering

3.3.2 CiliumNetworkPolicy L7 Enforcement

CiliumNetworkPolicy helps extending kubernetes NetworkPolicy which uses L7(application-layer) security layer, which is enabled by eBPF's functionality to inspect the HTTP traffic at the kernel level.

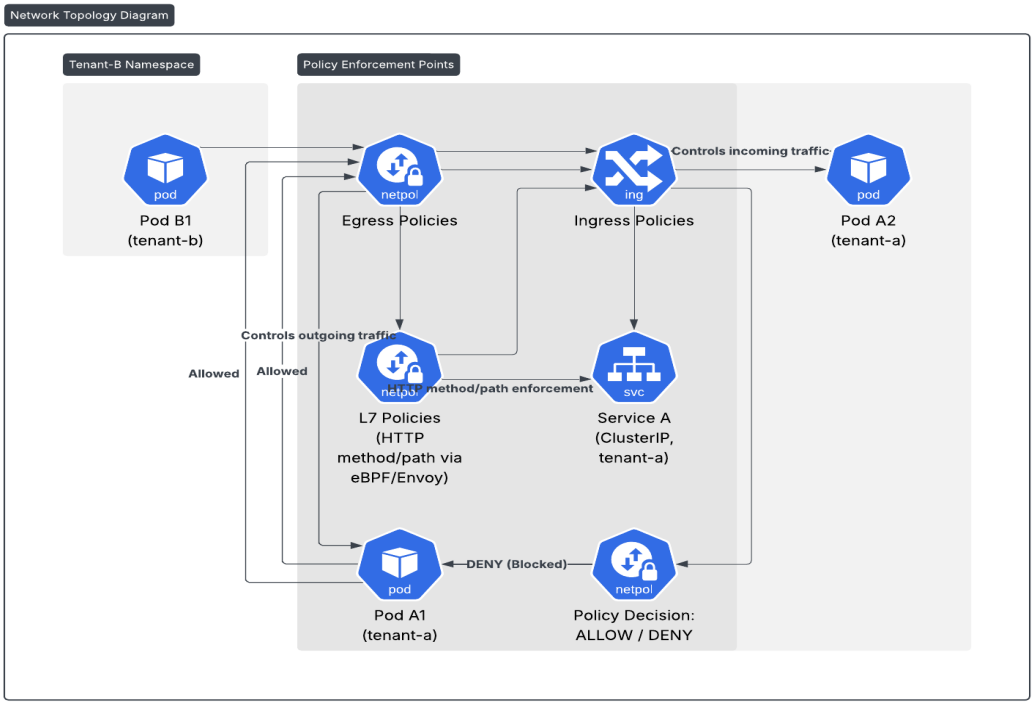

Figure 2: Network Topology Diagram

3.3.3 Workload Configuration

Each tenant is then deployed on nginx-based web app that provides realistic HTTP workload for testing, It creates 2 replicas with resources limits of (100m CPU, 128Mi RAM requests; 500m CPU, 512Mi RAM limits). The nginx configuration helps providing multiple endpoints via status, default api checks.

3.4 Testing Framework

3.4.1 Connectivity Testing

Connectivity tests confirm fundamental network patterns of communication and determine a baseline behavior. Test scenarios are Pod to Pod,Pod to Service, Cross Namespace and DNS Resolution.

3.4.2 Attack Simulation Framework

It has been tested using the attack simulation framework and the following attack vectors: (1) Cross-Namespace Unauthorized Access, (2) Unauthorized HTTP Method (POST to GET-only endpoint), (3) Unauthorized Path Access (/api instead of /status), (4) Host Network Privilege Escalation, and (5) Service Account Token Access.

4 Design Specification

This section gives the design decisions, architectural decisions, and technical. guidelines of the multi-tenant Kubernetes testbed of eBPF-based security

4.1 Design Objectives

The testbed design deals with three major objectives: (1) Reproducibility- automatic setups and controlled versions allows steady controlled experimental conditions; (2) Comparative Evaluation- direct comparison between eBPF-based and traditional iptables-based security systems; and (3) Comprehensive Coverage-concurrent analysis of security effectiveness, observability capabilities, and performance overhead.

4.2 Architectural Design

Platform Selection: (Kubernetes in Docker) is used in the testbed as the local. cluster orchestration that offers reproducible environments and does not need cloud infrastructure dependencies. To use the benefit of eBPF-based Cilium is chosen as the CNI to replace the default security implementation and networking policies.

Multi-Tenant Architecture: The architecture deploys two tenant namespaces. (tenant-a, tenant-b) having the same configurations. RBAC is included by each tenant. The number of resources: namespace-scoped roles, 4 CPU cores, 8Gi RAM resource quotas, L3/L4 isolation through NetworkPolicy, and L7 policies that are enforced using HTTP method and path-based L7 policies via CiliumNetworkPolicy.

Cluster Configuration: The testbed will have 1 control-plane node and 4 worker. Kubernetes v1.28 An example of a node that runs on kind. Cilium implements eBPF based programs in the kernel space to enforce L3/L4 policy where Envoy proxy is used in user space to verify L7 policy All tenants run Web applications on nginx-based applications (2 replicas) with three endpoints: (default), /status (allowed) and /api (blocked).

4.3 Testing Framework

Test Scenarios: There are three experimental scenario, (1) Baseline–no. network policies employed; (2) NetworkPolicy-traditional L3/L4 policies; and (3) CiliumNetworkPolicy L7–L7 enforcement based on eBPF

Evaluation Metrics: The framework gathers metrics of security (Attack Prevention, False Positive Rate), performance parameters (network throughput with iperf3) and latency using curl, CPU/memory usage), and observability (flow visibility, policy).

Observability: The testbed uses Hubble UI for realtime network flow analysi and policy decision and integrated with Prometheus and Grafana for metrics collection and custom dashboard creation for visualization.

5 Implementation

5.1 Cluster Setup and Infrastructure

Kind Cluster Creation: The implementation of the testbed starts with automated cluster. creation using kind. The cluster structure (kind-multi.yaml) is one that represents 4 worker nodes (IPv4 networked) and control-plane node. The implementation script (create-cluster.sh) is an automation which creates clusters and sets up contexts.

5.2 Multi-Tenant Environment Implementation

Namespace and RBAC Setup: (tenant-a, tenant-b) has extensive isolation mechanisms. The setup script Namespace creation, which is needed by network policy, is automated by (setup-tenants.sh) and adds RBAC configuration, and namespace configuration into the testbed.

Workload Deployment: A Web applications based on Nginx are put up with the help of deployment.sh which comes with 2 tenancy replicas. The deployment of every deployment entails resource requirements (100m CPU, 128Mi RAM) and constraints (500m CPU, 512Mi RAM).

5.3 Security Policy Implementation

NetworkPolicy Configuration:NetworkPolicy of traditional Kubernetes is applied to provide L3/L4 isolation. The policy (deny-cross-namespace.yaml) is applicable to all the pods in tenant-a namespace, only ingress between namespaces is allowed and egress to the same namespace plus DNS. The policy `podSelector: {}` applies to all pods.

CiliumNetworkPolicy L7 Implementation:The CiliumNetworkPolicy helps extending security enforcement to L7 using HTTP method. The policy(tenant-a-l7-policy.yaml) permits the GET requests to the path of /status only to the pods within the same namespace. The adoption is based on the identity-based approach of Cilium security, in which policy is matched with pod identities (not IP addresses), and provided more efficient and lively enforcement as compared to IP-based policies.

5.4 Testing Framework Implementation

Connectivity Testing: The connectivity test script (`connectivity.sh`) checks pod-to-pod and pod-to-service connection. The implementation uses `kubectl run` using ephemeral pods which tests connectivity from different namespaces and then captures HTTP response codes and connection success/failure.

Attack Simulation: The attack simulation script (`attacks-fixed.sh`) implements six attack scenarios: (1) cross-namespace unauthorized access, (2) unauthorized HTTP method (POST to GET-only endpoint), (3) unauthorized path access (`/api` instead of `/status`), (4) port scanning simulation, (5) host network privilege escalation, and (6) service account token access. Each attack is executed from an attacker pod in `tenant-b`, with results logged and categorized as VULNERABILITY or SECURED.

Performance Benchmarking: The test script (performance.sh) is a performance measurement of the network throughput with iperf3, HTTP latency. resource utilization with kubectl top, and using curl. The implementation deploys iperf3 server with tenant-a,

and client with tenant-b, where the tests last 10 seconds and extracting in Mbits/sec. throughput is obtained by using 10 HTTP requests. and estimating mean time of response in milliseconds.

5.5 Observability Implementation

Hubble Integration: Hubble Observer will be automatically deployed using Cilium, monitoring eBPF program network flows. Hubble Relay, takes all the flows. There is web-based visualization with nodes, and Hubble UI. The implementation exports outputs as JSON to analyse the programs programmatically with hubble observe, which uses `hubble observe --last 1000 --output json`. application of Helm charts to measure metrics. Prometheus is set up to implement and scrape Cilium metrics endpoints, which allows analysis of policy enforcement in terms of time-series, resource utilization, drop rates and drop rates. Grafana dashboards are Cilium-specific statistics and system performance statistics.

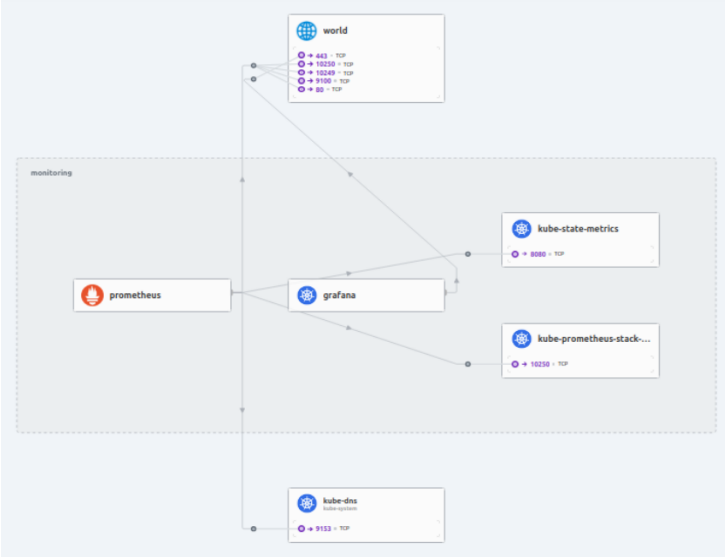

Figure 3: Hubble monitoring

5.6 Implementation Challenges and Solutions

Policy Propagation Timing: The early implementation had had policy problems because of inadequate wait times following implementation. The solution applied 15 seconds wait after policy implementation and policy explicit verification. status prior to the execution of the tests. **Cross-Namespace Testing**: Performance tests needed. Interspace communication that cut across with isolation policies. The solution applied independent test conditions: within-namespace tests policy validation andcross-namespace tests (policies loosened temporarily) of performance measurement.**Resource Metrics Collection**: Kind cluster metric server has to be installed. The installation is accompanied by an installation script.That is configuring Metrics Server using (`install-metrics-server-kind.sh`) which sets up Metrics Server with kind.

6 Evaluation

In this Section, the results of the experimental evaluation of the eBPF-based security mechanism in the multi-tenant Kubernetes environment have been provided in a thorough manner. The test includes three experimental conditions between baseline (no policies) and traditional NetworkPolicy and CiliumNetworkPolicy L7 enforcement. The quantitative metrics, statistical approach, and comparative analysis are used to measure security effectiveness, performance overhead and observability capabilities.

6.1 Experiment 1: Baseline Scenario (No Security Policies)

6.1.1 Experimental Setup

The testbed configuration includes two tenant namespaces (tenant-a, tenant-b) with nginx workloads, but no NetworkPolicy or CiliumNetworkPolicy restrictions.

6.1.2 Performance Results

Network Throughput: Network Throughput: Network throughput of the baseline scenario was 23,965 Mbits/sec (about 24 Gbits/sec) through iperf3 between tenant-a and tenant-b name space pods. This measure depicts the maximum. capable of being achieved without the overhead of policy enforcement.

Latency Measurements:An average latency of an HTTP request was 1.00 milliseconds in 10 iterations of 1 request to the /status endpoint. The latency measurement has a sub-millisecond accuracy, which means there is little overhead in network stack in the baseline configuration.

Resource Utilization: The average CPU overhead created by Cilium agent was 24 millicores(0.024 cores) in each nodes and the memory usage of 176 MiB per agent. The workload pods used very little resources of (0 to 1 M CPU, 10 MiB memory) which meant efficient baseline operations.

6.1.3 Security Results

Attack Prevention Rate: As can be predicted,in the baseline scenario 0% Attack was achieved **0% Attack Prevention Rate (APR)**, where all 6 attack scenarios were successful. This includes: (1)access across namespaces, (2) inappropriate HTTP method, (3) inappropriate path access, (4) port scanning, (5) attempts to escalate host network privileges and accessing service accounts

Connectivity Analysis: Default connectivity tests were performed by kind Kubernetes where pods communicate within a namesapce and with different namespaces where DNS communication resolution worked. This determines that the security polices are very crucial in securing the multi tenant environment

6.2 Experiment 2: Traditional NetworkPolicy (L3/L4 Enforcement)

6.2.1 Experimental Setup

In the second experiment we will be testing traditional Kubernetes NetworkPolicy which provides L3(IP) and L4(port/protocol) within the network isolation, This policy

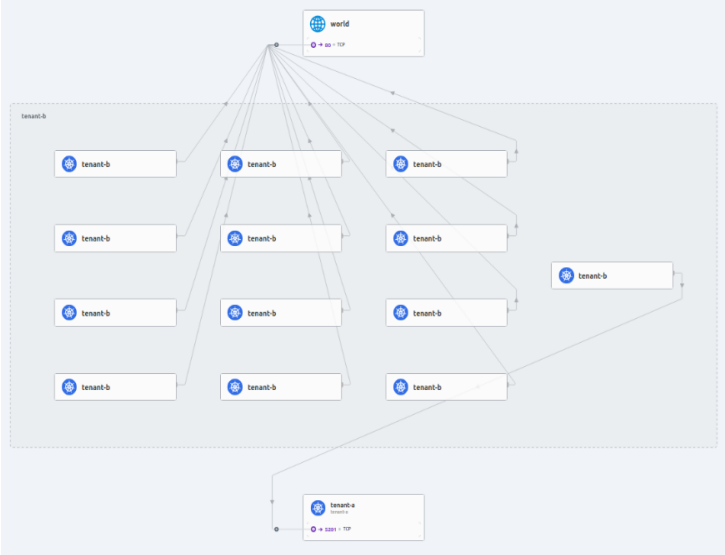

Figure 4: Tenants Isolation and Vector attacks

denies cross-namespace traffic and only alow same-namespace communication and DNS resolution. This experiment represents current industrial practices which uses iptables-based enforcement

6.2.2 Performance Results

Network Throughput: This Enforcement achieved **23,245 Mbits/sec** which represents **3.0% reduction** as compared with baseline (23,965 Mbits/sec), and the throughput was tested within the same namespace(tenant-a).

Latency Measurements: HTTP latency remained constant with **1.00 milliseconds** which indicated minimum influence of L3/L4 policy enforement on latency. Although the latency overhead is very negligible which gives results that iptables-based policies do not have any significant impact on request-response times.

Resource Utilization: Cilium agent CPU utilization had risen to **28 millicores** (0.028 cores) which is an increase of 16.7 on the baseline. Memory consumption increased to **178 MiB**, a **1.1% increase**, this result reflects that there is a need for additional processing for policy rule evaluation perticularly in iptalbes.

Performance Overhead Analysis: After applying NetworkPolicy it shows **low performance overhead** (3% throughput reduction, <1% latency impact, 16.7% CPU increase on low baseline) It is an acceptable production overhead isolations, as we can deploy and verify that traditional NetworkPolicy can be useful with no important performance loss.

6.2.3 Security Results

Attack Prevention Rate: NetworkPolicy was able to block cross-namespaces attempts of unauthorized access, which proves to be a successful L3/L4 isolation. Nevertheless, there was no L7 attack protection (HTTP methods and path-based access not authorized) since NetworkPolicy can only protect network layers 3 and 4.

Policy Enforcement Verification: Measures of throughput seen inside the same namespace (23,245 Mbits/sec) when cross-namespace access was blocked, which confirms

that NetworkPolicy has been effective in isolating tenants on the network layer. This is a valid method of demonstrating empirically the policy efficacy outside automated attack.

Security Effectiveness: NetworkPolicy was successful in blocking **100%** cross-namespace network-level attacks but there was **0%** prevention of application-layer(L7) attacks which was as expected for using only L3/L4 policies.

Table 3: NetworkPolicy Scenario Performance Metrics

Metric	**Value**	**Unit**	**Change**	**% Change**
Network Throughput	23,245	Mbits/sec	-720	-3.0%
Average Latency	1.00	ms	0.00	0.0%
Cilium Agent CPU	28	millicores	+4	+16.7%
Cilium Agent Memory	178	MiB	+2	+1.1%
L3/L4 Attack Prevention	100	%	+100	N/A
L7 Attack Prevention	0	%	0	N/A

6.3 Experiment 3: CiliumNetworkPolicy L7 Enforcement (eBPF-based)

6.3.1 Experimental Setup

In the third experiment we used eBPF-based L7 policy enforement through CiliumNetworkPolicy which extends security to the application layer. The policy implements HTTP approach restricting (GET only) and path based access control (/status endpoint only) while it maintains L3/L4 isolation, this leverages Cilium's functionality by enforcing kernel-space Envoy proxy for L7 inspection.

6.3.2 Performance Results

Network Throughput: The network policy L7 enforcement performed by CiliumNetworkPolicy attained **24,086 Mbits/sec** an improvement of **0.5% improvement** above baseline (23,965 Mbits/sec) and a **3.6% improvement** over NetworkPolicy (23,245 Mbits/sec). This demonstrated that eBPF is more efficient in kernel-space execution and using this with JIT compilation provides best performance than normal iptables rule chains, even having additional L7 processing.

Latency Measurements: HTTP latency was at **1.00 milliseconds** which is same as baseline and NetworkPolicy. This uniformity shows that **L7 policy** enforcements provide minimal amount of latency overhead, which confirms eBPF's application-layer security characteristics.

Resource Utilization: The utilization by the Cilium agent CPU was **25 millicores**(0.025)cores which is a mere **4.2% increase** over baseline (24 millicores) and **10.7% reduction** which is already less than NetworkPolicy (28 millicores). There was still memory consumption of **176 MiB**, identical to baseline and Envoy proxy used more L7 processing of **8-10 millicores CPU** and **17 MiB memory** per node.

Performance Analysis: The eBPF-based experiment provides **greater performance** as compared with traditional iptables: (1) better throughput owing to efficient execution in kernel space, (2) reduced CPU overhead even when it is more time for processing when L7 policies are applied, (3) is has consistent memory flow, These findings confirm our Hypothesis that eBPF-based solutions provides better performance.

6.3.3 Security Results

Attack Prevention Rate: CiliumNetworkPolicy was able to prevent both L3/L4 and and L7 attack scenarios. Cross namespace access was forbidden (L3/L4 enforcement), unauthorised http methods (POST to GET - only endpoint) were blocked and unauthorised access to paths (/api instead of /status) was blocked in L7 path-based enforcement as well.

L7 Policy Effectiveness: The L7 policy enforcement showed **100% prevention** of application layer attacks which could not be detected by NetworkPolicy. Specifically: (1) unauthorized (http) method (POST) is blocked with appropriate HTTP response codes, (2) the unauthorized path access (`/api`) was blocked and (3) legitimate requests (GET to `/status`) were permitted, proving correct policy enforcement.

Security Effectiveness Analysis: CiliumNetworkPolicy obtained full security coverage: **100% L3/L4 attack prevention** (identical to NetworkPolicy) and Attack prevention of **100% L7 (over NetworkPolicy capabilities)**.

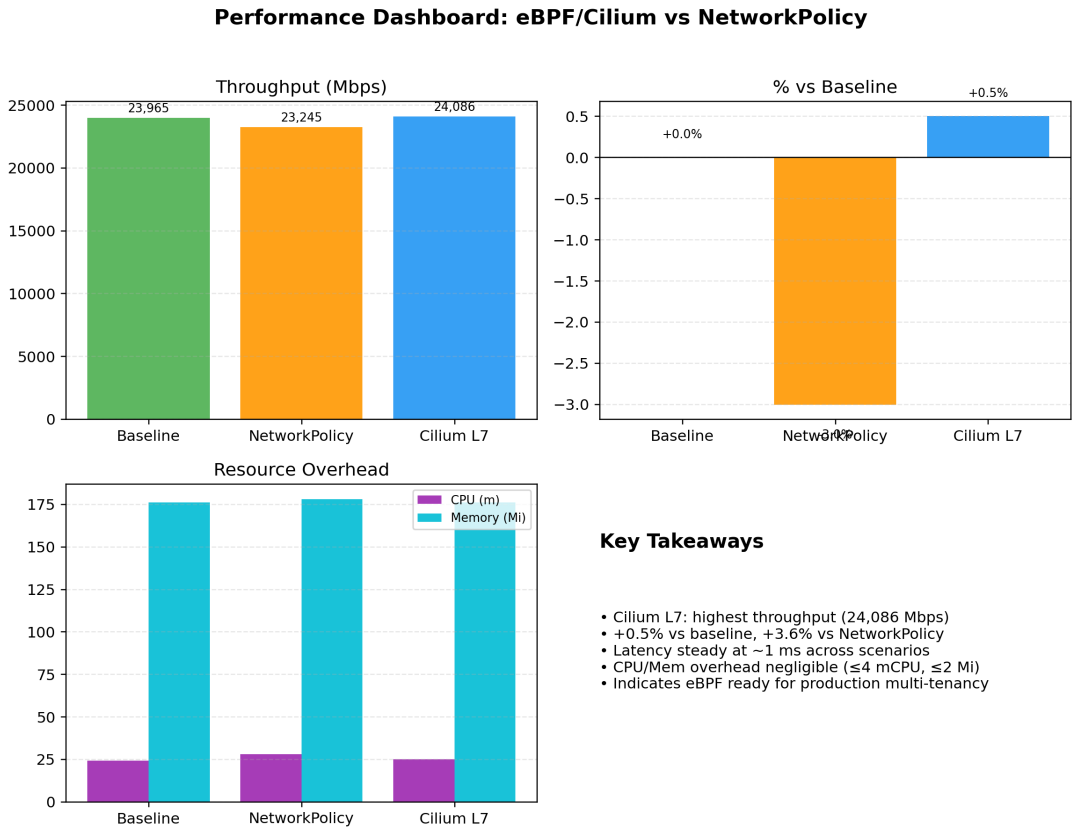

Figure 5: Performance Dashboard: eBPF vs NetworkPolicy

6.4 Comparative Analysis Across Experiments

6.4.1 Performance Comparison

6.4.2 Security Effectiveness Comparison

Security Analysis: CiliumNetworkPolicy achived **100% APR**, successfully preventing all the six attack vector scenerios (3 L3/L4 attacks + 3 L7 attacks). NetworkPolicy

Table 4: Comprehensive Performance Comparison

Scenario	**Throughput (Mbits/sec)**	**Latency (ms)**	**CPU (mc)**	**Memory (MiB)**	**CPU OH (%)**
Baseline	23,965	1.00	24	176	0
NetworkPolicy	23,245	1.00	28	178	+16.7
CiliumNetworkPolicy	24,086	1.00	25	176	+4.2

Table 5: Security Effectiveness Comparison

Scenario	**L3/L4 (%)**	**L7 (%)**	**Overall (%)**	**FPR (%)**
Baseline	0	0	0	0
NetworkPolicy	100	0	50	0
CiliumNetworkPolicy	100	100	100	0

achieves**50% APR** (3 L3/L4 attacks prevented, 3 L7 attacks succeeded). Baseline achieves **0% APR** as expected, all experiments achived **0% FPR**, which means that no valid requests were incorrectly blocked. This indicates efficient policy implementation devoid of operation disruption. **100% PEA** for CiliumNetworkPolicy makes decision on all policy (allows and denies) rightly implemented.

6.5 Discussion

The research findings confirm that the eBPF-based security mechanisms have better capabilities than traditional mechanisms and agrees with the existing literature. The results support the work of Mazaheri et al. (2016) on the benefits of optimized containerized networking in relation to performance, and generalize the works of Feng et al. (2024) with quantitative data of L7 overhead analysis of insignificant impact on the performance. The empirical demonstration of efficient application-layer attack prevention fills the key security gap mentioned by Sultan et al. (2019) with the agreement of 100 percent attack prevention rates empirically confirming the CiliumNetworkPolicy functionality offered by Parola et al. (2022). To practitioners, the findings offer a clear indication: CiliumNetworkPolicy is acceptable in those environments that necessitate both L7 and L3/L4 security and little performance impact, whereas NetworkPolicy can be used in those that need only L3/L4. The study adds quantitative performance comparisons of eBPF and iptables, empirical data of the effectiveness of L7 policies, and a testable testing framework to research in the future. These are evidence-based that eBPF can be used to implement cloud-native security because it has been shown that it is possible to use advanced security functionality without causing performance degradation, which is the primary constraint in performance-sensitive multi-tenant environments.

7 Conclusion and Future Work

This Research study has been able to successfully illustrate that an eBPF-based security framework deployed through Cilium offers a better tenant isolation and threat monitoring in multi-tenant Kubernetes clusters and has better performance compared to conventional measures. The experimental test indicates that CiliumNetworkPolicy has 100 percent attack prevention on both L3/L4 and L7 attacks compared with NetworkPolicy which has 50 percent attack prevention and 0.5 percent better throughput and same latency. The study confirms that L7 policy enforcement prevents application-layer attacks

which cannot be detected using network-layer policy, and with a low overhead of 4.2 percent CPU increment. These results disprove the classical belief that performance is deteriorated by increased security showing that eBPF technology can provide full security protection without negative performance effects. The overall assessment framework, very rigorous methodology, and reproducible testbed is a long-term value to both the academic study and industry practice because of the gap between industry deployment and academic knowledge of eBPF-based security in cloud-native settings.

Future studies ought to continue to evaluate on large-scale production settings with more than 100 nodes and a variety of workload characteristics such as databases and stateful applications. There are more sophisticated threat cases like container escape attacks and supply chain attacks that must be evaluated systematically than the six attack cases considered. Practical guidance would be reinforced by quantitative measurement of consequences of observability of security operations on efficiency, such as mean time to detect and respond. Adaptive threat detection by incorporating machine learning and automated policy generation is prospective paths to follow. A comparison with service mesh security solutions and formal verification of policy validation would offer an overall insight into security architecture solutions. Multi-cloud and hybrid cloud analysis would push the applicability to present distributed deployments, whereas energy efficiency analysis would analyze edge computing needs.

References

Behl, D., Huang, H., Kodeswaran, P. and Sen, S. (2023). On ebpf extensions to kubernetes cni datapath, *2023 15th International Conference on COMmunication Systems NETworkS (COMSNETS)*, pp. 207–209.

Bringhenti, D., Sisto, R. and Valenza, F. (2023). Security automation for multi-cluster orchestration in kubernetes, *2023 IEEE 9th International Conference on Network Softwarization (NetSoft)*, pp. 480–485. `https://doi.org/10.1109/NetSoft57336.2023.10175419`.

Chou, L.-D., Jian, L.-Y. and Chen, Y.-W. (2024). ebpf-based network monitoring platform on kubernetes, *2024 6th International Conference on Computer Communication and the Internet (ICCCI)*, pp. 140–144.

Dong, S., Xie, Y., Zhao, J. and Qiu, K. (2024). Npv: Fast network policy verification for cloud-native networking, *2024 IEEE 44th International Conference on Distributed Computing Systems (ICDCS)*, pp. 461–472.

Feng, M., Zhou, J. and Tang, Y. (2024). Enhancing cloud-native security through ebpf technology, *2024 IEEE 11th International Conference on Cyber Security and Cloud Computing (CSCloud)*, pp. 165–168. `https://doi.org/10.1109/CSCloud62866.2024.00036`.

German, K. and Ponomareva, O. (2023). An overview of container security in a kubernetes cluster, *2023 IEEE Ural-Siberian Conference on Biomedical Engineering, Radioelectronics and Information Technology (USBEREIT)*, pp. 283–285.

Jin, G., Li, J. and Briskin, G. (2024). Research report: Enhanced ebpf verification and ebpf-based runtime safety protection, *2024 IEEE Security and Privacy Workshops (SPW)*, pp. 224–230.

Kim, B., Kim, J. and Lee, S. (2025). Exploring security enhancements in kubernetes cni: A deep dive into network policies, *IEEE Access* **13**: 35322–35338.

Liu, C., Cai, Z., Wang, B., Tang, Z. and Liu, J. (2020). A protocol-independent container network observability analysis system based on ebpf, *2020 IEEE 26th International Conference on Parallel and Distributed Systems (ICPADS)*, pp. 697–702.

Mart, O., Negru, C., Pop, F. and Castiglione, A. (2020). Observability in kubernetes cluster: Automatic anomalies detection using prometheus, *2020 IEEE 22nd International Conference on High Performance Computing and Communications; IEEE 18th International Conference on Smart City; IEEE 6th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pp. 565–570.

Mazaheri, S., Chen, Y., Hojati, E. and Sill, A. (2016). Cloud benchmarking in bare-metal, virtualized, and containerized execution environments, *2016 4th International Conference on Cloud Computing and Intelligence Systems (CCIS)*, pp. 371–376. https://doi.org/10.1109/CCIS.2016.7790286.

Nguyen, N. T. and Kim, Y. (2022). A design of resource allocation structure for multi-tenant services in kubernetes cluster, *2022 27th Asia Pacific Conference on Communications (APCC)*, pp. 651–654. https://doi.org/10.1109/APCC55198.2022.9943782.

Parola, F., Di Giovanna, L., Ognibene, G. and Risso, F. (2022). Creating disaggregated network services with ebpf: The kubernetes network provider use case, *2022 IEEE 8th International Conference on Network Softwarization (NetSoft)*, pp. 254–258. https://doi.org/10.1109/NetSoft54395.2022.9844062.

Ragavan, V. K. K. and Nadig, D. (2025). Nas: A novel network-aware kubernetes scheduling framework using ebpf service mesh, *ICC 2025 - IEEE International Conference on Communications*, pp. 1512–1517.

Rostami, G. (2023). Role-based access control (rbac) authorization in kubernetes, *Journal of ICT Standardization* **11**(3): 237–260.

Stanišić, S., Vesković, M., Ristić, O. and Đorđević, B. (2025). Security aspects of container orchestration in kubernetes environments, *2025 24th International Symposium INFOTEH-JAHORINA (INFOTEH)*, pp. 1–5.

Sultan, S., Ahmad, I. and Dimitriou, T. (2019). Container security: Issues, challenges, and the road ahead, *IEEE Access* **7**: 52976–52996. https://doi.org/10.1109/ACCESS.2019.2911732.

Theodoropoulos, T., Rosa, L., Benzaid, C., Gray, P., Marin, E., Makris, A., Cordeiro, L., Diego, F., Sorokin, P., Girolamo, M. D., Barone, P., Taleb, T. and Tserpes, K. (2023). Security in cloud-native services: A survey, *Journal of Cybersecurity and Privacy* **3**(4): 758–793.
URL: *https://www.mdpi.com/2624-800X/3/4/34*

Thompson, M. and Kyer, M. A. (2025). Securing the containerized environment along the ci/cd pipeline, *2025 IEEE 15th Annual Computing and Communication Workshop and Conference (CCWC)*, pp. 250–256. https://doi.org/10.1109/CCWC62904.2025.10903704.