GCPS

ENHANCING POLICY COMPLIANCE MONITORING WITH GRAYLOG IN OPEN POLICY AGENT

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
ATILIM UNIVERSITY

BY

AHMED SALEM AHMED SHIBANI

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

JUNE 2024

Approval of the Graduate School of Natural and Applied Sciences, Atilim University.

Prof. Dr. Ender Keskinkılıç
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of **Master of Science in Computer Engineering Department, Atilim University**.

Prof. Dr. Gökhan Şengül
Head of Department

This is to certify that we have read the thesis ENHANCING POLICY COMPLIANCE MONITORING WITH GRAYLOG IN OPEN POLICY AGENT submitted by AHMED SALEM AHMED SHIBANI and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Asst. Prof. Dr. Selma Nazlıoğlu
Supervisor

Examining Committee Members:

Assoc. Prof. Dr. Gül Tokdemir
Computer Engineering, Çankaya University

Asst. Prof. Dr. Selma Nazlıoğlu
Software Engineering, Atılım University

Asst. Prof. Dr. Arda Sezen
Computer Engineering, Atılım University

Date: June 11, 2024

I declare and guarantee that all data, knowledge and information in this document has been obtained, processed and presented in accordance with academic rules and ethical conduct. Based on these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name : AHMED SALEM AHMED SHIBANI

Signature :

ABSTRACT

ENHANCING POLICY COMPLIANCE MONITORING WITH GRAYLOG IN OPEN POLICY AGENT

Shibani, Ahmed Salem Ahmed

M.S., Department of Computer Engineering

Supervisor : Asst. Prof. Dr. Selma Nazlıoğlu

June 2024, 155 pages

Ensuring strict adherence to security policies is of importance as modern IT environments become increasingly complex. Traditional monitoring methods often fall short in providing the necessary visibility and real-time insights required to manage dynamic and distributed systems. This research addresses these challenges by leveraging Graylog's robust log management capabilities to monitor and analyze OPA decision logs, thereby improving the detection and management of policy violations.

The primary aim of this study is to develop a monitoring solution that enhances the visibility and management of policy enforcement activities. The research identifies the critical data elements from OPA decision logs necessary for detecting policy violations and develops an OPA Decision Log Parser and Data Extractor. Through the integration of Graylog with OPA, the study designs and implements intuitive dashboards and alerting mechanisms, providing real-time insights into policy compliance and enabling proactive responses to security issues. Additionally, the thesis contributes a shareable content pack to the Graylog and OPA communities, facilitating broader adoption and implementation of effective policy monitoring solutions. The effectiveness of the proposed solution is validated through three use cases. Each application demonstrates the system's capability to detect policy violations accurately

and efficiently, confirming the utility of the integration in diverse operational environments. Key findings indicate that the integration significantly enhances real-time policy compliance monitoring, improves detection and alerting of policy violations, and provides actionable insights through customizable dashboards. Future research directions include expanding application use cases to other container orchestration platforms, incorporating advanced analytics and machine learning techniques, and developing detailed user guidance for configuring and utilizing OPA's logging capabilities. This work demonstrates that the integration of Graylog with OPA offers a powerful solution for managing policy compliance in modern IT environments, ultimately strengthening organizational security and compliance frameworks.

Keywords: policy violation, security, policy as code, violation detection, logging

ÖZ

OPEN POLICY AGENT İLKE UYUMUNUN İZLEMESININ GRAYLOG KULLANILARAK İYILEŞTIRILMESI

Shibani, Ahmed Salem Ahmed

Yüksek Lisans, Bilgisayar Mühendisliği

Tez Yöneticisi : Dr. Öğr. Üyesi Selma Nazlıoğlu

Haziran 2024, 155 sayfa

Modern BT ortamlarının giderek daha karmaşık hale gelmesiyle güvenlik politikalarına sıkı sıkıya uyulmasının sağlanması önem kazanmaktadır. Geleneksel izleme yöntemleri, dinamik ve dağıtık sistemleri yönetmek için gerekli görünürlük ve gerçek zamanlı içgörüler sağlama konusunda genellikle yetersiz kalmaktadır. Bu araştırma, Graylog'un güçlü günlük yönetim yeteneklerinden yararlanarak OPA karar günlüklerini izleyip analiz ederek bu zorluklara çözüm bulmayı amaçlamaktadır. Bu sayede, politika ihlallerinin tespiti ve yönetimi iyileştirilmektedir.

Bu çalışmanın temel amacı, politika uygulama faaliyetlerinin görünürlüğünü ve yönetimini artıran bir izleme çözümü geliştirmektir. Bu araştırma ile politika ihlallerini tespit etmek için gerekli olan OPA karar günlüklerinin içerisindeki kritik veri unsurları çıkarılmış ve bir OPA Karar Günlüğü Ayrıştırıcı ve Veri Çıkarıcı geliştirilmiştir. Graylog'un OPA ile entegrasyonu sayesinde,, politika uyumuna ilişkin gerçek zamanlı içgörüler sağlayan ve güvenlik sorunlarına proaktif yanıtlar verilmesini mümkün kılan panolar ve uyarı mekanizmaları tasarlanmış ve uygulanmıştır. Ayrıca, politika izleme çözümlerinin Graylog ve OPA toplulukları tarafından daha geniş çapta benimsenmesini ve etkili bir şekilde kullanılmasını kolaylaştıracak paylaşımlı bir içerik paketi ortaya çıkarılmıştır. Önerilen çözümün etkinliği, üç kullanım durumu aracılığıyla

doğrulanmıştır. Her uygulama, sistemin politika ihlallerini doğru ve verimli bir şekilde tespit etme yeteneğini göstererek, entegrasyonun çeşitli operasyonel ortamlardaki faydasını doğrulamaktadır. Entegrasyonun, gerçek zamanlı politika uyumunun izlemesini önemli ölçüde artırdığı, politika ihlallerinin tespiti ve uyarılmasını iyileştirdiği ve özelleştirilebilir panolar aracılığıyla uygulanabilir içgörüler sağladığı tespit edilmiştir. Gelecek araştırma konuları arasında kullanım durumlarının diğer konteyner düzenleme platformlarına uygulaması ile genişletilmesi, ileri analiz ve makine öğrenimi tekniklerinin dahil edilmesi ve OPA'nın günlük kaydı yeteneklerinin yapılandırılması ve kullanılması için detaylı kullanıcı kılavuzlarının geliştirilmesi yer almaktadır. Bu çalışma, Graylog'un OPA ile entegrasyonunun, modern BT ortamlarında politika uyumunu yönetmek için güçlü bir çözüm sunduğunu ve nihayetinde kurumsal güvenlik ve uyum çerçevelerini güçlendirdiğini göstermektedir.

Anahtar Kelimeler: ilke ihlali, güvenlik, kod tabanlı ilke, ihlal tespiti, günlük kaydı

This work is dedicated to the loving memory of my father, Dr. Salem Shibani. His unwavering support, boundless wisdom, and enduring love have been the guiding light in my life. Though he is no longer with us, his spirit continues to inspire and motivate me every day. I owe my achievements to his belief in me and his relentless pursuit of knowledge and excellence.

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my advisor, Dr. Selma Nazlıoğlu, for her continuous support, guidance, and invaluable insights throughout the course of this research. Her encouragement and expertise have been instrumental in the completion of this thesis.

I am profoundly thankful to my wife and kids for their unwavering love, patience, and understanding. Their support and sacrifices have been my source of strength and motivation during this challenging journey.

My heartfelt thanks go to my mom, whose endless love and prayers have always been my pillar of strength. Her unwavering belief in me has been a constant source of inspiration.

I would also like to extend my sincere appreciation to the Libyan Ministry of Higher Education and Scientific Research and the Libyan Academic Attaché in Ankara for their financial support and encouragement, which have been vital in enabling me to pursue my academic goals.

Thank you all for your contributions and support.

TABLE OF CONTENTS

ABSTRACT iii

ÖZ v

DEDICATION vii

ACKNOWLEDGMENTS viii

TABLE OF CONTENTS ix

LIST OF TABLES xv

LIST OF FIGURES xvi

LIST OF ABBREVIATIONS xix

CHAPTER

1 INTRODUCTION 1

1.1 Problem Statement 2

1.2 Aim and Purpose 3

1.3 Contribution 4

1.4 Research Questions 5

1.5 Threats to Validity 6

1.6 Outline of Thesis 7

2 BACKGROUND 8

2.1 Security Goals and Mechanisms 8

2.2 Access Control Policies 9

2.2.1 Access Control Models 9

2.2.2 Mandatory Access Control (MAC) 10

2.2.3 Discretionary Access Control (DAC) 11

2.2.4 Role-Based Access Control (RBAC) 11
2.2.5 Attribute-Based Access Control (ABAC) 12
2.2.6 Other Access Control Models 13
2.3 Open Policy Agent 14
2.4 Graylog 15
2.5 Docker 16
2.5.1 Docker Swarm Mode 16
3 RELATED WORK 19
3.1 Policy Compliance and Auditing 19
3.2 Open Policy Agent (OPA) Applications 21
3.3 Log Management and Analysis Tools 23
3.4 Gap Analysis 24
3.5 Summary 25
4 METHODOLOGY 26
4.1 Research Methodology Outline 26
4.2 Analysis 28
4.2.1 OPA Architecture 29
4.2.1.1 The Rego Language 29
4.2.2 OPA Use Cases 31
4.2.3 OPA Decision Logs 32
4.2.4 Graylog Architecture 35
4.2.4.1 Streams 36
4.2.4.2 Inputs 36
4.2.4.3 Pipelines 36
4.2.4.4 Dashboards 38
4.2.4.5 Event Definitions and Alerts 39
4.2.4.6 Content Packs 41
4.3 Design 42
4.3.1 Environment Architecture Design 42
4.3.1.1 Docker Swarm Cluster 42

4.3.1.2 Infrastructure Services 43
4.3.1.3 Authorization Service 45
4.3.1.4 Identification of Minimum Resource Requirements 46
4.3.1.5 Identification of Secure Versions for Primary Components 47
4.3.2 Identification of Essential Data to Detect Violations 49
4.3.3 Use Case Applications Design 49
4.3.3.1 App1 - HTTP-API 49
4.3.3.2 App2 - SSH and Sudo Authorization . 50
4.3.3.3 App3 - Event-Driven MSA 50
4.3.4 Validation Design 51
4.3.4.1 Crafting Validation Applications . . . 51
4.3.4.2 Methodology for Validation 52
4.4 Summary 53
5 IMPLEMENTATION AND DEPLOYMENT 54
5.1 Environment Setup Implementation 55
5.1.1 Setting up Docker Swarm Cluster 55
5.1.2 Integration of Traefik 58
5.1.3 Deploying Portainer for Cluster Management . . . 58
5.1.4 Installing and Configuring the OPA Stack 59
5.1.5 Graylog Stack Implementation 60
5.2 Implementation of Use Cases Applications 61
5.2.1 HTTP API Application 62
5.2.2 SSH and Sudo Authorization Application 65
5.2.3 Event-driven Microservice Architecture Overview . 70
5.2.4 Deployment of Applications 73
5.3 Validation Setup 74
5.3.1 Crafting Validation Applications 74
5.3.1.1 HTTP-API Traffic Generator 74

5.3.1.2 SSH-Sudo Traffic Generator 76
5.3.1.3 Event-driven MSA Traffic Generator . 77
5.3.2 Deployment of Validation Applications 78
5.4 Graylog Configuration 79
5.4.1 Configuring Graylog Inputs 80
5.4.2 Configuring Graylog Streams 81
5.4.2.1 Setting up the Streams 82
5.4.3 Creating OPA Pipelines and Rules 82
5.4.3.1 Pipeline 1 - OPA Normalization 83
5.4.3.2 Pipeline 2 - OPA General Routing . . 85
5.4.3.3 Pipeline 3 - OPA Application Routing 87
5.5 Summary 89
6 DEMONSTRATION 91
6.1 Designing and Creating Dashboards 91
6.1.1 Overall OPA Decisions Dashboard 91
6.1.2 General Application Policy Metrics 93
6.1.3 Application-Specific Metrics Tab 96
6.2 Configuring Event Definitions for OPA 98
6.2.1 HTTP API Policy Violation Event 99
6.2.2 SSH Policy Violation Event 99
6.2.3 Sudo Policy Violation Event 100
6.2.4 High Latency in Policy Decisions Event 100
6.2.5 Excessive Latency in Policy Decisions Event . . . 101
6.3 Summary 101
7 EVALUATION AND RESULTS 106
7.1 Methodology for Collecting Results 106
7.1.1 Baseline Dashboard Observation 107
7.2 Results for HTTP-API Application 109
7.2.1 Running the Traffic Generator in Attacker Mode . . 109
7.2.2 Observing the Dashboards 109

7.2.2.1 Monitoring Alerts Channel 109
7.2.2.2 Data Analysis 111
7.3 Results for SSH-Sudo Application 112
7.3.1 Running the Traffic Generator in Attacker Mode . . 112
7.3.2 Observing the Dashboards 112
7.3.3 Monitoring Alerts Channel 113
7.4 Results for Event-Driven Microservice Architecture Application 116
7.4.1 Running the Traffic Generator for Policy Violations 116
7.4.2 Observing the Dashboards 116
7.4.3 Monitoring Alerts Channel 116
7.5 Answers to Research Questions 119
7.6 Summary 123
8 CONCLUSION 125
8.1 Discussion and Findings 125
8.2 Contributions to the Field 126
8.3 Future Work 126
8.4 Final Thoughts 127
REFERENCES 133
A FULL DOCKER COMPOSE CONFIGURATION 134
A.1 Traefik Docker Compose File 134
A.2 Portainer CE Docker Compose File 136
A.3 OPA Docker Compose File 137
A.4 Graylog Docker Compose File 138
A.5 HTTP-API Docker Compose File 141
A.5.1 HTTP-API Attacker Docker Compose File 142
A.6 SSH-Sudo Docker Compose File 143
A.6.1 SSH Attacker Docker Compose File 145
A.6.2 Sudo Attacker Docker Compose File 145
A.7 Event-Driven MSA Docker Compose File 146

A.7.1 Kafka Docker Compose File 150
A.7.2 MSA Attacker Docker Compose File 154

LIST OF TABLES

Table 4.1 Distribution of Respondents by Open Policy Agent (OPA) Use Cases 32

Table 4.2 Minimum Resource Requirements Based on Community Feedback . 46

Table 4.3 Summary of Secure Versions for Primary Components 48

Table 4.4 Essential Data from OPA Decision Log 49

Table 5.1 Key Configurations in OPA Docker Compose File 60

Table 5.2 HTTP API Traffic Generator Cases 65

Table 5.3 Configured hosts and Created Users for SSH and Sudo Application . 67

Table 5.4 HTTP API Traffic Generator Cases 70

Table 5.5 HTTP API Traffic Generator Attacker-Mode Cases 76

Table 5.6 HTTP API Traffic Generator Attacker-Mode Cases 77

Table 5.7 Utilized Graylog Streams 81

Table 6.1 HTTP API Policy Violation Event Definition 104

Table 6.2 SSH Policy Violation Event Definition 104

Table 6.3 Sudo Policy Violation Event Definition 104

Table 6.4 High Latency in Policy Decisions Event Definition 105

Table 6.5 Excessive Latency in Policy Decisions Event Definition 105

LIST OF FIGURES

Figure 4.1 Research Methodology 27
Figure 4.2 Workflow of OPA [1] 29
Figure 4.3 Graylog Architecture 35
Figure 4.4 Pipeline Stages and Rules Execution Order 38
Figure 4.5 Environment Architecture Design 43
Figure 4.6 Validation Methodology 52

Figure 5.1 Environment Setup Steps 55
Figure 5.2 Docker Swarm Architecture 57
Figure 5.3 HTTP API Application Architecture 62
Figure 5.4 SSH and Sudo Authorization Application Architecture 66
Figure 5.5 Event-Driven Microservice Application Architecture 71
Figure 5.6 Graylog Input Configuration 80
Figure 5.7 Creating a Graylog Stream 83
Figure 5.8 Graylog OPA Pipeline Workflow 84

Figure 6.1 Total Decisions Count (left-hand-side) and Total Policy Violations Count (right-hand-side) 92
Figure 6.2 Real-Time Alerts 93
Figure 6.3 Decision Logs Over Time 94
Figure 6.4 Top Applications by Policy Violations Count 95
Figure 6.5 Violation Trends Over Time 95
Figure 6.6 Policy Violations by Client IP 96

Figure 6.7 Decision Latency 97
Figure 6.8 Overall OPA Decision Logs Dashboard 98
Figure 6.9 Total Policy Decisions (left-hand-side) and Total Policy Violations (right-hand-side) 98
Figure 6.10 Policy Queries Over Time 99
Figure 6.11 Decision Latency 99
Figure 6.12 Client Activity and Overall Policy Compliance 100
Figure 6.13 Policy Result by Client 100
Figure 6.14 HTTP API Application General Policy Metrics 101
Figure 6.15 Policy Requests by User and Policy Results by User 101
Figure 6.16 Policy Requests by Method and Policy Results by Method 102
Figure 6.17 Policy Requests by Path & Policy Results by Path 102
Figure 6.18 HTTP API Application - Application Specific Metrics 103

Figure 7.1 OPA Decision Logs Dashboard during normal operations, showing policy-compliant traffic. 108
Figure 7.2 OPA Decision Logs Dashboard showing policy violations 110
Figure 7.3 HTTP-API Application Dashboard - General Policy Metrics with metrics on policy violations 110
Figure 7.4 HTTP-API Application Dashboard - Application Specific Metrics with metrics on policy violations 111
Figure 7.5 Slack alert notification for a detected policy violation in HTTP-API Application 112
Figure 7.6 OPA Decision Logs Dashboard showing policy violations for SSH and Sudo 113
Figure 7.7 SSH-Sudo Application Dashboard - General Policy Metrics with metrics on policy violations 114
Figure 7.8 SSH-Sudo Application Dashboard - Application Specific Metrics with metrics on policy violations 114
Figure 7.9 Slack alert notification for a detected SSH policy violation 115

Figure 7.10 Slack alert notification for a detected Sudo policy violation 115
Figure 7.11 OPA Decision Logs Dashboard showing policy violations for the Event-Driven Microservice Architecture 117
Figure 7.12 Event-Driven Microservice Application Dashboard - General Policy Metrics with metrics on policy violations 118
Figure 7.13 Event-Driven Microservice Application Dashboard - Application Specific Metrics with metrics on policy violations 118
Figure 7.14 Slack alert notification for a detected policy violation 119

LIST OF ABBREVIATIONS

ABAC	:	Attribute-Based Access Control
ACL	:	Access Control List
API	:	Application Programming Interface
CE	:	Community Edition
CNCF	:	Cloud Native Computing Foundation
CPU	:	Central Processing Unit
CQRS	:	Command Query Responsibility Segregation
CVE	:	Common Vulnerabilities and Exposures
DAC	:	Discretionary Access Control
ELK	:	Elasticsearch, Logstash, Kibana
FQDN	:	Fully Qualified Domain Name
GDPR	:	General Data Protection Regulation
HIPAA	:	Health Insurance Portability and Accountability Act
HR	:	Human Resources
HTTP	:	Hypertext Transfer Protocol
IDS	:	Intrusion Detection System
IP	:	Internet Protocol
JSON	:	JavaScript Object Notation
JWT	:	JSON Web Token

LBAC	:	Lattice-Based Access Control
MAC	:	Mandatory Access Control
MSA	:	Microservice Architecture
NIST	:	National Institute of Standards and Technology
OPA	:	Open Policy Agent
PAM	:	Pluggable Authentication Module
PCI-DSS	:	Payment Card Industry Data Security Standard
POSIX	:	Portable Operating System Interface
RADOS	:	Reliable Autonomic Distributed Object Store
RAM	:	Random Access Memory
RBAC	:	Role-Based Access Control
REST	:	Representational State Transfer
RFC	:	Request for Comments
RGW	:	RADOS Gateway
SIEM	:	Security Information and Event Management
SOAR	:	Security Orchestration, Automation, and Response
SSH	:	Secure Shell
SSL	:	Secure Sockets Layer
TCP	:	Transmission Control Protocol
TLS	:	Transport Layer Security
UDP	:	User Datagram Protocol
URI	:	Uniform Resource Identifier
URL	:	Uniform Resource Locator

CHAPTER 1

INTRODUCTION

The evolution of technology and the proliferation of cloud-based services have accelerated the need for robust authorization systems that can manage complex and dynamic environments effectively. This transformation has given rise to Policy-as-Code [2], an innovative approach that embeds governance and security directly into the IT infrastructure through code. Central to this paradigm shift, Open Policy Agent (OPA) [1] is introduced as an open-source, general-purpose policy engine that decouples policy decision-making from policy enforcement. OPA has emerged as a pivotal tool in defining and enforcing policies across diverse cloud-native environments, enabling organizations to seamlessly automate and streamline their security protocols.

As enterprises increasingly adopt microservices and distributed systems [3, 4], ensuring strict policy adherence becomes paramount. Violations of policy can lead to severe security breaches [5], non-compliance penalties, and potential data loss, making the monitoring of policy compliance not just beneficial but essential. Effective policy compliance monitoring safeguards against these risks by continuously verifying and validating that the operational activities align with the defined policies. The benefits extend beyond security to encompass compliance, auditing, and adherence to standards, proving critical for organizations needing to demonstrate regulatory compliance and maintain operational integrity [6].

In contrast to traditional monitoring methods, log analysis presents a promising avenue for enhancing policy compliance monitoring [7, 8, 9]. By leveraging log data generated by OPA's decision-making processes, organizations can gain valuable insights into policy evaluations, enforcement decisions, and potential compliance violations. Through advanced log parsing techniques and analytics, organizations can

uncover patterns, trends, and anomalies that may signify policy-related issues. This proactive approach to monitoring enables organizations to identify and mitigate security and compliance risks promptly, thereby strengthening their overall governance framework.

1.1 Problem Statement

The complexity of modern IT environments, particularly in containerized and distributed architectures, exacerbates the challenges associated with policy compliance monitoring. Ephemeral infrastructure, microservices, and orchestration platforms introduce additional layers of abstraction and dynamism, making it increasingly difficult for organizations to gain visibility into policy enforcement activities. As a result, organizations face heightened risks of security breaches, compliance violations, and operational disruptions due to undetected policy misconfigurations or anomalies.

OPA (Open Policy Agent) exposes a set of APIs that enable unified, logically centralized policy management [10]. However, despite its robust framework, OPA lacks a built-in control plane service for collecting crucial telemetry data, such as decision logs [11]. This absence poses a challenge for organizations seeking comprehensive insights into policy evaluations, enforcement decisions, and compliance status across their infrastructure. Furthermore, a recent survey highlighted on the OPA blog reveals that a substantial portion of OPA users—36% do not track OPA decisions, and 39% do not monitor their OPA status, despite the availability of management APIs designed for these purposes [12]. This finding underscores a critical gap in the current utilization of OPA's capabilities, suggesting a need for improved user guidance on configuring monitoring and logging.

Addressing this problem will have practical benefits for organizations and users of the applications, including enhanced security, compliance, and operational stability. It will also contribute to future research in policy compliance monitoring by providing a framework for effective decision logs data collection and analysis.

To effectively monitor policy violations in OPA, several key needs must be addressed:

- **Mechanism for Collecting OPA Decision Logs**: There must be a reliable and timely method to collect decision logs to ensure real-time visibility into policy enforcement activities.
- **Integration with Logging and Monitoring Tools**: Seamless integration with tools like Graylog to visualize and analyze decision logs.
- **Automated Alerting Systems**: Implementation of automated systems to alert administrators about policy violations in real-time, ensuring prompt response and mitigation.
- **User Guidance on Monitoring and Logging**: Clear documentation and best practices to help users configure and utilize OPA's logging and monitoring capabilities effectively.

1.2 Aim and Purpose

The primary aim of this thesis is to leverage Graylog [13], a robust log management and analysis platform, to enhance OPA policy monitoring capabilities. By integrating Graylog with OPA's decision logs, organizations can establish a streamlined pipeline for collecting, parsing, and analyzing policy-related telemetry data. This integration provides real-time visibility into policy enforcement activities, allowing stakeholders to proactively identify and address compliance issues, thereby mitigating risks associated with policy misconfigurations and security breaches.

Graylog's advanced search capabilities, customizable dashboards, and alerting mechanisms enable the construction of tailored monitoring solutions to meet specific policy compliance requirements. This flexibility allows organizations to scale their policy monitoring efforts dynamically, accommodating evolving IT environments. Graylog's extensibility ensures seamless integration with existing infrastructure components, consolidating policy-related telemetry data from diverse sources, simplifying operational workflows, and enhancing the overall security posture.

The thesis investigates the potential of utilizing Graylog's log monitoring capabilities to extract valuable insights from OPA decision logs. By tapping into this information,

the aim is to enhance visibility into policy evaluations, enforcement decisions, and potential compliance violations. Through systematic log aggregation, parsing, and analysis, organizations can proactively address policy-related issues.

Central to this effort is the design and implementation of intuitive dashboards offering real-time visibility into policy invocations and their outcomes. These dashboards enable stakeholders to monitor policy enforcement trends, detect anomalies, and track compliance metrics, facilitating informed decision-making and rapid responses to security and compliance challenges.

Recognizing the unique challenges posed by containerized environments, the thesis demonstrates the value of centralized log aggregation and analysis through the integration of Graylog with container orchestration platforms like Docker Swarm. By consolidating logs from disparate sources and applying advanced analytics, organizations gain deeper insights into their policy enforcement landscape, strengthening their security posture and regulatory compliance efforts.

Finally, this thesis aims to contribute the created dashboards, data extractors, and alert rules—collectively known as a ”content pack”—to the Graylog and OPA communities. This contribution provides organizations with a starting point for integrating OPA with Graylog, facilitating the adoption and implementation of effective policy monitoring solutions across a broader spectrum of organizations.

1.3 Contribution

Key contributions of this study are highlighted below:

- The essential data from OPA decision logs necessary for detecting policy violations have been identified. Along with that, OPA Decision Log Parser and Data Extractor are developed.
- A solution for integrating OPA with Graylog is developed, including designing and implementing intuitive dashboards and alerting mechanisms
- Built-in alerts are defined and generated as a result of policy violations.

- A shareable and downloadable ”Content Pack” in GrayLog is developed and released.

1.4 Research Questions

In this thesis, we delve into specific research questions that aim to further understand and optimize the integration of log monitoring systems with policy enforcement tools. These questions are designed to explore the effective deployment of Graylog alongside Open Policy Agent (OPA) to enhance policy compliance monitoring across various IT environments. Our inquiry focuses on the practical application of log analysis to improve security and compliance outcomes, the potential of real-time data visualization for policy management, and the overall impact of these technologies on the operational dynamics of modern, containerized infrastructures. Through this investigation, we seek to determine the efficacy of Graylog in extracting actionable insights from OPA decision logs, the capability of constructed dashboards to accurately reflect policy invocation activities and the advantages of centralized log aggregation in maintaining robust security and compliance standards. These questions form the cornerstone of our research, guiding our exploration into innovative methods for fortifying policy compliance through advanced log monitoring techniques:

1. **How can the integration of Graylog with Open Policy Agent (OPA) enhance real-time policy compliance monitoring?** This question aims to explore the effectiveness of combining Graylog's log management capabilities with OPA's policy enforcement to improve real-time monitoring and response to policy violations.

2. **How effective are the policy decision log trails generated by Open Policy Agent in providing actionable insights on policy compliance?** This question aims to assess the effectiveness and completeness of the information included in OPA decision logs.

3. **In what ways can customizable dashboards in Graylog provide actionable insights into policy compliance trends and anomalies in different OPA use cases?** This question investigates the role of visual dashboards in Graylog for tracking and understanding policy compliance metrics, helping stakeholders make informed deci-

sions based on real-time data.

4. **What specific features of Graylog contribute most to enhancing the visibility and management of policy compliance in containerized environments?** This question explores which aspects of Graylog (e.g., dashboard customization, log filtering, alerting mechanisms) are most beneficial for managing policies in complex, dynamic systems like containerized architectures.

5. **What are the main challenges and limitations faced during the deployment of Graylog and OPA integration in a Docker Swarm environment, and how can these be mitigated?** This question seeks to identify and address the practical difficulties encountered during the implementation of the integrated system in a container orchestration setup, focusing on solutions to enhance performance and reliability.

1.5 Threats to Validity

In the course of this thesis, certain assumptions and limitations have shaped the scope and depth of our investigation into enhancing policy compliance monitoring with Graylog in Open Policy Agent (OPA). It is essential to delineate these parameters to understand the context and the boundaries within which the research findings apply.

Firstly, while Open Policy Agent is versatile with numerous potential applications across different environments, this research has focused selectively on three specific use cases: managing authorization for HTTP APIs, SSH and Sudo authentication, and event-driven microservices using Kafka. These scenarios were chosen due to their prevalence and significance in typical enterprise environments, providing a relevant and impactful foundation for analyzing the integration of policy enforcement with log monitoring.

Additionally, the deployment environments for our testing and implementation have been limited to Docker Swarm. Although Kubernetes is a popular choice for orchestrating containerized applications, it was not utilized in this study. This decision was influenced by the specific requirements and existing infrastructure of the research setting. It is important to acknowledge that the results and conclusions drawn from

Docker Swarm environments may not be wholly transferable to Kubernetes or other orchestration platforms without additional consideration and testing.

Furthermore, the research assumes that the log data generated and collected is sufficiently verbose and accurate, which may not always be the case in different operational settings. Issues such as incomplete or incorrect logging can significantly impact the effectiveness of Graylog in monitoring and analyzing policy compliance.

Finally, it is worth noting that the integration complexity between Graylog and Open Policy Agent may vary significantly with different system architectures and configurations. The findings of this thesis may not encapsulate the full range of challenges or opportunities that might arise in other architectures or more complex integration scenarios.

These assumptions and limitations underscore the necessity for further research and testing across various environments and configurations to generalize the findings more broadly and to deepen the understanding of policy compliance monitoring in diverse operational contexts.

1.6 Outline of Thesis

Chapter 2 delves into security goals, various access control policies and models, the Open Policy Agent (OPA) and its declarative language Rego, the Graylog log management platform, and Docker along with Docker Swarm for containerization. Chapter 3 reviews existing work on the integration and effectiveness of policy compliance monitoring systems, evaluates the role and sufficiency of log management tools, and discusses the process of policy compliance and auditing. Chapter 4 provides a detailed, step-by-step description of the activities conducted to enhance policy compliance monitoring using Graylog with OPA, covering the analysis and design aspects. Chapter 5 focuses on the implementation and deployment details of the framework. In Chapter 6, three impactful use cases for OPA are demonstrated. Chapter 7 presents the evaluation and results of the study, including answers to the research questions. Finally, Chapter 8 discusses this study's extensions and suggests future work directions.

CHAPTER 2

BACKGROUND

Policy compliance monitoring is a crucial component in any organization's governance and security framework, particularly as systems become more complex and dispersed across various environments. Ensuring that all operational activities conform to established policies mitigates risks associated with data breaches and unauthorized access and ensures adherence to regulatory requirements and internal standards. Effective policy compliance monitoring acts as both a deterrent and a mechanism for early detection of deviations or malicious activities, thereby safeguarding organizational assets and maintaining trust. The subsequent sections describe concepts that are relevant to this thesis work, providing a better understanding of the methods adopted.

2.1 Security Goals and Mechanisms

The most common definitions of computer security emphasize three main goals: *confidentiality* which concerns preventing unauthorized information disclosure; *integrity* which addresses unauthorized information modification; and *availability* which involves preventing unauthorized denial of access to information. To achieve these goals, four interrelated mechanisms are utilized: *(i) Identification* focuses on determining the source of any external input to the system. *(ii) Authentication* makes sure that an actor is actually who or what it purports to be. *(iii) Authorization* ensures that an authenticated actor has the right to access and modify either data or services. Authorization is achieved by an access control mechanism within a system, which comprises three fundamental elements: the subjects, the targets, and the rules that de-

lineate how the subjects may access the targets. The overarching guidelines governing how access should be controlled are generally referred to as the access control policy [14]. *(iv) Auditing* keeps a record of user and system actions and their effects helping trace the actions of, and to identify, an attacker.

2.2 Access Control Policies

Access control focuses on limiting the activities of legitimate users who have already been successfully authenticated [15]. Access Control Policies are guidelines and rules that define how access to system resources is granted or denied [16]. These policies act as a framework for ensuring that only authorized users or systems can access certain information or perform specific actions, based on predefined conditions [14]. Access Control Policies play a critical role in the security infrastructure of an organization, helping to protect sensitive data from unauthorized access, modification, or destruction. Research in access control has developed various models that formally represent security policies and verify properties of access control systems.

2.2.1 Access Control Models

Access control models play a crucial role in the security frameworks of information systems, traditionally categorized into two primary types: discretionary access control (DAC) and mandatory access control (MAC) [17]. DAC allows the resource owner to decide who can access their resources, providing flexibility and personalized security management. This model is most common in environments where individual users control their own data. In contrast, MAC is more rigid, wherein access decisions are dictated by a central authority based on predefined security classifications of both users and data. This model is typically employed in highly secure environments where the need for confidentiality and control overrides flexibility, such as in military or government settings.

As an alternative to DAC and MAC, Role-Based Access Control (RBAC) [18] has emerged as a widely adopted model that organizes access controls around roles within an organization, rather than individual user identities or the rigid classifications used

in MAC [19]. In RBAC, permissions are associated with roles, and users are assigned to these roles, thereby simplifying the administration of permissions as users take on different responsibilities. Another sophisticated model, Attribute-Based Access Control (ABAC) [20], extends this flexibility further by defining access permissions based on a wide range of attributes, including user attributes, resource attributes, and environmental conditions. ABAC provides a more dynamic and context-sensitive approach to access control, allowing for highly granular and adaptable security policies that can respond in real-time to varying operational contexts. Both RBAC and ABAC offer robust frameworks that adapt well to the complexities of modern enterprise environments, where user roles and access needs can frequently change.

2.2.2 Mandatory Access Control (MAC)

Mandatory Access Control (MAC) is a stringent security model that restricts the rights to access information based on the levels of security classification associated with information and users. Originating from government and military contexts [14], the MAC model is designed to prevent unauthorized users from accessing sensitive or confidential information, regardless of their intentions or other permissions they might have. Its history is deeply rooted in protecting national security interests, where it was imperative to control information flow strictly based on clearance levels and need-to-know bases. Over time, the principles of MAC have been adapted for use in various high-security environments, such as in intelligence agencies and research institutions dealing with sensitive data.

The operational mechanism of MAC revolves around labels and clearances. Every piece of information and system resource is assigned a classification label, such as ”confidential,” ”secret,” or ”top secret.” Similarly, users are given clearances that define the highest classification of data they can access. A central authority, typically a security administrator, sets these classifications and the policies governing them. Access decisions are then made automatically by the system based on these predefined rules, without any discretion left to individual users or resource owners. This ensures a highly controlled environment where access to information is strictly managed according to the security policy, providing a high level of security but at the cost of

flexibility and user autonomy.

2.2.3 Discretionary Access Control (DAC)

Discretionary Access Control (DAC) is a type of access control system where the control over access to system resources is left to the discretion of the owners of those resources or to the individuals specified by those owners. Originating in less rigidly secured environments compared to those that use Mandatory Access Control (MAC), DAC allows users to make their own decisions about who should have access to the resources they control. This approach is grounded in the principle of individual ownership and responsibility, making it a common choice in collaborative environments where flexibility and user autonomy are prioritized. Historically, DAC has been the preferred model in commercial operating systems, as it provides a simple and intuitive framework for access management.

In contrast to MAC, where access decisions are based on centralized policies and predefined security attributes assigned to all entities and resources, DAC operates on the premise that resource owners have the knowledge and authority to manage access as they see fit. For example, in a DAC model, a document owner might grant read access to specific colleagues directly, using an access control list (ACL) that specifies who can or cannot access that document. This contrasts with MAC, where access to the document would be controlled based on security levels or classifications managed by a central authority. While DAC offers greater flexibility and easier administration in environments with fewer or less stringent security requirements, it is generally considered less secure than MAC, as it is more susceptible to Trojan horse attacks and other security breaches where user discretion can be exploited [14].

2.2.4 Role-Based Access Control (RBAC)

Role-Based Access Control (RBAC) is an access control paradigm where permissions are assigned to roles within an organization rather than directly to individual users. This approach simplifies access management by allowing administrators to allocate permissions according to job functions rather than on a user-by-user basis.

The concept of RBAC emerged in the 1990s as a response to the limitations of Discretionary Access Control (DAC) and Mandatory Access Control (MAC) systems, particularly in large organizational contexts where user roles frequently dictate access needs [21]. RBAC's primary appeal lies in its ability to efficiently manage user permissions through roles that correspond to the organization's structure, which reduces the complexity and cost of security administration.

RBAC operates by grouping permissions into roles that reflect the authority and responsibilities inherent in the organization. Users are then assigned to these roles, typically based on their job functions and the principle of least privilege, which ensures they receive access only to the resources necessary for their roles. For instance, a human resources manager will have access to personnel files, while a sales manager will not. This model also includes concepts such as role hierarchies, allowing for inherited permissions, and separation of duties, which restricts the ability of a single user to perform conflicting sensitive tasks, thereby enhancing security and compliance. Over time, RBAC has evolved to support more dynamic conditions in security-sensitive environments, integrating with other models like Attribute-Based Access Control (ABAC) to provide more granular access controls based on additional attributes beyond roles.

2.2.5 Attribute-Based Access Control (ABAC)

Attribute-Based Access Control (ABAC) is a flexible access control methodology where decisions to grant or deny access are made based on attributes associated with users, resources, and the environment. Unlike Role-Based Access Control (RBAC), which relies on predefined roles within an organization, ABAC uses policies that can evaluate multiple characteristics, or attributes, making it highly adaptable to various scenarios. This model supports dynamic and fine-grained control, allowing for complex rules that consider a variety of conditions. ABAC's roots trace back to the need for more nuanced access control mechanisms that could address limitations found in RBAC and other traditional models, especially in environments requiring rigorous compliance with diverse regulatory frameworks.

The concept of Attribute-Based Access Control (ABAC) has been discussed in schol-

arly literature for several years, reflecting a shift towards more dynamic and fine-grained access control mechanisms. ABAC as a formal model has been increasingly cited and developed in academic discussions since the early 2000s [22]. This growth in interest aligns with the broader evolution of IT environments towards more complex and highly distributed systems, where traditional access control models like Discretionary Access Control (DAC) and Mandatory Access Control (MAC) could not efficiently meet the flexible and context-dependent security requirements.

ABAC works by utilizing policies that take into account multiple attributes to make access decisions. These attributes could include user attributes such as department or job title, resource attributes like classification levels, and contextual attributes such as the time of day or location of access. For example, an ABAC policy might allow access to sensitive financial records only for users in the finance department (user attribute) and only during regular business hours (contextual attribute). This approach not only ensures that access control measures are closely aligned with an organization's security policies but also provides the flexibility to easily adapt to changes in the operational environment or business requirements. By enabling such precise control, ABAC can effectively minimize the risk of unauthorized access while accommodating legitimate user needs more efficiently.

2.2.6 Other Access Control Models

In addition to principal models, several other access control frameworks have been developed to meet specific security needs and operational contexts, including:

- **Lattice-Based Access Control (LBAC)**: LBAC is an advanced model designed to prevent illegal information flow. It extends RBAC with lattice structures to define the levels of data classification and the clearances assigned to users, ensuring that users can only access objects at or below their level of clearance [23]. This model is particularly effective in environments that require fine-grained control of information flow, such as government or military applications.

- **Temporal Access Control**: This model incorporates time as a crucial factor in access control decisions. It restricts user access to resources based on time

constraints, enabling scenarios where access permissions can be automatically adjusted based on time of day, week, or other temporal conditions [24]. This is especially useful in business environments where access needs may vary at different times.

- **Location-Based Access Control (LBAC)**: Increasingly relevant in mobile environments [25], this model restricts access based on the physical location of the user. It ensures that sensitive information or resources are only accessible in secure locations, adding a geographical dimension to security policies that are critical for organizations with mobile workforces or multi-site facilities.

- **Risk-Adaptive Access Control (RAdAC)**: This model adjusts access controls dynamically in response to assessed risks [26]. RAdAC systems assess the risk of a particular access event in real-time and adapt security measures accordingly, providing a flexible and responsive approach to potential security threats.

2.3 Open Policy Agent

The Open Policy Agent (OPA) is an open-source, general-purpose policy engine that decouples policy decision-making from enforcement, allowing for versatile and unified policy enforcement across various platforms [1]. OPA is particularly effective in environments such as microservices, Kubernetes, CI/CD pipelines, and API gateways, providing a scalable solution for managing policies that control access to resources based on a rich set of contextual data.

The development of OPA was motivated by the need to decouple policy from software services so that the people responsible for policy can read, write, analyze, version, distribute, and in general manage policy separate from the service itself [27]. In modern IT environments, where systems are often large, distributed, and multifaceted, managing policy compliance traditionally can lead to duplicated effort, inconsistencies, and security gaps as each system might have its unique way of handling policies. OPA provides a unified toolset to streamline this process, enabling central policy management without sacrificing performance and security. It is commonly used for scenarios such as providing unified authorization for microservices, comprehensive access control

for Kubernetes deployments, and policy-based control for cloud-native environments, ensuring that policies are consistently enforced no matter where they are applied.

2.4 Graylog

Graylog is a powerful open-source log management platform designed to help with the rapid collection, storage, and analysis of the massive amounts of log data generated by IT systems and applications [13]. It is built on a server-node architecture that makes it highly scalable and capable of handling extensive data inputs efficiently. At its core, Graylog uses Elasticsearch or OpenSearch as the storage backend, which provides powerful full-text search capabilities and indexing that are ideal for the complex querying of large datasets. MongoDB is used to store metadata and configuration information, supporting the operational management of the Graylog environment. The Graylog server itself processes incoming log messages, performs message filtering, and manages log storage and retrieval. Ranking the best log management and analysis tools in 2024, a recent survey [28] placed Graylog at the 4th rank.

Graylog's use cases are diverse, encompassing system and network monitoring, security information and event management (SIEM), and compliance reporting. It is particularly valuable in environments where quick identification of operational issues or security threats is critical. By providing powerful tools for data visualization, such as dashboards and reports, Graylog enables organizations to gain actionable insights from their data, enhancing decision-making and operational efficiency.

Building on its versatile capabilities, Graylog is extensively utilized across various industries for specific use cases that demand meticulous data analysis and real-time alerting [29]. For instance, in the realm of cybersecurity, Graylog serves as a pivotal component in intrusion detection systems (IDS), where it aggregates and analyzes logs from multiple sources to detect unusual activities that could indicate a security breach [30]. Additionally, in IT operations, Graylog is employed to manage logs from distributed systems, helping IT teams troubleshoot application errors and performance bottlenecks quickly [31]. For compliance purposes, organizations leverage Graylog to maintain and monitor logs required by regulatory frameworks such as GDPR, HIPAA,

or PCI-DSS, ensuring that they can easily produce reports during audits and maintain compliance with legal standards [32]. These diverse applications underscore Graylog's adaptability and effectiveness in addressing complex logging and monitoring needs across different functional areas within an organization.

2.5 Docker

Docker is an innovative open-source platform designed to simplify the process of developing, shipping, and running applications using containerization technology [33]. Introduced in 2013, Docker has revolutionized software deployment by allowing developers to package applications and their dependencies into a standardized unit, called a container, that is portable across any system running the Docker engine. This technology abstracts and automates the deployment of applications, allowing them to run isolated in a shared operating system. Docker containers are lightweight, have reduced overhead, and provide a consistent runtime environment, which solves the common issue of *"it works on my machine"* syndrome. Containers support granular control over resources, making Docker a powerful tool for maximizing resource utilization and simplifying software updates and scalability.

2.5.1 Docker Swarm Mode

Docker Swarm is the native clustering and orchestration tool for Docker, introduced to handle the coordination and administration of multiple containers deployed across multiple host machines. It turns a group of Docker engines into a single, virtual Docker engine, where the standard Docker application programming interfaces are used to control several containers deployed across multiple hosts [34]. This orchestration layer is responsible for managing the lifecycle of containers, scaling in and out as per the demands, and handling container placement strategies based on predefined constraints and configurations. Docker Swarm ensures high availability and redundancy, offering built-in service discovery and robust load balancing capabilities. It is especially praised for its simplicity and ease of use, integrating deeply with the Docker ecosystem and leveraging its native APIs for seamless, scalable deployments.

Docker Swarm allows developers and system administrators to manage a cluster of Docker nodes as a single virtual system, enhancing the operational capabilities of Docker in production environments.

Docker Swarm provides several key features that enhance its functionality as a container orchestration tool [34]:

- **Cluster Management:** Docker Swarm automatically turns multiple docker hosts into a single, virtual Docker host, providing a high level of abstraction to manage a cluster as if it were a single virtual Docker engine.
- **Decentralized Design:** It operates on a decentralized design which means that it can use the Docker API as its front end which makes it easier to plug into any tools that already communicate with a Docker daemon.
- **Scaling:** Allows for the automatic scaling of applications up or down, depending on the demand, without manual intervention, ensuring efficient use of resources.
- **Load Balancing:** Automatically distributes container workloads uniformly across the cluster to optimize resource utilization and maximize redundancy.
- **Service Discovery:** Containers in a Docker Swarm can automatically discover each other, which simplifies the process of connecting containers and creating complex multi-container applications.
- **Rolling Updates:** Facilitates rolling updates to applications across the cluster, allowing for seamless updates and minimizing downtime.
- **High Availability:** Supports high availability by ensuring that the application services are duplicated across multiple nodes in the cluster, enhancing fault tolerance.
- **Security:** Provides secure node communication and ensures that only authorized nodes can join the cluster through the use of mutual TLS and digital signatures.

These features make Docker Swarm an effective tool for managing containerized applications at scale, promoting automation, ease of use, and efficient resource management.

CHAPTER 3

RELATED WORK

This chapter provides a review of the existing literature pertinent to the integration and effectiveness of policy compliance monitoring systems, with a particular focus on the use of the Open Policy Agent (OPA) and log management tools like Graylog. The selection of these works is based on their relevance to the key research questions posed in this thesis, specifically examining the applications of Open Policy Agent and the role of log management systems in enhancing security and monitoring capabilities, and the evolving practices in policy compliance and auditing.

3.1 Policy Compliance and Auditing

Research on policy compliance and auditing within IT environments has shown a progression from manual auditing methods toward more automated and continuous approaches. To automate cloud compliance, a framework for integrating policy enforcement within the CI/CD pipelines using open-source tools is proposed by utilizing the Policy as Code methodology [2]. The work demonstrates the practical application of these tools in enhancing security compliance in cloud environments, effectively showing how Policy as Code can be operationalized to ensure continuous compliance and security in cloud deployments. The open-source tools incorporated are (i) *tfsec and Regula* for static analysis of Infrastructure as Code (IaC), (ii) *Cloud Custodian* for ongoing cloud security posture management (CSPM), and (iii) *OPA Gatekeeper* for enforcing policies directly on Kubernetes clusters. The work's approach aligns closely with our thesis in using OPA for policy enforcement. However, their approach

focuses on automating compliance verification in cloud environments and integrating these checks into the CI/CD pipeline, while our thesis aims at enhancing real-time monitoring and logging of policy compliance using Graylog alongside OPA.

Focusing on enhancing security in cloud-native environments through effective policy management, [6] explores how integrated policy management systems can significantly enhance security and compliance in cloud-native environments. Namely, Azure Policy is used to show how it can be incorporated with Gatekeeper to apply enforcements and safeguards on Azure Kubernetes Service. The work's approach is specific to cloud-native environments and involves tools like Kubernetes and Falco for policy enforcement. While leveraging OPA, our thesis integrates Graylog for enhanced logging and monitoring, focusing on real-time compliance monitoring and alerting in various IT environments, not limited to cloud-native infrastructures.

As the number of cloud users and security incidents rises, a new model, POVIDE (Policy Violation Detection Model), to detect policy violations leveraging anomaly detection techniques is proposed in [5], addressing the increasing threats and policy violations in cloud computing environments. The model leverages a real-time operational framework to terminate suspicious activities, aiming to overcome the identified shortcomings of existing systems like IDS and firewalls. The detection framework focuses on continuous monitoring and uses both supervised and unsupervised learning methods to improve the accuracy of violation detection. While the proposed model and our thesis share the common goal of improving policy compliance monitoring, the work's approach emphasizes machine learning and anomaly detection to identify policy violations. In contrast, our thesis approach focuses on utilizing existing log management tools for policy compliance.

Misuse of network protocols within security-restricted environments is identified in [35], particularly through traffic analysis of the Secure Shell (SSH) protocol. The research investigates how different types of SSH traffic can be distinguished based on observable characteristics, even when the content is encrypted. A method is proposed to analyze encrypted SSH traffic to determine its compliance with defined security policies. By focusing on the traffic's meta-information like packet sizes and inter-arrival times, the study aims to detect and classify the nature of the SSH traf-

fic—whether it is being used for acceptable tasks or potentially violating policy by carrying non-permitted types of traffic. The work focuses on network traffic analysis to detect policy violations, particularly within SSH sessions using traffic analysis techniques. In contrast, our thesis approach is more centered on log management and real-time alerting.

The existing work underscores a significant emphasis on automating policy compliance monitoring, with various approaches tailoring this automation to specific technologies and frameworks in cloud environments.

3.2 Open Policy Agent (OPA) Applications

Recent studies have increasingly highlighted the versatility of OPA across various IT environments, underscoring its adaptability and efficacy in managing complex policies. In [36] a RESTful approach to tape management within the Storage Resource Manager (StoRM) system, developed at INFN-CNAF, is introduced which utilizes Open Policy Agent (OPA) for authorization policy decisions. The newly developed StoRM Tape service implements a simplified HTTP interface for tape-stored files management and utilizes OPA to define and enforce authorization policies effectively. The integration of OPA streamlines the security framework by providing a dynamic, policy-driven authorization mechanism that adapts to modern web technologies. This deployment illustrates how OPA facilitates complex security policy enforcement in large-scale data management environments within scientific communities. However, it is identified that there are no defined mechanisms to monitor policy violations.

As Telemedicine applications proliferate, securing sensitive healthcare data across diverse and often fragmented systems becomes paramount. [37] addresses critical challenges of complex access control requirements within the Telemedicine sector, particularly under offline use cases. The study proposes a solution leveraging Open Policy Agent (OPA) to enforce policies dynamically across a telemedicine infrastructure, ensuring data security even in offline scenarios where necessary. However, the proposed system lacks the capability to detect policy violations.

The INFN-CNAF data center, serving a wide range of scientific collaborations, seeks

to enhance access to S3 and WebDAV protocols by incorporating secure authentication mechanisms and ensuring compliance with federated authorization standards. The focus is to integrate POSIX-like capabilities with modern cloud storage services through the use of OpenID Connect (OIDC) and JSON Web Tokens (JWT) for authentication and authorization. [38] highlights the use of Open Policy Agent (OPA) as a critical tool for managing security in cloud storage environments that integrate POSIX capabilities. Specifically, OPA is employed alongside the Ceph RADOS Gateway (RGW) to ensure that the access controls and authorization procedures align with the security policies dictated by the managing identity providers like INDIGO-IAM. While the paper focuses on integrating secure, federated access to remote storage using Open Policy Agent (OPA), our thesis emphasizes integration with OPA across various IT environments.

To enhance security compliance and manage component dependencies in cloud environments, the integration of Open Policy Agent (OPA) within a DevSecOps framework is addressed in [39]. The work proposes the Policy Champion Model, which utilizes policy-as-code to support continuous integration and deployment processes, thereby aligning with best practices and security measures. As a part of the solution, OPA is employed as a core component in ensuring that all cloud configuration files and operational procedures meet predefined security policies and compliance standards. The policies enforced by OPA also serve as a form of ”grey documentation,” which aids new developers in understanding the system's architecture and security protocols through interactive and responsive policy evaluation. The integration of a custom tool named Wand alongside OPA automates the remediation processes when discrepancies between configurations and policy standards are detected. This automation supports agile development practices by reducing the manual overhead for developers. The integration shows the versatility of using OPA in different use cases, similar to what is shown in our thesis.

The existing literature on Open Policy Agent (OPA) showcases its application across diverse fields such as scientific data management, telemedicine, and cloud services, highlighting its adaptability and effectiveness in various complex environments. These studies focus on OPA's role in enabling dynamic, policy-driven authorization that accommodates the specific needs of different sectors.

3.3 Log Management and Analysis Tools

Existing work on log management tools, such as Graylog, ELK Stack, and Splunk, reveals a significant focus on their critical roles in monitoring and ensuring system security. A comprehensive analysis of log management processes using open-source tools is conducted in [40], with a particular focus on using Graylog for handling log data efficiently. The study highlights the importance of log management in maintaining security, and its critical role in collecting, analyzing, and managing log data. The authors advocate for the use of Graylog, an open-source log management tool, due to its efficiency in indexing, high performance, and cost-effectiveness. The integration of Graylog with systems such as Active Directory and File Servers is showcased, demonstrating its capability to collect and analyze logs. This integration is crucial for preemptively identifying and mitigating unauthorized access and other security threats. Graylog employs sophisticated filtering and query mechanisms to effectively sift through large volumes of log data. Furthermore, the paper highlights Graylog's ability to set up real-time alerts based on specified log data conditions, aiding in proactive security management.

A comparison of two popular open-source log management tools, *ELK Stack* and *Graylog* in [41] focuses on various performance metrics like response time, CPU and memory usage, as well as their capabilities in security, alerting, monitoring, and data visualization. The assessment of ELK Stack and Graylog in managing logs within network environments indicates that ELK Stack outperforms Graylog in stability, faster response time, and reliability under stress conditions. This makes the ELK Stack better suited for environments requiring robust performance. Conversely, Graylog is noted for its user-friendly interfaces, out-of-the-box features, ease of maintenance, and superior suitability for centralized log management, particularly excelling in security and alerting functionalities. Graylog offers more accessible and integrated alerting features out-of-the-box, which is advantageous for security monitoring and incident response.

In the white paper [42], Graylog discusses the importance and benefits of integrating automation into security operations. The paper highlights the challenges faced by security operations teams, such as high volumes of security alerts and manual process-

ing tasks, which can lead to inefficiencies and alert fatigue. Graylog proposes the use of Security Orchestration, Automation, and Response (SOAR) systems to automate repetitive tasks, enhance threat detection, and improve response times. The integration of log management tools, like Graylog, with SOAR platforms, is emphasized for creating a comprehensive security environment. By centralizing and enriching log data, these systems can automate complex workflows, reduce false positives, and enable more effective threat intelligence and incident response. The paper underscores that automation is essential for keeping pace with the evolving threat landscape and maintaining robust security postures.

3.4 Gap Analysis

Despite the extensive research on OPA applications, policy compliance, auditing, and log management tools, there remains a noticeable gap in studies that specifically explore the integration of these three components to enhance policy compliance monitoring. Furthermore, a recent survey highlighted on the OPA blog [43] reveals that a substantial portion of OPA users—36% do not track OPA decisions, and 39% do not monitor their OPA status, despite the availability of management APIs designed for these purposes. This finding underscores a critical gap in the current utilization of OPA's capabilities, suggesting a need for improved user guidance on configuring monitoring and logging.

This thesis aims to bridge the gap in monitoring OPA decision logs by providing empirical evidence on the effectiveness of employing Graylog. It offers insights into real-time alerting on policy violations and enhances the overall visibility of policy compliance in containerized environments. By simplifying the integration process of OPA with Graylog, users can set up and maintain comprehensive monitoring of OPA's operational status and decision logs. This streamlined solution for capturing and analyzing OPA decision logs within Graylog's powerful analytical framework contributes to reducing the oversight gap identified in the literature.

Ultimately, this integration enhances the visibility and traceability of policy enforcement actions and promotes a more proactive stance towards policy compliance mon-

itoring and management in diverse operational contexts. This approach ensures continuous oversight and actionable insights into the operational status and effectiveness of policy applications across various OPA use cases.

Additionally, Graylog's Content Packs significantly enhance collaborative cybersecurity efforts by allowing the packaging and sharing of configurations such as dashboards, alert conditions, and input settings. This feature enables the tools and rules developed during the research to be easily shared and reused by the broader community, fostering a collaborative approach to refining and improving these configurations for more effective threat detection and response.

3.5 Summary

The literature reviewed in this chapter lays a substantial groundwork for this thesis, demonstrating the critical roles that OPA and log management tools like Graylog play in policy compliance monitoring. However, the unique contribution of this research lies in its exploration of the synergy between OPA and Graylog, aimed at improving policy compliance monitoring. This work is poised to offer insights into the practical challenges and benefits of this integration, potentially guiding future innovations in IT security management practices.

CHAPTER 4

METHODOLOGY

This chapter provides step by step description of the methodology adopted in this thesis work to enhance policy compliance monitoring with Graylog in Open Policy Agent (OPA). The research was structured into several key phases, each contributing to the overall objective of improving the understanding and implementation of policy enforcement and log monitoring systems, to find answers to the problems stated in Section 1.1. First two phases, *Analysis* and *Design* are elaborated here while detailed explanations of *Implementation & Deployment*, and *Demonstration* are given in their own separate chapters, in Chapter 5 and in Chapter 6 respectively.

4.1 Research Methodology Outline

The methodology adopted in this thesis is structured into four key phases: Analysis, Design, Implementation & Deployment, and Demonstration. This structured approach is illustrated in Figure 4.1. Each phase encompasses specific activities which are briefly explained in the following paragraphs.

The **Analysis** phase involves a thorough examination of the OPA architecture, decision logs, and Graylog architecture. This foundational step aims to understand the components and interactions within OPA and Graylog, and how they can be leveraged for effective policy compliance monitoring. *Analysis of OPA Architecture* includes a detailed examination of its setup, Application Programming Interfaces (APIs), and logging capabilities to understand how policies are enforced and logged. This step is crucial for understanding how policies are enforced and logged within OPA. The

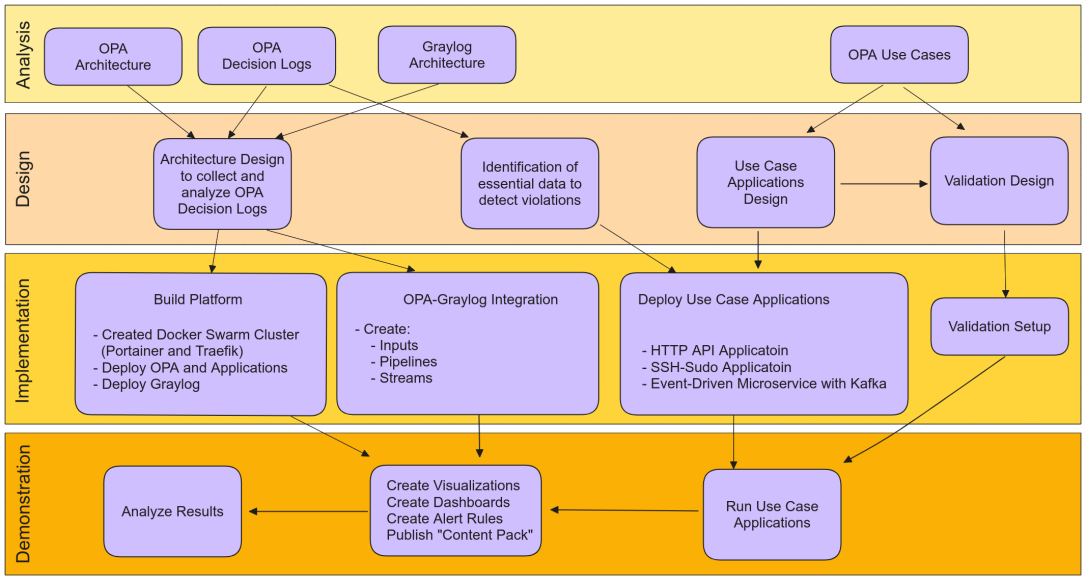

Figure 4.1: Research Methodology

content and structure of *decision logs generated by OPA* are analyzed to identify the key data elements necessary for detecting policy violations. *Graylog's features* for log ingestion, parsing, storage, and visualization are analyzed to determine how it can be integrated with OPA for effective policy compliance monitoring. This involves assessing Graylog's capabilities to handle and process logs efficiently, ensuring that it can support real-time alerting and comprehensive visibility. Additionally, *relevant OPA use cases* are identified where OPA's policy enforcement could benefit from enhanced logging and monitoring. This provides real-world context and relevance to the analysis, ensuring that the findings are applicable and practical. The results of this analysis phase provide the context and groundwork for the subsequent phases of the methodology.

The **Design** phase focuses on activities to form *the architecture* that integrates OPA with Graylog, specifying how decision logs are collected, parsed, and analyzed. This includes *identifying essential data points from OPA decision logs* that need to be monitored to detect policy violations effectively. Use case applications are designed to showcase OPA integrations, providing practical examples of how the system functions. Additionally, the validation design is developed to ensure that the system can be effectively tested and evaluated. This includes designing traffic generators to produce both policy-compliant and non-compliant traffic for testing purposes.

The **Implementation** phase has the key activities to bring the architecture to life. *Building the platform* includes setting up the necessary infrastructure, including creating a Docker Swarm cluster with Portainer and Traefik, deploying OPA and the use case applications, and configuring Graylog for log management. *OPA-Graylog Integration* requires the creation of inputs, pipelines, and streams in Graylog to handle the data from OPA. *Deployment of Use Case Applications* includes implementation of the selected use case applications, namely HTTP API Application, SSH-Sudo Authorization Application, and Event-Driven Microservice Architecture (MSA) with Kafka to generate relevant policy decision logs. In the *Validation Setup*, the environment is set and the required tools are configured for validating the integrated system. This includes developing specially crafted applications to generate policy violations on demand.

In the **Demonstration** phase, the integrated system is put into action. *Use case applications are executed* to generate both compliant and non-compliant policy traffic. *Various dashboards and widgets* are created in Graylog to visualize the data collected from OPA decision logs, and *alert rules and notification channels* are set up. This step also included the development and release of a *"Content Pack"* for Graylog. Finally, *the results are analyzed* to assess the effectiveness of the proposed solution in enhancing policy compliance monitoring. This involves examining the alerts, and dashboards to determine how well they capture and report policy violations.

By following this structured methodology, this thesis provides a comprehensive approach to improving policy compliance monitoring with Graylog and OPA, ensuring robust, real-time alerting and visibility in containerized environments.

4.2 Analysis

This section provides a detailed analysis of OPA Architecture and decision logs, Rego Language, and Graylog architecture along with its capabilities.

4.2.1 OPA Architecture

The workflow of the Open Policy Agent (OPA) system, illustrated in Figure 4.2, The Open Policy Agent (OPA) workflow involves a service sending a JSON-formatted query to OPA, which evaluates it against predefined Rego policies and contextual JSON data. Based on this evaluation, OPA then generates a decision and returns it to the service. This workflow is integral to OPA's functionality within various services, handling requests and events to ensure compliance with the specified access control policies. OPA utilizes a declarative language called Rego, inspired by the older query language Datalog, to articulate policies.

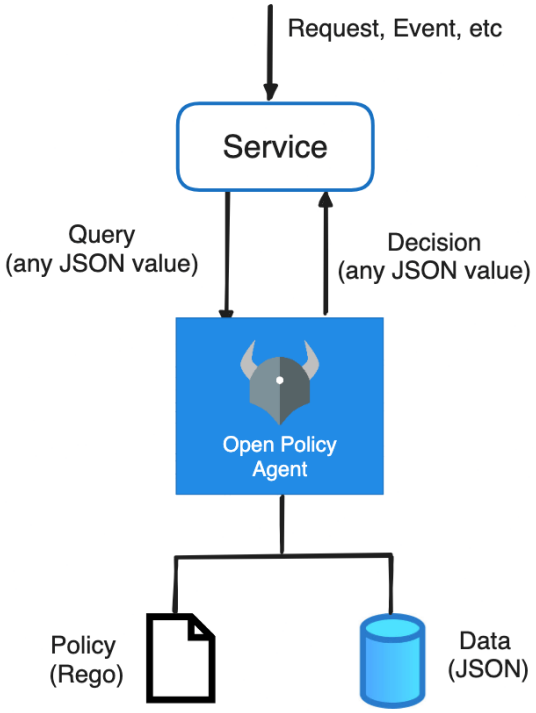

Figure 4.2: Workflow of OPA [1]

4.2.1.1 The Rego Language

Rego is the declarative language used by OPA to express policies. Rego allows policy authors to write clear and logical rules that decide who can do what under which conditions across the stack. It is designed to interact with structured data (JSON) and supports both low-level data filtering and high-level policy decisions [44].

Rego's syntax and structure are inspired by Datalog, which is renowned for its simplicity and power in dealing with logical queries. However, Rego extends these capabilities with additional features tailored to modern policy management needs, such as support for JSON. This makes Rego particularly adept at managing and interpreting the JSON-formatted data prevalent in many contemporary applications and services.

One of the key strengths of Rego is its ability to perform fine-grained access control. For example, a policy might specify that only certain users can access specific resources under defined conditions. Consider a scenario where access to a sensitive resource is allowed only if the user is an administrator or if the request comes during business hours. A Rego policy enforcing this rule would query the data describing the user's role and the time of the request, ensuring compliance with these conditions.

Rego is well-suited for writing both Role-Based Access Control (RBAC) and Attribute-Based Access Control (ABAC) policies. In RBAC, access decisions are made based on the roles assigned to users within an organization. An example policy implemented in Rego is given in Listing 4.1, specifying that only users with the role of *"manager"* (line 6) can *approve expenses* (line 7), while all other requests are rejected (line 3). Assume Jane is working in a company as a manager and is assigned manager role in the company's payroll system. This policy enforced for the payroll system allows Jane to approve expenses.

```
1 package rbac
2
3 default allow = false
4
5 allow {
6     input.role == "manager"
7     input.action == "approve_expense"
8 }
```

Listing 4.1: RBAC Policy Example in Rego

For ABAC, attributes of the user, resource, or environment dictate access controls. This model is more flexible and context-aware. For example, the ABAC policy implemented in Rego, given in Listing 4.2, allows users to *access* (line 7) a resource only if *the user's department* matches the department attribute of the resource (line

6) and the *current time* is within business hours from 9 am to 5 pm (lines 8-9). Assume Bob is working as a developer in R&D department of a company where analysis documents are stored in a local repository to be used by R&D department. This policy enforced for the local repository allows Bob to access analysis documents during business hours.

```
1  package abac
2
3  default allow = false
4
5  allow {
6      input.user.department == input.resource.department
7      input.action == "access"
8      now.hour  = 9
9      now.hour   17
10 }
```

Listing 4.2: ABAC Policy Example in Rego

4.2.2 OPA Use Cases

The OPA project has gained significant traction in production environments across notable organizations such as Goldman Sachs, Netflix, Pinterest, T-Mobile, among others [45]. A recent survey involving over 240 organizations [43] revealed that 43% are at the production stage with their OPA implementation. Over half of the respondents utilize OPA for multiple applications. The primary applications of OPA are configuration authorization, including Kubernetes admission control, and API authorization. Additionally, the project has effectively partnered with several CNCF projects like Kubernetes, Envoy, CoreDNS, Helm, and SPIFFE/SPIRE. It is also compatible with Gatekeeper, enhancing Kubernetes with native capabilities for admission policy enforcement and auditing.

A 2021 survey [46] encompassing responses from 300 participants identified the most prevalent use cases for Open Policy Agent (OPA). The findings, as summarized in Table 4.1, categorize these use cases by the percentage of respondents employing each method.

Table 4.1: Distribution of Respondents by Open Policy Agent (OPA) Use Cases

Use Case	% of Respondents
Kubernetes admission control	54%
Application authorization	39%
Microservice authorization	39%
Terraform validation	25%
Other	5%

In this thesis, the focus is on three primary applications of OPA, which collectively encompass 78% of its use cases. These applications include (i) HTTP API Authorization, encapsulating general application authorization scenarios, (ii) SSH and Sudo Authorization, which also fall under application authorization, and (iii) Event-Driven Microservice Architecture integrated with Kafka, pertaining to microservice authorization. This targeted approach highlights the predominant areas where OPA is implemented within industry practices.

4.2.3 OPA Decision Logs

Open Policy Agent (OPA) decision logs [12] provide detailed insights into policy execution. Each decision log records an event that details a policy query, capturing comprehensive information that aids in the auditing and debugging of policy decisions. These logs are structured to include the policy queried, the inputs to those queries, bundle metadata, and other pertinent details that enhance transparency and facilitate detailed analysis. Key fields are explained below to understand the structure and information encapsulated within an OPA Decision Log.

- `bundles` describes the policy bundles involved in the decision-making process.
- `decision_id` refers to a unique identifier for each decision, enhancing traceability.
- `input` specifies the input data used in the policy evaluation.
- `labels` are identifiers that provide context about the OPA instance, such as version and instance ID.
- `level` indicates the log level set on the OPA server.

- `metrics` provides performance metrics related to the decision-making process, useful for performance tuning and troubleshooting.
- `path` specifies the hierarchical path to the policy that was evaluated.
- `req_id` is the request identifier unique to the OPA instance, aiding in correlating logs with specific policy evaluations.
- `requested_by` shows the client's identifier that made the request, typically an IP address or a server identifier.
- `result` contains the outcome of the policy evaluation and additional data utilized during the decision-making process.
- `timestamp` is a RFC3999-compliant timestamp when the policy decision was made.

A concrete example of a decision log generated by an HTTP API application is given in Listing 4.3, it indicates that the nginx policy bundle was loaded (lines 1-4), `decision_id` is a unique identifier for this specific policy decision (line 5). The `input` field indicates the HTTP method used (line 7), the path being accessed (lines 9-11), and the user making the request (line 13). The `path` field indicates the policy path that was evaluated (line 28), and the `result` data structure indicates the outcome of the policy decision and the data used in the policy decision-making process (lines 31-45).

```
1  {
2    "bundles": {
3      "nginx": {}
4    },
5    "decision_id": "75dee627-7faa-4a4f-94ee-c7b5ebc924ba",
6    "input": {
7      "method": "GET",
8      "path": [
9        "finance",
10       "salary",
11       "charlie"
12     ],
13     "user": "bettie"
```

```
  },
  "labels": {
    "id": "2b059a9a-e0a9-42a2-94f0-832b98393329",
    "version": "0.63.0"
  },
  "level": "info",
  "metrics": {
    "counter_server_query_cache_hit": 1,
    "timer_rego_external_resolve_ns": 800,
    "timer_rego_input_parse_ns": 40983588,
    "timer_rego_query_eval_ns": 59372014,
    "timer_server_handler_ns": 100465733
  },
  "msg": "Decision Log",
  "path": "httpapi/authz",
  "req_id": 996347,
  "requested_by": "10.0.5.6:38812",
  "result": {
    "allow": false,
    "hr": [
      "david"
    ],
    "subordinates": {
      "alice": [],
      "betty": [
        "charlie"
      ],
      "bob": [
        "alice"
      ],
      "charlie": []
    }
  },
  "time": "2024-05-14T07:28:34Z",
  "timestamp": "2024-05-14T07:28:34.343139436Z",
  "type": "openpolicyagent.org/decision_logs"
}
```


Listing 4.3: Example of an OPA Decision Log

This structured approach to logging ensures that each aspect of a policy decision is

recorded and available for review, making OPA decision logs a vital tool for ensuring policy compliance and for debugging and optimizing policy performance.

4.2.4 Graylog Architecture

The system architecture of Graylog is designed to be scalable and robust, utilizing Elasticsearch or OpenSearch for log storage and MongoDB for metadata and configuration data, illustrated in Figure 4.3. This allows Graylog to perform fast data retrieval, facilitating complex searches that are crucial for detailed analysis and reporting. The platform's server nodes can be clustered to enhance performance and fault tolerance, ensuring reliable service even under heavy loads.

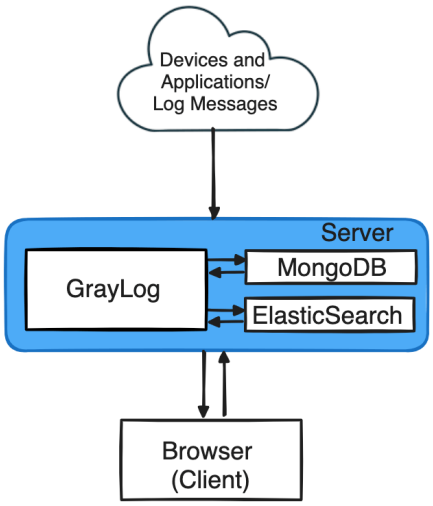

Figure 4.3: Graylog Architecture

The architecture of Graylog is further enhanced with a web interface that provides a user-friendly way to configure inputs, index sets, streams, and dashboards. This interface allows users to create complex searches and alerts, facilitating real-time analysis and monitoring. Graylog also supports a plugin system, which extends its capabilities through integrations with other tools and applications. For instance, plugins can provide additional message input sources, output destinations, or custom functions for processing log data. The system is designed to be highly available and fault-tolerant, with features such as load balancing and the ability to scale horizontally across multiple servers. This robust architecture ensures that Graylog can serve as the central

component of a comprehensive logging strategy, providing insights and operational intelligence critical for modern IT operations.

4.2.4.1 Streams

Graylog Streams [47] allow users to segment logs into manageable groups, allowing for more targeted analysis and quicker access to relevant data. Streams are beneficial for managing large volumes of logs by enabling the segmentation of log messages that match certain patterns or criteria. This functionality not only helps in organizing logs more efficiently but also in monitoring and alerting on specific events within the log data. This systematic categorization aids in quicker searches and more focused analysis, enhancing the overall log management process.

4.2.4.2 Inputs

Inputs in Graylog [48] are essential for capturing logs transmitted over the network using various protocols. For instance, the *"Syslog UDP Input"* operates as a *"listener input"*, which implies that it actively listens on a designated port, awaiting data transmissions from external applications to the Graylog platform. This type of input is designed to handle syslog messages that comply with RFC 5424 [1] and RFC 3164 [2] standards.

4.2.4.3 Pipelines

In Graylog, pipelines [49] are mechanisms designed to process messages as they flow through the logging system. These pipelines facilitate the parsing, transformation, and enrichment of messages before they are either stored or forwarded. By linking pipelines to specific streams, users can tailor processing tasks to the unique characteristics of the incoming messages, enhancing the system's efficiency and capability to handle complex data operations.

[1] https://www.ietf.org/rfc/rfc5424.txt
[2] https://www.ietf.org/rfc/rfc3164.txt

Pipeline Rules

Pipeline rules are essential components within a pipeline, setting the conditions and actions that manage how messages are processed. These rules follow a simple *"if-then"* logic where the *when* clause specifies the conditions under which the rule applies, and the *then* clause defines the actions to be taken if those conditions are met. An example of a basic pipeline rule is given in Listing 4.4 where rule name is specified as *"example_rule"* (line 1). Conditions [50], such as the presence of a field or specific patterns within a message field, are expected to be written between *when* (line 2) and *then* (line 4). They also support Boolean and comparison operators to enable complex logical constructions. This flexibility allows rules to be intricately tailored to specific requirements, ensuring precise control over message processing. Actions are listed between *then* (line 4) and the *end* (line 6).

```
1 rule "example_rule"
2 when
3     // Conditions to be met
4 then
5     // Actions to be performed
6 end
```

Listing 4.4: Example of a basic pipeline rule

Pipeline Stages and Rules Execution Order

Pipelines are segmented into multiple stages, sequentially organizing the execution of rules. Each stage contains a set of rules processed in a defined order, allowing dependencies between rules to dictate their arrangement. For example, a common practice is to place data normalization rules in the early stages to prepare the data for subsequent processing.

Messages are processed in parallel across corresponding stages of different pipelines, enhancing throughput and efficiency. However, this parallelism requires careful orchestration to maintain data consistency across the system. For illustration, Stage 1 of Pipeline 1 runs concurrently with Stage 1 of Pipeline 2, as shown in Figure 4.4 [51]. This setup ensures that data handling is both rapid and aligned with other concurrent

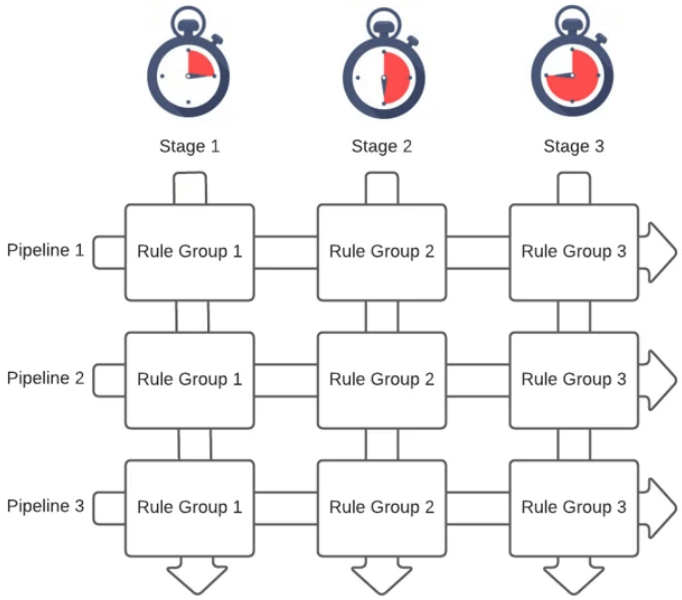

Figure 4.4: Pipeline Stages and Rules Execution Order

processes. By structuring data handling in this manner, Graylog effectively supports detailed log analysis and sophisticated data manipulation tasks, making it an indispensable tool for comprehensive data management strategies. The use of pipelines not only streamlines the processing of voluminous log data but also reinforces the system's capacity to perform under complex operational demands.

4.2.4.4 Dashboards

In Graylog, dashboards [52] play a major role in visualizing and analyzing log data. Graylog dashboards are customizable interfaces that allow users to define and share search queries and visualizations. They provide a consolidated view of log data, enabling users to monitor, analyze, and report on various aspects of their log data in real time. Dashboards support multiple tabs for different use cases and allow full-screen display for presentations. Dashboards can be used for various purposes. Such examples include (i) monitoring system performance metrics, (ii) tracking security events and alerts, (iii) visualizing application log data for troubleshooting, and reporting on compliance and audit logs.

- *Widgets* are the building blocks of Graylog dashboards. They are versatile elements that can display various types of data visualizations such as charts,

graphs, and tables. Each widget can be configured with specific search criteria, time ranges, and visual representation formats, allowing users to tailor the data presentation to their needs.

- *Aggregations* are used within widgets to summarize and analyze large volumes of log data. By grouping data based on specific fields and applying metrics like count, sum, or average, aggregations help in extracting meaningful insights from the logs. This capability is essential for identifying trends, patterns, and anomalies in the data.

- *Visualizations* in Graylog include a range of graphical representations such as line charts, bar charts, pie charts, and tables. These visual tools enhance the interpretability of log data by providing clear and concise graphical summaries. Users can select the most appropriate visualization type for their data to effectively communicate insights.

4.2.4.5 Event Definitions and Alerts

Event Definitions in Graylog [53] allows users to create specific triggers based on log data, enabling proactive monitoring and alerting for various conditions within an IT environment. An event refers to specific circumstances that cause a deviation from normal behavior, such as changes to firewall policies or failed login attempts from blacklisted IP addresses. By defining events, users can automate the detection of these conditions and set up alerts to notify stakeholders, ensuring prompt response to potential security threats or operational issues. There are two primary methods to create event definitions in Graylog:

1. **Event Definition Wizard**: A guided process within the Graylog interface that allows users to specify event criteria, set priorities, and define remediation steps.

2. **Direct Creation from Search Results**: This method allows users to generate event definitions directly from the log search results, making it easy to pinpoint specific log patterns and set up alerts based on them.

Event Types: Graylog supports different types of events depending on the edition

being used. In Graylog Open, which is our current focus, only the Filter & Aggregation event type is supported. However, other editions such as Graylog Enterprise and Graylog Security offer additional support for Event Correlation, which provides more advanced capabilities for detecting complex event patterns.

Event Priorities: Setting a priority for each event definition helps in categorizing the importance of the event. This ensures that critical events are addressed with the urgency they require, while less critical events can be handled appropriately without overwhelming the response teams.

Remediation Steps: Event definitions can include optional remediation steps that provide actionable guidance on how to address the detected issue. The steps will be included in the corresponding alert and are valuable for ensuring that the appropriate response is taken promptly, minimizing the impact of the event.

Event Criteria: The conditions that must be met for an event to be triggered are specified as event criteria. These are crucial for accurately capturing relevant events and avoiding false positives. Criteria can include specific log message patterns, thresholds for metric values, or combinations of conditions using Boolean logic.

Event Fields: Events can have custom fields that allow for the extraction and storage of specific data points from log messages. This can be useful for enriching events with additional context or for using these fields in subsequent analysis and alerting.

Event Alerts: Alerts in Graylog [54] are notifications configured to inform users when an event is triggered. They are crucial for proactive monitoring and rapid response to potential issues, ensuring that administrators are promptly informed about critical incidents. Types of Alerts Supported by Graylog are explained below.

1. **Email Alerts -** These alerts send notifications via email to specified recipients, providing a straightforward way to inform stakeholders about important events.
2. **Slack Alerts -** Integrates with Slack to send alert notifications to specified channels, facilitating real-time communication within teams.
3. **Webhook Alerts -** Allows for integration with various third-party services by sending HTTP POST requests to specified URLs, enabling automated work-

flows and integrations.

4. **PagerDuty Alerts -** For users of Graylog Enterprise, these alerts integrate with PagerDuty to provide robust incident management capabilities, ensuring critical issues are escalated and addressed promptly.

5. **Microsoft Teams Alerts -** Similar to Slack alerts, these send notifications to Microsoft Teams channels, supporting team collaboration and quick response.

Graylog's alerting system is highly extensible, supporting various plugins and integrations available in the Graylog Marketplace. This flexibility allows organizations to tailor their alerting mechanisms to their specific operational needs, ensuring efficient and effective monitoring and incident response.

4.2.4.6 Content Packs

Graylog Content Packs are pre-configured support packages that make it easier to set up Graylog for specific environments or tasks. These packs can include streams, inputs, outputs, dashboards, and even alert conditions [55], providing a comprehensive solution for particular logging needs or data analyses. Content Packs are particularly useful because they allow Graylog users to share their configurations and setups with the community, enabling others to quickly deploy and utilize proven configurations without the need to start from scratch. This community-driven approach significantly reduces the time and effort required to implement complex logging solutions and allows for the sharing of best practices among users.

The availability of these Content Packs facilitates a collaborative ecosystem where both new and experienced users can benefit from the collective expertise of the Graylog community. For instance, if a user has configured Graylog to monitor Apache web servers efficiently, they can package this configuration and share it as a Content Pack. Other community members can then download and deploy this pack to get up and running with similar monitoring needs quickly. This not only fosters a sense of community but also enhances the overall utility and effectiveness of Graylog deployments. The ability to share complex configurations and analytical dashboards

democratizes access to advanced log management capabilities, making high-level IT operations support accessible to a broader range of users and organizations.

4.3 Design

This section outlines the detailed design of the system for monitoring policy compliance. The aim is to construct a robust, scalable environment that facilitates policy compliance monitoring across various applications.

4.3.1 Environment Architecture Design

The environment architecture is designed to provide a robust and scalable framework for monitoring policy compliance across various applications. The architecture's primary components are categorized as follows: Docker Swarm Cluster for container orchestration; infrastructure services including Traefik for load balancing, Portainer for cluster management, and Graylog for log management and analysis; an authorization service with Open Policy Agent (OPA) for policy enforcement; and use case applications selected in Section 4.2.2 and designed in Section 4.3.3. Figure 4.5 illustrates the overall architecture design, where access requests are routed to relevant applications by Traefik and handled by OPA, generating a unique decision log for each request. These logs are then collected and processed by Graylog to identify policy violations, which trigger predefined alert mechanisms.

4.3.1.1 Docker Swarm Cluster

At the core of the architecture is the Docker Swarm cluster, which is composed of multiple Docker engines running on virtual machines (VMs). The Docker Swarm cluster provides a scalable and resilient platform for deploying containerized applications. It includes the following nodes: *Docker Engine 1* on Host VM1, *Docker Engine 2* on Host VM2, and *Docker Engine 3* on Host VM3. Each Docker engine runs on a separate virtual machine with its own host operating system, providing an isolated environment for container execution.

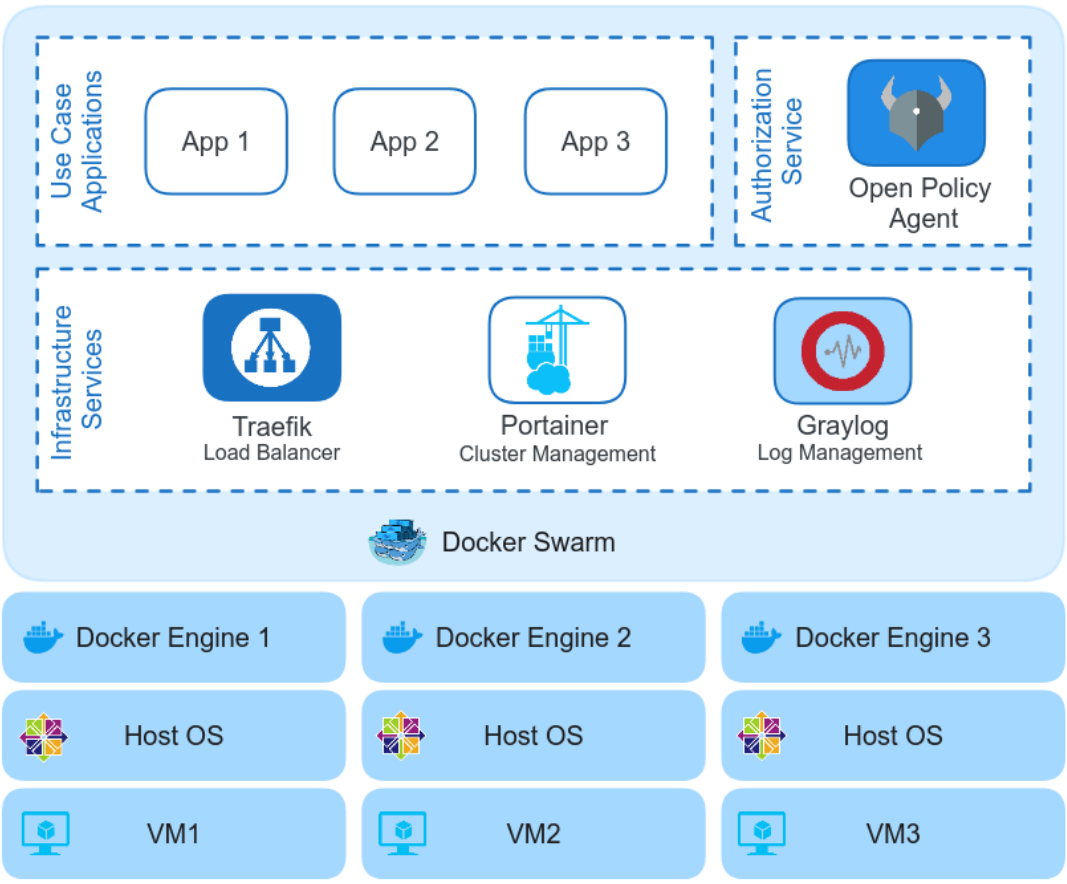

Figure 4.5: Environment Architecture Design

4.3.1.2 Infrastructure Services

Three main services are employed to manage the cluster easily, load-balance incoming requests, and monitor policy violations by collecting and analyzing OPA decision logs.

Traefik (Load Balancer)

Traefik [56] is a modern HTTP reverse proxy and load balancer designed to ease the deployment of containerized applications. It simplifies networking complexity associated with deploying and securing containerized architectures by automatically discovering the right configuration for services based on their deployment context. Traefik seamlessly integrates with major container management systems like Kubernetes, Docker, Docker Swarm, and others, dynamically configuring itself and responding as services are added, removed, or scaled. It also includes built-in mechanisms for SSL

certificate management through integration with *Let's Encrypt* for automatic SSL certificate issuance and renewal.

In our design, Traefik acts as a dynamic reverse proxy and load balancer with the role of routing incoming traffic to the appropriate services within the Docker Swarm cluster. Traefik automatically discovers services and manages SSL certificates to secure communications.

Portainer (Cluster Management)

Portainer Community Edition (CE) [57] is an open-source, lightweight management tool that provides a comprehensive graphical interface to manage Docker environments, including Docker Swarm. It simplifies the operations associated with Docker container management by offering an intuitive dashboard that facilitates easy deployment, monitoring, and maintenance of containers, images, networks, and volumes across cluster nodes.

Portainer was chosen for this design to simplify the deployment, monitoring, and maintenance of containers, images, networks, and volumes within the Docker Swarm cluster. Portainer enhances accessibility and operational efficiency.

Graylog (Log Management and Policy Violation Monitoring)

The Graylog stack is composed of four main services, each integral to the functionality of the system that helps in the collection, processing, management, and monitoring of OPA decision logs.

1. **Graylog Server**: The core component of the stack, the Graylog server, processes and manages log messages received from various sources. It provides a user interface for log querying, analysis, and visualization. The server also handles user management, processing pipelines, alerting, and reporting.

2. **MongoDB:** MongoDB serves as the database backend for Graylog. It stores the configuration and meta information such as users, roles, stream rules, dashboards, and alert settings.

3. **OpenSearch:** OpenSearch is used for indexing and searching the log data. This

service enables Graylog to perform rapid search operations across large volumes of logs, facilitating quick retrieval of relevant log entries based on user queries.

4. **Logspout:** Logspout is a lightweight, minimalistic container log router for Docker environments that streams Docker container logs to a centralized log endpoint (Graylog). As a log management tool, Logspout is designed to capture all Docker logs without requiring any modifications to the Docker containers themselves. This makes it an ideal choice for environments where modifying running containers is not feasible or desired.

In our design, Logspout plays a crucial role as the log shipper, specifically configured to monitor and forward logs from Docker containers. Its primary function is to aggregate and ship logs generated by the OPA (Open Policy Agent) service within the OPA stack. This selective logging is achieved by configuring Logspout to filter and transmit only those logs that originate from containers associated with the OPA service.

This targeted log collection is facilitated through Logspout's ability to dynamically recognize and connect to the Docker daemon, pulling logs directly from Docker's logging driver and forwarding them to the Graylog stack. This configuration ensures that we capture and analyze runtime data and decision logs from OPA, without overwhelming our log management system with extraneous data.

4.3.1.3 Authorization Service

Open Policy Agent (OPA) is the policy decision point that decouples policy enforcement from the application logic. It evaluates incoming requests against predefined policies and generates decisions in real-time. OPA Decision Logs are captured and forwarded to Graylog via Logspout.

4.3.1.4 Identification of Minimum Resource Requirements

Given the absence of specific official resource requirements for Graylog and Open Policy Agent (OPA), we synthesized recommendations from community feedback and user experiences [58, 59]. Although not official, these insights offer practical guidelines for setting up the environment. The community-derived specifications, elaborated below, provide a practical approach to hardware allocation, ensuring the environment can manage the expected load and data volume.

CPU: Users suggest at least 4 cores to handle the computational load of Graylog and Docker Swarm.

RAM: For optimal performance, community feedback indicates a minimum of 16 GB of RAM, combining the needs for log management and container operations.

Storage: Practical use cases recommend 100 GB of SSD storage for managing log data and application requirements effectively.

The summary of collected resource requirements is given in Table 4.2. It's important to recognize that hardware sizing for Graylog is highly dependent on factors such as the volume and rate of log ingestion, which can vary significantly across different environments. Consequently, the provided estimations should be regarded as approximate guidelines for a minimal setup. These recommendations are based on practical user experiences and feedback rather than definitive specifications, highlighting the need for adjustments according to specific operational demands and log management requirements.

Table 4.2: Minimum Resource Requirements Based on Community Feedback

Resource	Community Specification
CPU	4 cores
RAM	16 GB
Storage	100 GB SSD
Network	1 Gbps Ethernet

4.3.1.5 Identification of Secure Versions for Primary Components

Ensuring the security of the software components within our architecture is paramount, especially considering the potential for known vulnerabilities that could compromise the system. Each primary component is assessed by providing a summary of its vulnerability history and the latest secure version for deployment is recommended.

Graylog has encountered several vulnerabilities over time.

- **CVE Summary**: Graylog has a total of 11 CVEs reported [60], including issues such as remote code execution and privilege escalation.
- **Secure Version**: The latest secure version, *Graylog 6.0.3*, has no known CVEs. This version includes significant security enhancements and fixes for prior vulnerabilities, making it the optimal choice for our setup.

Open Policy Agent (OPA) is a crucial component for policy management and enforcement, with a history of some vulnerabilities.

- **CVE Summary**: OPA has 5 reported CVEs [61], primarily involving denial of service and logic bypass vulnerabilities.
- **Secure Version**: The most recent version, *OPA 0.65.0*, is free of known CVEs. It includes several security improvements and fixes, ensuring robust security for policy management.

Docker Engine is a key tool for container orchestration, which has faced numerous security challenges.

- **CVE Summary**: Docker Engine has 37 reported CVEs [62], including arbitrary code execution and container escape issues.
- **Secure Version**: The latest version, *Docker Engine 25.0.5*, has no published CVEs. This version includes security patches and updates, making it the safest choice for our architecture.

Portainer is a management tool for Docker environments, which has also experienced security vulnerabilities.

- **CVE Summary**: Portainer has 16 reported CVEs [63], including issues related to incorrect access control and arbitrary code execution.

- **Secure Version**: The most recent version, *Portainer 2.19.5*, has no known CVEs. It incorporates security improvements and patches, ensuring a secure management interface for Docker environments.

Traefik Traefik is a dynamic reverse proxy and load balancer that has encountered some security issues.

- **CVE Summary**: Traefik has 16 reported CVEs [64], including vulnerabilities related to security misconfigurations and denial of service issues.

- **Secure Version**: The latest version, *Traefik 3.0.2*, has no published CVEs. This version is fortified with security patches and enhancements, making it a secure choice for our architecture.

The summary of the identified secure versions of the environment architecture components is given in Table 4.3. By selecting these versions, we ensure that our architecture is safeguarded against known vulnerabilities, providing a secure foundation for our system operations. These choices reflect the most current and secure software options available, reducing the risk of potential security breaches.

Table 4.3: Summary of Secure Versions for Primary Components

Component	**Latest Secure Version**
Graylog	Graylog 6.0.3
OPA	OPA 0.65.0
Docker Engine	Docker 25.0.5
Portainer	Portainer 2.19.5
Traefik	Traefik 3.0.2

4.3.2 Identification of Essential Data to Detect Violations

By analyzing several OPA decision logs from the different use case applications given in Section 4.2.2, and referencing OPA Decision Logs in Section 4.2.3, the data fields given Table 4.4 are identified as the most essential to detect violations:

Table 4.4: Essential Data from OPA Decision Log

Field Name	Description	Usage
`type`	The type of log entry	To determine this is a decision log from OPA
`result`	Boolean value	To identify policy violation in the case of a value of false
`path`	The specific OPA API endpoint invoked	To understand what policy check was performed
`metrics`	Performance metrics related to the policy decision-making process	To help in performance monitoring and debugging
`input`	Dictionary of input items used in the policy decision-making process	To understand the application behavior and policy query details
`time` `timestamp`	The time when the decision was made	To audit and track when the policy violation occurred

4.3.3 Use Case Applications Design

This section details the design of three baseline applications, each tailored to a specific OPA use case. These applications simulate real-world scenarios to assess the practicality and impact of policy compliance within the architecture. For each application, a specific Traffic Generator is developed to generate valid policy queries which creates a baseline for validation purposes.

4.3.3.1 App1 - HTTP-API

Design Overview The HTTP API application demonstrates policy enforcement using OPA within a controlled environment. The application operates by exposing an HTTP API that accepts GET requests to simulate real-world interactions requiring authorization for data access.

Policy Implementation Design A Rego policy file specifies the authorization logic, allowing individuals to access their own salary information, managers to access the salaries of their direct reports, and HR members to access any employee's salary.

Traffic Generation A traffic generator simulates HTTP requests to the API at randomized intervals, generating decision logs for visualization in Graylog.

4.3.3.2 App2 - SSH and Sudo Authorization

Design Overview This application leverages OPA to enforce fine-grained, host-level access controls for SSH and sudo operations. It integrates OPA with Linux-PAM to dynamically evaluate and enforce access policies based on user roles and specific host conditions.

Policy Implementation Design The policy framework differentiates access levels between administrative and developer roles. Three distinct Rego policies manage file loading, SSH access, and sudo access.

Traffic Generation A traffic generator systematically initiates SSH sessions and executes sudo commands at randomized intervals, generating decision logs for visualization in Graylog.

4.3.3.3 App3 - Event-Driven MSA

Design Overview Based on the *"python-kafka-microservices"* project, this application utilizes OPA to enforce access policies across Kafka resources. It demonstrates the potential for visualizing policy invocations in Graylog.

Policy Implementation Design A simplified Rego policy denies operations if the client ID is "misuser," focusing on monitoring authorization requests from Apache Kafka.

Traffic Generation A traffic generator creates sessions with the dashboard application, placing orders every five seconds to generate decision logs for visualization in Graylog.

4.3.4 Validation Design

This section outlines the design of the validation stage to test the robustness and effectiveness of the OPA and Graylog integration in the identification of policy violations and taking actions according to the defined alerting mechanisms.

4.3.4.1 Crafting Validation Applications

For each of the Use Case Applications, which have different policy implementations, the objective is to craft applications that allow us to generate policy-violating requests based on the application architecture and its specific policies. The validation applications play a crucial role in ensuring the robustness and effectiveness of the policy compliance monitoring system. These applications are designed to generate policy-violating traffic on demand, simulating a range of scenarios that mimic real-world violations.

To achieve comprehensive validation, we have developed a suite of applications coded in Python. Each application is crafted to replicate different use cases that might occur in a typical operational environment. These applications provide a controlled setting where various policy violations can be triggered intentionally, allowing us to observe how the monitoring and alerting mechanisms respond. The applications can be used to represent *misuse* of use case applications, acting as a *threat actor* to break authorization to access a service or an artifact.

Moreover, these validation applications are designed to be repeatable and scalable. They can be easily modified to include new scenarios as policies evolve. This adaptability ensures that our monitoring system remains relevant and effective over time, continuously improving its ability to safeguard against policy breaches.

In summary, the development of these validation applications is a critical step in enhancing the reliability of our policy compliance monitoring framework. By providing a reliable method to test and validate the system's response to policy violations, we ensure that our monitoring and alerting mechanisms are robust, accurate, and capable of maintaining high standards of security and compliance.

4.3.4.2 Methodology for Validation

The methodology for validation involves a systematic approach to ensure the robustness and effectiveness of the policy compliance monitoring system integrated with the Open Policy Agent (OPA) and Graylog. The outline of the detailed process for validating the effectiveness of the policy compliance monitoring system methodology is illustrated in Figure 4.6 and elaborated below:

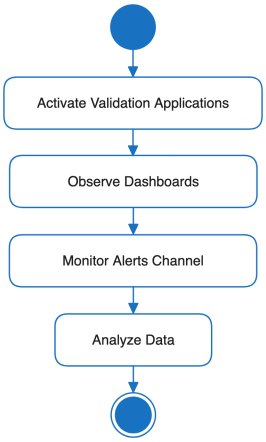

Figure 4.6: Validation Methodology

1. **Activate Validation Application**: For each application under study, the validation application will be activated. This application will be designed to simulate policy violations by generating 10 non-compliant requests each time it is run.

2. **Observe Dashboards**: Once the policy violation generator is activated, the corresponding dashboards in Graylog will be monitored. These dashboards will be configured to display key metrics such as total decisions, policy violations, and real-time alerts.

3. **Monitor Alerts Channel**: In addition to the dashboards, alert channels configured in Graylog will be closely observed. These channels will be set to notify stakeholders of significant policy violations.

4. **Analyze Data**: The data collected from the dashboards and alert channels will be analyzed to evaluate the accuracy and efficiency of the policy compliance monitoring setup. Specific metrics, such as the number of detected policy violations and the latency of alerts, will be examined to conclude the system's

performance.

4.4 Summary

Chapter 4 details the comprehensive methodology employed to enhance policy compliance monitoring by integrating Graylog with the Open Policy Agent (OPA). The methodology is systematically divided into four key phases: Analysis, Design, Implementation, and Demonstration. In the Analysis phase, the chapter delves into understanding the architectures of OPA and Graylog, examining OPA decision logs, and identifying significant use cases where enhanced logging and monitoring can provide substantial benefits. The Design phase focuses on architecting a robust system that leverages both OPA and Graylog, detailing the collection and analysis of decision logs, identifying essential data for detecting violations, and planning the validation approach.

CHAPTER 5

IMPLEMENTATION AND DEPLOYMENT

This chapter dives into the detailed design and implementation processes for the thesis work. The aim is to construct a robust, scalable environment to facilitate policy compliance monitoring across various applications based on the design elaborated in Section 4.3. The implementation is divided into four main sections: Environment Setup, Implementation of Use Case Applications, Validation Setup, and Graylog Configuration.

Each section builds upon the infrastructure established in the preceding sections, to achieve a systematic development flow that is both logical and efficient.

Section 5.1: Environment Setup involves the foundational setup of our Docker Swarm cluster, integrated with critical tools such as Traefik for load balancing, Portainer for cluster management, and the essential installations of the OPA and Graylog stacks. This setup forms the backbone of our validation environment, providing the necessary groundwork for deploying our applications and policies.

Section 5.2: Implementation of Use Cases Applications focuses on the creation and deployment of three applications tailored to OPA use cases. This stage simulates real-world application scenarios to assess the practicality and impact of policy compliance within our architecture.

Section 5.3: Validation Setup focuses on validating the integration of Open Policy Agent (OPA) and Graylog, assessing the usefulness of the created dashboards, and evaluating the effectiveness of the event definitions.

Section 5.4: Graylog Configuration is dedicated to fine-tuning the Graylog system to maximize its potential for capturing, analyzing, and visualizing policy-related events and metrics where policy monitoring will takes place. This final stage aims to transform raw data into actionable insights, thereby enabling precise monitoring and enhanced decision-making regarding policy compliance and system security.

5.1 Environment Setup Implementation

Five main steps to build the execution environment is illustrated in Figure 5.1 and detailed descriptions of each are provided in the subsequent subsections.

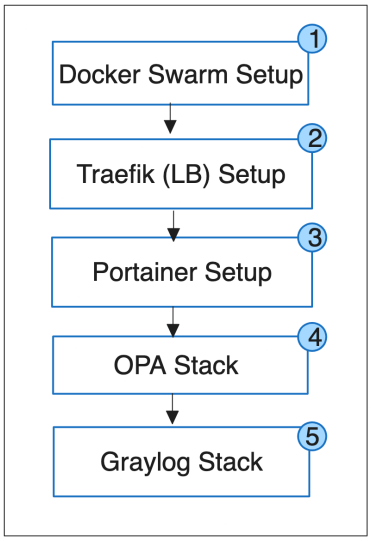

Figure 5.1: Environment Setup Steps

5.1.1 Setting up Docker Swarm Cluster

This section outlines the high-level process of establishing the Docker Swarm cluster consisting of one manager node and two worker nodes, which serves as the foundation for deploying and managing the containerized environments in our design.

Overview of Swarm Initialization The initialization of the Docker Swarm mode transforms individual Docker engines into a multi-host clustering system. The manager node is configured to orchestrate and manage the state of the cluster, which

involves scheduling container deployments and maintaining the desired state as specified in service definitions.

Node Configuration Following the initialization, the worker nodes are incorporated into the cluster using a secure token generated by the manager node. This token-based system ensures secure communication and authorization as nodes join the cluster, maintaining the integrity of the managerial operations. The manager node oversees the cluster's activities and distributes tasks to the worker nodes, leveraging Docker's built-in orchestration capabilities.

Network Configuration An essential component of setting up a Docker Swarm cluster is configuring the network overlays. These overlays enable containers distributed across multiple Docker hosts to communicate as if they were on the same host, thereby providing the necessary network infrastructure to support high availability and scalability across the cluster.

The architecture of the Docker Swarm cluster architecture is illustrated in Figure 5.2 providing the interactions between a *Manager node* and two *Worker nodes*.

Manager Node is central to the Docker Swarm and handles the orchestration and management of the Swarm. It includes:

- Docker Daemon: The Docker Daemon listens for Docker API requests and manages various Docker objects, its functionalities include: API, Orchestrator, Allocator, Scheduler, and Dispatcher functions [65].
- Service Discovery: Service discovery is the process employed by Docker to direct a request from the external clients of your service to a specific swarm node, without the need for clients to be aware of the number of nodes involved in the service or their specific IP addresses or ports.
- Manager nodes implement the Raft Consensus Algorithm [66] to manage the global cluster state.
- Containers: The manager node can also host containers akin to a worker node.
- Network ports: The manager node communicates on specific network ports for various functions [67]:

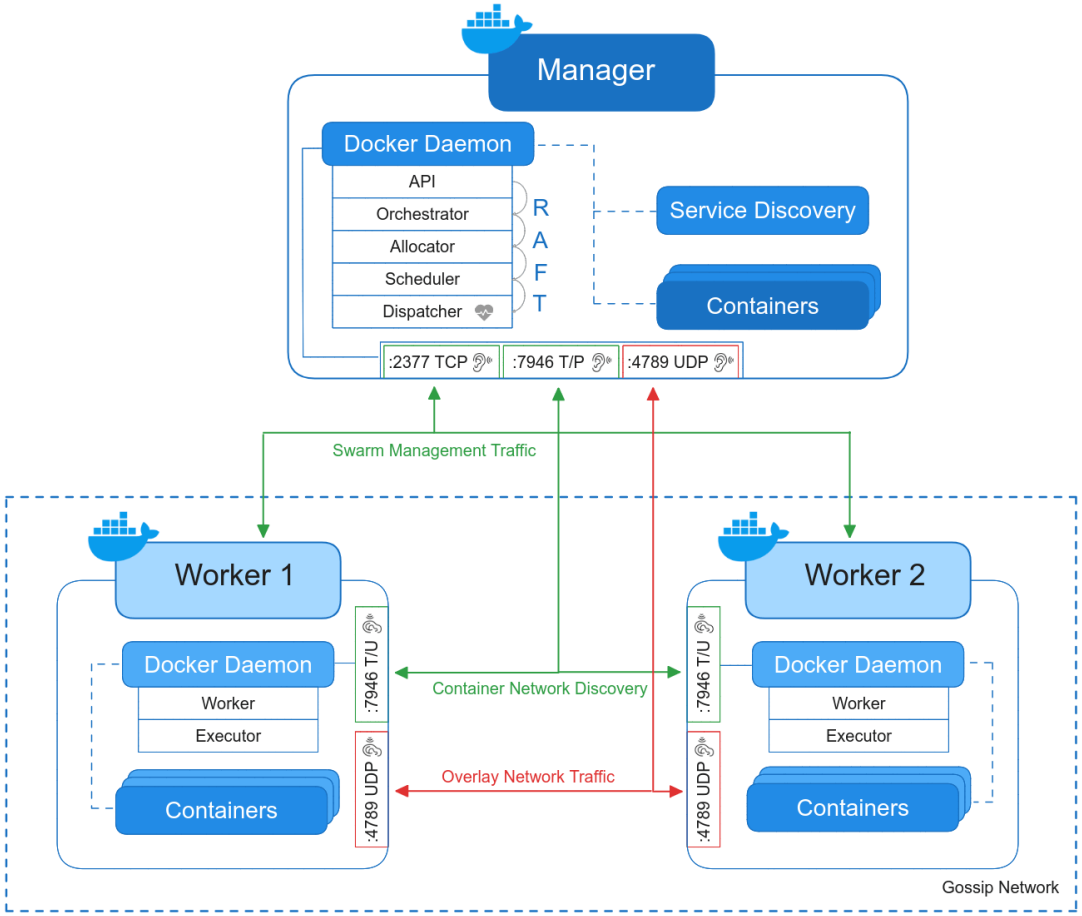

Figure 5.2: Docker Swarm Architecture

- Port 2377 (TCP) is used for Swarm management communications.
- Port 7946 (TCP/UDP) is used for overlay network node discovery
- Port 4789 (UDP) is utilized for overlay network traffic

Worker Nodes (Worker 1 and Worker 2) run a Docker Daemon configured for handling tasks assigned by the Manager, including a Worker process and an Executor. Each node hosts multiple containers that are managed by Docker.

The communication across nodes involves the same ports as the *Manager* for seamless integration and communication in the Swarm. The overlay network traffic occurs through Port 4789 (UDP), facilitating communication between containers across different nodes. Port 7946 (TCP/UDP) enables container network discovery and node communication.

Network Configuration includes two main types of network traffic [34]:

1. Control and management plane traffic includes (i) *Swarm Management Traffic* managed by TCP connections, primarily for control and configuration data exchanged between the Manager and Worker nodes, and (ii) *Gossip Network* underpining the cluster-wide peer-to-peer network overlay, managing container network state.

2. Application data plane traffic includes an *Overlay Network Traffic* which utilizes UDP for the transport of data packets across the container network, facilitating communication between Docker containers on different nodes and traffic to and from external clients.

5.1.2 Integration of Traefik

Firstly, we will deploy Traefik on our Docker Swarm cluster utilizing a docker-compose file. This step involves configuring Traefik as a reverse proxy and load balancer directly within the Swarm environment, streamlining the process of routing external requests to the appropriate services.

As we progress, we will utilize *"Docker object labels"* [68] to specify which services within the Swarm are to be exposed to the public. These labels inform Traefik about the services that require ingress traffic configuration, thereby enabling Traefik to automatically configure and manage ingress routes for exposed Docker Swarm services. The *docker-compose files* are provided in Appendix A.

5.1.3 Deploying Portainer for Cluster Management

By integrating Portainer into our environment, we enable more efficient management of the cluster's resources, streamline the deployment workflows, and improve the accessibility of cluster management and cluster statistics for more informed decision-making. The Portainer stack comprises two main components: the Portainer Agent and the Portainer Server.

The Portainer Agent should be deployed on every node (on both the managers and the workers) and communicate with the local Docker daemon. This communication

is facilitated through a Docker socket, which is exposed as a *"volume"*. The second component, the Portainer Server, acts as the control plane and provides the user interface. This interface allows users to set up, manage, and interact with the underlying container orchestration platforms.

The Portainer Server service is configured to automatically discover all the Portainer Agents in the cluster via Docker's service discovery mechanisms and communicates with them using TCP port 9001. Additionally, we have configured Traefik-related labels on the Portainer Server service to publicly expose it using a specified Fully-Qualified Domain Name (FQDN). Traefik is also set up to handle SSL certificate issuance and renewals for the FQDN domain, ensuring secure communications.

5.1.4 Installing and Configuring the OPA Stack

Docker images for Open Policy Agent are available on DockerHub[1], we used the latest image tag to construct our local OPA cluster. Listing 5.1 shows an excerpt of the docker-compose file we used for the implementation.

```
...
    command:
      - "run"
      - "--server"
      - "--log-format=json"
      - "--set=decision_logs.console=true"
      - "--set=services.nginx.url=http://bundle_server"
      - "--set=bundles.nginx.service=nginx"
      - "--set=bundles.nginx.resource=bundle.tar.gz"
      - "--set=bundles.nginx.polling.min_delay_seconds=10"
      - "--set=bundles.nginx.polling.max_delay_seconds=30"
...
networks:
  opa:
    external: true
...
```

Listing 5.1: OPA Docker Compose File Excerpt

[1] https://hub.docker.com/r/openpolicyagent/opa/

Our OPA stack consists of two main services: the OPA service (line 14) and the *"bundle-server"* (line 7). The OPA service operates with three replicated instances, ensuring high availability and load distribution. The bundle-server, implemented using a straightforward *Nginx* web server (line 8), serves an important role by hosting the policy bundle—a compressed file containing all our policies (line 9). This architecture is designed to simplify updates to the policy bundle as we integrate and evolve our baseline applications. Additionally, an external volume attached to the bundle-server facilitates easy uploading and distribution of new policy bundle versions through the Portainer UI, ensuring that all OPA servers have access to the latest policies. Key configurations with critical parameters from in Listing 5.1 is listed in Table 5.1.

Table 5.1: Key Configurations in OPA Docker Compose File

Configuration	line #	Explanation
OPA Server	5	Sets the logging format to JSON, which will later facilitate the parsing of these logs in Graylog
	6	Ensures that decision logs are output to the console.
	7-11	Enables OPA to periodically fetch the policy bundle from the Nginx server, ensuring that the latest policies are always applied.
OPA Network	14-	Designates opa as an external network and allows future baseline application stacks to join this network, promoting seamless interaction with the OPA service

5.1.5 Graylog Stack Implementation

The deployment of the Graylog stack involves a multi-service setup orchestrated using Docker Compose. The deployment leverages MongoDB, OpenSearch, and Graylog as core components.

The **MongoDB** service is configured to store Graylog metadata and configuration data. It uses the MongoDB 6.0.14 image. The MongoDB data is persisted in a Docker volume named `mongo_data`, ensuring data durability across service restarts.

OpenSearch is deployed as the primary data storage and search engine for log data. It

uses the OpenSearch 2.12.0 image, configured with Java options to manage memory allocation effectively. The service is designed to operate as a single-node cluster with security features like SSL disabled for simplicity. Its data is stored in the `es_data` volume. It is also configured to restart on failure, ensuring high availability.

The **Graylog** service acts as the central log management system. It is set up using the Graylog 6.0 image and is configured to bind its web interface and REST API to port 9000. Graylog connects to MongoDB for metadata and OpenSearch for log data storage. The service includes configurations for secure access and integration with Traefik for load balancing and routing. The configuration and journal data are stored in `graylog_config` and `graylog_journal` volumes, respectively. Graylog's entrypoint script ensures that it waits for OpenSearch to be fully operational before starting, and it is configured to restart automatically to maintain service continuity.

Additionally, **Logspout** is deployed globally to capture Docker container logs and forward them to Graylog. This service ensures that logs from all containers are collected and sent to Graylog for processing and analysis. Logspout is configured to restart on failure and has resource limits set to ensure it runs efficiently without overloading the system.

The deployment utilizes Docker's overlay network for Graylog components, ensuring seamless inter-service communication, and integrates with the external Traefik network for public-facing services. This setup ensures a scalable, resilient, and comprehensive logging solution capable of handling diverse operational scenarios and maintaining high standards of log management and analysis.

5.2 Implementation of Use Cases Applications

This section is dedicated to the development and deployment of three distinct applications designed to ensure that the forthcoming configurations of Graylog inputs, dashboards, extractors, and alert rules, are sufficiently comprehensive to encompass a wide range of OPA use cases. Each application is crafted to address a unique OPA use case, mirroring real-world operational scenarios. This is intended to guarantee that our Graylog implementation can effectively capture, analyze, and display the diverse data

interactions and policy enforcement activities generated across various scenarios.

5.2.1 HTTP API Application

The HTTP API Application serves as a practical demonstration of policy enforcement using OPA within a controlled environment. This application operates by exposing an HTTP API that accepts GET requests, which simulate real-world interactions with a web service that requires authorization for data access. In our setup, the API simulates a scenario where users request their salary information, and OPA evaluates whether the request should be allowed based on predefined policies. The implemented HTTP API Application is fundamentally based on the methodologies and Docker Images outlined in the official Open Policy Agent documentation [69]. The application architecture and its integration with OPA is illustrated in Figure 5.3.

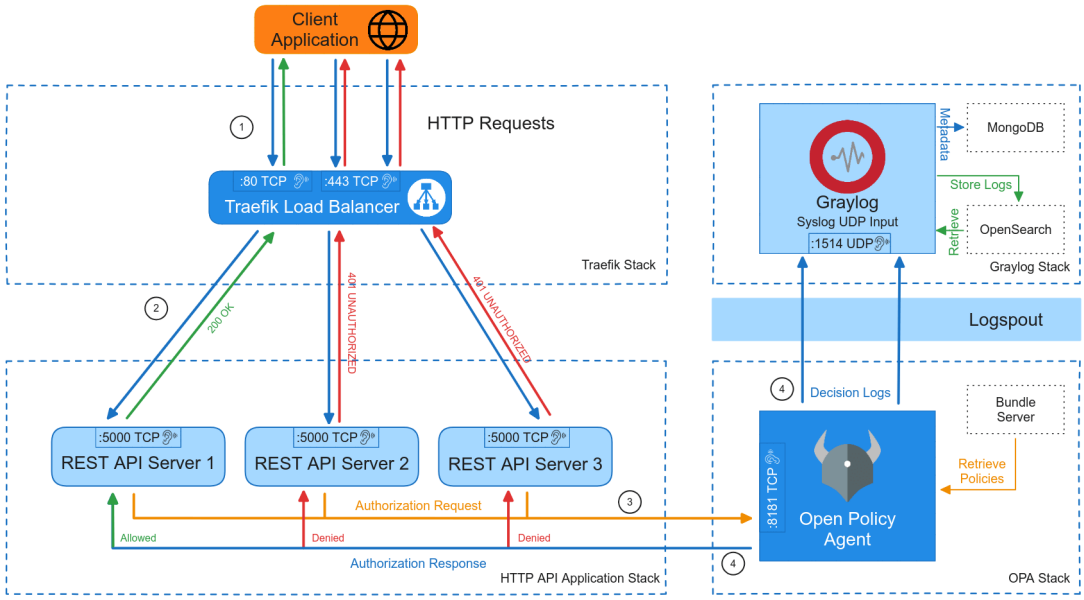

Figure 5.3: HTTP API Application Architecture

The HTTP API Application is designed using a Python Flask framework to demonstrate how Open Policy Agent (OPA) can be integrated to enforce access control policies in a RESTful service environment. This application exemplifies a realistic use case where:

- Individuals are authorized to view their own salary details, for instance, accessing `GET /finance/salary/{user}` is permitted for the user specified in the

`{user}` parameter.

- Managers are granted permission to view the salaries of their direct reports, meaning a request to `GET /finance/salary/{user}` by a user's manager is allowed.
- Members of the HR department are granted permission to view the salary of any employee.

Application Workflow

Each time an HTTP request is received, the REST API server consults OPA to determine if the request should be authorized. This interaction is achieved through a RESTful API call to OPA with the necessary request details. Listing 5.2 provides a simplified representation of how the server processes an incoming request.

```
# Extract the user from the form data.
http_api_user = request.form['user']

# Convert the path into a list format by removing slashes and splitting.
http_api_path_list = request.path.strip("/").split("/")

# Construct the input dictionary for OPA.
input_dict = {
    "input": {
        "user": http_api_user,
        "path": http_api_path_list,  # Example: ["finance", "salary", "alice"]
        "method": request.method  # HTTP method (GET, POST, etc.)
    }
}

# Make a POST request to OPA for an authorization decision.
response = requests.post("http://opa:8181/v1/data/httpapi/authz", json=input_dict)

# Process the response from OPA.
if response.json()["allow"]:
    # Permission granted
```

```
22 else:
23     # Permission denied
```

Listing 5.2: HTTP API Application Excerpt

Policy Implementation

A policy file written in Rego (OPA's native query language) specifies the authorization logic of HTTP API application. The policy in Listing 5.3 includes rules that allow individuals to access their own salary information (line 10-13), managers to access the salaries of their direct reports (line 16-21), and members of the HR department are allowed to access any employee's salary (line 24-28). The *subordinates* where hierarchical relationships (line 3) and *hr* list are defined at the beginning.

```
1  package httpapi.authz
2  # Define relationships, e.g., bob is alice's manager.
3  subordinates := {"alice": [], "charlie": [], "bob": ["alice"], "betty":
       ["charlie"]}
4  hr = ["david",]
5
6  # Default to denying access.
7  default allow := false
8
9  # Rule to allow users to access their own salary information.
10 allow if {
11   input.method == "GET"
12   input.path == ["finance", "salary", input.user]
13 }
14
15 # Rule to allow managers to access their subordinates' salary information
       .
16 allow if {
17   some username
18   input.method == "GET"
19   input.path = ["finance", "salary", username]
20   subordinates[input.user][_] == username
21 }
22
23 # Allow HR members to get anyone's salary.
```

```
allow {
  input.method == "GET"
  input.path = ["finance", "salary", _]
  input.user == hr[_]
}
```

Listing 5.3: HTTP API Application Rego Policy

Traffic Generation

To generate decision logs for visualization in Graylog, a traffic generator simulates HTTP requests to the API at randomized intervals, ranging between 1 and 10 seconds, using scenarios outlined in Table 5.2. This helps in visualizing the practical implementation of OPA decision logs under varied conditions in Graylog in the subsequent stage of this study.

Table 5.2: HTTP API Traffic Generator Cases

Case	Description	Authorized?
1	A user requesting his own salary	Authorized
2	A manager request to get his subordinates salaries	Authorized
3	An HR employee requests to get the salary of any employee	Authorized

5.2.2 SSH and Sudo Authorization Application

The SSH and Sudo Authorization Application leverages OPA to enforce fine-grained, host-level access controls for SSH and sudo operations. This application acts as a practical demonstration of integrating policy-based access controls into SSH and sudo sessions using OPA, coupled with Linux-PAM (Pluggable Authentication Modules). The implemented SSH and Sudo Authorization Application is fundamentally based on the methodologies and Docker images outlined in the official Open Policy Agent documentation [70]. The application architecture and its integration with OPA is illustrated in Figure 5.4.

The SSH and Sudo Application is based on a custom Docker image [2], containing

[2] https://hub.docker.com/r/openpolicyagent/demo-pam

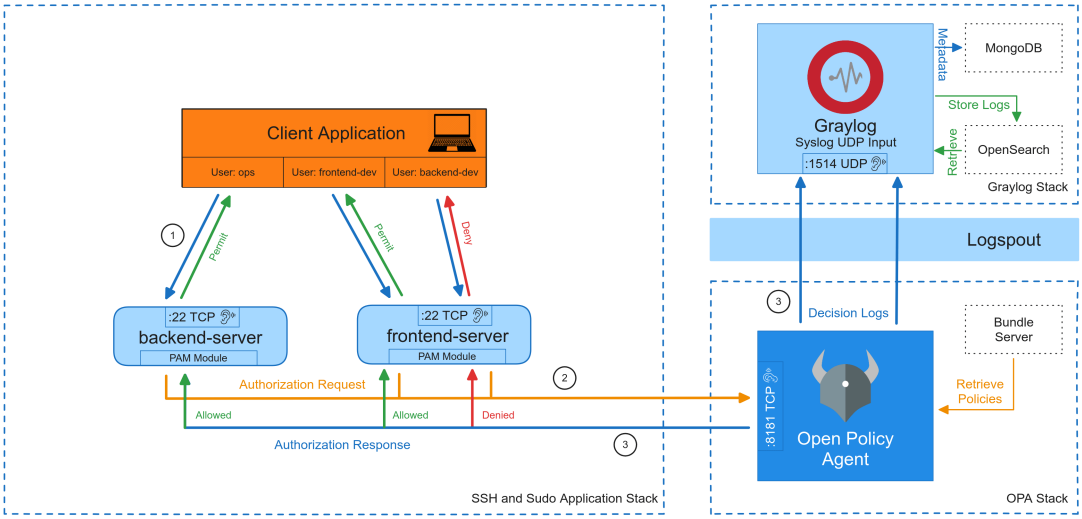

Figure 5.4: SSH and Sudo Authorization Application Architecture

pre-created Linux accounts for the users. Additionally, the required PAM module is pre-configured inside the sudo and sshd files in `/etc/pam.d`.

Application Workflow

Open Policy Agent (OPA) and Linux-PAM can be integrated to implement host-level access controls for SSH and sudo commands within a controlled environment. Linux-PAM is configured to outsource authorization decisions to a specialized plugin, in this case, an OPA-based plugin, which evaluates access permissions based on predefined policies.

In this setup, OPA is utilized to dynamically evaluate and enforce access policies based on user roles and specific host conditions. For example, the system allows administrative users to SSH into any host and execute sudo commands while restricting normal users to only those hosts to which they have explicitly contributed.

This is accomplished by interfacing OPA with the Linux-PAM system using the *"OPA-based Linux-PAM"* [3] plugin, where OPA's decisions dictate the authorization of SSH and sudo requests based on predefined policies.

While this setup focuses on authorization, it is worth noting that authentication (the verification of user identities) is not managed by OPA. In practical implementations,

[3] `https://github.com/open-policy-agent/contrib/tree/main/pam_opa`

authentication might be handled directly by SSH or via external identity management systems. Table 5.3 lists configured hosts and created users along with their explanations.

Table 5.3: Configured hosts and Created Users for SSH and Sudo Application

Type	Name	Explanation
Hosts	`backend-server`	A server configured to use OPA Plugin with Linux-PAM to offload authorization decisions for SSH and Sudo
	`frontend-server`	A server configured to use OPA Plugin with Linux-PAM to offload authorization decisions for SSH and Sudo
Users	`frontend-dev`	A developer who contributed to apps running on the frontend host
	`backend-dev`	A developer who contributed to apps running on the backend host
	`ops`	An administrator with access across both hosts

Policy Implementation

The policy framework within the SSH and Sudo Authorization Application is designed to differentiate access levels between administrative and developer roles, addressing their distinct responsibilities and the overarching security requirements of the system. The objectives and specific implementations of the administrative and developer access policies are detailed below.

- **Administrative Access**: Administrative users are permitted to SSH into any host and execute sudo commands, reflecting their high-level oversight and system maintenance responsibilities.
- **Developer Access**: Developer access is more restricted, allowing SSH and sudo actions only on specific hosts where they have directly contributed. This targeted access control aligns with their specific project roles and responsibilities, ensuring that developers have the necessary tools to be productive without compromising the system's security.

To support these objectives, three distinct Rego policies are used which are explained below. All three policies are bundled and provided to the Open Policy Agent (OPA) via the bundle server.

1. pull.rego

This policy, implemented in Listing 5.4, is responsible for informing PAM on what files and environment variables need to be dynamically loaded into the policy evaluation context to inform subsequent authorization decisions. Specifically, it directs the loading of the file `/etc/host_identity.json`, which contains critical host identity data (line 4).

```
1 package pull
2
3 # Define which files should be loaded into the context
4 files := ["/etc/host_identity.json"]
5
6 # Define which environment variables should be loaded into the context
7 env_vars := []
```

Listing 5.4: PAM Configuration Rego Policy

2. sshd_authz.rego

This policy, implemented in Listing 5.5, manages SSH access by distinguishing between administrative access and developer-specific access, this is done based on their contributions as recorded in `/etc/host_identity.json` (line 18) which is loaded through the `pull.rego` policy. It grants SSH access if the user name has ”admin” role (lines 12-14), allows SSH access for developers only if they have contributed to the specific host (lines 17-19), and generates an error message ”Request denied by administrative policy” if the `allow` condition is not met, meaning access is denied (lines 22-24).

```
1 package sshd.authz
2
3 import rego.v1
4 import input.pull_responses
5 import input.sysinfo
6 import data.hosts
7
```

```
# Set default authorization to false
default allow := false

# Allow administrators unrestricted SSH access
allow if {
    data.roles.admin[_] == input.sysinfo.pam_username
}

# Allow developers SSH access only to hosts they have contributed to
allow if {
  hosts[pull_responses.files["/etc/host_identity.json"].host_id].
    contributors[_] == sysinfo.pam_username
}

# Include an error message if the access is not authorized
errors contains "Request denied by administrative policy" if {
    not allow
}
```

Listing 5.5: SSH Rego Policy

3. sudo_authz.rego

Similar to the SSH authorization policy, implemented in Listing 5.6, this policy governs sudo access, granting privileges primarily to users with an ”admin” role (lines 9-11), thereby ensuring that administrative commands are safeguarded and only accessible to authorized personnel.

```
package sudo.authz

import rego.v1

# Set default authorization to false
default allow := false

# Allow administrative users to execute sudo commands
allow if {
    data.roles.admin[_] == input.sysinfo.pam_username
}

# Include an error message if the access is not authorized
```

```
errors contains "Request denied by administrative policy" if {
    not allow
}
```

Listing 5.6: Sudo Rego Policy

Traffic Generation

To generate decision logs for visualization in Graylog, we employ a traffic generator developed in Python. The traffic generator systematically initiates SSH sessions to the *"backend-server"* and *"frontend-server"* at randomized intervals, ranging between 1 and 10 seconds, and optionally executes a series of sudo commands using scenarios outlined in Table 5.4. This helps in visualizing the practical implementation of OPA decision logs under varied conditions in Graylog in the subsequent stage of this study.

Table 5.4: HTTP API Traffic Generator Cases

Case	Description	Authorized?
1	Admin user (ops) logging to frontend-server	Authorized
2	Admin user (ops) logging to backend-server	Authorized
3	Non-admin user (frontend-dev) logging to frontend-server	Authorized
4	Non-admin user (backend-dev) logging to backend-server	Authorized
5	Admin user (ops) executing sudo command on frontend-server	Authorized
6	Admin user (ops) executing sudo command on backend-server	Authorized

5.2.3 Event-driven Microservice Architecture Overview

The Event-driven Microservice Architecture implemented is based upon the *"python-kafka-microservices"* open-source project [4]. Additionally, it incorporates the *"Open Policy Agent plugin for Kafka authorization"* [5] plugin.The application architecture and its integration with OPA is illustrated in Figure 5.5.

OPA is employed to enforce access policies across various Kafka resource types. Despite its complexity in setup, this application operates under a relatively straightfor-

[4] https://github.com/ifnesi/python-kafka-microservices
[5] https://github.com/StyraInc/opa-kafka-plugin

ward policy where OPA predominantly permits all operations. This configuration, while seemingly simplistic, effectively demonstrates the potential of visualizing policy invocations in Graylog, which is explored in further detail in the subsequent stage of this study.

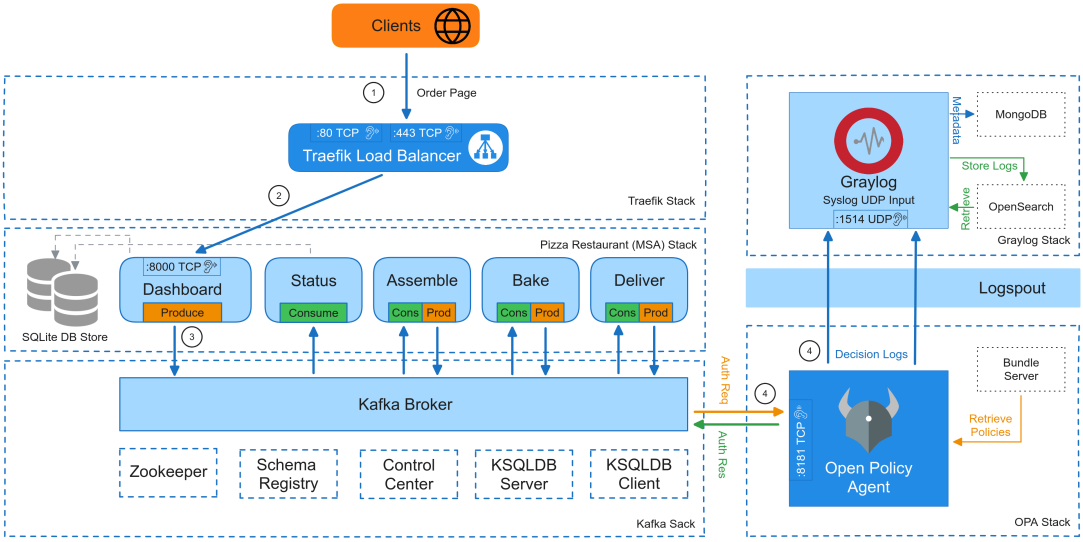

Figure 5.5: Event-Driven Microservice Application Architecture

Application Workflow

"python-kafka-microservices" is utilized which is an exemplary implementation of the Command and Query Responsibility Segregation (CQRS) and Event Sourcing patterns [71]. This architecture is demonstrated through a conceptual model of a pizza takeaway shop, making the complex principles more relatable and easier to understand. The workflow begins with a Flask-based web application (Dashboard) that serves as the Command interface in the CQRS pattern. Here, users can log in, customize their pizza, place orders, and monitor the status of their orders. The web application interacts with a SQLite3 state store used as a materialized view, which bridges the Command and Query components of the CQRS pattern. While SQLite3 is used here for simplicity and demonstration purposes, more robust scenarios might employ in-memory data stores or specialized processing like ksqlDB or Flink.

Microservices Pipeline Once a pizza order is placed through the web application, it progresses through a series of dedicated microservices, each reflecting a stage

in the pizza preparation and delivery process as follows.

1. **Assembly**: The `msvc\_assemble.py` service takes the order details and assembles the pizza accordingly.
2. **Baking**: The `msvc_bake.py` service is responsible for baking the assembled pizza.
3. **Delivery**: The `msvc\_delivery.py` service manages the logistics of delivering the pizza to the customer.
4. **Status Processing**: The `msvc\_status.py` service acts as the Query component in the CQRS architecture. This service updates the web application with the status of the order as each of the previous stages completes its task.

Data and Communication Flow All communications between these microservices are facilitated through an Apache Kafka cluster, which is configured with the "OPA Authorizer Plugin" [6]. This plugin intercepts all Kafka message requests and forwards them to the OPA service for authorization decisions based on predefined policies.

OPA Authorization Workflow When a request is made (e.g., a service attempts to publish or subscribe to a Kafka topic), the "authorizer plugin" queries the OPA with details of the request. OPA evaluates this information against the policies it has loaded and returns a decision to Kafka, either permitting or denying the request based on its compliance with the policy rules.

Policy Implementation

In the implementation of our Event-Driven Microservice Architecture, the primary focus is not on the intricate details of crafting complex authorization policies but rather on observing and analyzing the data flow of authorization requests and decision logs within the system. Consequently, for this study, a simplified policy approach has been adopted to streamline the demonstration and evaluation of the Kafka integration decision logs visualization in Graylog in the subsequent stage of this study.

[6] `https://github.com/StyraInc/opa-kafka-plugin`

Given this focus, a straightforward Rego policy is implemented in Listing 5.7 that denies operations if the client ID is equal to ”misuser” for the Kafka policy path (line 6-8). This policy will be utilized later in the validation stage. By doing so, we can concentrate on monitoring how authorization requests from Apache Kafka are processed and how decisions are logged by OPA. This approach ensures that all Kafka messages, except those from ”misuser,” are authorized seamlessly, allowing for unrestricted data flow and facilitating comprehensive tracking and visualization of authorization activities.

```
1 package kafka.authz
2
3 default allow = false
4
5 # Deny if clientId is blacklisted
6 allow if {
7     input.requestContext.header.name.clientId != "misuser"
8 }
```

Listing 5.7: Kafka Rego Policy

Traffic Generation

To generate decision logs for visualization in Graylog, we employ a traffic generator developed in Python. The traffic generator creates a session with the dashboard application and places a Pizza order every five seconds. This helps in visualizing the practical implementation of OPA decision logs in Graylog in the subsequent stage of this study.

5.2.4 Deployment of Applications

For each of the three applications discussed previously, along with their corresponding traffic generators, Docker images and Docker Compose files are developed to facilitate their deployment as separate stacks. This strategic configuration ensures that only those components that necessitate interaction with the Open Policy Agent (OPA) are connected to the designated OPA network, optimizing both security and performance.

To streamline the deployment process and ensure consistency across different environments, all source files, Docker Compose configurations, and additional pertinent details are centrally maintained and accessible through our GitHub repository [7]. This repository serves as a comprehensive resource for all deployment-related materials, allowing for transparent, reproducible setups, and easy updates.

5.3 Validation Setup

In this section, the setup for validating our integration of Open Policy Agent (OPA) and Graylog is elaborated. To achieve a thorough validation, we have developed specially crafted applications designed to generate policy violations on demand. These validation applications leverage the traffic generators created for each application in Section 5.2. These applications simulate a variety of scenarios, providing a controlled environment to test the robustness and effectiveness of our monitoring and alerting system. The validation setup involves deploying these applications within our Docker Swarm environment, configuring them to trigger specific policy violations, and then using Graylog to capture, analyze, and visualize the resulting data.

5.3.1 Crafting Validation Applications

The traffic generators are designed to read environment variables that dictate their behavior, allowing us to simulate various scenarios seamlessly.

5.3.1.1 HTTP-API Traffic Generator

An excerpt of HTTP API Application Traffic generator is provided in Listing 5.8. The application operates in two modes, controlled by the environment variable `ROLE` (line 3), which can be set to `normal` or `attacker`. In the `normal` mode, the traffic generator produces policy-compliant requests (line 20-), ensuring continuous generation of legitimate traffic coming from Table 5.2. This default mode runs consistently

[7] `https://github.com/shumbashi/policy-compliance-monitoring`

to maintain a baseline of policy-compliant activities, simulating standard operational conditions.

Conversely, setting the `ROLE` variable to `attacker` switches the application to generate only policy-violating traffic (lines 11-17) coming from Table 5.5. In this mode, the application will generate 10 policy violations, with a random interval of 1 to 10 seconds between each violation. This mode is crucial for testing the system's ability to detect and respond to unauthorized actions. By focusing solely on policy violations, we can thoroughly evaluate the effectiveness of our event definitions and alerting mechanisms.

```
1  ...
2  # Read environment variable ROLE. Either normal or attacker. Default is
       normal
3  ROLE = os.getenv('ROLE', 'normal')
4  ...
5  def main():
6      authorized_cases = [case1, case2, case3]
7      unauthorized_cases = [case4, case5]
8
9      # if ROLE is attacker, only run unauthorized cases 10 times.
10     if ROLE == 'attacker':
11         for i in range(10):
12             # Pick an a random unauthorized case
13             case = random.choice(unauthorized_cases)
14             # Execure the case
15             case()
16             time.sleep(random.randint(1, 10))
17         exit(0)
18     else:
19         # Run forever
20         while True:
21             # Pick an a random authorized case
22             case = random.choice(authorized_cases)
23             # Execure the case
24             case()
25             # Sleep for a random time between 1 and 10 seconds
26             time.sleep(random.randint(1, 10))
```

```
27 ...
```

Listing 5.8: HTTP API Application Traffic Generator Excerpt

Table 5.5: HTTP API Traffic Generator Attacker-Mode Cases

Case	Description	Authorized?
4	A user requesting the salary of another employee	Denied
5	A manager request to get salary of a non-subordinate employee	Denied

5.3.1.2 SSH-Sudo Traffic Generator

An excerpt of this traffic generator is provided in Listing 5.9. In addition to the `ROLE` variable, an environment-controlled variable called `ATTACK\_TYPE` is added (line 5), further specifying the nature of the policy violation. This variable can be set to either `ssh` or `sudo`, directing the application to generate specific types of unauthorized access attempts. The `ssh` option triggers violations related to SSH access (line 16), while the `sudo` option focuses on unauthorized sudo command executions (line 18). This granularity allows for detailed validation of our policies and monitoring setup, ensuring that all potential violation scenarios, listed in Table 5.6, are adequately covered.

```
1 ...
2 # Read environment variable ROLE. Either normal or attacker. Default is
    normal
3 # Read environment variable ATTACK_TYPE. Either ssh or sudo. Default is
    ssh
4 ROLE = os.getenv('ROLE', 'normal')
5 ATTACK_TYPE = os.getenv('ATTACK_TYPE', 'ssh')
6 ...
7 def main():
8   authorized_cases = [case1, case2, case3, case4]
9   ssh_unauthorized_cases = [case5, case6]
10  sudo_unauthorized_cases = [case7, case8]
11
12  # if ROLE is attacker, only run unauthorized cases 10 times with a
      random sleep time between 1 and 10 seconds
13  if ROLE == 'attacker':
```

```
14      for i in range(10):
15        # if ATTACK_TYPE is ssh, run ssh unauthorized cases, else run sudo
        unauthorized cases
16        if ATTACK_TYPE == 'ssh':
17          case = random.choice(ssh_unauthorized_cases)
18        elif ATTACK_TYPE == 'sudo':
19          case = random.choice(sudo_unauthorized_cases)
20        case()
21        time.sleep(random.randint(1, 10))
22      exit(0
23    # Else, run normal cases forever
24    else:
25      while True:
26        # Pick an a random authorized case
27        case = random.choice(authorized_cases)
28        # Execute the case
29        case()
30        # Sleep for a random time between 1 and 10 seconds
31        time.sleep(random.randint(1, 10))
32  ...
```

Listing 5.9: SSH-Sudo Application Traffic Generator Excerpt

Table 5.6: HTTP API Traffic Generator Attacker-Mode Cases

Case	Description	Authorized?
5	Non-admin user (frontend-dev) logging to backend-server	Denied
6	Non-admin user (backend-dev) logging to frontend-server	Denied
7	Non-admin user (frontend-dev) executing sudo command on frontend-server	Denied
8	Non-admin user (backend-dev) executing sudo command on backend-server	Denied

5.3.1.3 Event-driven MSA Traffic Generator

An excerpt of this traffic generator is provided in Listing 5.10. A client has been developed to attempt to subscribe to a specific Kafka topic (pizzia-ordered) using the client ID “misuser” (line 8). This will ensure OPA will deny the request thereby generating a policy violation event as per the policy developed for the application in Listing 5.7.

```
...
def main():
    # Kafka Consumer Configuration
    conf = {
        'bootstrap.servers': 'broker:9092',
        'group.id': 'mygroup',
        'auto.offset.reset': 'earliest',
        'client.id': 'misuser',
    }

    # Poll for messages 10 times before exiting
    for i in range(10):
        try:
            consumer = Consumer(conf)
            consumer.subscribe(['pizza-ordered'])
            consumer.poll(timeout=1.0)
        except Exception as e:
            raise KafkaException(e)
        finally:
            consumer.close()
            time.sleep(random.randint(10, 15))
    exit(0)
...
```

Listing 5.10: Kafka Attack Traffic Generator

By systematically deploying these validation applications within our Docker Swarm environment and configuring them to generate controlled policy violations, we can rigorously test the robustness of our OPA and Graylog integration.

5.3.2 Deployment of Validation Applications

For each application, the traffic generator operating in *"normal"* mode is deployed within the respective application stack as illustrated in Section 5.2. This integration ensures that legitimate traffic is constantly generated, creating a baseline of policy-compliant activities. By embedding the traffic generator within the application stack, we can maintain a consistent flow of normal operations, which is essential for accurate monitoring and analysis.

Conversely, the *"attacker"* mode traffic generators are deployed in separate Docker Swarm stacks. This segregation allows us to isolate the violation generators from the main application environment while maintaining the necessary network configurations to ensure they can interact with the application components. Deploying the *"attacker"* mode in a separate stack provides the flexibility to start and stop the stack on demand, facilitating the generation of policy violations whenever needed. This approach enables us to test the responsiveness and effectiveness of our monitoring and alerting system without disrupting the normal operational flow. The following steps are taken to deploy validation applications.

1. Create separate Docker Swarm stacks for the "attacker" mode traffic generators.
2. Configure the traffic generators with the necessary network settings to allow communication with the main application stack.
3. Set the `ROLE` environment variable to `attacker` to ensure the traffic generator produces only policy-violating traffic.
4. For the SSH-Sudo Authorization Application, additionally set the `ATTACK_TYPE` environment variable to specify the type of policy violation (`ssh` or `sudo`).

By deploying the traffic generators in this manner, we achieve a validation environment that allows us to test our OPA and Graylog integration. The continuous *"normal"* mode traffic provides a steady baseline, while the on-demand *"attacker"* mode traffic enables targeted testing of our policy enforcement and monitoring mechanisms. This deployment strategy ensures comprehensive validation of our system's ability to detect policy violations effectively.

5.4 Graylog Configuration

This section focuses on configuring Graylog to handle, analyze, and visualize the logs generated by the applications deployed in the previous section. This stage is critical for demonstrating how Graylog can be leveraged to gain insights from the data generated by policy enforcement points and application interactions.

5.4.1 Configuring Graylog Inputs

For our specific application, the Syslog UDP [8] is used input to collect logs that are forwarded by Logspout from the OPA stack. To configure the Syslog UDP input in Graylog, the following steps are undertaken.

1. **Accessing Input Settings**: Navigate to the ”System/Inputs” section within the Graylog Dashboard.
2. **Input Selection**: From the dropdown menu, select ”Syslog UDP” and then click on ”Launch new input.”
3. **Configuration**: Enter a descriptive title for the input to easily identify it later. All other configuration settings are maintained at their default values to ensure standard functionality.
4. **Illustration**: Detailed illustration is given in Figure 5.6 for input configuration.

Figure 5.6: Graylog Input Configuration

By correctly setting up the Syslog UDP input, we ensure that Graylog is well-equipped to receive and process the Syslog messages emitted from OPA, thus facilitating effective log management and analysis.

[8] https://go2docs.graylog.org/current/getting_in_log_data/syslog_inputs.html

5.4.2 Configuring Graylog Streams

Graylog Streams are employed primarily as a method for categorizing decision logs by application. For each of our applications, a stream is created specifically to collect and organize logs relevant to the particular application. Unlike typical setups where streams use filter rules to automatically sort incoming messages, our approach involves categorizing these logs through Pipeline rules instead. Pipelines are configured to assess and route the logs to the appropriate streams based on the content and context of the messages. This method enhances the organization and segregation of decision logs, making it easier to monitor, analyze, and manage logs specific to each application within our system. Graylog Streams, listed in Table 5.7, will be established to categorize various types of logs.

Table 5.7: Utilized Graylog Streams

Stream Name	**Description**
Decision Logs	This stream captures all OPA decision logs, serving as a central repository for all policy decision activities
Operation Logs	This stream is dedicated to logging non-decision-related operations from OPA, helping differentiate operational logs from decision-making logs
HTTP API Decision Logs	This stream specifically filters OPA decision logs related to the *"HTTP API Application"*. These logs are also included in the Decision Logs stream, illustrating Graylog's capability to house messages in multiple streams simultaneously.
SSH-Sudo Decision Logs	This stream focuses on collecting OPA decision logs pertaining to the *"SSH and Sudo Authorization Application"*, ensuring easy access to logs relevant to SSH and sudo policy evaluations
Kafka Decision Logs	This stream caters to OPA decision logs associated with the *"Event-Driven Microservice Architecture"*, segregating these from the broader decision logs for more focused analysis.

5.4.2.1 Setting up the Streams

The following steps are undertaken to configure the streams in Graylog.

1. **Access the Streams Interface**: Begin by navigating to the "Streams" section accessible from the navigation bar at the top of the Graylog interface.
2. **Initiate Stream Creation**: Click on the "Create stream" button to start the configuration process.
3. **Define Stream Details**: In the creation menu, provide a Title and Description for the stream. These details are crucial as they define the stream's purpose and will be referenced later in the Pipeline configurations. Leave all other settings at their default values to ensure the stream functions according to standard specifications.
4. **Save the Stream**: After entering the necessary information, click on "Create stream" to save your new stream. At this point, the stream is created but remains inactive.
5. **Activate the Stream**: Since "Stream rules" are not required for our setup, proceed to activate the stream by clicking on "Start stream."
6. **Illustration**: Details of the stream creation form is provided in Figure 5.7.

In the next section, how log messages are precisely routed to these streams via configured Pipelines is explored, demonstrating the integration and functionality of our logging strategy within Graylog. This streamlined setup not only optimizes log management but also aligns with the thesis' goals of enhancing policy compliance monitoring across different applications.

5.4.3 Creating OPA Pipelines and Rules

Three pipelines are designed to process OPA-generated logs, enhancing their readability and usefulness for indexing and analysis. The pipeline workflow applied for each log message received is detailed in Figure 5.8. It visually maps out the progression and function of each stage within these pipelines.

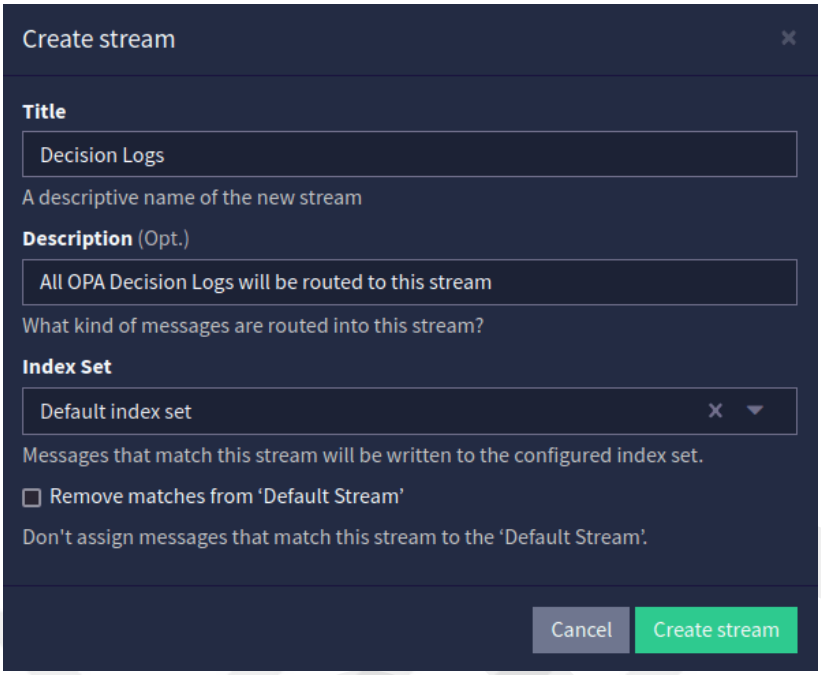

Figure 5.7: Creating a Graylog Stream

5.4.3.1 Pipeline 1 - OPA Normalization

The initial pipeline in our configuration is the OPA Normalization Pipeline whose goal is to transform the JSON formatted logs generated by OPA into indexable fields and to perform necessary data cleaning. This pipeline is attached to the *"Default Stream"* which is the default system-created stream where all incoming messages are routed. Two main stages of this pipeline are provided below.

Stage 1 - Rule 1: Flatten OPA JSON Fields The first stage includes a rule named *"Flatten OPA JSON Fields"*, implemented in Listing 5.11, which is triggered if the log message is received by our specifically designated OPA Input. The main functions of this rule are given below.

1. **JSON Parsing and Flattening**: This function parses the JSON structure of the log message and flattens it, transforming nested JSON containers into a single-level structure to simplify subsequent data handling (line 7).
2. **Field Renaming**: Due to a naming clash with the "level" field commonly used in the Syslog protocol, this rule renames the "level" field from OPA logs to "level_opa" to avoid confusion and ensure compatibility with other

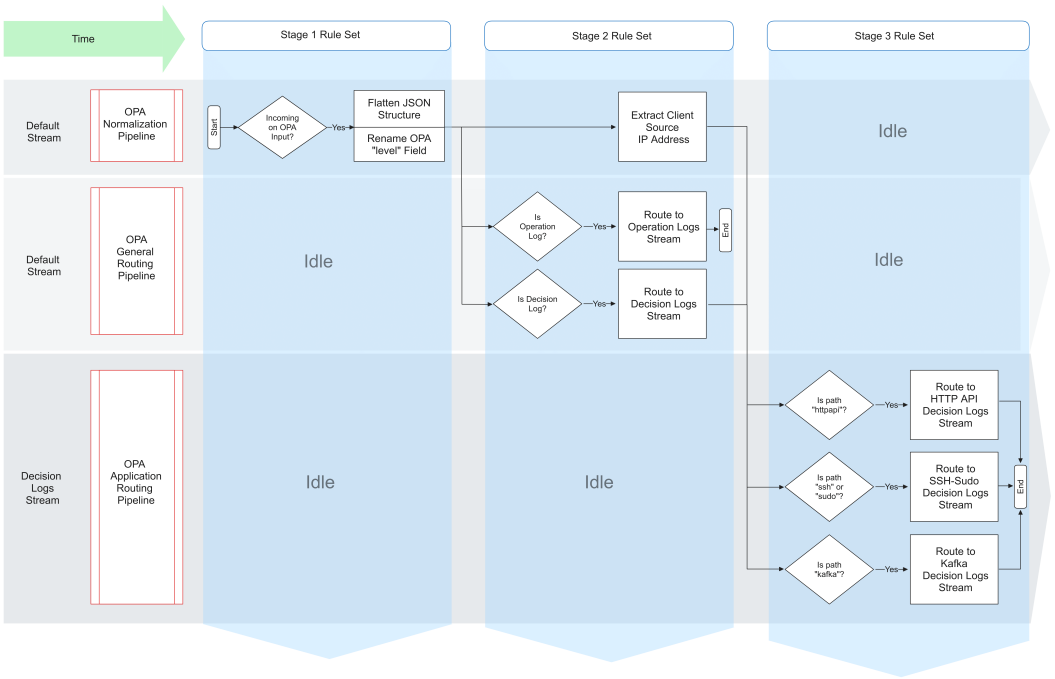

Figure 5.8: Graylog OPA Pipeline Workflow

log data (lines 9-12).

```
1  rule "Flatten OPA JSON Fields"
2  when
3    from_input(
4      name : "OPA Logs"
5    )
6  then
7    let fragment = flatten_json(to_string(message."message"), "
   flatten");
8    set_fields(to_map(fragment));
9    rename_field(
10     old_field : "level",
11     new_field : "level_opa"
12   );
13 end
14
```

Listing 5.11: Flatten OPA JSON Fields Rule

Stage 2 - Rule 1: Extract Source IP Address The second stage in this pipeline features a rule, implemented in Listing 5.12, aimed at extracting the client's IP address from the socket format (e.g., "IP:PORT") found in the "requested_by"

field of the OPA log (lines 2-7). The IP address is then stored in a newly created field named ”requested_by_src” (line 8).

```
1  rule "Extract Source IP Address from IP:PORT Pair"
2  when
3    has_field(field : "requested_by")
4    AND
5    contains(value: to_string(message."requested_by"),search:
   ":")
6  then
7    let fragment = split(":", to_string(message."requested_by"))
   [0];
8    set_field("requested_by_src", fragment);
9  end
10
```

Listing 5.12: Extract Source IP Address Rule

5.4.3.2 Pipeline 2 - OPA General Routing

To effectively manage the diverse types of OPA logs received, including decision logs and other operational logs, it is crucial to implement a routing strategy that ensures logs are directed to the appropriate streams for targeted analysis. Pipeline 2, known as the General Routing Pipeline, contains a single stage, *”Stage 2”*, which operates concurrently with Stage 2 in Pipeline 1 without any conflict, as they process different fields. This pipeline is also attached to the *“Default Stream”*. This structured routing mechanism within Pipeline 2 ensures that each type of log is systematically categorized and stored in the corresponding stream. By separating decision and operational logs, we enhance the manageability and analysis of data within Graylog, enabling more precise monitoring and quicker access to relevant log information.

Stage 2 - Rule 1: Route Decision Logs to Stream The first rule in this stage, ”Route Decision Logs to Stream”, implemented in Listing 5.13, is designed to filter and direct decision logs specifically. It checks for the presence of a `msg` field within the log message (line 3) and verifies that this field contains the exact phrase ”Decision Log” (line 5). Upon validation, the rule routes these logs to the pre-

defined ”Decision Logs” stream (lines 7-9). This focused routing ensures that decision logs are isolated for specialized analysis.

```
1  rule "Route Decision Logs to Stream"
2  when
3    has_field(field : "msg")
4    AND
5    contains(value: to_string(message."msg"),search: "Decision
   Log")
6  then
7    route_to_stream(
8      name : "Decision Logs"
9    );
10 end
11
```

Listing 5.13: Route Decision Logs to Stream Rule

Stage 2 - Rule 2: Route Operation Logs to Stream The second rule, ”Route Operation Logs to Stream”, implemented in Listing 5.14, targets operational logs by examining the `msg` field for specific operational phrases using a regular expression (line 3). This rule matches logs containing the phrases `Sent response` or `Received request` (lines 5-6). Once a match is found, the log is directed to the ”Operation Logs” stream, which is dedicated to capturing and storing operational activities (lines 9-11).

```
1  rule "Route Operation Logs to Stream"
2  when
3    has_field(field : "msg")
4    AND
5    regex(value: to_string(message."msg"),
6    pattern: "Sent response Received request"
7  ).matches == true
8  then
9    route_to_stream(
10     name : "Operation Logs"
11   );
12 end
13
```

Listing 5.14: Route Operation Logs to Stream Rule

5.4.3.3 Pipeline 3 - OPA Application Routing

This pipeline is specifically designed to segment OPA decision logs by application, enhancing the granularity of log analysis by routing them to application-specific streams. It operates at Stage 3, which ensures that it processes logs after the initial normalization and general routing tasks completed in Pipelines 1 and 2. This strategic placement in the processing order leverages the clean and categorized logs from the *"Decision Logs"* stream, ensuring that only relevant decision logs are further classified.

Each rule within this stage is crafted to identify decision logs related to specific applications based on the `path` field in the log entries. These rules then route the logs to designated streams created for each application, facilitating focused monitoring and analysis.

By segmenting decision logs based on their related application, this pipeline facilitates a more structured and efficient analysis process, allowing stakeholders to quickly pinpoint and respond to specific log data pertinent to each application's operational context.

Stage 3 - Rule 1: Route HTTP API Decision Logs to Stream This rule, implemented in Listing 5.15, filters decision logs associated with the HTTP API application. It checks for the presence of `httpapi` within the `path` field (line 5) and routes these specific logs to the *"HTTP API Decision Logs"* stream (lines 7-9). This ensures that all decision logs relevant to HTTP API interactions are consolidated in a single stream for easier access and analysis.

```
1  rule "Route HTTP API Decision logs to Stream"
2  when
3    has_field(field : "path")
4    AND
5    contains(value: to_string(message."path"),search: "httpapi")
6  then
7    route_to_stream(
8      name : "HTTP API Decision Logs"
9    );
10 end
```

```
11
```

Listing 5.15: Route HTTP API Decision logs to Stream Rule

Stage 3 - Rule 1: Route SSH Decision Logs to Stream This rule, implemented in Listing 5.16, targets logs pertaining to SSH authorization decisions. It precisely matches logs where the `path` field equals `sshd/authz` (line 5) and directs them to the *"SSH-Sudo Decision Logs"* stream (lines 7-9), thereby isolating SSH-related decision data for specialized scrutiny.

```
1  rule "Route SSH Decision Logs to Stream"
2  when
3    has_field(field : "path")
4    AND
5    to_string(message."path") == "sshd/authz"
6  then
7    route_to_stream(
8      name : "SSH-Sudo Decision Logs"
9    );
10 end
11
```

Listing 5.16: Route SSH Decision Logs to Stream Rule

Stage 3 - Rule 3: Route Sudo Decision Logs to Stream Similarly, this rule, implemented in Listing 5.17, filters for logs specifically related to Sudo authorization decisions, using a path match for `sudo/authz` (line 5). Logs identified by this rule are also routed to the *"SSH-Sudo Decision Logs"* stream (lines 7-9), maintaining a focused collection of all sudo-related activities within the same stream.

```
1  rule "Route Sudo Decision Logs to Stream"
2  when
3    has_field(field : "path")
4    AND
5    to_string(message."path") == "sudo/authz"
6  then
7    route_to_stream(
8      name : "SSH-Sudo Decision Logs"
9    );
```

```
10    end
11
```

Listing 5.17: Route Sudo Decision Logs to Stream Rule

Stage 3 - Rule 4: Route Kafka Decision Logs to Stream The final rule , implemented in Listing 5.18, is designed to capture decision logs related to the Event-Drive Microservice Architecture Application. It identifies these logs by searching for `kafka` in the `path` field (line 5) and routes them to the *"Kafka Decision Logs"* stream, which is dedicated to storing all Kafka-related decision logs (lines 7-9).

```
1   rule "Route Kafka Decision Logs to Stream"
2   when
3     has_field(field : "path")
4     AND
5     contains(value: to_string(message."path"),search: "kafka")
6   then
7     route_to_stream(
8       name : "Kafka Decision Logs"
9     );
10  end
11
```

Listing 5.18: Route Kafka Decision Logs to Stream Rule

5.5 Summary

This chapter gives details of the comprehensive setup, implementation, and configuration of our Docker Swarm environment, the deployment of baseline applications, and the integration of Graylog for log management and analysis.

1. **Environment Setup**: Establishing the Docker Swarm cluster and integrating essential tools like Traefik, Portainer, OPA, and Graylog.
2. **Implementation of Use Case Applications**: Developing and deploying applications to simulate real-world scenarios and assess policy compliance.
3. **Validation Setup**: Developing and deploying applications to simulate policy violations and validate the effectiveness of the integration.

4. **Graylog Configuration**: Configuring Graylog inputs, understanding OPA decision logs, setting up streams, and creating pipelines to route and process logs efficiently.

CHAPTER 6

DEMONSTRATION

This chapter gives details including how dashboards are created and designed, and how policy violation events are configured for each of the use case applications given in Section 5.2. This will present the applicability of our work to several cases and aid in the validation of use case applications.

6.1 Designing and Creating Dashboards

For demonstration purposes, four main Graylog dashboards are designed and implemented. The first dashboard is dedicated to monitoring overall OPA decision logs. The remaining three are each focused on one of our case study applications: the HTTP API Application, the SSH and Sudo Authorization Application, and the Event-Driven Microservice Architecture Application. Each dashboard is structured uniformly with two primary tabs: (i) General Policy Metrics and (ii) Application-Specific Metrics. The dashboards are based on the essential decision log fields identified in Section 4.3.2. This uniform structure provides a consistent blueprint for visualizing the fundamental policy elements while allowing for customization to meet specific end-user requirements.

6.1.1 Overall OPA Decisions Dashboard

The Overall OPA Decisions Dashboard is designed to provide an overarching overview of all OPA Decision Logs across all applications. This dashboard includes widgets

that display vital metrics described below.

Total Decisions Count: This widget, given in Figure 6.1 on the left-hand side, displays the total number of policy decisions made by OPA. It provides a cumulative count to give a high-level view of policy evaluation activities.

Total Policy Violations Count: This widget, given in Figure 6.1 on the right-hand side, shows the total number of policy violations detected. It helps in monitoring the number of instances where policies were not adhered to.

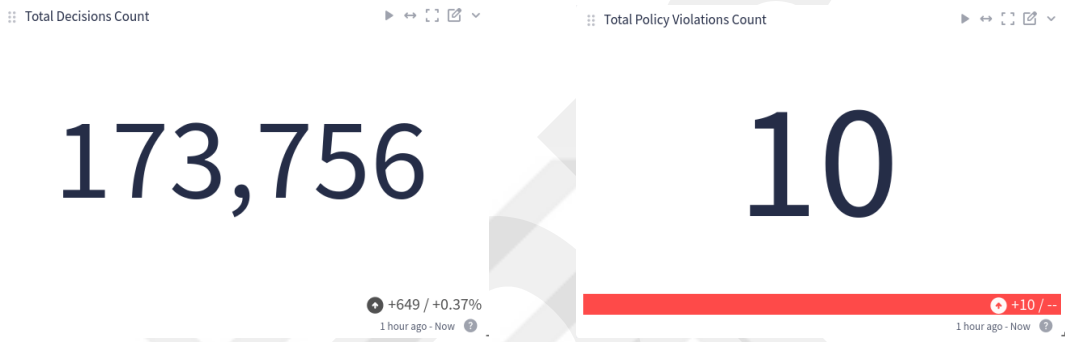

Figure 6.1: Total Decisions Count (left-hand-side) and Total Policy Violations Count (right-hand-side)

Real-Time Alerts: This widget, given in Figure 6.2, lists real-time alerts generated based on specific conditions defined in the event definitions. It provides immediate visibility into critical issues that require attention.

Decision Logs Over Time: This bar chart, given in Figure 6.3, visualizes the volume of decision logs over time, categorized by the type of policy decision. It helps in identifying trends and patterns in policy evaluation activities.

Top Applications by Policy Violations Count: This pie chart, given in Figure 6.4, shows the distribution of policy violations across different applications. It helps in identifying which applications are generating the most policy violations.

Violation Trends Over Time: This bar chart, given in Figure 6.5, visualizes the trend of policy violations over time. It helps in identifying periods with higher violation frequencies.

Policy Violations by Client IP: A bar chart, given in Figure 6.6, displaying the number of policy violations grouped by client IP addresses. Helps identify poten-

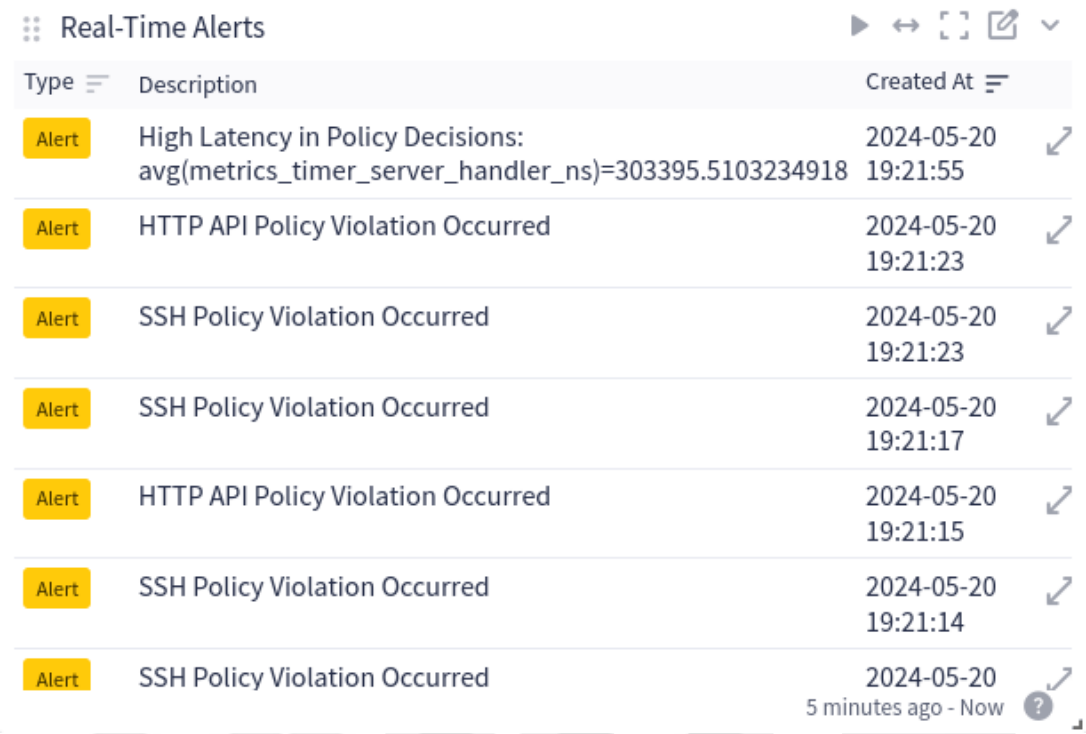

Figure 6.2: Real-Time Alerts

tial sources of policy violations, whether they are specific clients or IP addresses.

Decision Latency: An area chart, given in Figure 6.7, showing different latency metrics (Input Parse Time, Query Evaluation Time, Total Request Processing Time). Provides insights into the performance of the policy decision process, helping to identify latency issues.

These metrics offer a high-level view of policy compliance and help identify trends and patterns in policy enforcement activities. Figure 6.8 illustrates the full dashboard tab *"Overall OPA Decision Logs"*.

6.1.2 General Application Policy Metrics

The *General Application Policy Metrics* tab is designed to provide an overview of policy enforcement across the application. This tab includes widgets and visualizations that display vital metrics explained below.

Total Policy Decisions: This widget, given in Figure 6.9 on the left-hand side, displays the total number of policy decisions made by the OPA in a given timeframe. It

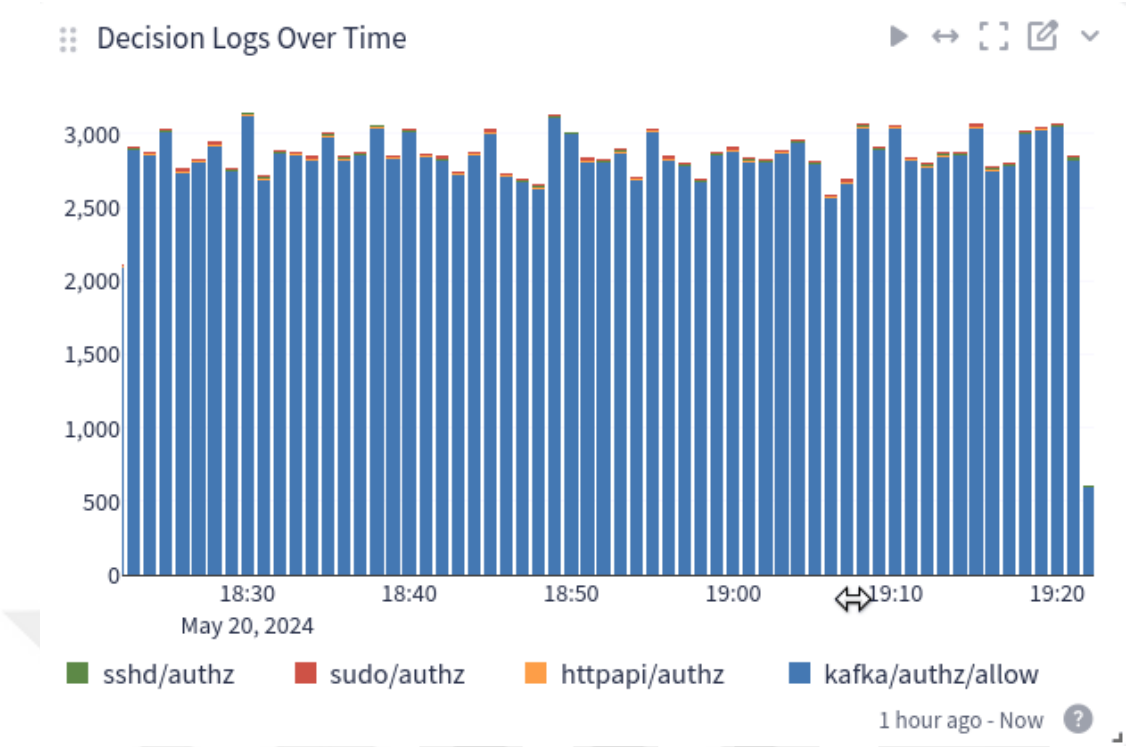

Figure 6.3: Decision Logs Over Time

provides a quick overview of the volume of policy evaluations performed, helping to assess the load on the policy engine.

Total Policy Violations: This widget, given in Figure 6.9 on the right-hand-side, shows the total number of policy violations detected. It helps in identifying the extent of non-compliance issues within the monitored environment.

Policy Queries Over Time: The bar chart, given in Figure 6.10, illustrates the number of policy queries processed over time. It helps in understanding the temporal distribution of policy evaluations and can highlight peak times of activity.

Decision Latency: This line, given in Figure 6.11, chart visualizes the latency of policy decisions, broken down into various components such as input parse time and query evaluation time. Monitoring decision latency is crucial for performance optimization and ensuring timely policy enforcement.

Client Activity: This pie chart, given in Figure 6.12 on the left-hand side, shows the distribution of policy queries by client IP addresses. It helps in identifying which clients are most actively interacting with the policy engine.

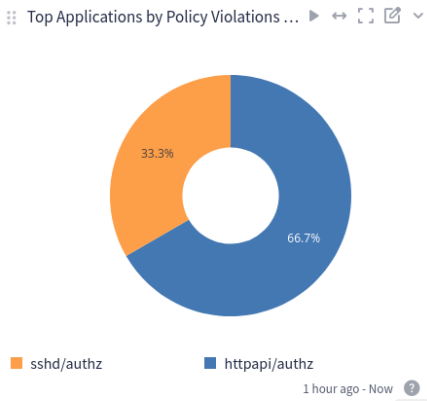

Figure 6.4: Top Applications by Policy Violations Count

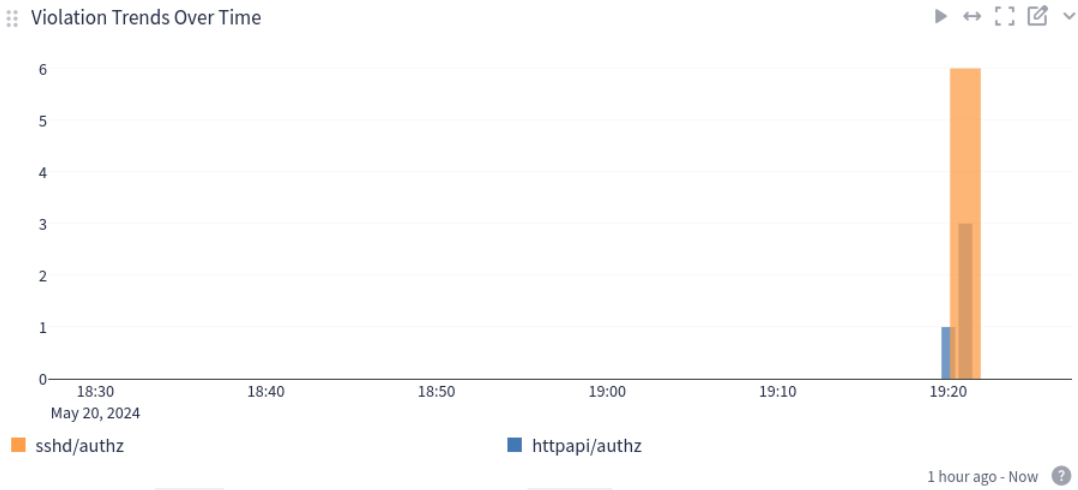

Figure 6.5: Violation Trends Over Time

Overall Policy Compliance: This pie chart, given in Figure 6.12 on the right-hand side, provides an overview of policy compliance, indicating the proportion of allowed versus denied requests. It offers a snapshot of the compliance status across the monitored environment.

Policy Result by Client: This bar chart, given in Figure 6.13, breaks down the policy results (allowed or denied) by client IP addresses. It helps in understanding how different clients are interacting with the policies and identifying any clients that are frequently causing violations.

These metrics offer a high-level view of policy compliance and help identify trends and patterns in policy enforcement activities. Figure 6.14 illustrates the full dashboard tab *"General Policy Metrics"* for the HTTP API Application.

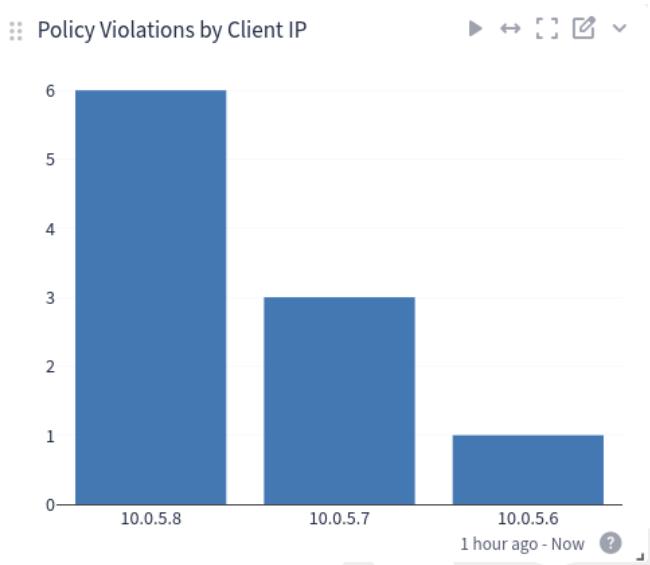

Figure 6.6: Policy Violations by Client IP

6.1.3 Application-Specific Metrics Tab

The Application-Specific Metrics tab focuses on metrics unique to each application, including widgets tailored to the specific operational and security needs of each application. For example, for the HTTP API Application dashboard in Graylog this tab contains several widgets that provide detailed insights into the policy requests and results specific to this application. Each widget visualizes particular aspects of the HTTP API interactions, helping to monitor and analyze policy compliance at a granular level.

Policy Requests by User: This donut chart, given in Figure 6.15 on the left-hand side, displays the distribution of policy requests by user. It provides a quick overview of which users are making the most requests, helping to identify user activity patterns.

Policy Results by User: This bar chart, given in Figure 6.15 on the right-hand side, shows the policy results (allowed or denied) for each user. It helps in understanding how different users are interacting with the policies and identifying any users who are frequently causing policy violations.

Policy Requests by Method: This donut chart, given in Figure 6.16 on the left-hand side, illustrates the distribution of policy requests by HTTP method (GET, POST, etc.). It provides insights into the types of operations being requested, which can help

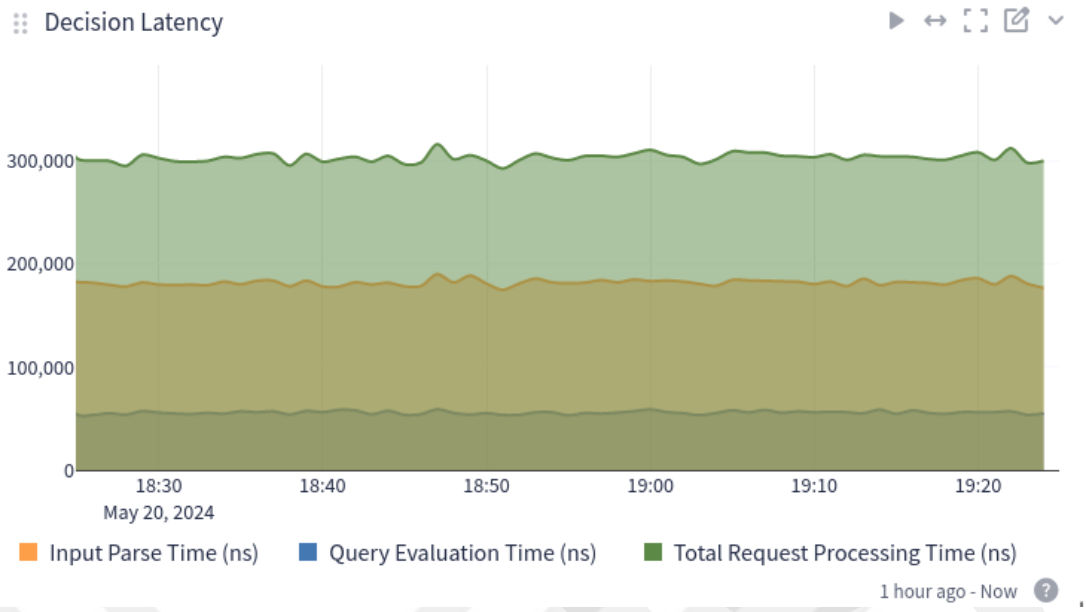

Figure 6.7: Decision Latency

in understanding usage patterns and identifying potential areas of concern.

Policy Results by Method: This bar chart, given in Figure 6.16 on the right-hand side, shows the policy results (allowed or denied) for each HTTP method. It helps in analyzing how different types of requests are being evaluated by the policies, providing insights into method-specific compliance issues.

Policy Requests by Path: This bar chart, given in Figure 6.17 on the left-hand side, displays the number of policy requests for different API paths. It helps in understanding which parts of the API are being accessed most frequently and identifying any paths that are particularly prone to policy violations.

Policy Results by Path: This bar chart, given in Figure 6.17 on the right-hand side, shows the policy results (allowed or denied) for each API path.

By segmenting metrics in this way, the dashboards provide detailed insights that are relevant to the operational context of each application, enabling stakeholders to monitor and manage specific aspects of their applications more effectively. Figure 6.18 illustrates the full dashboard tab *"Application Specific Metrics"* for the HTTP API Application.

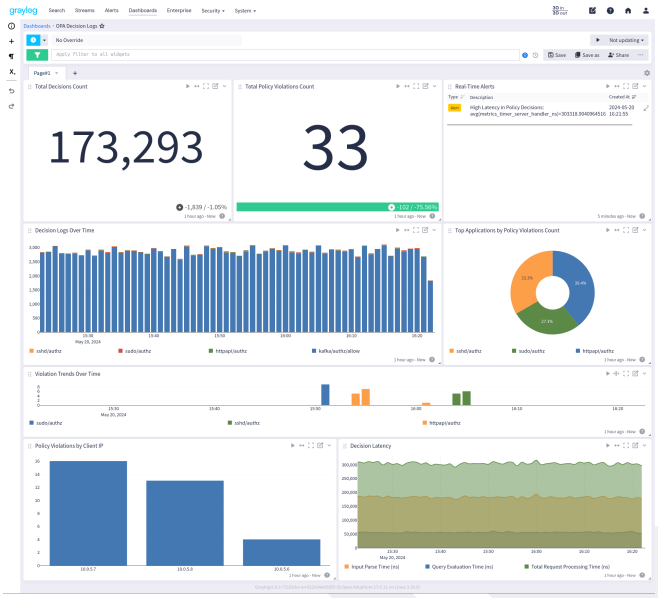

Figure 6.8: Overall OPA Decision Logs Dashboard

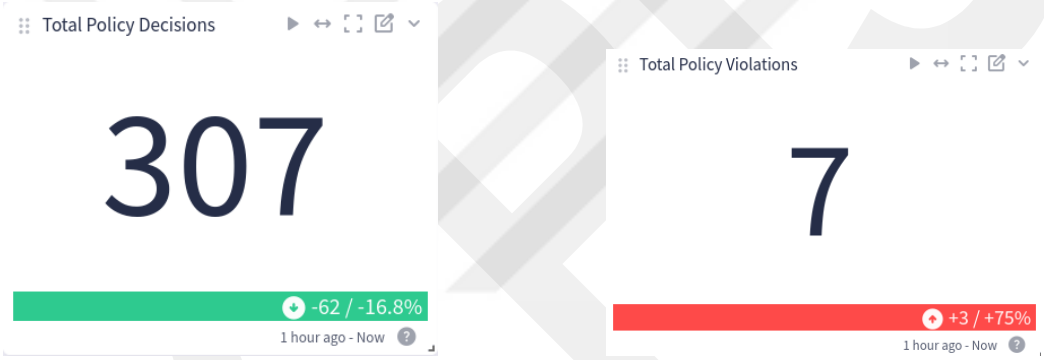

Figure 6.9: Total Policy Decisions (left-hand-side) and Total Policy Violations (right-hand-side)

6.2 Configuring Event Definitions for OPA

To effectively monitor Open Policy Agent (OPA) integration with the case study applications, a set of event definitions are created, providing comprehensive oversight of policy enforcement activities. These event definitions are crafted to cover both general OPA and application-specific events. The focus will be on capturing essential telemetry data and triggering alerts based on specific conditions related to policy evaluations and compliance status. By doing so, we ensure continuous monitoring and prompt identification of potential issues, thereby enhancing the overall security and compliance posture of the designed system. Below are detailed descriptions of each event definition.

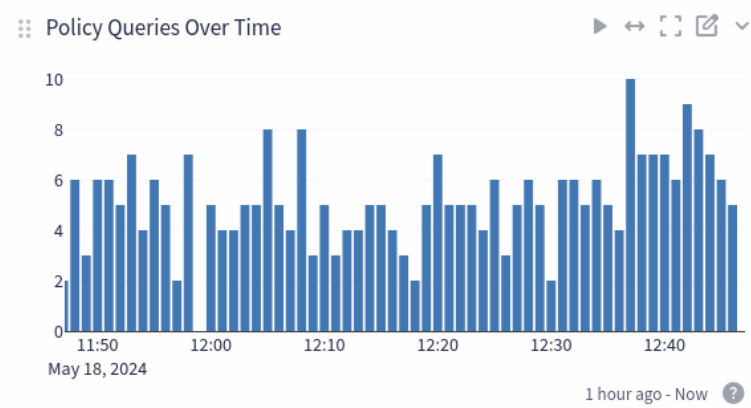

Figure 6.10: Policy Queries Over Time

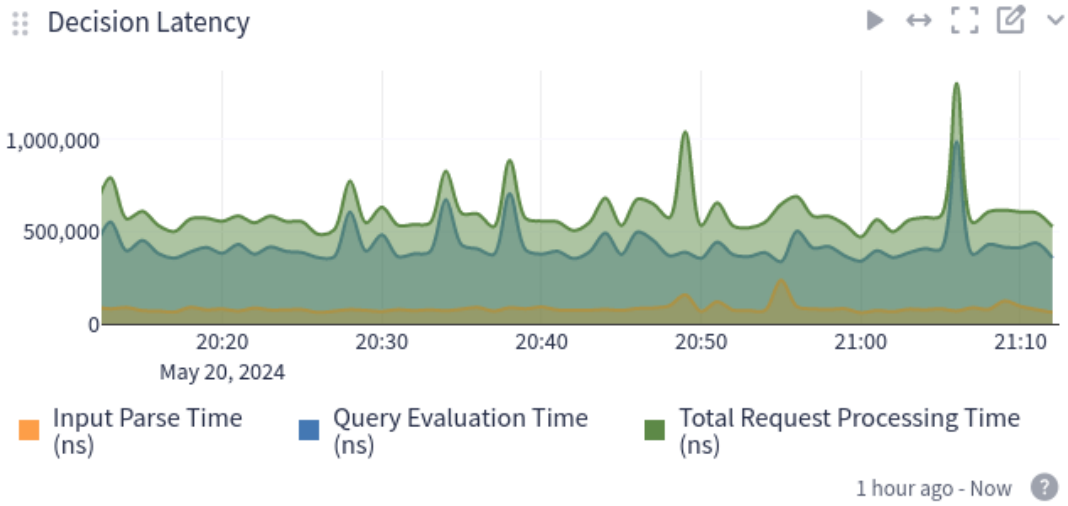

Figure 6.11: Decision Latency

6.2.1 HTTP API Policy Violation Event

This event is configured to trigger when a policy violation occurs in the HTTP API application, ensuring immediate attention to unauthorized API requests. An overview of the event definitions is listed in Table 6.1, including all relevant details.

6.2.2 SSH Policy Violation Event

This event captures violations of SSH policies, particularly when unauthorized SSH access attempts are detected, ensuring security measures are promptly enforced. An overview of the event definitions is listed in Table 6.2, including all relevant details.

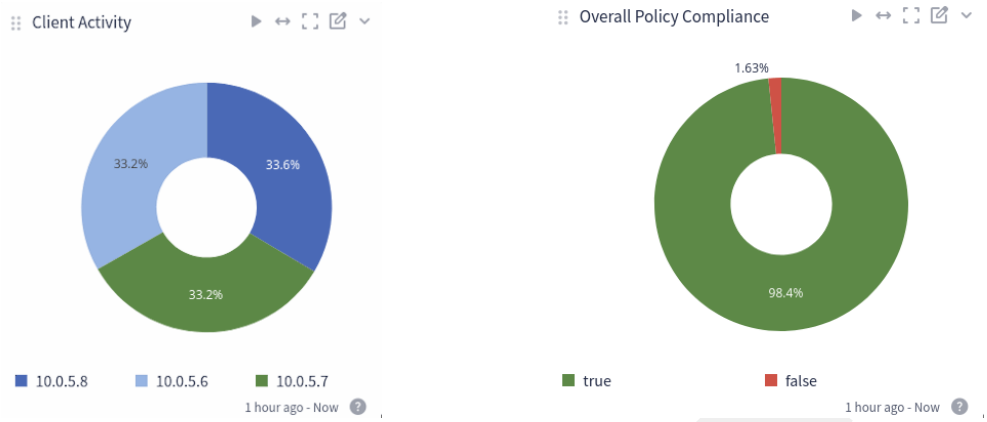

Figure 6.12: Client Activity and Overall Policy Compliance

Figure 6.13: Policy Result by Client

6.2.3 Sudo Policy Violation Event

This event monitors for unauthorized Sudo commands in the SSH-Sudo application, highlighting potential privilege escalation attempts. An overview of the event definitions is listed in Table 6.3, including all relevant details.

6.2.4 High Latency in Policy Decisions Event

This event identifies high latency in policy decision processes, enabling proactive measures to address performance bottlenecks. An overview of the event definitions is listed in Table 6.4, including all relevant details.

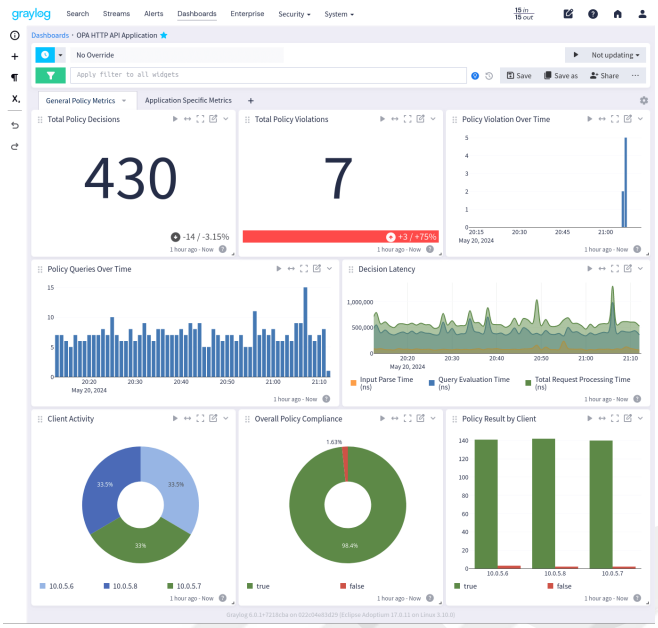

Figure 6.14: HTTP API Application General Policy Metrics

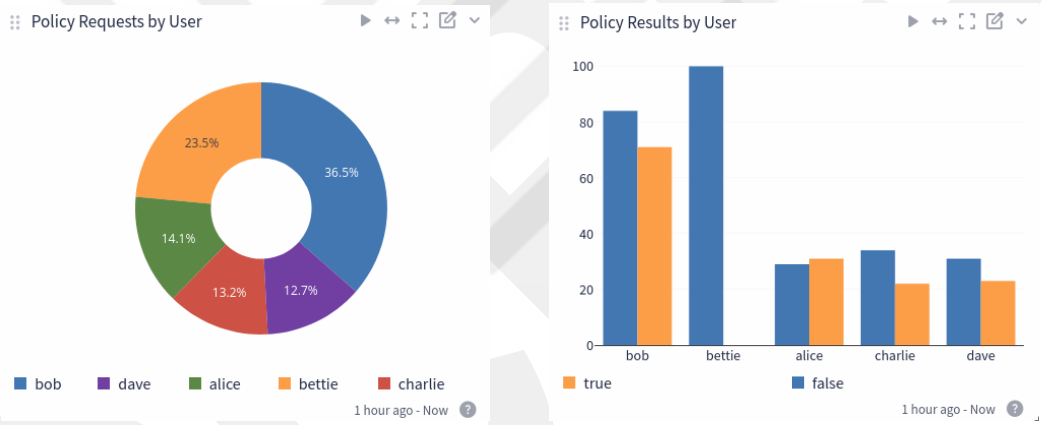

Figure 6.15: Policy Requests by User and Policy Results by User

6.2.5 Excessive Latency in Policy Decisions Event

This event is critical for detecting excessive latency in policy decisions, ensuring that significant performance issues are promptly addressed. An overview of the event definitions is listed in Table 6.5, including all relevant details.

6.3 Summary

The focus of this chapter is visualization and alerting capabilities within Graylog to enhance the monitoring of Open Policy Agent (OPA). Graylog dashboards are intro-

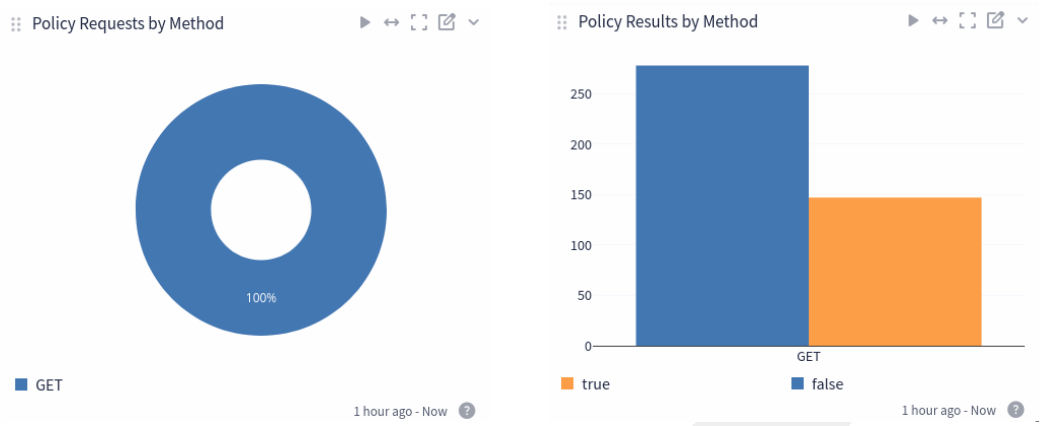

Figure 6.16: Policy Requests by Method and Policy Results by Method

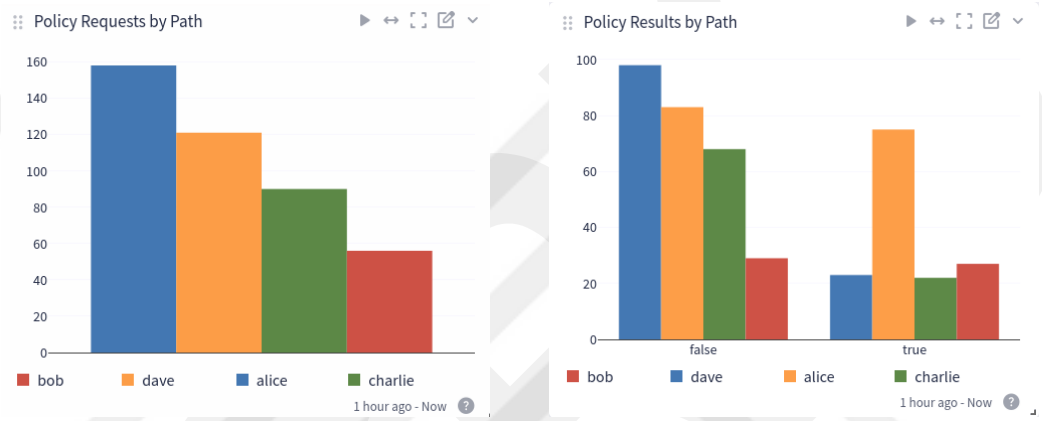

Figure 6.17: Policy Requests by Path & Policy Results by Path

duced, detailing how widgets, aggregations, and visualizations are utilized to present log data effectively. Dashboards in Graylog allow for real-time monitoring and analysis, providing a consolidated view of critical metrics.

The design and creation of four dashboards are described: one overarching dashboard for all OPA decision logs and three specific dashboards for our case study applications (*HTTP API Application, SSH and Sudo Authorization Application, and Event-Driven Microservice Architecture Application*). Each dashboard is structured with two primary tabs: General Policy Metrics and Application-Specific Metrics, ensuring a consistent and comprehensive monitoring framework.

Following the dashboard implementation, the configuration of event definitions are given. Event definitions are crucial for proactive monitoring and alerting, enabling the detection of specific conditions within log data. Event definitions to monitor general OPA activities and application-specific events are created, ensuring thorough over-

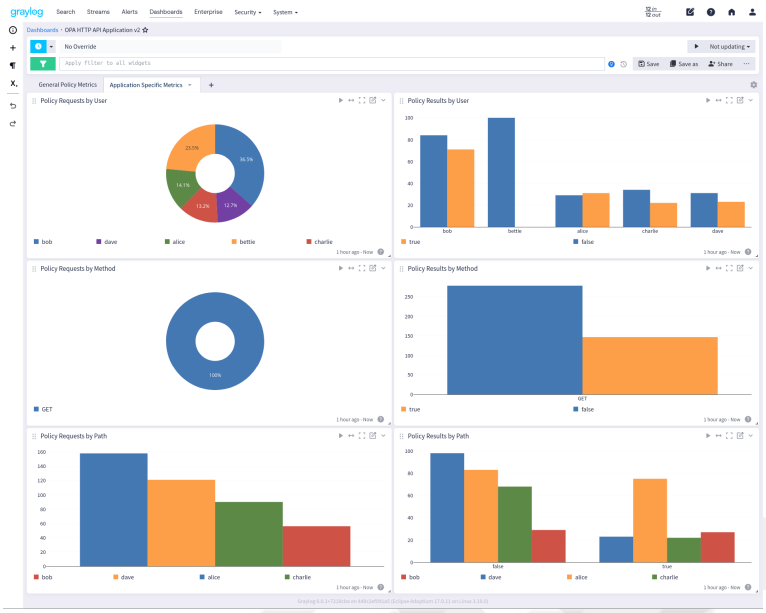

Figure 6.18: HTTP API Application - Application Specific Metrics

sight of policy enforcement and compliance.

This chapter also highlights the creation of alerting mechanisms to notify stakeholders of significant events using Slack. Alerts in Graylog can be configured to send notifications via various channels, such as email, Slack, and webhooks, ensuring timely responses to potential issues.

Table 6.1: HTTP API Policy Violation Event Definition

Title	HTTP API Policy Violation Occurred
Description	An event indicating a violation of API policy in the HTTP API application has been detected
Priority	High
Condition Type	Filter & Aggregation
Search Query	`result_allow:false`
Streams	HTTP API Decision Logs
Search within the last	1 second
Execute search every	1 second
Create Events for Definition if	Filter has results
Event Fields	`input_method`, `input_path`, `input_user`
Notifications	Alert to Slack
Grace Period	5 minutes

Table 6.2: SSH Policy Violation Event Definition

Title	SSH Policy Violation Occurred
Description	An event indicating a violation of SSH policy in the SSH-Sudo application has been detected
Priority	High
Condition Type	Filter & Aggregation
Search Query	`result_allow:false` AND `input_sysinfo_pam_service:sshd`
Streams	SSH-Sudo Decision Logs
Search within the last	1 second
Execute search every	1 second
Create Events for Definition if	Filter has results
Event Fields	`error`, `username`, `host`
Notifications	Alert to Slack
Grace Period	5 minutes

Table 6.3: Sudo Policy Violation Event Definition

Title	Sudo Policy Violation Occurred
Description	An event indicating a violation of Sudo policy in the SSH-Sudo application has been detected
Priority	High
Condition Type	Filter & Aggregation
Search Query	`result_allow:false` AND `input_sysinfo_pam_service:sudo`
Streams	SSH-Sudo Decision Logs
Search within the last	1 second
Execute search every	1 second
Create Events for Definition if	Filter has results
Event Fields	`error`, `username`
Notifications	Alert to Slack
Grace Period	5 minutes

Table 6.4: High Latency in Policy Decisions Event Definition

Title	High Latency in Policy Decisions
Description	Triggered when the decision latency for policy evaluations exceeds a certain threshold (400,000 nanoseconds)
Priority	Normal
Condition Type	Filter & Aggregation
Search Query	None
Streams	Decision Logs
Search within the last	5 minutes
Execute search every	5 minutes
Create Events for Definition if	Aggregation of results reaches a threshold
Event Conditions	`avg(..._server_handler_ns) >= 400000` AND `avg(...rver_handler_ns) < 600000`
Event Fields	None
Notifications	Alert to Slack
Grace Period	5 minutes

Table 6.5: Excessive Latency in Policy Decisions Event Definition

Title	Excessive Latency in Policy Decisions
Description	Triggered when the decision latency for policy evaluations exceeds a certain threshold (600,000 nanoseconds)
Priority	High
Condition Type	Filter & Aggregation
Search Query	None
Streams	Decision Logs
Search within the last	5 minutes
Execute search every	5 minutes
Create Events for Definition if	Aggregation of results reaches a threshold
Event Conditions	`avg(..._server_handler_ns) >= 600000`
Event Fields	None
Notifications	Alert to Slack
Grace Period	5 minutes

CHAPTER 7

EVALUATION AND RESULTS

This chapter presents the results obtained from the validation setup and demonstration outlined in Chapter 6. The aim is to showcase the effectiveness of the integrated Open Policy Agent (OPA) and Graylog system in monitoring policy compliance within selected use case applications. By utilizing specially crafted validation applications designed to generate both compliant and non-compliant policy traffic, we assess the utility of the proposed monitoring solution.

The results are organized to reflect the insights gained from the three primary applications under study: *the HTTP API Application, the SSH and Sudo Authorization Application, and the Event-Driven Microservice Architecture Application.* We focus on the data captured in Graylog dashboards, the triggered alerts from predefined event definitions, and the overall system's ability to detect and report policy violations in real-time.

7.1 Methodology for Collecting Results

In this section, we present the methodology employed for collecting results to evaluate the effectiveness of Graylog in detecting and alerting policy violations within our integrated system. The following steps were undertaken for each application:

1. **Activation of Policy Violation Generator**: For each application under study, a specially crafted policy violation generator application was activated. This application was designed to simulate policy violations by generating 10 non-

compliant requests each time it's run.

2. **Observation of Dashboards**: Once the policy violation generator was activated, the corresponding dashboards in Graylog were monitored, *"OPA Decision Logs"* dashboard and the application-specific dashboard. These dashboards were configured to display key metrics such as total decisions, policy violations, and real-time alerts.

3. **Monitoring Alerts Channel**: In addition to the dashboards, alert channels configured in Graylog were closely observed. These channels were set to notify stakeholders of significant policy violations via Slack.

4. **Data Analysis**: The data collected from the dashboards and alert channels were analyzed to evaluate the accuracy and efficiency of the policy compliance monitoring setup. Specific metrics, such as the number of detected policy violations and the latency of alerts, were examined to conclude the system's performance.

7.1.1 Baseline Dashboard Observation

Before activating the *attacker mode* on any of the applications, *"OPA Decision Logs"* dashboard was observed under normal operations. This baseline observation provided a reference point for the system's behavior when only policy-compliant traffic is generated. It allowed us to compare the normal operational state with the state during policy violation testing. The *"OPA Decision Logs"* dashboard, given in Figure 7.1, during normal operations displayed the key metrics below.

Total Decisions Count displays the overall number of policy decisions made by OPA. During normal operations, this count *reflects the typical volume of policy-compliant requests*.

Total Policy Violations Count indicates the number of policy violations detected. Under normal conditions, this count remains *at zero*, demonstrating policy compliance.

Real-Time Alerts provides notifications for significant events. During normal operations, there are *minimal or no alerts*, indicating a stable and compliant state.

Decision Logs Over Time shows a timeline of policy decisions made, categorized by different types of authorization requests (e.g., SSHD, Sudo, HTTP API, Kafka). This metric helps in understanding the distribution and frequency of policy decisions over time.

Violation Trends Over Time tracks the occurrence of policy violations over time. In the baseline state, this graph remains *flat*, indicating no policy violations.

Policy Violations by Client IP maps the policy violations to specific client IP addresses. Under normal operations, this graph shows *no data points*, reinforcing the absence of violations.

Decision Latency displays the time taken for input parsing, query evaluation, and total request processing. This metric helps in assessing the performance and efficiency of the policy engine.

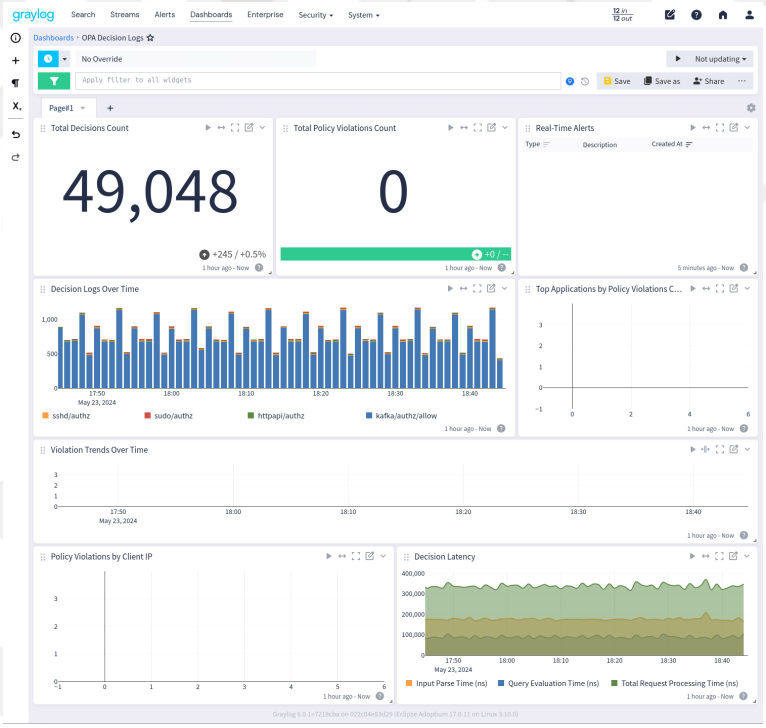

Figure 7.1: OPA Decision Logs Dashboard during normal operations, showing policy-compliant traffic.

This baseline observation provides a clear comparison to evaluate the impact and effectiveness of the system when subjected to policy violations generated in Sections 7.2, 7.3, and 7.4.

7.2 Results for HTTP-API Application

This section presents the results obtained from running the HTTP-API traffic generator application in the `attacker` mode. The policy violation generator was configured to produce 10 policy violations, with a random interval of 1 to 10 seconds between each violation. This setup was designed to test the effectiveness of our Graylog and OPA integration in detecting and alerting on policy violations.

7.2.1 Running the Traffic Generator in Attacker Mode

The HTTP-API traffic generator application was set to `attacker` mode by setting the `ROLE` environment variable accordingly in the designated Docker Swarm Stack, provided in Section 5.3.1.1. Once activated, the traffic generator produced policy-violating requests, which were then logged and monitored via our configured Graylog setup.

7.2.2 Observing the Dashboards

We monitored the ”OPA Decision Logs” dashboard and the application-specific dashboard in Graylog, given in Figure 7.2, Figure 7.3, and Figure 7.4. These dashboards provided real-time insights into the policy compliance status of the HTTP-API application. Key metrics included the count of policy violations detected, the frequency of real-time alerts, and details about users and paths involved in the violation incidents.

7.2.2.1 Monitoring Alerts Channel

The Slack alerts channel configured in Graylog notified stakeholders immediately upon detecting policy violations. Alerts were triggered by predefined event definitions, given in Figure 7.5, ensuring prompt notification of unauthorized actions.

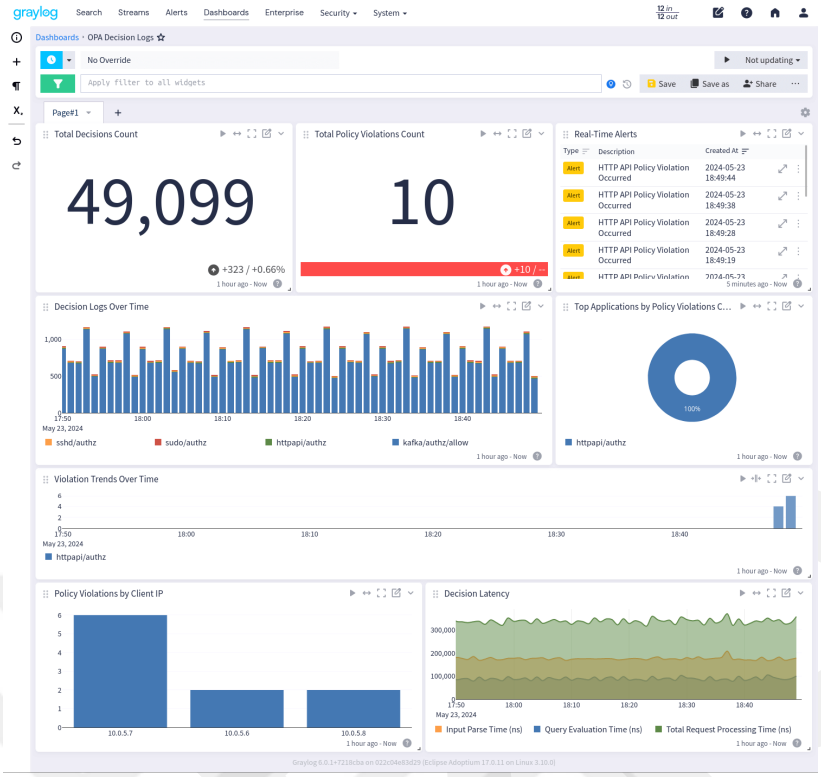

Figure 7.2: OPA Decision Logs Dashboard showing policy violations

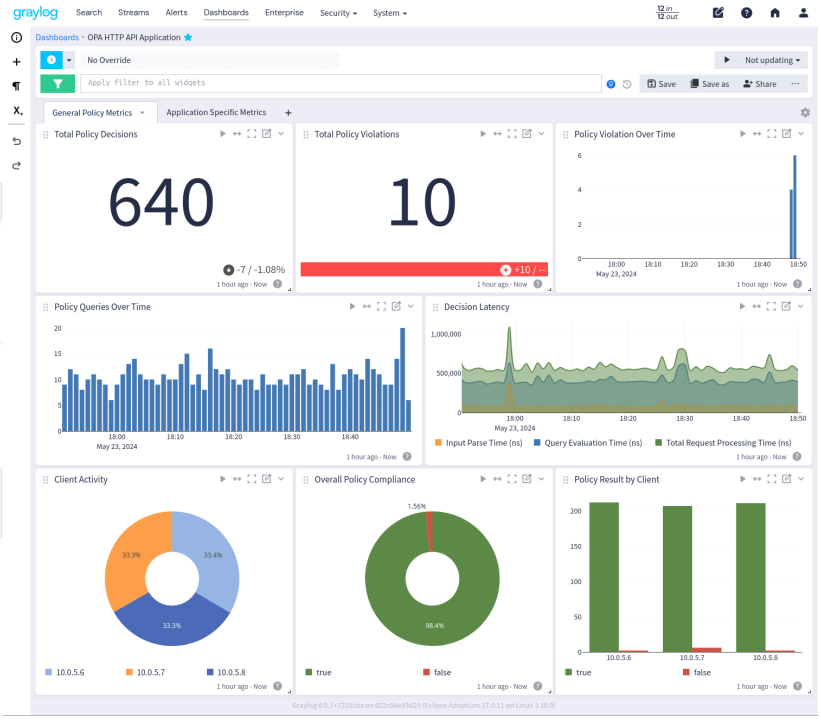

Figure 7.3: HTTP-API Application Dashboard - General Policy Metrics with metrics on policy violations

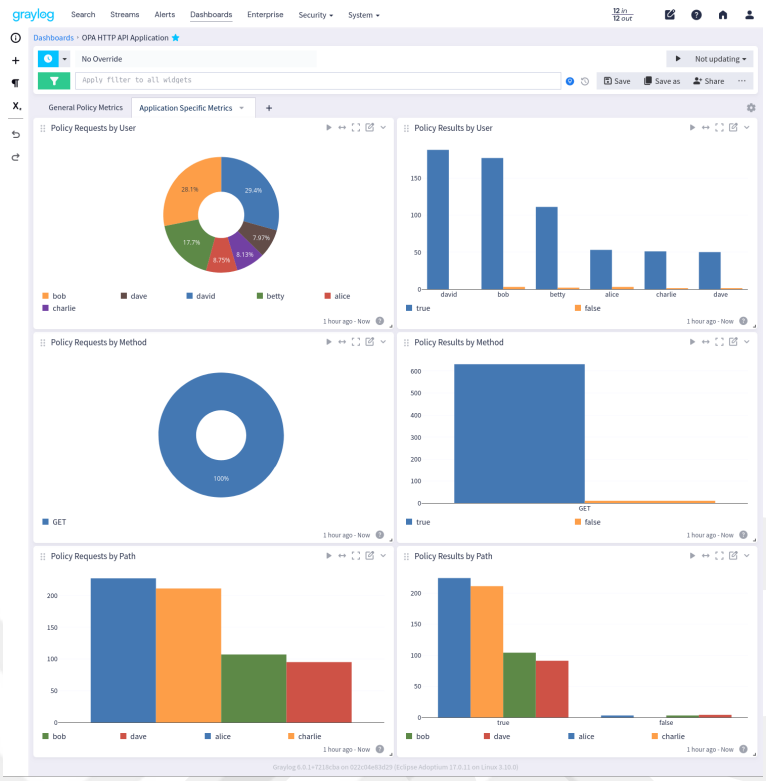

Figure 7.4: HTTP-API Application Dashboard - Application Specific Metrics with metrics on policy violations

7.2.2.2 Data Analysis

The data collected from the dashboards and Slack alerts were analyzed to evaluate the system's performance. Key findings are listed below.

- The system successfully detected all 10 policy violations generated by the HTTP-API application.
- Alerts were received in Slack with minimal latency, demonstrating the system's real-time monitoring capabilities.
- The dashboards provided clear and actionable insights into the nature and frequency of policy violations.

These results confirm the effectiveness of the integrated OPA and Graylog system in monitoring policy compliance for the HTTP-API application.

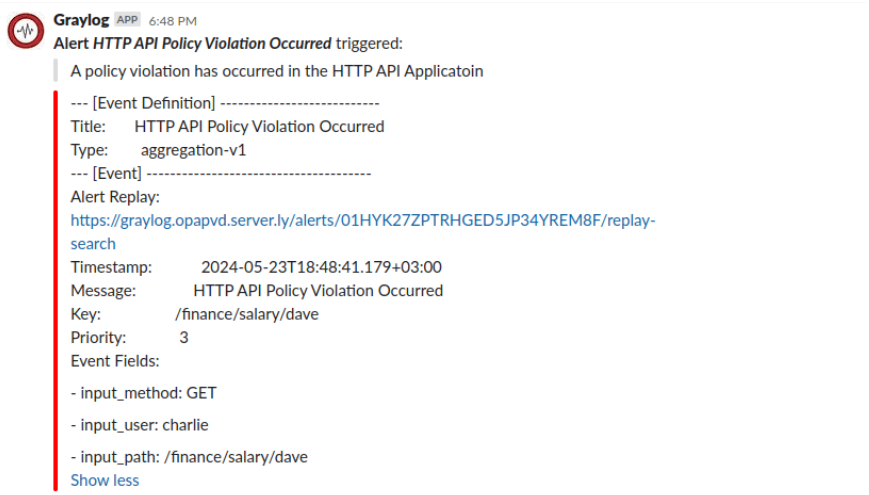

Figure 7.5: Slack alert notification for a detected policy violation in HTTP-API Application

7.3 Results for SSH-Sudo Application

This section presents the results obtained from running the SSH-Sudo traffic generator application in the `attacker` mode. The policy violation generator was configured to produce policy violations for both SSH and Sudo authorization requests.

7.3.1 Running the Traffic Generator in Attacker Mode

The SSH-Sudo traffic generator application was set to `attacker` mode by adjusting the `ROLE` environment variable accordingly the designated Docker Swarm Stacks, in Section 5.3.1.2. The `ATTACK_TYPE` variable was set to `ssh` and `sudo` respectively, and the traffic generator was run simultaneously for both types. Each run produced 10 policy violations with a random interval of 1 to 10 seconds between each violation. Due to the default configuration of Sudo, each authorization attempt is made three times before failing, resulting in a total of 30 policy violations for the Sudo attack.

7.3.2 Observing the Dashboards

The *"OPA Decision Logs"* dashboard and the application-specific dashboard are observed in Graylog, given in Figure 7.6, Figure 7.7, and Figure 7.8. These dashboards

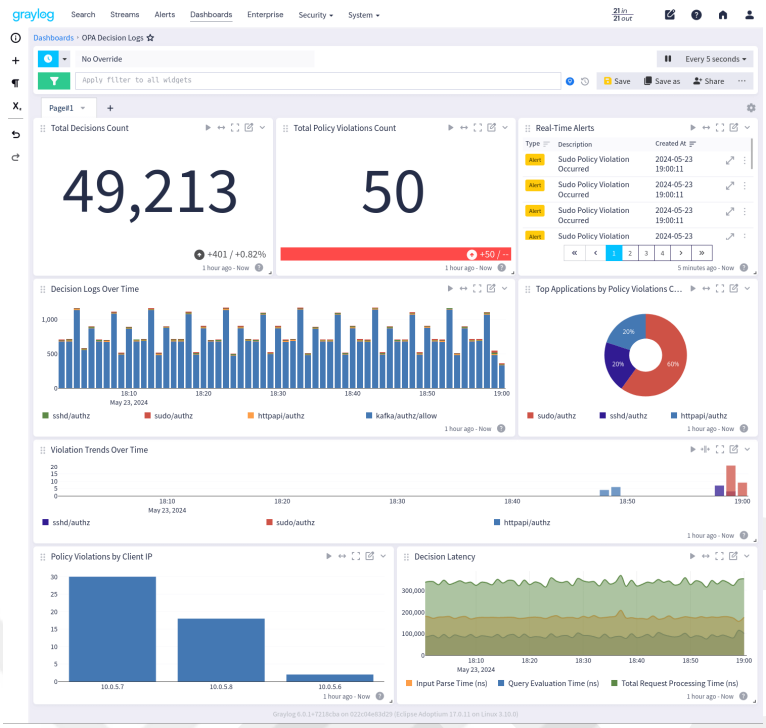

Figure 7.6: OPA Decision Logs Dashboard showing policy violations for SSH and Sudo

provided real-time insights into the policy compliance status of the SSH-Sudo application. Key metrics observed included the total number of decisions made, the count of policy violations detected, the frequency of real-time alerts, and details about users and hosts involved in the violation incident.

7.3.3 Monitoring Alerts Channel

The Slack alerts channel configured in Graylog showed minimal latency in delivering alerts, demonstrating the system's real-time monitoring capabilities, given in Figure 7.9, and Figure 7.10.

Data Analysis

The data collected from the dashboards and Slack alerts were analyzed to evaluate the system's performance. Key findings are listed below.

- The system detected all 10 SSH policy violations and all 30 Sudo policy viola-

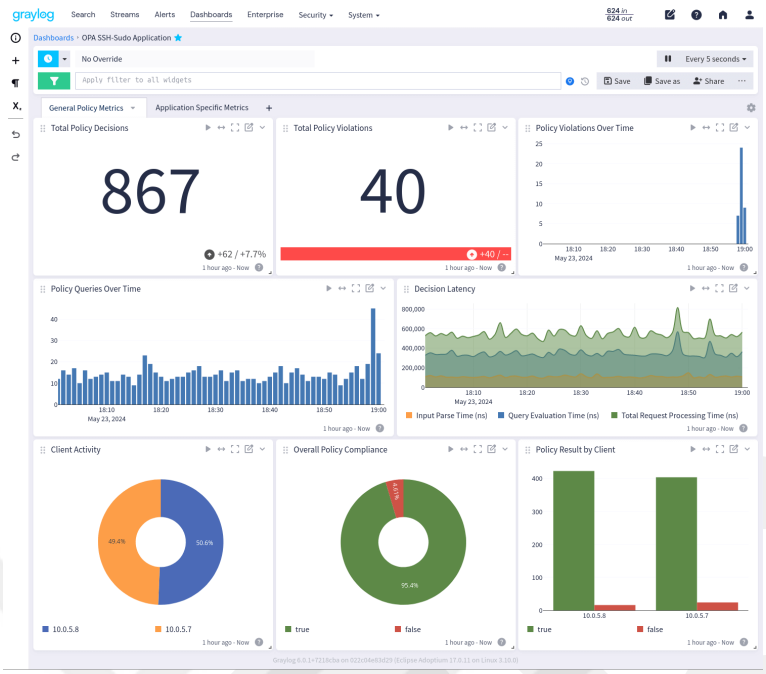

Figure 7.7: SSH-Sudo Application Dashboard - General Policy Metrics with metrics on policy violations

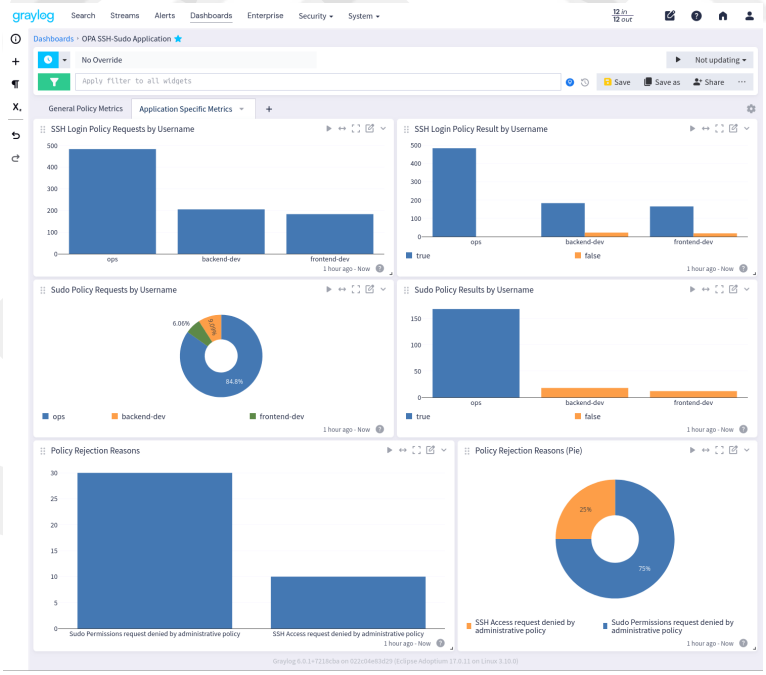

Figure 7.8: SSH-Sudo Application Dashboard - Application Specific Metrics with metrics on policy violations

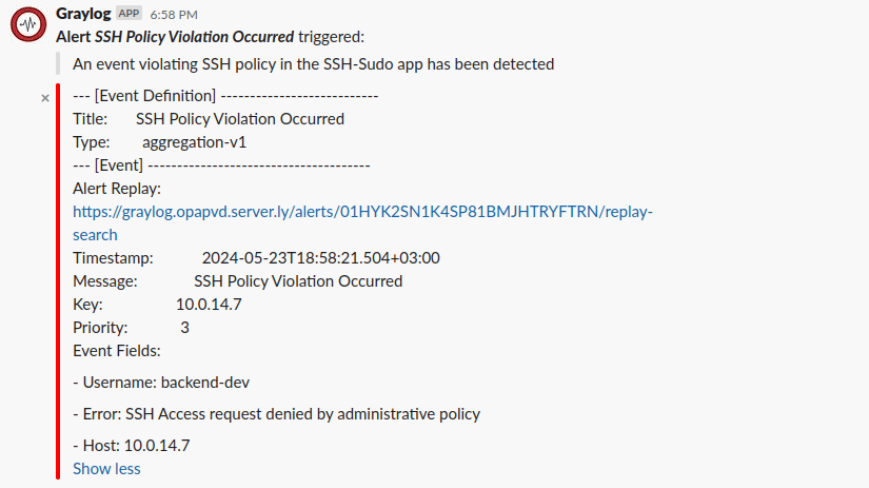

Figure 7.9: Slack alert notification for a detected SSH policy violation

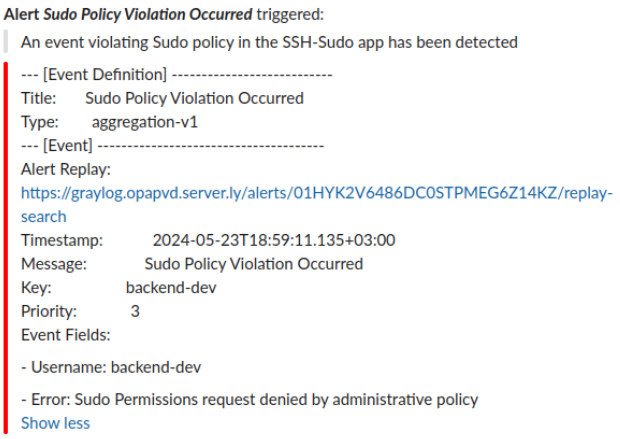

Figure 7.10: Slack alert notification for a detected Sudo policy violation

tions.

- Alerts were received in Slack with minimal latency.

- The dashboards provided clear and actionable insights into the nature and frequency of policy violations.

These results confirm the effectiveness of the integrated OPA and Graylog system in monitoring policy compliance for the SSH-Sudo application.

7.4 Results for Event-Driven Microservice Architecture Application

This section presents the results of running the policy violation generator within the Event-Driven Microservice Architecture application. The generator, a standalone application, performed policy-violating actions on the Kafka Broker, which has an OPA policy to deny requests from client ID ”misuser”.

7.4.1 Running the Traffic Generator for Policy Violations

The traffic generator for policy violations was started as a stack configured on Docker Swarm. It attempted policy-violating actions by sending requests to the Kafka Broker using the client ID ”misuser”, explained in Section 5.3.1.3. To ensure all authorization decisions were processed by OPA rather than relying on Kafka's local cache, the application was designed to sleep for a random interval between 10 and 15 seconds between each policy-violating request.

7.4.2 Observing the Dashboards

The *”OPA Decision Logs”* dashboard and the application-specific dashboard are monitored in Graylog, given in Figure 7.11, Figure 7.12 and Figure 7.13. These dashboards provided real-time insights into the policy compliance status of the Event-Driven Microservice Architecture application.

7.4.3 Monitoring Alerts Channel

The Slack alerts channel configured in Graylog is also observed. This channel was set to notify stakeholders immediately upon detection of any policy violations. The alerts were triggered by predefined event definitions, ensuring prompt notification of unauthorized actions, given in Figure 7.14.

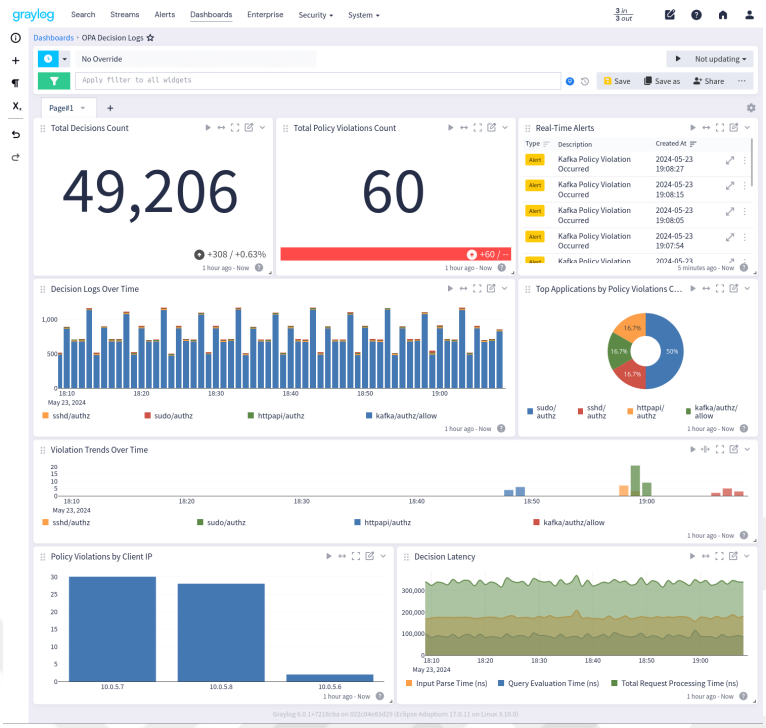

Figure 7.11: OPA Decision Logs Dashboard showing policy violations for the Event-Driven Microservice Architecture

Data Analysis

The data collected from the dashboards and Slack alerts were analyzed to evaluate the system's performance. Key findings are listed below.

- The system detected all policy violations generated by the Event-Driven Microservice Architecture application.
- Alerts were received in Slack with minimal latency.
- The dashboards provided clear and actionable insights into the nature and frequency of policy violations.

These results confirm the effectiveness of the integrated OPA and Graylog system in monitoring policy compliance for the Event-Driven Microservice Architecture application, providing a robust framework for detecting and responding to policy violations.

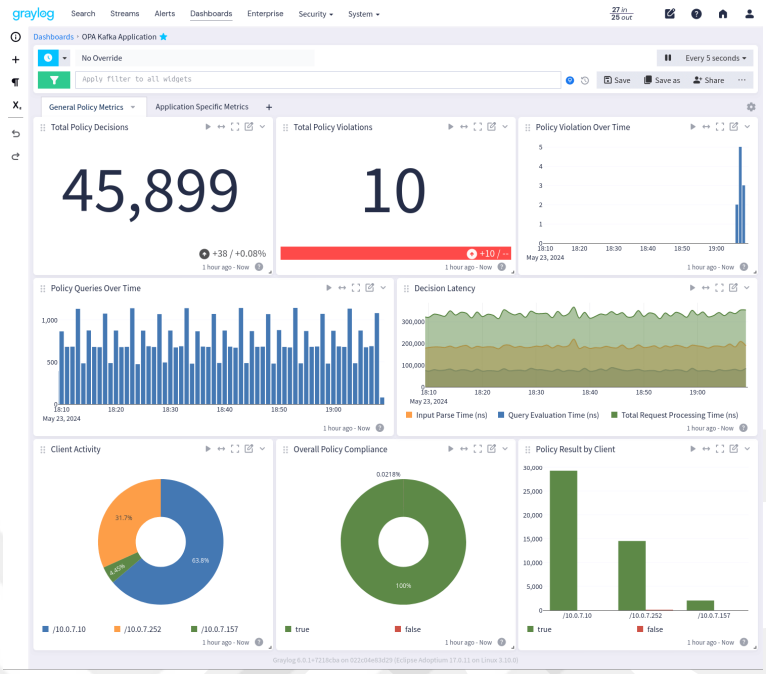

Figure 7.12: Event-Driven Microservice Application Dashboard - General Policy Metrics with metrics on policy violations

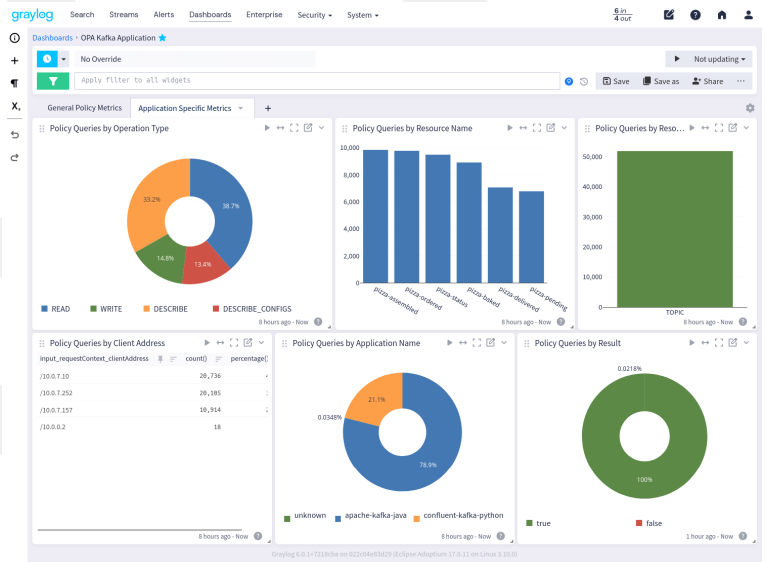

Figure 7.13: Event-Driven Microservice Application Dashboard - Application Specific Metrics with metrics on policy violations

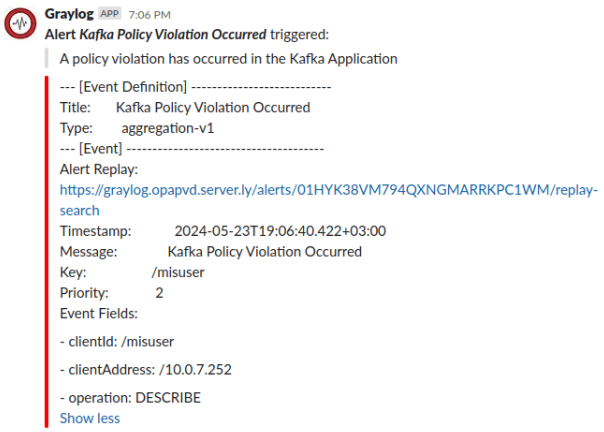

Figure 7.14: Slack alert notification for a detected policy violation

7.5 Answers to Research Questions

This section addresses the key research questions formulated in Section 1.4. Each question is revisited and answered based on the findings and analysis presented throughout the chapters. The purpose of this section is to consolidate the insights gained and to demonstrate how the research objectives have been achieved.

RQ1: How can the integration of Graylog with Open Policy Agent (OPA) enhance real-time policy compliance monitoring?

Real-time policy compliance monitoring is significantly enhanced by the integration of Graylog with OPA through several mechanisms, as detailed below:

- **Centralized Log Aggregation and Analysis**: Graylog's logging capabilities enable the centralized collection and parsing of OPA decision logs, which contain detailed information about policy evaluations and enforcement decisions. This setup, as shown in Section 4.3 allows for comprehensive analysis and real-time detection of policy violations, providing a robust framework for monitoring compliance effectively.
- **Real-Time Alerting**: The integration allows for real-time alerting on policy violations. Administrators can receive immediate alerts upon detection of non-compliant activities by configuring Graylog to send notifications via various channels, such as Slack. This prompt notification system ensures that policy violations are addressed swiftly, minimizing potential security risks, detailed in

Section 6.2.

- **Scalability in Containerized Environments**: Deploying the integrated solution in containerized environments, such as Docker Swarm, ensures scalability. This setup can handle the dynamic nature of modern IT infrastructures, making it suitable for large-scale deployments where policy compliance monitoring needs to adapt to changing conditions and workloads. The Graylog deployment in Section 4.3 ensures that we can process decision logs from any number of OPA servers when they are scaled up without requiring any changes on the Graylog side.

- **Enhanced Visualization and Dashboards**: Graylog provides robust visualization capabilities through customizable dashboards. As shown in Chapter 6, these dashboards offer real-time insights into policy compliance status, displaying critical metrics such as the number of policy decisions, the frequency of violations, and the distribution of these violations across various parameters. This visual representation aids administrators in quickly identifying trends and anomalies, facilitating informed decision-making.

- **Improved Operational Workflows**: By integrating Graylog with OPA, the operational workflows for managing policy compliance are significantly streamlined. Graylog's search and filtering capabilities allow for efficient querying of logs, making it easier to track compliance issues across different applications and services. This integration also supports the creation of tailored monitoring solutions to meet specific compliance requirements, enhancing the overall security posture of the organization.

RQ2: How effective are the policy decision log trails generated by Open Policy Agent in providing actionable insights on policy compliance?

Based on the detailed analysis of OPA decision logs performed in Section 4.3.2, it is evident that the decision logs generated by Open Policy Agent contain sufficient information to effectively detect policy violations. The critical data elements in OPA decision logs include the decision result (allow or deny), the policy path, input attributes (such as user role, resource, and action), and timestamps. These elements are

essential for identifying non-compliant activities. Graylog can be configured to parse these elements using extractors and pipelines. Custom rules and alert conditions can be set up to trigger notifications when specific patterns indicative of policy violations are detected. By structuring the logs and leveraging Graylog's search and filtering capabilities, organizations can gain detailed insights into the nature and frequency of policy violations.

RQ3: In what ways can customizable dashboards in Graylog provide actionable insights into policy compliance trends and anomalies in different OPA use cases?

Customizable dashboards in Graylog allow for the creation of visual representations of log data, making it easier to track policy compliance trends and detect anomalies as shown in Chapter 6. Dashboards can display metrics such as the total number of policy decisions, the number of violations over time, and the distribution of violations by user or resource. They can also visualize decision latencies and other performance metrics. By providing a real-time, interactive interface, dashboards enable administrators to quickly identify unusual patterns, such as spikes in policy violations or delays in decision-making, which could indicate underlying issues that need to be addressed.

RQ4: What specific features of Graylog contribute most to enhancing the visibility and management of policy compliance?

Drawing upon the Graylog configuration detailed in Section 5.4, we assert the following Graylog features are the most contributing to enhancing the visibility and management of policy compliance:

- **Powerful Search and Filtering Capabilities**: Graylog's robust search and filtering features allow users to quickly query logs using a wide range of criteria. This is particularly useful for identifying specific policy violations or tracking compliance across different applications and services. Users can create complex search queries to pinpoint issues, which enhances the ability to detect and investigate policy breaches.
- **Customizable Dashboards**: One of Graylog's most impactful features is its customizable dashboards. These dashboards provide real-time visualizations of log data, making it easier to monitor policy compliance metrics and detect

anomalies. Dashboards can display a variety of widgets, such as charts, graphs, and counters, which provide actionable insights into policy enforcement trends, violation counts, and system performance.

- **Alerting and Notifications**: Graylog's alerting system is crucial for proactive compliance management. Administrators can configure alerts to trigger notifications based on specific log patterns or threshold breaches. For example, alerts can be set up to notify administrators immediately when a policy violation occurs, ensuring rapid response and mitigation.

- **Power and Flexibility of Graylog Pipelines**: The configuration of Graylog Pipelines, as explored in Section 5.4.3, demonstrates their power and flexibility in enhancing and enriching log messages. Pipelines enable the dynamic transformation of incoming log data, applying rules that can modify, tag, and route logs based on specific criteria. This capability not only improves the clarity and relevance of log data but also allows for sophisticated data processing workflows that support comprehensive policy compliance monitoring.

RQ5: What are the main challenges and limitations faced during the deployment of Graylog and OPA integration in a Docker Swarm environment, and how can these be mitigated?

Some challenges faced during the deployment include ensuring the reliable collection and forwarding of logs, managing the scalability of the logging infrastructure, and handling the performance overhead introduced by logging. To mitigate these challenges, it is essential to optimize the configuration of Docker Swarm for resource management, ensure that Graylog and OPA are correctly configured for high availability, and use efficient log shipping methods (such as Logspout) to reduce latency. Additionally, regular monitoring and tuning of the system can help maintain performance and scalability as the environment grows.

Careful planning for storage space is also crucial. In our use case, the applications were generating an average of 5GB of logs per day. This volume of data requires thoughtful consideration of storage management strategies to avoid running out of space and to ensure smooth operation. Graylog provides mechanisms to manage log

storage through indices retention and rotation policies [72]. These policies allow administrators to define how long logs should be retained and when old logs should be deleted or archived. Designing these policies carefully is essential to balance the need for historical log data with the available storage capacity. By configuring indices retention and rotation policies appropriately, organizations can ensure that enough storage space is available to accommodate ongoing log generation without impacting system performance.

Implementing these strategies helps maintain the efficiency and reliability of the integrated Graylog and OPA system, ensuring that it can scale effectively and continue to provide valuable insights into policy compliance and security monitoring.

7.6 Summary

This chapter presented the results of our validation setup and demonstration of the integrated Open Policy Agent (OPA) and Graylog system for monitoring policy compliance across three primary applications: the HTTP-API Application, the SSH and Sudo Authorization Application, and the Event-Driven Microservice Architecture Application.

For the **HTTP-API application**, the system effectively detected all generated policy violations and provided timely alerts through Slack, showcasing the real-time monitoring capabilities of the integrated setup. The dashboards offered comprehensive insights into the nature and frequency of violations, confirming the system's ability to maintain policy compliance.

In the **SSH-Sudo application**, the system successfully identified all policy violations generated for both SSH and Sudo requests. The minimal latency in Slack alerts and the detailed metrics available on the dashboards underscored the system's efficiency and responsiveness in detecting unauthorized actions.

For the **Event-Driven Microservice Architecture application**, the traffic generator's policy-violating actions were promptly detected by the integrated system. The real-time insights from the dashboards and the immediate alerts sent via Slack validated

the system's effectiveness in monitoring complex, event-driven environments.

Across all applications, the integrated OPA and Graylog system demonstrated a robust framework for real-time policy compliance monitoring. The dashboards provided clear and actionable data, while the alerting mechanisms ensured stakeholders were promptly informed of any violations. These results confirm that the system can effectively detect, report, and respond to policy violations, enhancing overall security and compliance within diverse application environments.

CHAPTER 8

CONCLUSION

8.1 Discussion and Findings

This thesis investigated the integration of Graylog with Open Policy Agent (OPA) to enhance real-time policy compliance monitoring. The findings from this study demonstrate the effectiveness of this integration in several key areas:

- **Enhanced Real-Time Monitoring**: The integration of Graylog with OPA significantly improved real-time policy compliance monitoring by providing a centralized platform for log aggregation, analysis, and visualization. The ability to collect and parse OPA decision logs enabled real-time alerting and visualization of policy violations, allowing administrators to promptly identify and respond to compliance issues.
- **Effective Detection and Alerting**: Across all tested applications—HTTP API, SSH-Sudo, and Event-Driven Microservice Architecture—the integrated system effectively detected policy violations and provided timely alerts. This confirms the system's capability to maintain policy compliance through real-time monitoring and immediate stakeholder notification via channels like Slack.
- **Actionable Insights through Dashboards**: Customizable dashboards in Graylog provided actionable insights into policy compliance trends and anomalies. These dashboards displayed critical metrics such as the total number of policy decisions, the number of violations over time, and the distribution of violations by user or resource, which facilitated informed decision-making and rapid responses to security and compliance challenges.

8.2 Contributions to the Field

This thesis aimed to leverage Graylog's log management capabilities to enhance OPA's policy monitoring features. The primary contributions of this work are:

1. **Identification of Essential Data Elements**: The study identified the critical data elements from OPA decision logs necessary for detecting policy violations and developed an OPA Decision Log Parser and Data Extractor.
2. **Development of a Monitoring Solution**: A comprehensive solution for integrating OPA with Graylog was developed. This included designing and implementing intuitive dashboards and alerting mechanisms. This solution provides real-time visibility into policy enforcement activities and enables proactive management of compliance issues.
3. **Validation through Case Studies**: The system's effectiveness was validated using three case study applications. Each application demonstrated the system's ability to detect and alert on policy violations accurately and efficiently, thereby confirming the integration's utility in diverse operational environments.
4. **Contribution to the Community**: The thesis aimed to contribute the developed dashboards, data extractors, and alert rules—collectively known as a "content pack"—to the Graylog and OPA communities. This contribution provides organizations with a starting point for integrating OPA with Graylog, facilitating broader adoption and implementation of effective policy monitoring solutions.

8.3 Future Work

While this research has provided significant insights and practical solutions for enhancing policy compliance monitoring, several areas remain open for future investigation:

- **Scalability and Performance Optimization**: Future research should focus on optimizing the scalability and performance of the integrated system. This in-

cludes exploring more efficient log shipping methods and enhancing the configuration of Docker Swarm for better resource management.

- **Broader Application Use Cases**: Expanding the range of application use cases to include other container orchestration platforms like Kubernetes would provide a more comprehensive understanding of the system's effectiveness across different environments.
- **Advanced Analytics and Machine Learning**: Incorporating advanced analytics and machine learning techniques could further enhance the system's ability to detect and predict policy violations, providing even more proactive compliance monitoring.
- **Enhanced User Guidance and Documentation**: Developing detailed user guidance and best practices for configuring and utilizing OPA's logging and monitoring capabilities can help bridge the current gap in user knowledge and improve the overall effectiveness of policy compliance monitoring.

8.4 Final Thoughts

In conclusion, the integration of Graylog with OPA presents a powerful solution for real-time policy compliance monitoring. The findings and contributions of this thesis provide a solid foundation for future advancements in this field, ultimately enhancing the security and compliance posture of organizations.

REFERENCES

[1] Introduction to open policy agent. Cloud Native Computing Foundation. (accessed: 11.06.2024). [Online]. Available: https://www.openpolicyagent.org/docs/latest/

[2] M. Caracciolo, "Policy as code, how to automate cloud compliance verification with open-source tools," Master Thesis, Politecnico di Torino, Italy, 2023.

[3] S. Hassan, R. Bahsoon, and R. Kazman, "Microservice transition and its granularity problem: A systematic mapping study," *Software: Practice and Experience*, vol. 50, no. 9, pp. 1651–1681, 2020.

[4] O.V. Talaver and T.A. Vakaliuk, "Reliable distributed systems: Review of modern approaches," *Journal of Edge Computing*, vol. 2, no. 1, pp. 84–101, 2023.

[5] R.A. Oginga, "A Model For Detecting Information Technology Infrastructure Policy Violations in a cloud environment," PhD Thesis, Kabarak University, Kenya, 2019.

[6] J.G.C.d.C. Machado, "Assurance and Compliance of Security Policies in Cloud-Native Environments," in *Assurance and Compliance of Security Policies in Cloud-Native Environments*, 2022.

[7] R. Rinnan, "Benefits of Centralized Log File Correlation," Masters Thesis, Gjovik University College, Norway, 2005.

[8] K.O. Detken, T. Rix, C. Kleiner, B. Hellmann, and L. Renners, "SIEM approach for a higher level of IT security in enterprise networks," in *2015 IEEE 8th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, 2025, pp. 322–327.

[9] A. Oliner, A. Ganapathi, and W. Xu, "Advances and challenges in log analysis," *Communications of the ACM*, vol. 55, no. 2, pp. 55–61, 2012.

[10] Integrating OPA. Open Policy Agent. (accessed: 23.06.2024). [Online]. Available: https://www.openpolicyagent.org/docs/latest/integration/

[11] Overview & Architecture. Open Policy Agent. (accessed: 23.06.2024). [Online]. Available: https://www.openpolicyagent.org/docs/latest/management-introduction/

[12] Decision Logs. Open Policy Agent. (accessed: 23.06.2024). [Online]. Available: https://www.openpolicyagent.org/docs/latest/management-decision-logs/

[13] What is graylog. Graylog. (accessed: 23.06.2024). [Online]. Available: https://go2docs.graylog.org/5-2/what_is_graylog/what_is_graylog.htm

[14] P. Samarati and S. Vimercati, "Access Control: Policies, Models, and Mechanisms," in *Foundations of Security Analysis and Design*, R. Focardi and R. Gorrieri, Eds., 2001, pp. 137–196.

[15] R. Sandhu and P. Samarati, "Access control: Principle and practice," *IEEE Communications Magazine*, vol. 32, no. 9, pp. 40–48, 1994.

[16] T.Y.C. Woo and S.S. Lam, "Authorization in Distributed Systems: A Formal Approach," in *IEEE Symposium on Security and Privacy*, 1992.

[17] Department of Defense Trusted Computer System Evaluation Criteria. Department of Defense. (accessed: 11.06.2024). [Online]. Available: http://csrc.nist.gov/publications/history/dod85.pdf

[18] D. Ferraiolo and R. Kuhn. Role-Based Access Controls. NIST. (accessed: 11.06.2024). [Online]. Available: http://csrc.nist.gov/groups/SNS/rbac/documents/ferraiolo-kuhn-92.pdf

[19] C. Wright, "Information gathering," in *The IT Regulatory and Standards Compliance Handbook*, C. Wright, Ed. Syngress, 2008, pp. 73–114.

[20] Guide to Attribute Based Access Control (ABAC) Definition and Considerations. NIST. (accessed: 23.06.2024). [Online]. Available: https://nvlpubs.nist.gov/nistpubs/specialpublications/NIST.SP.800-162.pdf

[21] R.W. Baldwin. Naming and Grouping Privileges to Simplify Security Management in Large Databases. Standford University. (accessed: 23.06.2024). [Online]. Available: https://theory.stanford.edu/~ninghui/courses/Fall03/papers/baldwin.pdf

[22] V.C. Hu, D. Ferraiolo, R. Kuhn, A. Schnitzer, K. Sandlin, R. Miller, and K. Scarfone, "Guide to attribute based access control (abac) definition and considerations," *NIST Special Publication*, vol. 800, p. 162, 2014.

[23] R. Sandhu, "Lattice-based access control models," *Computer*, vol. 26, no. 11, pp. 9–19, 1993.

[24] E. Bertino, C. Bettini, E. Ferrari, and P. Samarati, "A temporal access control mechanism for database systems," *IEEE Transactions on Knowledge and Data Engineering*, vol. 8, no. 1, pp. 67–80, 1996.

[25] M. Decker, "Location-aware Access Control: An Overview," in *Proceedings of Informatics*, 2009, pp. 75–82.

[26] H. F. Atlam, A. Alenezi, R. Khalid Hussein, and G. B. Wills, "Validation of an Adaptive Risk-based Access Control Model for the Internet of Things," *International Journal of Computer Network and Information Security*, vol. 10, no. 1, pp. 26–35, 2018.

[27] Philosophy. Open Policy Agent. (accessed: 23.06.2024). [Online]. Available: https://www.openpolicyagent.org/docs/latest/philosophy/

[28] 16 Best Log Management Tools for 2024 (Free + Paid). Comparitech. (accessed: 23.06.2024). [Online]. Available: https://www.comparitech.com/net-admin/log-management-tools/

[29] Case Studies. Graylog. (accessed: 23.06.2024). [Online]. Available: https://dev-graylog.pantheonsite.io/resources/case-studies/

[30] Threat Detection and Incident Response. Graylog. (accessed: 23.06.2024). [Online]. Available: https://dev-graylog.pantheonsite.io/use-cases/threat-detection-and-response/

[31] Centralized Log Management. Graylog. (accessed: 23.06.2024). [Online]. Available: https://dev-graylog.pantheonsite.io/use-cases/centralized-log-management/

[32] Audit & Regulatory Compliance. Graylog. (accessed: 23.06.2024). [Online]. Available: https://dev-graylog.pantheonsite.io/use-cases/audit-and-regulatory-compliance/

[33] Docker overview. Docker Documentation. (accessed: 23.06.2024). [Online]. Available: https://docs.docker.com/get-started/overview/

[34] Swarm mode overview. Docker Documentation. (accessed: 23.06.2024). [Online]. Available: https://docs.docker.com/engine/swarm/

[35] J. Horton and R. Safavi-Naini, "Detecting Policy Violations through Traffic Analysis," in *2006 22nd Annual Computer Security Applications Conference (ACSAC'06)*, 2006, pp. 109–120. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/4041159

[36] E. Vianello, F. Agostini, L. Cappelli, D. Tommaso, A. Galavotti, J. Gasparetto, and F. Giacomini, "A RESTful approach to tape management in StoRM," *CHEP*, 2023.

[37] Z. Szabó, "Evaluation of a policy enforcement solution in telemedicine with offline use cases," *Pollack Periodica*, vol. 17, no. 1, pp. 12–17, 2021.

[38] F. Fornari, A. Alkhansa, A. Costantini, C. Pellegrino, and D. Salomoni, "Posix access to remote storage via openid connect," in *EPJ Web of Conferences*, vol. 295, 2024, p. 01043.

[39] J. Tan *et al.*, "Ensuring component dependencies and facilitating documentation by applying open policy agent in a devsecops cloud environment," Masters Thesis, Aalto University, Finland, 2022.

[40] E. Gül and E.N. Yılmaz, "Log management with open source tools," in *SETSCI-Conference Proceedings*, vol. 4, 2019, pp. 164–171.

[41] R.D. Șuștic, A. Moraru, A.B. Rus, and V. Dobrota, "Performance Evaluation of ELK Stack versus Graylog as Open Source Log Mangement Tools," *Acta Technica Napocensis. Electronica-Telecomunicatii*, vol. 62, no. 1, 2022.

[42] Automating Security Operations. Graylog. (accessed: 23.06.2024). [Online]. Available: http://graylog.org/docs/automating-security-operations/

[43] P. O'Neill. Open Policy Agent 2022 User Survey Summary. Medium. (accessed: 23.06.2024). [Online]. Available: https://blog.openpolicyagent.org/open-policy-agent-2022-user-survey-summary-370cf0243bb7

[44] Policy Language. Open Policy Agent. (accessed: 23.06.2024). [Online]. Available: https://www.openpolicyagent.org/docs/latest/policy-language/

[45] Adopters. GitHub. (accessed: 23.06.2024). [Online]. Available: https://github.com/open-policy-agent/opa/blob/master/ADOPTERS.md

[46] T. Sandall. Open Policy Agent 2021 Survey Summary. Medium. (accessed: 23.06.2024). [Online]. Available: https://blog.openpolicyagent.org/open-policy-agent-2021-survey-summary-e749bbd7b824

[47] Streams. Graylog. (accessed: 23.06.2024). [Online]. Available: https://go2docs.graylog.org/5-2/making_sense_of_your_log_data/streams.html?Highlight=streams

[48] Inputs. Graylog. (accessed: 23.06.2024). [Online]. Available: https://go2docs.graylog.org/5-2/getting_in_log_data/inputs.htm?TocPath=Graylog%20Inputs%7C____0

[49] Pipelines. Graylog. (accessed: 23.06.2024). [Online]. Available: https://go2docs.graylog.org/5-2/making_sense_of_your_log_data/pipelines.html?Highlight=pipelines

[50] Rules. Graylog. (accessed: 23.06.2024). [Online]. Available: https://go2docs.graylog.org/current/making_sense_of_your_log_data/rules.html

[51] Detecting Threats with Graylog Pipelines - Part 1. Reconinfosec. (accessed: 23.06.2024). [Online]. Available: https://blog.reconinfosec.com/detecting-threats-with-graylog-pipelines

[52] Dashboards. Graylog. (accessed: 23.06.2024). [Online]. Available: https://go2docs.graylog.org/current/interacting_with_your_log_data/dashboards.html?tocpath=Visualizing%20Log%20Data%7C____3

[53] Event definitions. Graylog. (accessed: 23.06.2024). [Online]. Available: https://go2docs.graylog.org/current/interacting_with_your_log_data/event_definitions.html?tocpath=Managing%20Events%7CEvent%20Definitions%7C____0

[54] Alerts. Graylog. (accessed: 23.06.2024). [Online]. Available: https://go2docs.graylog.org/current/interacting_with_your_log_data/alerts.html?tocpath=Managing%20Events%7CEvent%20Definitions%7CAlerts%7C____0

[55] Content Packs. Graylog. (accessed: 23.06.2024). [Online]. Available: https://go2docs.graylog.org/5-2/what_more_can_graylog_do_for_me/content_packs.html

[56] Traefik Proxy Documentation - Traefik. Traefik. (accessed: 23.06.2024). [Online]. Available: https://doc.traefik.io/traefik/

[57] Portainer documentation. Portainer. (accessed: 23.06.2024). [Online]. Available: https://docs.portainer.io

[58] Graylog Open system and hardware requirements - Development. Graylog Community. (accessed 23.06.2024). [Online]. Available: https://community.graylog.org/t/graylog-open-system-and-hardware-requirements/29886

[59] OpenSearch installation and configuration [current]. PandoraFMS. (accessed 23.06.2024). [Online]. Available: https://pandorafms.com/manual/!current/en/documentation/pandorafms/technical_annexes/38_opensearch_installation

[60] Cve list for graylog. CVEDetails. (accessed: 23.06.2024). [Online]. Available: https://www.cvedetails.com/vulnerability-list/vendor_id-19067/Graylog.html

[61] Cve list for open policy agent. CVEDetails. (accessed: 23.06.2024). [Online]. Available: https://www.cvedetails.com/vulnerability-list/vendor_id-25923/Openpolicyagent.html

[62] Cve list for docker. CVEDetails. (accessed: 23.06.2024). [Online]. Available: https://www.cvedetails.com/vulnerability-list/vendor_id-13534/product_id-28125/Docker-Docker.html

[63] Cve list for portainer. CVEDetails. (accessed: 23.06.2024). [Online]. Available: https://www.cvedetails.com/vulnerability-list/vendor_id-19294/product_id-50211/Portainer-Portainer.html

[64] Cve list for traefik. CVEDetails. (accessed: 23.06.2024). [Online]. Available: https://www.cvedetails.com/vulnerability-list/vendor_id-24981/product_id-98472/Traefik-Traefik.html

[65] Docker 1.12 Swarm Mode - Under the hood. Collabnix. (accessed: 23.06.2024). [Online]. Available: https://collabnix.com/docker-1-12-swarm-mode-under-the-hood/

[66] Raft consensus in swarm mode. Docker Documentation. (accessed: 23.06.2024). [Online]. Available: https://docs.docker.com/engine/swarm/raft/

[67] Getting started with Swarm mode. Docker. (accessed: 23.06.2024). [Online]. Available: https://docs.docker.com/engine/swarm/swarm-tutorial/

[68] Docker object labels. Docker. (accessed: 23.06.2024). [Online]. Available: https://docs.docker.com/config/labels-custom-metadata/

[69] Http apis. Open Policy Agent. (accessed: 23.06.2024). [Online]. Available: https://www.openpolicyagent.org/docs/latest/http-api-authorization/

[70] SSH and sudo. Open Policy Agent. (accessed: 23.06.2024). [Online]. Available: https://www.openpolicyagent.org/docs/latest/ssh-and-sudo-authorization/

[71] I. Nesi. Python-kafka-microservices implementation. Github. (accessed: 23.06.2024). [Online]. Available: https://github.com/ifnesi/python-kafka-microservices

[72] Index time size optimizing. Graylog. (accessed: 23.06.2024). [Online]. Available: https://go2docs.graylog.org/current/setting_up_graylog/index_time_size_optimizing.htm?Highlight=indices

[73] J. Brunel, F. Cuppens, N. Cuppens, T. Sans, and J.P. Bodeveix, "Security policy compliance with violation management," in *Proceedings of the 2007 ACM Workshop on Formal Methods in Security Engineering*, ser. FMSE '07, 2007, pp. 31–40.

[74] R.E. Smith, “A Contemporary Look at Saltzer and Schroeder’s 1975 Design Principles,” *IEEE Secur and Privacy*, vol. 10, no. 6, pp. 20–25, 2012.

[75] Manage swarm service networks. Docker. (accessed: 23.06.2024). [Online]. Available: https://docs.docker.com/engine/swarm/networking/

Appendix A

FULL DOCKER COMPOSE CONFIGURATION

This appendix includes the complete docker-compose files used in the implementation of our containerized environment. Each service configuration is detailed with annotations to aid in understanding the purpose and function of each setting.

A.1 Traefik Docker Compose File

```
version: '3.3'

services:
  traefik:
    image: traefik:v3.0
    ports:
      - target: 80
        published: 80
        mode: host
      - target: 443
        published: 443
        mode: host
    deploy:
      placement:
        constraints:
          - node.labels.traefik-public.traefik-public-certificates ==
    true
      labels:
        - traefik.enable=true
        - traefik.docker.network=traefik-public
        - traefik.constraint-label=traefik-public
```

```
      - traefik.http.middlewares.admin-auth.basicauth.users= {USERNAME?Variable not set}: {HASHED_PASSWORD?Variable not set}
      - traefik.http.middlewares.https-redirect.redirectscheme.scheme=https
      - traefik.http.middlewares.https-redirect.redirectscheme.permanent=true
      - traefik.http.routers.traefik-public-http.rule=Host( {DOMAIN?Variable not set} )
      - traefik.http.routers.traefik-public-http.entrypoints=http
      - traefik.http.routers.traefik-public-http.middlewares=https-redirect
      - traefik.http.routers.traefik-public-https.rule=Host( {DOMAIN?Variable not set} )
      - traefik.http.routers.traefik-public-https.entrypoints=https
      - traefik.http.routers.traefik-public-https.tls=true
      - traefik.http.routers.traefik-public-https.service=api@internal
      - traefik.http.routers.traefik-public-https.tls.certresolver=le
      - traefik.http.routers.traefik-public-https.middlewares=admin-auth
      - traefik.http.services.traefik-public.loadbalancer.server.port=8080
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock:ro
      - traefik-public-certificates:/certificates
    command:
      - --providers.docker
      - --providers.docker.constraints=Label( traefik.constraint-label , traefik-public )
      - --providers.docker.exposedbydefault=false
      - --providers.swarm.endpoint=unix:///var/run/docker.sock
      - --entrypoints.http.address=:80
      - --entrypoints.https.address=:443
      - --certificatesresolvers.le.acme.email= {EMAIL?Variable not set}
      - --certificatesresolvers.le.acme.storage=/certificates/acme.json
      - --certificatesresolvers.le.acme.tlschallenge=true
      - --accesslog
      - --log
      - --api
    networks:
      - traefik-public
```

```
volumes:
  traefik-public-certificates:
networks:
  traefik-public:
    external: true
```

Listing A.1: Traefik Docker Compose File

A.2 Portainer CE Docker Compose File

```
version: '3.2'

services:
  agent:
    image: portainer/agent:2.19.4
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
      - /var/lib/docker/volumes:/var/lib/docker/volumes
    networks:
      - agent_network
    deploy:
      mode: global
      placement:
        constraints: [node.platform.os == linux]
  portainer:
    image: portainer/portainer-ce:2.19.4
    command: -H tcp://tasks.agent:9001 --tlsskipverify
    volumes:
      - portainer_data:/data
    networks:
      - agent_network
      - traefik-public
    deploy:
      mode: replicated
      replicas: 1
      placement:
        constraints: [node.role == manager]
      labels:
        - traefik.enable=true
        - traefik.docker.network=traefik-public
        - traefik.constraint-label=traefik-public
```

```
      - traefik.http.routers.portainer-http.rule=Host( portainer.opapvd.server.ly )
      - traefik.http.routers.portainer-http.entrypoints=http
      - traefik.http.routers.portainer-http.middlewares=https-redirect
      - traefik.http.routers.portainer-https.rule=Host( portainer.opapvd.server.ly )
      - traefik.http.routers.portainer-https.entrypoints=https
      - traefik.http.routers.portainer-https.tls=true
      - traefik.http.routers.portainer-https.tls.certresolver=le
      - traefik.http.services.portainer.loadbalancer.server.port=9000
networks:
  agent_network:
    driver: overlay
    attachable: true
  traefik-public:
    external: true
volumes:
  portainer_data:
```

Listing A.2: Portainer Docker Compose File

A.3 OPA Docker Compose File

```
version: '3.4'
services:
  opa:
    image: openpolicyagent/opa:latest
    ports:
      - "8181:8181"
    command:
      - "run"
      - "--server"
      - "--log-format=json"
      - "--set=decision_logs.console=true"
      - "--set=services.nginx.url=http://bundle_server"
      - "--set=bundles.nginx.service=nginx"
      - "--set=bundles.nginx.resource=bundle.tar.gz"
      - "--set=bundles.nginx.polling.min_delay_seconds=10"
      - "--set=bundles.nginx.polling.max_delay_seconds=30"
    depends_on:
      - bundle_server
```

```
    deploy:
      mode: replicated
      replicas: 3
      update_config:
        parallelism: 1
        delay: 10s
      restart_policy:
        condition: on-failure
    networks:
      - opa

  bundle_server:
    image: nginx:1.20.0-alpine
    ports:
      - 8888:80
    volumes:
      - bundles:/usr/share/nginx/html/
    deploy:
      mode: replicated
      replicas: 1
      update_config:
        parallelism: 1
        delay: 10s
      restart_policy:
        condition: on-failure
    networks:
      - opa
networks:
  opa:
    external: true
volumes:
  bundles:
    external: true
```

Listing A.3: OPA Docker Compose File

A.4 Graylog Docker Compose File

```
version: '3.4'
services:
  mongodb:
```

```
    image: mongo:6.0.14
    networks:
      - graylog
    volumes:
      - mongo_data:/data/db
    deploy:
      mode: replicated
      replicas: 1
  opensearch:
    image: "opensearchproject/opensearch:2.12.0"
    environment:
      - "OPENSEARCH_JAVA_OPTS=-Xms1g -Xmx1g"
      - "bootstrap.memory_lock=true"
      - "discovery.type=single-node"
      - "action.auto_create_index=false"
      - "plugins.security.ssl.http.enabled=false"
      - "plugins.security.disabled=true"
      - OPENSEARCH_INITIAL_ADMIN_PASSWORD=SECRETPASSWORDPLACEHOLDER
    ulimits:
      memlock:
        hard: -1
        soft: -1
      nofile:
        soft: 65536
        hard: 65536
    restart: "on-failure"
    deploy:
      mode: replicated
      replicas: 1
    networks:
      - graylog
  graylog:
    image: graylog/graylog:5.2
    environment:
    - GRAYLOG_NODE_ID_FILE=/usr/share/graylog/data/config/node-id
    - GRAYLOG_HTTP_BIND_ADDRESS=0.0.0.0:9000
    - GRAYLOG_ELASTICSEARCH_HOSTS=http://opensearch:9200
    - GRAYLOG_MONGODB_URI=mongodb://mongodb:27017/graylog
    - GRAYLOG_PASSWORD_SECRET=SECRETPASSWORDPLACEHOLDER
    - GRAYLOG_ROOT_PASSWORD_SHA2=SECRETENCRYPTEDPASSWORDPLACEHOLDER
```

```
    - GRAYLOG_HTTP_EXTERNAL_URI=https://graylog.opapvd.server.ly/
    entrypoint: /usr/bin/tini -- wait-for-it opensearch:9200 -- /docker-
    entrypoint.sh
    networks:
      - graylog
      - traefik-public
    volumes:
      - graylog_journal:/usr/share/graylog/data/journal
      - graylog_config:/usr/share/graylog/data/config
    restart: always
    deploy:
      mode: replicated
      replicas: 1
      labels:
          - traefik.enable=true
          - traefik.docker.network=traefik-public
          - traefik.constraint-label=traefik-public
          - traefik.http.routers.graylog-http.rule=Host( graylog.opapvd.
    server.ly )
          - traefik.http.routers.graylog-http.entrypoints=http
          - traefik.http.routers.graylog-http.middlewares=https-redirect
          - traefik.http.routers.graylog-https.rule=Host( graylog.opapvd.
    server.ly )
          - traefik.http.routers.graylog-https.entrypoints=https
          - traefik.http.routers.graylog-https.tls=true
          - traefik.http.routers.graylog-https.tls.certresolver=le
          - traefik.http.services.graylog.loadbalancer.server.port=9000
    depends_on:
      - mongodb
      - opensearch
    ports:
      # Graylog web interface and REST API
      - 9000:9000
      # Syslog UDP
      - 1514:1514/udp
  logspout:
    image: gliderlabs/logspout:latest
    environment:
      DOCKER_LABELS: 'on'
      MULTILINE_ENABLE_DEFAULT: 'false'
```

```
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
      - /etc/hostname:/etc/host_hostname:ro
    command:
      multiline+syslog://graylog:514?filter.name=opa_opa
    depends_on:
      - graylog
    networks:
      - graylog
    deploy:
      mode: global
      restart_policy:
        condition: on-failure
        delay: 10s
        window: 5s
      resources:
        limits:
          cpus: '0.10'
          memory: 256M
        reservations:
          memory: 64M
networks:
  graylog:
    driver: overlay
  traefik-public:
    external: true
volumes:
  mongo_data: {}
  es_data: {}
  graylog_journal: {}
  graylog_config: {}
```

Listing A.4: Graylog Docker Compose File

A.5 HTTP-API Docker Compose File

```
version: '3.4'

services:
  api_server:
    image: openpolicyagent/demo-restful-api:0.2
```

```
    ports:
      - "5000:5000"
    environment:
      - OPA_ADDR=http://opa:8181
      - POLICY_PATH=/v1/data/httpapi/authz
    deploy:
      mode: replicated
      replicas: 3
      update_config:
        parallelism: 1
        delay: 10s
      restart_policy:
        condition: on-failure
    networks:
      - opa
      - netowrk

  client:
    image: shumbashi/opa_http_api_client:0.3
    deploy:
      mode: replicated
      replicas: 1
      update_config:
        parallelism: 1
        delay: 10s
      restart_policy:
        condition: on-failure
    networks:
      - network

networks:
  opa:
    external: true
  network:
    driver: overlay
    attachable: true
```

Listing A.5: HTTP-API Application Docker Compose File

A.5.1 HTTP-API Attacker Docker Compose File

```
version: '3.4'

services:
  attacker:
    image: shumbashi/opa_http_api_client:0.7
    environment:
      - ROLE=attacker
    deploy:
      mode: replicated
      replicas: 1
      update_config:
        parallelism: 1
        delay: 10s
      restart_policy:
        condition: on-failure
    networks:
      - http-api_network

networks:
  http-api_network:
    external: true
```

Listing A.6: HTTP-API Attacker Application Docker Compose File

A.6 SSH-Sudo Docker Compose File

```
version: '3.4'

services:
  frontend-server:
    image: openpolicyagent/demo-pam
    networks:
      - opa
      - network
    deploy:
      mode: replicated
      replicas: 1
      update_config:
        parallelism: 1
        delay: 10s
      restart_policy:
```

```
        condition: on-failure
    configs:
      - source: frontend_host_id.json
        target: /etc/host_identity.json

  backend-server:
    image: openpolicyagent/demo-pam
    networks:
      - opa
      - network
    deploy:
      mode: replicated
      replicas: 1
      update_config:
        parallelism: 1
        delay: 10s
      restart_policy:
        condition: on-failure
    configs:
      - source: backend_host_id.json
        target: /etc/host_identity.json

  client:
    image: shumbashi/opa_ssh_sudo_client:0.8
    networks:
      - network
    deploy:
      mode: replicated
      replicas: 1
      update_config:
        parallelism: 1
        delay: 10s
      restart_policy:
        condition: on-failure

configs:
  frontend_host_id.json:
    external: true

  backend_host_id.json:
```

```
    external: true

networks:
  opa:
    external: true
  network:
    driver: overlay
    attachable: true
```

Listing A.7: SSH-Sudo Application Docker Compose File

A.6.1 SSH Attacker Docker Compose File

```
version: '3.4'

services:
  ssh_attacker:
    image: shumbashi/opa_ssh_sudo_client:0.9
    environment:
      - ROLE=attacker
      - ATTACK_TYPE=ssh
    networks:
      - ssh-sudo_network
    deploy:
      mode: replicated
      replicas: 1
      update_config:
        parallelism: 1
        delay: 10s
      restart_policy:
        condition: on-failure

networks:
  ssh-sudo_network:
    external: true
```

Listing A.8: SSH Attacker Application Docker Compose File

A.6.2 Sudo Attacker Docker Compose File

```
version: '3.4'
```

```
2
3 services:
4   sudo_attacker:
5     image: shumbashi/opa_ssh_sudo_client:0.9
6     environment:
7       - ROLE=attacker
8       - ATTACK_TYPE=sudo
9     networks:
10      - ssh-sudo_network
11    deploy:
12      mode: replicated
13      replicas: 1
14      update_config:
15        parallelism: 1
16        delay: 10s
17      restart_policy:
18        condition: on-failure
19
20 networks:
21   ssh-sudo_network:
22     external: true
```

Listing A.9: Sudo Attacker Application Docker Compose File

A.7 Event-Driven MSA Docker Compose File

```
1 version: '3.4'
2 services:
3   assemble:
4     image: 'shumbashi/msa_app:0.3'
5     command: bash -c 'python3 run_me_first.py swarm.ini    python3 msvc_
      assemble.py swarm.ini'
6     networks:
7       - kafka
8     volumes:
9       - pizza_logs:/usr/app/src/logs
10    deploy:
11      mode: replicated
12      replicas: 1
13      update_config:
14        parallelism: 1
```

```
      delay: 10s
    restart_policy:
      condition: on-failure

status:
  image: 'shumbashi/msa_app:0.3'
  command: bash -c 'python3 run_me_first.py swarm.ini    python3 msvc_
  status.py swarm.ini'
  networks:
    - kafka
  volumes:
    - pizza_dbs:/usr/app/src/db
    - pizza_logs:/usr/app/src/logs
  deploy:
    mode: replicated
    replicas: 1
    update_config:
      parallelism: 1
      delay: 10s
    restart_policy:
      condition: on-failure

bake:
  image: 'shumbashi/msa_app:0.3'
  command: bash -c 'python3 run_me_first.py swarm.ini    python3 msvc_
  bake.py swarm.ini'
  networks:
    - kafka
  volumes:
    - pizza_logs:/usr/app/src/logs
  deploy:
    mode: replicated
    replicas: 1
    update_config:
      parallelism: 1
      delay: 10s
    restart_policy:
      condition: on-failure

delivery:
```

```
    image: 'shumbashi/msa_app:0.3'
    command: bash -c 'python3 run_me_first.py swarm.ini    python3 msvc_
    delivery.py swarm.ini'
    networks:
      - kafka
    volumes:
      - pizza_dbs:/usr/app/src/db
      - pizza_logs:/usr/app/src/logs
    deploy:
      mode: replicated
      replicas: 1
      update_config:
        parallelism: 1
        delay: 10s
      restart_policy:
        condition: on-failure

  dashboard:
    image: 'shumbashi/msa_app:0.3'
    command: 'python3 webapp.py swarm.ini'
    ports:
      - "8000:8000"
    networks:
      - kafka
      - traefik-public
      - network
    volumes:
      - pizza_dbs:/usr/app/src/db
      - pizza_logs:/usr/app/src/logs
    deploy:
      mode: replicated
      replicas: 1
      update_config:
        parallelism: 1
        delay: 10s
      restart_policy:
        condition: on-failure
      labels:
        - traefik.enable=true
        - traefik.docker.network=traefik-public
```

```
        - traefik.constraint-label=traefik-public
        - traefik.http.routers.pizza-http.rule=Host( pizza.mydomain.ly )
        - traefik.http.routers.pizza-http.entrypoints=http
        - traefik.http.routers.pizza-http.middlewares=https-redirect
        - traefik.http.routers.pizza-https.rule=Host( pizza.mydomain.ly )
        - traefik.http.routers.pizza-https.entrypoints=https
        - traefik.http.routers.pizza-https.tls=true
        - traefik.http.routers.pizza-https.tls.certresolver=le
        - traefik.http.services.pizza.loadbalancer.server.port=8000
  client:
    image: shumbashi/opa_pizza_client:0.3
    environment:
      - DASHBOARD_URL=http://dashboard:8000
    networks:
      - network
    deploy:
      mode: replicated
      replicas: 1
      update_config:
        parallelism: 1
        delay: 10s
      restart_policy:
        condition: on-failure

networks:
  kafka:
    external: true
  traefik-public:
    external: true
  network:
    driver: overlay

volumes:
  pizza_dbs:
   external: true
  pizza_logs:
   external: true
```

Listing A.10: Event-Driven MSA Application Docker Compose File

A.7.1 Kafka Docker Compose File

```
version: '3.4'

services:
  zookeeper:
    image: confluentinc/cp-zookeeper:7.2.2
    ports:
      - "2181:2181"
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181
      ZOOKEEPER_TICK_TIME: 2000
    networks:
      - network
    deploy:
      mode: replicated
      replicas: 1
      update_config:
        parallelism: 1
        delay: 10s
      restart_policy:
        condition: on-failure

  broker:
    image: confluentinc/cp-server:7.2.2
    depends_on:
      - zookeeper
    ports:
      - "9092:9092"
      - "9101:9101"
    environment:
      KAFKA_OPA_AUTHORIZER_CACHE_EXPIRE_AFTER_SECONDS: 10
      KAFKA_OPA_AUTHORIZER_URL: http://opa:8181/v1/data/kafka/authz/allow
      KAFKA_AUTHORIZER_CLASS_NAME: org.openpolicyagent.kafka.
    OpaAuthorizer
      KAFKA_BROKER_ID: 1
      KAFKA_ZOOKEEPER_CONNECT: 'zookeeper:2181'
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: PLAINTEXT:PLAINTEXT,PLAINTEXT
    _HOST:PLAINTEXT
      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://broker:9094,PLAINTEXT_HOST
    ://broker:9092
```

```
      KAFKA_METRIC_REPORTERS: io.confluent.metrics.reporter.
    ConfluentMetricsReporter
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
      KAFKA_GROUP_INITIAL_REBALANCE_DELAY_MS: 0
      KAFKA_CONFLUENT_LICENSE_TOPIC_REPLICATION_FACTOR: 1
      KAFKA_CONFLUENT_BALANCER_TOPIC_REPLICATION_FACTOR: 1
      KAFKA_TRANSACTION_STATE_LOG_MIN_ISR: 1
      KAFKA_TRANSACTION_STATE_LOG_REPLICATION_FACTOR: 1
      KAFKA_JMX_PORT: 9101
      KAFKA_JMX_HOSTNAME: localhost
      KAFKA_CONFLUENT_SCHEMA_REGISTRY_URL: http://schema-registry:8081
      CONFLUENT_METRICS_REPORTER_BOOTSTRAP_SERVERS: broker:9094
      CONFLUENT_METRICS_REPORTER_TOPIC_REPLICAS: 1
      CONFLUENT_METRICS_ENABLE: 'true'
      CONFLUENT_SUPPORT_CUSTOMER_ID: 'anonymous'
      CLASSPATH: "/plugin/*"
    networks:
      - kafka
      - network
      - opa
    volumes:
      - kafka_plugins:/plugin
    deploy:
      mode: replicated
      replicas: 1
      update_config:
        parallelism: 1
        delay: 10s
      restart_policy:
        condition: on-failure

  schema-registry:
    image: confluentinc/cp-schema-registry:7.2.2
    depends_on:
      - broker
    ports:
      - "8081:8081"
    environment:
      SCHEMA_REGISTRY_HOST_NAME: schema-registry
      SCHEMA_REGISTRY_KAFKASTORE_BOOTSTRAP_SERVERS: 'broker:9094'
```

```
      SCHEMA_REGISTRY_LISTENERS: http://0.0.0.0:8081
    networks:
      - network
    deploy:
      mode: replicated
      replicas: 1
      update_config:
        parallelism: 1
        delay: 10s
      restart_policy:
        condition: on-failure

  control-center:
    image: confluentinc/cp-enterprise-control-center:7.2.2
    depends_on:
      - broker
      - schema-registry
    ports:
      - "9021:9021"
    environment:
      CONTROL_CENTER_BOOTSTRAP_SERVERS: 'broker:9094'
      CONTROL_CENTER_CONNECT_CONNECT-DEFAULT_CLUSTER: 'connect:8083'
      CONTROL_CENTER_KSQL_KSQLDB1_URL: "http://ksqldb-server:8088"
      CONTROL_CENTER_KSQL_KSQLDB1_ADVERTISED_URL: "http://localhost:8088"
      CONTROL_CENTER_SCHEMA_REGISTRY_URL: "http://schema-registry:8081"
      CONTROL_CENTER_REPLICATION_FACTOR: 1
      CONTROL_CENTER_INTERNAL_TOPICS_PARTITIONS: 1
      CONTROL_CENTER_MONITORING_INTERCEPTOR_TOPIC_PARTITIONS: 1
      CONFLUENT_METRICS_TOPIC_REPLICATION: 1
      PORT: 9021
    networks:
      - network
    deploy:
      mode: replicated
      replicas: 1
      update_config:
        parallelism: 1
        delay: 10s
      restart_policy:
        condition: on-failure
```

```
116
117 ksqldb-server:
118   image: confluentinc/cp-ksqldb-server:7.2.2
119   depends_on:
120     - broker
121   ports:
122     - "8088:8088"
123   environment:
124     KSQL_CONFIG_DIR: "/etc/ksql"
125     KSQL_BOOTSTRAP_SERVERS: "broker:9094"
126     KSQL_HOST_NAME: ksqldb-server
127     KSQL_LISTENERS: "http://0.0.0.0:8088"
128     KSQL_CACHE_MAX_BYTES_BUFFERING: 0
129     KSQL_KSQL_SCHEMA_REGISTRY_URL: "http://schema-registry:8081"
130     KSQL_PRODUCER_INTERCEPTOR_CLASSES: "io.confluent.monitoring.clients
  .interceptor.MonitoringProducerInterceptor"
131     KSQL_CONSUMER_INTERCEPTOR_CLASSES: "io.confluent.monitoring.clients
  .interceptor.MonitoringConsumerInterceptor"
132     KSQL_KSQL_CONNECT_URL: "http://connect:8083"
133     KSQL_KSQL_LOGGING_PROCESSING_TOPIC_REPLICATION_FACTOR: 1
134     KSQL_KSQL_LOGGING_PROCESSING_TOPIC_AUTO_CREATE: 'true'
135     KSQL_KSQL_LOGGING_PROCESSING_STREAM_AUTO_CREATE: 'true'
136   networks:
137     - kafka
138     - network
139   deploy:
140     mode: replicated
141     replicas: 1
142     update_config:
143       parallelism: 1
144       delay: 10s
145     restart_policy:
146       condition: on-failure
147
148 ksqldb-cli:
149   image: confluentinc/cp-ksqldb-cli:7.2.2
150   depends_on:
151     - broker
152     - ksqldb-server
153   entrypoint: /bin/sh
```

```
    tty: true
    networks:
      - network
    deploy:
      mode: replicated
      replicas: 1
      update_config:
        parallelism: 1
        delay: 10s
      restart_policy:
        condition: on-failure
networks:
  kafka:
    external: true
  network:
    driver: overlay
  opa:
    external: true

volumes:
  kafka_plugins:
   external: true
```

Listing A.11: Kafka Docker Compose File

A.7.2 MSA Attacker Docker Compose File

```
version: '3.4'

services:
  attacker:
    image: shumbashi/opa_msa_misuse_client:0.1
    environment:
      - ROLE=attacker
    deploy:
      mode: replicated
      replicas: 1
      update_config:
        parallelism: 1
        delay: 10s
      restart_policy:
```

```
        condition: on-failure
    networks:
      - kafka

networks:
  kafka:
    external: true
```

Listing A.12: MSA Attacker Docker Compose File