Republic of Tunisia
Ministry of Higher Education
and Scientific Research

University of Sfax

National School of Electronics
and Telecommunications of Sfax

Engineer in :
Telecommunictaions
Engineering

Option :
CyberSecurity

Ordre Nb: GT-aaa-0-00-00

Audit Report

Presented at

National School of Electronics and Telecommunications of Sfax

in order to

Validate the Audit process of the Firma'IQ architecture.

Option:
CyberSecurity

by

Achraf ben Abdallah
Ahmed Rekik
Heni Yousfi
Mayssa Larguech

Comprehensive Security and Performance Audit of a Microservices-Based Architecture: Enhancing Resilience, Automation, and Scalability

Presented on: 10/12/2024, before the jury of commission.

Acknowledgement

I would like to begin by expressing my heartfelt gratitude to my supervisors, whose invaluable guidance, encouragement, and constructive feedback have been instrumental in the successful completion of this project. Their expertise and willingness to share their knowledge have not only enriched this work but also provided me with insights that will undoubtedly benefit my future endeavors. I am deeply appreciative of the time and effort they invested in mentoring me throughout this journey.

I am also immensely grateful to the professors and administrative staff at ENET'Com for their unwavering support and for creating an environment conducive to research and innovation. Their dedication to providing access to resources, facilities, and academic expertise has played a significant role in enabling the progress of this project. Special thanks go to the technical team, who ensured that all necessary tools and platforms were available and operational during the project.

Furthermore, I would like to extend my deepest thanks to my colleagues and teammates, whose collaboration, enthusiasm, and commitment were vital in overcoming challenges and achieving project milestones. Their diverse perspectives and problem-solving skills added immense value to this endeavor. Lastly, my heartfelt appreciation goes to my family and friends for their endless support, encouragement, and understanding, which motivated me to persevere and complete this project with determination and optimism.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

CI/CD Continuous Integration and Continuous Deployment

CPU Central Processing Unit

GDPR General Data Protection Regulation

KPI Key Performance Indicator

mTLS Mutual Transport Layer Security

RBAC Role-Based Access Control

TLS Transport Layer Security

VPN Virtual Private Network

VPS Virtual Private Server

GENERAL INTRODUCTION

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed non risus. Suspendisse lectus tortor, dignissim sit amet, adipiscing nec, ultricies sed, dolor. Cras elementum ultrices diam. Maecenas ligula massa, varius a, semper congue, euismod non, mi. Proin porttitor, orci nec nonummy molestie, enim est eleifend mi, non fermentum diam nisl sit amet erat. Duis semper. Duis arcu massa, scelerisque vitae, consequat in, pretium a, enim. Pellentesque congue. Ut in risus volutpat libero pharetra tempor. Cras vestibulum bibendum augue. Praesent egestas leo in pede. Praesent blandit odio eu enim. Pellentesque sed dui ut augue blandit sodales. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Aliquam nibh. Mauris ac mauris sed pede pellentesque fermentum. Maecenas adipiscing ante non diam sodales hendrerit.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed non risus. Suspendisse lectus tortor, dignissim sit amet, adipiscing nec, ultricies sed, dolor. Cras elementum ultrices diam. Maecenas ligula massa, varius a, semper congue, euismod non, mi. Proin porttitor, orci nec nonummy molestie, enim est eleifend mi, non fermentum diam nisl sit amet erat. Duis semper. Duis arcu massa, scelerisque vitae, consequat in, pretium a, enim. Pellentesque congue. Ut in risus volutpat libero pharetra tempor. Cras vestibulum bibendum augue. Praesent egestas leo in pede. Praesent blandit odio eu enim. Pellentesque sed dui ut augue blandit sodales. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Aliquam nibh. Mauris ac mauris sed pede pellentesque fermentum. Maecenas adipiscing ante non diam sodales hendrerit.

Chapter

1 Introduction and Context

Contents

1.1 Objectives and Scope of the Audit

1.1.1 Purpose of the Audit

The adoption of microservices architectures and DevOps methodologies has revolutionized how modern applications are developed, deployed, and maintained. However, this shift also introduces new challenges, particularly in ensuring that systems are secure, scalable, and operationally efficient. This audit was initiated to comprehensively evaluate the architecture, deployment practices, and operational readiness of the system under review.

The primary purpose of this audit is to validate that the system is:

- **Secure**: Ensuring that sensitive data is protected and potential attack surfaces are minimized.
- **Scalable**: Confirming that the system can efficiently handle increased workloads and future growth without performance degradation.
- **Automated**: Minimizing manual intervention and human error through robust automation of deployment and management tasks.
- **Resilient**: Ensuring fault tolerance and rapid recovery from failures to maintain high availability.

The findings from this audit will provide actionable insights into potential vulnerabilities, inefficiencies, and opportunities for improvement. These insights will help ensure that the system aligns with industry best practices and organizational goals.

High-Level Objectives:

- *Identify Weaknesses*: Evaluate areas such as security, infrastructure, CI/CD pipelines, and microservices design to uncover gaps.
- *Enforce Best Practices*: Validate adherence to standards like ISO/IEC 27001, OWASP, and Kubernetes best practices.

- *Enable Continuous Improvement*: Establish a feedback loop for iterative enhancements based on audit findings.
- *Support Business Goals*: Ensure the technical architecture aligns with business priorities such as speed, reliability, and cost-efficiency.

1.1.2 Audit Scope

The audit's scope was carefully defined to focus on the most critical aspects of the system. Four key domains were identified:

a) Infrastructure

The underlying infrastructure, hosted on OVH VPS, consists of two virtual machines configured using Vagrant:

- **Master Node**: Responsible for managing the Kubernetes control plane, including scheduling, monitoring, and cluster state maintenance.
- **Worker Node**: Hosts containerized workloads, including microservices, databases, and external APIs.

Key areas evaluated:

- *Resource Allocation*: Ensuring optimal CPU, memory, and disk utilization.
- *Automation*: Assessing the effectiveness of Ansible playbooks in provisioning and configuration.

b) Applications

The system is built around **seven distinct microservices**, each performing specific business functions. The audit examined:

- *Modularity*: Independence and loose coupling of services.

- *Scalability*: Readiness for increased workloads through horizontal and vertical scaling.
- *Security*: Vulnerabilities within containerized services and their communication protocols.

c) CI/CD Pipelines

The CI/CD pipelines facilitate seamless integration and deployment. The audit focused on:

- *Jenkins Pipelines*: Efficiency of builds, tests, and image creation workflows.
- *Argo CD*: Automation of deployments and rollback capabilities.

d) Security

Key aspects reviewed included:

- Kubernetes RBAC (Role-Based Access Control): Ensuring minimal privileges.
- Docker Image Security: Vulnerability scanning and base image optimization.
- Secret Management: Secure storage of sensitive data using Kubernetes Secrets.

This scope provided a well-rounded evaluation while maintaining focus on actionable insights.

1.2 Description of the Project

1.2.1 Infrastructure Setup

The infrastructure was designed with a focus on scalability, reliability, and automation. OVH VPS served as the hosting environment, leveraging virtualization and containerization technologies.

a) Hosting Environment

- **Master Node**: Controls Kubernetes orchestration by managing the API server, scheduler, and etcd.

- **Worker Node**: Executes workloads, hosting containerized microservices and handling inter-service communications.

b) Configuration Automation with Ansible

Ansible playbooks automated the setup of the environment:

- Installed Docker and Kubernetes components.
- Configured Kubernetes clusters using `kubeadm`.
- Automated node registration to the cluster, reducing manual errors and ensuring consistency.

c) Networking

- Kubernetes network policies restricted inter-pod communication to authorized components only.
- Encrypted traffic ensured secure communications between nodes and services.

1.2.2 Microservices Architecture

a) Design Principles

- **Single Responsibility**: Each service was responsible for a specific function.
- **Autonomy**: Services operated independently, ensuring resilience to failures.
- **Scalability**: Horizontal scaling was configured to handle varying loads efficiently.

b) Containerization with Docker

- Lightweight base images like Alpine Linux were used to reduce vulnerabilities.
- Multi-stage builds optimized image sizes and removed unnecessary layers.

c) Deployment with Kubernetes

- *Declarative Configurations*: Kubernetes manifests defined deployments, services, and ingress routes.
- *Secrets Management*: ConfigMaps and Secrets securely handled sensitive data like API keys and credentials.

1.2.3 CI/CD Pipeline

a) Continuous Integration (CI)

- **Jenkins Pipelines**: Automated builds and static code analysis with SonarQube.
- **Docker Image Security**: Images were scanned for vulnerabilities before deployment.

b) Continuous Delivery (CD)

- **Argo CD**: Automated deployment of container images to Kubernetes clusters.
- **Rollback Capabilities**: Ensured recovery from deployment failures.

1.3 Methodology

1.3.1 Techniques Employed

The audit combined several methods to ensure a thorough evaluation:

- *Log Analysis*: CI/CD pipeline logs were reviewed for errors and inefficiencies.
- *Configuration Reviews*: Ansible playbooks and Kubernetes manifests were examined against best practices.
- *Security Assessments*: Tools like SonarQube and Trivy were used for vulnerability detection.

1.3.2 Tools and Standards

Industry standards and modern tools were employed:

- **ISO/IEC 27001**: Ensured compliance with information security management standards.
- **OWASP Guidelines**: Addressed web application vulnerabilities.
- **Kubernetes Best Practices**: Ensured efficient and secure cluster configurations.

1.3.3 Challenges and Solutions

a) Challenge 1: Managing Secrets and Sensitive Data

Description of the Problem: A common vulnerability in modern system architectures is the improper handling of secrets such as API keys, database credentials, and other sensitive information. During the audit, it was observed that:

- Secrets were being stored in plain text within Jenkins pipelines (`Jenkinsfile`) or in Kubernetes manifest files.
- There was no centralized mechanism to securely manage or rotate secrets, increasing the risk of exposure due to human error or system breaches.

Risks Identified:

- Unencrypted secrets in files could be accessed by unauthorized individuals.
- Static secrets (e.g., hardcoded API keys) increase exposure time and the attack surface.

Solution Implemented:

- **Kubernetes Secrets:** Sensitive data was migrated to Kubernetes Secrets, which are natively encrypted and accessible only by authorized pods. Examples include:
  - Database credentials used by microservices.

  - API keys for external integrations (e.g., Weather API, Satellite Imagery API).

- **HashiCorp Vault:** A dedicated secret management tool was introduced to further enhance security. Features included:
  - Dynamic secret generation (e.g., temporary database credentials that expire after a set duration).
  - Rotational policies to automatically update secrets and ensure freshness.
- **RBAC Enforcement:** Role-Based Access Control (RBAC) policies restricted access to secrets. Only specific Kubernetes Service Accounts were allowed to retrieve relevant secrets.

b) Challenge 2: Ensuring Secure Docker Images

Description of the Problem: Docker images used for containerizing microservices were found to contain unnecessary layers or packages, which:

- Increased the attack surface.
- Introduced vulnerabilities due to outdated dependencies.

Risks Identified:

- Larger image sizes increased deployment times and storage requirements.
- Vulnerable packages within Docker images created opportunities for exploitation.

Solution Implemented:

- **Multi-Stage Builds:** Dockerfiles were refactored to use multi-stage builds, resulting in:
  - Smaller final images containing only runtime dependencies.
  - Removal of intermediate artifacts, reducing the attack surface.

- **Regular Vulnerability Scans:** Tools such as Trivy and Docker's own vulnerability scanner were integrated into the CI/CD pipeline to:
  - Identify and report known vulnerabilities.
  - Ensure that only secure images were pushed to Docker Hub.
- **Base Image Optimization:** Switched to minimal base images like Alpine Linux to further reduce unnecessary dependencies.

c) Challenge 3: Automating the Deployment Process

Description of the Problem: Manual deployment processes often lead to inconsistencies, errors, and delays. The audit revealed:

- Some steps in the deployment process were still reliant on human intervention, particularly during Kubernetes manifest validation.
- Rollback mechanisms were insufficiently automated, making recovery from failed deployments cumbersome.

Risks Identified:

- Human errors in deployment configurations or commands could lead to downtime.
- Delayed rollbacks during failed deployments increased system unavailability.

Solution Implemented:

- **Argo CD Integration:** Argo CD was introduced to manage declarative, GitOps-based deployments. Features included:
  - Automated synchronization of Kubernetes manifests from a Git repository.
  - Rollbacks to the last successful deployment in case of failures.
- **Pipeline Enhancements:** Jenkins pipelines were extended to include:

  - Automated validation of Kubernetes manifests using tools like `kubeval` and `kube-linter`.
  - Integration with Argo CD to trigger deployments directly after successful pipeline completion.

d) Challenge 4: Optimizing CI/CD Pipelines for All Microservices

Description of the Problem: The audit revealed inefficiencies in the CI/CD pipeline, including:

- Static pipeline configurations that did not adapt to the unique requirements of each microservice.
- Lack of comprehensive testing, particularly for integration and end-to-end scenarios.

Risks Identified:

- Failure to catch integration issues early led to increased debugging time.
- Static pipelines increased maintenance efforts when microservices evolved.

Solution Implemented:

- **Microservice-Specific Pipelines:** Each microservice was assigned a dedicated Jenkinsfile, tailored to its specific requirements. Steps included:
  - Unit and integration tests for each service.
  - Static code analysis with SonarQube to ensure quality gates were met.
- **Testing Enhancements:**
  - JUnit for robust unit testing.
  - End-to-end testing frameworks to simulate real-world scenarios.
- **Pipeline Templates:** Common pipeline templates were created to ensure consistency while allowing customization for each service.

e) Challenge 5: Aligning Security with Kubernetes RBAC Policies

Description of the Problem: The Kubernetes cluster's RBAC policies were overly permissive, granting broad access rights to users and services. This created potential risks of:

- Unauthorized access to sensitive resources.
- Accidental misconfigurations by users with elevated permissions.

Risks Identified:

- Breach of sensitive resources due to compromised service accounts.
- Difficulty in auditing access due to poorly defined roles.

Solution Implemented:

- **Principle of Least Privilege:** RBAC roles were redefined to minimize permissions granted to users and services.
- **Service Account Isolation:** Each microservice was assigned its own dedicated Service Account with specific permissions.
- **Auditing and Monitoring:** Tools like `kube-audit` were configured to log access attempts and identify anomalies.

f) Challenge 6: Scaling the Infrastructure for Future Growth

Description of the Problem: The current single-node setup for the worker node was insufficient to handle future growth or increased workloads.

Risks Identified:

- Limited scalability due to hardware constraints.
- Single points of failure for workloads running on the worker node.

Solution Implemented:

- **Cluster Expansion:** Migrated to a multi-node Kubernetes cluster to distribute workloads more effectively.
- **Horizontal Pod Autoscaler (HPA):** Configured HPA to automatically adjust the number of replicas for each microservice based on resource usage (e.g., CPU, memory).
- **Cloud-Native Storage:** Integrated scalable storage solutions, such as Persistent Volumes, to handle increased data storage demands.

g) Challenge 7: Monitoring and Observability

Description of the Problem: The system lacked sufficient monitoring and observability, making it difficult to:

- Proactively identify issues before they impacted users.
- Gain insights into system performance and resource utilization.

Risks Identified:

- Prolonged downtime due to delayed incident detection.
- Limited visibility into the root cause of performance issues.

Solution Implemented:

- **Prometheus and Grafana:** Introduced as the monitoring and visualization stack for:
  - Tracking system metrics (e.g., CPU, memory, disk usage).
  - Setting up alerts for abnormal conditions (e.g., high resource usage, pod failures).
- **Distributed Tracing:** Integrated tools like Jaeger for tracing requests across microservices, identifying performance bottlenecks.

1.4 Conclusion

This chapter outlined the objectives, scope, and methodology of the audit, addressing key challenges in the system architecture. Solutions such as Kubernetes Secrets, HashiCorp Vault, and multi-stage Docker builds enhanced security, while CI/CD optimizations improved deployment efficiency. These implementations strengthened the foundation of the microservices-based application, ensuring scalability, security, and automation.

By addressing critical vulnerabilities and integrating advanced monitoring tools like Prometheus and Grafana, the system is now better equipped to handle operational demands and future growth. These findings establish a solid baseline for the next steps in technical analysis and long-term optimization strategies.

Chapter

2 Analysis of Risks and Evaluation of the Architecture

Contents

2.1 Overview of the Audit Findings

This chapter provides a comprehensive analysis of the security, scalability, and operational resilience of the Firma'IQ microservices architecture. The audit evaluated the system across six critical domains:

- Access control and authentication.
- Network security.
- Data security.
- CI/CD pipeline security.
- Microservices architecture risks.
- Container and Kubernetes security.

Each domain was assessed for vulnerabilities, potential impacts, and alignment with industry best practices. This analysis forms the foundation for implementing a prioritized action plan to enhance the architecture and ensure production readiness.

2.2 Analysis of Identified Risks

This section categorizes and elaborates on each risk identified during the audit. By addressing these vulnerabilities, the architecture can achieve enhanced security, performance, and scalability.

2.2.1 Access Control and Authentication

a) Identified Risks

Access control is critical to protecting sensitive resources and maintaining the integrity of the system. The audit revealed:

- **No Microservices Authentication:** Many microservices lacked authentication mechanisms such as OAuth 2.0 or JSON Web Tokens (JWT).
- **Overly Permissive RBAC Policies:** Kubernetes RBAC configurations allowed users and service accounts unnecessary access to resources.
- **Weak SSH Key Management:** A single shared SSH key was used across multiple systems without rotation or revocation policies.

b) Implications

- **Unauthorized Access:** Lacking microservices authentication could expose endpoints to unauthorized users.
- **Privilege Escalation:** Weak RBAC configurations allow attackers to escalate privileges within the cluster.
- **Key Compromise:** Poor SSH key management creates a single point of failure, risking unauthorized system access.

c) Recommendations

- Implement OAuth 2.0 and JWT for authenticating requests to microservices.
- Refine RBAC policies to enforce the principle of least privilege.
- Introduce SSH key management with automated rotation and unique keys per user.

2.2.2 Network Security

a) Identified Risks

The network security assessment revealed vulnerabilities that could compromise data integrity and confidentiality:

- **Unsegmented Environments:** Development, testing, and production environments shared the same network.
- **Unsecured Inter-Service Communication:** Microservices communicated without TLS encryption.
- **Unverified VPN Protocols:** VPN configurations lacked defined encryption standards.

b) Implications

- Unsegmented networks increase the risk of data exposure between environments.
- Unencrypted communication is vulnerable to interception, compromising data confidentiality.
- Weak VPN protocols leave traffic susceptible to brute-force and man-in-the-middle attacks.

c) Recommendations

- Use Kubernetes namespaces to segment environments and apply network policies to restrict inter-pod communication.
- Integrate a service mesh (e.g., Istio) to encrypt inter-service traffic with mTLS.
- Adopt secure VPN standards such as OpenVPN with AES-256 encryption.

2.2.3 Data Security

a) Identified Risks

Critical gaps in data protection mechanisms included:

- **Unencrypted Data at Rest:** Sensitive data was stored without encryption.
- **Lack of Audit Logging:** Database access and modifications were not logged.
- **Compliance Gaps:** GDPR compliance measures were either undefined or poorly implemented.

b) Implications

- Unencrypted data increases the risk of exposure during unauthorized access.
- Lack of audit logs impedes incident detection and forensic analysis.
- Non-compliance with regulations risks heavy fines and reputational damage.

c) Recommendations

- Encrypt data at rest using AES-256 and enforce TLS 1.2 or higher for data in transit.
- Enable database audit logs to record all access and modifications.
- Perform regular GDPR audits and implement measures like data anonymization where required.

2.2.4 CI/CD Pipeline Security

a) Identified Risks

CI/CD pipelines were critical to the deployment process but introduced several vulnerabilities:

- **Secrets in Plain Text:** Jenkinsfiles contained hardcoded credentials.
- **No Automated Scanning:** Docker images and code repositories were not scanned for vulnerabilities.
- **Weak SonarQube Configurations:** Quality gates allowed high-severity vulnerabilities to pass.

b) Implications

- Exposed secrets could lead to unauthorized access and privilege escalation.
- Vulnerable components might enter production, increasing the system's attack surface.

- Weak quality gates could allow subpar or insecure code to pass into production.

c) Recommendations

- Integrate HashiCorp Vault for secrets management.
- Add Trivy to CI/CD pipelines to scan Docker images for known vulnerabilities.
- Strengthen SonarQube policies to block builds containing high-severity issues.

2.2.5 Microservices Risks

a) Identified Risks

The modularity of the microservices architecture introduced specific risks:

- **Vulnerabilities in Docker Images:** Images lacked regular scanning for known vulnerabilities.
- **Overly Large Images:** Containers included unnecessary dependencies and libraries.
- **Inconsistent Communication Patterns:** Over-reliance on synchronous calls caused bottlenecks under load.
- **External Dependency Failures:** APIs such as Weather API lacked retries or fallback mechanisms.
- **Limited Observability:** Monitoring and logging of microservices were fragmented across tools.

b) Implications

- Unscanned images risk introducing exploitable vulnerabilities.
- Large images slow down deployment times and increase storage costs.
- Synchronous communication reduces fault tolerance and increases latency.

- Unhandled API failures propagate system-wide disruptions.
- Insufficient logging complicates debugging and issue resolution.

c) Recommendations

- Secure Docker images by integrating vulnerability scanning into CI/CD pipelines.
- Optimize images using multi-stage builds and minimal base images like Alpine Linux.
- Transition to asynchronous communication with Kafka or RabbitMQ for critical workflows.
- Implement circuit breakers for API failures using libraries like Resilience4j.
- Centralize logging with the ELK Stack and enable distributed tracing with Jaeger.

2.2.6 Container and Kubernetes Security

a) Identified Risks

The container orchestration layer presented risks, including:

- **Public Registries:** Containers were pulled from public registries without verification.
- **Unrestricted Kubernetes API Access:** No IP whitelisting or authentication was enforced.
- **Undefined Resource Quotas:** Namespaces lacked CPU and memory limits, risking cluster instability.

b) Implications

- Public registries increase exposure to compromised images.
- Weak API server security risks unauthorized access to the entire cluster.
- Undefined quotas could lead to resource exhaustion during traffic surges.

c) Recommendations

- Use private registries with integrated vulnerability scanning.
- Enforce IP whitelisting and role-based authentication for API server access.
- Define resource quotas to manage CPU, memory, and storage utilization per namespace.

2.3 Risk Prioritization and Evaluation

Risk prioritization is essential to allocate resources effectively and address the most critical vulnerabilities first. In this section, we evaluate each identified risk based on its likelihood of occurrence and its potential impact on the system. The prioritization is guided by a quantitative risk assessment model, assigning scores to each risk and visualizing the results for clear decision-making.

2.3.1 Risk Assessment Methodology

The assessment methodology is based on two primary factors:

- **Likelihood:** The probability of the risk materializing, evaluated on a scale from 1 (rare) to 5 (almost certain).
- **Impact:** The severity of the consequences if the risk occurs, evaluated on a scale from 1 (negligible) to 5 (catastrophic).

Each risk is assigned a **risk rating**, calculated as:

$$\text{Risk Rating} = \text{Likelihood} \times \text{Impact}$$

This approach ensures a structured and repeatable evaluation process.

2.3.2 Risk Scoring and Categorization

The risks identified during the audit were scored based on the above methodology. To prioritize effectively, risks were categorized into the following levels:

- **Critical (Rating: 20–25):** Immediate action required to mitigate the risk.
- **High (Rating: 15–19):** Action should be taken as a priority.
- **Medium (Rating: 10–14):** Mitigation is recommended within planned cycles.
- **Low (Rating: 1–9):** Monitor and address if resources allow.

2.3.3 Risk Evaluation Table

The table below summarizes the evaluated risks, their likelihood, impact, and overall rating:

Risk Description	**Likelihood (1-5)**	**Impact (1-5)**	**Risk Rating**
Unsecured Microservices Access (No authentication or RBAC policies)	5	5	25
Lack of Network Segmentation Between Environments	5	4	20
Secrets Stored in Plain Text in Jenkins Pipelines	4	4	16
Unscanned Docker Images for Known Vulnerabilities	4	4	16
Undefined Resource Quotas in Kubernetes Namespaces	3	4	12
External API Failures Without Retry Mechanisms	3	3	9

Table 2.1: Risk Prioritization Table

2.3.4 Visualization of Risk Prioritization

To provide a clearer understanding of the risks and their relative priorities, a heatmap is presented below. The x-axis represents the likelihood of a risk occurring, while the y-axis represents its

impact. Risks falling in the top-right quadrant are considered critical and require immediate attention.

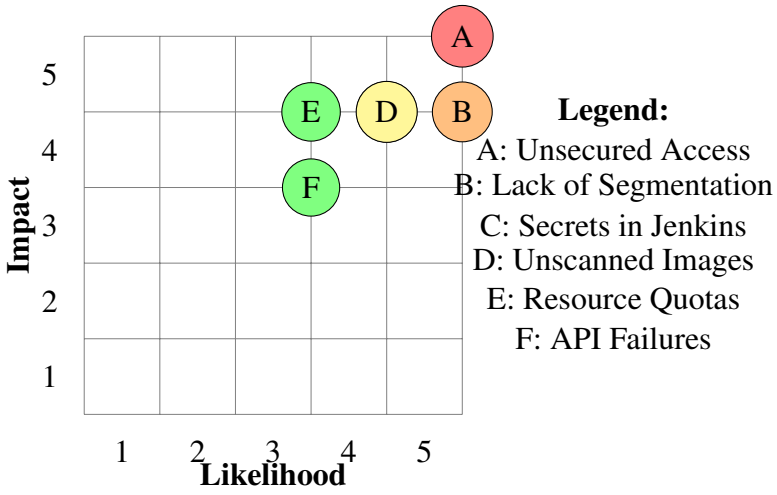

Figure 2.1: Risk Prioritization Heatmap

2.3.5 Insights and Actionable Priorities

From the evaluation:

- **Critical Risks (A, B):** Risks related to unsecured microservices access and network segmentation require immediate mitigation. These pose the highest potential for data breaches and unauthorized access.
- **High Risks (C, D):** Address secrets management and implement regular vulnerability scanning for Docker images. These are essential for securing the CI/CD pipeline and containerized workloads.
- **Medium Risks (E, F):** Define resource quotas in Kubernetes namespaces and strengthen external API handling with retry and fallback mechanisms.

2.3.6 Benefits of Risk Prioritization

Risk prioritization ensures that the organization:

- Allocates resources effectively to address the most pressing vulnerabilities.

- Reduces the attack surface by tackling high-likelihood, high-impact risks first.
- Maintains a systematic approach to long-term security improvements.

This structured evaluation provides a roadmap for aligning technical improvements with organizational goals.

2.3.7 Audit Test Results

During the audit process, various tests were conducted to assess the system's security posture, performance, and compliance with industry standards. The key findings from these tests are summarized below:

- **Operating System Hardening:** The system achieved a score of 430 out of 555, with recommendations to upgrade three vulnerable packages, configure a firewall, and enhance password policies.
- **Intrusion Detection:** No intrusion detection software was detected, requiring immediate action to integrate tools like Fail2Ban.
- **Network Security:** DNS configurations were verified as secure, but NodePort services exposed public ports, posing a potential risk.
- **Container Security:** Outdated Docker and Kubernetes versions were identified, requiring upgrades to the latest stable releases.

For detailed screenshots of the test outputs, refer to Appendix A.3.

2.4 Conclusion

This chapter detailed the vulnerabilities across the Firma'IQ architecture, including access control, network security, microservices, and Kubernetes configurations. By implementing the proposed recommendations, the system can achieve enhanced security, operational resilience, and compliance. The next chapter will focus on a detailed action plan to address these risks.

Chapter

3 Recommendations and Action Plan

Contents

3.1 Introduction

Based on the findings and risk prioritization detailed in Chapter 2, this chapter presents a comprehensive set of recommendations and a detailed action plan to address identified vulnerabilities. The recommendations focus on improving the system's security, scalability, and operational resilience while aligning with industry best practices. Each recommendation includes specific actions, proposed tools, estimated timelines, and measurable success criteria.

3.2 Recommendations by Domain

3.2.1 Access Control and Authentication

a) Overview of Issues

Access control vulnerabilities, including weak RBAC policies and lack of microservices authentication, were identified as critical risks. These gaps expose the system to unauthorized access and potential privilege escalation.

b) Recommended Actions

- **Implement OAuth 2.0 for Microservices Authentication:**
  - Introduce a centralized authentication server, such as Keycloak, to manage OAuth tokens.
  - Enforce token validation in all microservices using libraries like Spring Security or Express.js middleware.
- **Strengthen Kubernetes RBAC Policies:**
  - Define roles based on the principle of least privilege.

  - Apply namespace-specific roles to segregate access for development, testing, and production environments.

- **Automate SSH Key Management:**
  - Use HashiCorp Vault or AWS Systems Manager for automated key rotation.
  - Enforce unique keys per user and enable Multi-Factor Authentication (MFA) for SSH access.

c) Estimated Timeline

- OAuth 2.0 implementation: 4 weeks.
- RBAC policy refinement: 2 weeks.
- Automated SSH management: 3 weeks.

d) Success Criteria

- All microservices require OAuth tokens for access.
- Unauthorized access attempts are logged and blocked.
- SSH keys are rotated every 30 days, and MFA is enforced for all users.

3.2.2 Network Security

a) Overview of Issues

The lack of network segmentation and unencrypted inter-service communication increases the risk of data breaches and unauthorized access.

b) Recommended Actions

- **Introduce Network Segmentation:**

  - Use Kubernetes namespaces to isolate development, testing, and production environments.
  - Define network policies to restrict communication between namespaces.

- **Implement a Service Mesh:**
  - Deploy Istio to enforce mutual TLS (mTLS) between microservices.
  - Configure fine-grained traffic control to monitor and limit inter-service requests.

- **Enhance VPN Security:**
  - Adopt OpenVPN or WireGuard with AES-256 encryption.
  - Enforce strong authentication mechanisms for all VPN users.

c) Estimated Timeline

- Network segmentation and policies: 3 weeks.
- Service mesh deployment: 5 weeks.
- VPN configuration: 2 weeks.

d) Success Criteria

- Separate network zones exist for all environments.
- All inter-service communication is encrypted using mTLS.
- VPN connections comply with defined encryption and authentication standards.

3.2.3 Data Security

a) Overview of Issues

Unencrypted data and lack of audit logging present significant risks to data integrity and compliance.

b) Recommended Actions

- **Encrypt Data at Rest and in Transit:**
  - Use AES-256 for database encryption and configure Transparent Data Encryption (TDE) where supported.
  - Enforce TLS 1.2 or higher for all inter-service communication.
- **Enable Audit Logging:**
  - Activate database audit logs to record read, write, and administrative operations.
  - Centralize audit logs using a logging framework like the ELK Stack.
- **Ensure Compliance with GDPR:**
  - Conduct a GDPR compliance review.
  - Implement data anonymization and pseudonymization for sensitive personal information.

c) Estimated Timeline

- Data encryption: 3 weeks.
- Audit logging: 2 weeks.
- GDPR compliance measures: 6 weeks.

d) Success Criteria

- All sensitive data is encrypted at rest and in transit.
- Audit logs capture 100% of critical database operations.
- A compliance report is generated to verify GDPR adherence.

3.2.4 CI/CD Pipeline Security

a) Overview of Issues

The absence of secrets management and vulnerability scanning within the CI/CD pipeline introduces risks during software deployment.

b) Recommended Actions

- **Integrate Secrets Management:**
  - Use HashiCorp Vault to securely store and retrieve credentials in Jenkins pipelines.
  - Enforce strict access controls for accessing secrets.
- **Automate Vulnerability Scanning:**
  - Integrate Trivy to scan Docker images for vulnerabilities during builds.
  - Add SonarQube plugins for detecting security issues in code.
- **Strengthen Quality Gates:**
  - Configure SonarQube to block builds with high-severity issues or poor test coverage.

c) Estimated Timeline

- Secrets management: 2 weeks.
- Vulnerability scanning integration: 2 weeks.
- SonarQube quality gates: 1 week.

d) Success Criteria

- No secrets are stored in plain text within Jenkins pipelines.

- All builds pass vulnerability scans before deployment.
- Quality gates enforce strict thresholds for code security and test coverage.

3.2.5 Microservices Architecture

a) Overview of Issues

Risks include unscanned Docker images, inconsistent communication patterns, and insufficient observability for debugging and performance analysis.

b) Recommended Actions

- **Secure Docker Images:**
  - Scan Docker images using Trivy for vulnerabilities before deployment.
  - Use minimal base images like Alpine Linux to reduce attack surfaces.
- **Enhance Communication Resilience:**
  - Shift from synchronous calls to asynchronous patterns using Kafka.
  - Implement circuit breakers for API failures using Resilience4j.
- **Improve Observability:**
  - Deploy distributed tracing tools like Jaeger to trace requests across microservices.
  - Use centralized logging platforms, such as the ELK Stack, for comprehensive log aggregation.

c) Estimated Timeline

- Docker image security: 3 weeks.
- Communication enhancements: 4 weeks.
- Observability improvements: 5 weeks.

d) Success Criteria

- All Docker images are scanned and free from critical vulnerabilities.
- Asynchronous communication reduces API failure rates by 50%.
- Distributed tracing and centralized logging improve debugging times by 30%.

3.2.6 Container and Kubernetes Security

a) Overview of Issues

Public container registries and weak Kubernetes configurations increase the risk of unauthorized access and resource exhaustion.

b) Recommended Actions

- **Transition to Private Registries:**
  - Use Harbor or AWS Elastic Container Registry (ECR) to manage private images.
- **Secure Kubernetes API Access:**
  - Enforce IP whitelisting and mutual TLS authentication for API server access.
- **Define Resource Quotas:**
  - Apply CPU, memory, and storage limits per namespace to prevent resource exhaustion.

c) Estimated Timeline

- Private registry setup: 3 weeks.
- Kubernetes API security: 2 weeks.
- Resource quota configuration: 2 weeks.

d) Success Criteria

- All container images are stored in private registries.
- Unauthorized API server access attempts are blocked and logged.
- Resource utilization remains within defined quotas across all namespaces.

3.3 Integrated Action Plan

To ensure timely execution, the following integrated action plan consolidates all recommendations and assigns responsibilities.

3.3.1 Integrated Action Plan Timeline

To ensure a structured and effective implementation of the recommendations outlined in this report, the project timeline is divided into three distinct phases: **Preparation and Initial Setup**, **Core Implementation**, and **Monitoring and Optimization**. This phased approach ensures each task is addressed systematically, with measurable progress tracked throughout the project duration from **September 10th, 2024**, to **December 10th, 2024**.

a) Phase 1: Preparation and Initial Setup (September 10th – September 30th, 2024)

This phase establishes the foundation for implementation by aligning the project team, configuring essential tools, and assessing the current system state.

Key Activities:

- **Team Onboarding:**
  - Assign roles and responsibilities for Security, DevOps, and Data Protection teams.
  - Conduct a kick-off meeting to align stakeholders on the objectives, scope, and timeline.

- **Tool Configuration:**
  - Deploy monitoring tools, including Prometheus, Grafana, and the ELK Stack, within the Kubernetes environment.
  - Configure initial dashboards to track resource utilization, error rates, and system health.
- **Baseline Assessments:**
  - Conduct a baseline security audit using Trivy, kube-bench, and SonarQube.
  - Document current system performance metrics, such as API latency, uptime, and resource usage.
- **Training and Workshops:**
  - Organize hands-on sessions for the project team on tools like Prometheus, Jaeger, and Falco to ensure familiarity with their functionality.

Deliverables:

- A fully operational monitoring stack with basic dashboards.
- A baseline report detailing system performance and security metrics.
- A trained team prepared for implementation tasks.

b) Phase 2: Core Implementation (October 1st – November 20th, 2024)

This phase focuses on implementing core recommendations, including access control enhancements, network security measures, CI/CD pipeline hardening, and microservices optimization.

Key Activities:

- **Access Control Enhancements (October 1st – October 10th):**
  - Implement OAuth 2.0 authentication for microservices.

  - Refine Kubernetes RBAC policies to enforce least-privilege access.
  - Automate SSH key management and enable Multi-Factor Authentication (MFA) for critical access points.

- **Network Security Improvements (October 11th – October 20th):**
  - Segment environments by creating separate Kubernetes namespaces for development, testing, and production.
  - Deploy Istio as a service mesh to enforce mutual TLS (mTLS) and traffic control.
  - Strengthen VPN configurations using OpenVPN or WireGuard with AES-256 encryption.

- **CI/CD Pipeline Security (October 21st – November 5th):**
  - Integrate HashiCorp Vault for secure secrets management in Jenkins pipelines.
  - Automate vulnerability scanning for Docker images using Trivy.
  - Enhance SonarQube quality gates to block builds with critical security issues.

- **Microservices Optimization (November 6th – November 20th):**
  - Optimize Docker images with multi-stage builds to reduce size and vulnerabilities.
  - Transition to asynchronous communication using Kafka for resilient workflows.
  - Deploy Jaeger for distributed tracing to debug interactions between microservices.

Deliverables:

- Implemented OAuth 2.0, RBAC policies, and automated SSH management.
- Fully segmented network with namespaces secured by a service mesh.
- CI/CD pipelines with enhanced secrets management and security checks.
- Optimized microservices architecture with secure images and improved communication.

c) Phase 3: Monitoring and Optimization (November 21st – December 10th, 2024)

This phase focuses on evaluating implemented solutions, refining monitoring systems, and preparing the system for long-term sustainability.

Key Activities:

- **Monitoring Enhancements (November 21st – November 30th):**
  - Expand Prometheus and Grafana dashboards to include detailed metrics for API latency, resource utilization, and pod health.
  - Configure advanced alerts integrated with Slack or PagerDuty for actionable incident notifications.
  - Enable runtime security policies with Falco for Kubernetes nodes.
- **Evaluation and Testing (December 1st – December 5th):**
  - Conduct a final security assessment, including penetration testing.
  - Compare current performance metrics to baseline data to measure improvements.
  - Verify compliance with GDPR and other applicable data protection regulations.
- **Handover and Documentation (December 6th – December 10th):**
  - Prepare detailed documentation of configurations, processes, and best practices.
  - Handover system monitoring and maintenance responsibilities to the operations team.
  - Conduct a project closing meeting to review outcomes, deliverables, and next steps.

Deliverables:

- Advanced monitoring framework with actionable alerts and refined dashboards.
- A comprehensive final assessment report comparing pre- and post-implementation metrics.
- Complete system documentation and operational handover.

d) Summary Timeline

Phase	Activities	Timeline
Phase 1: Preparation and Initial Setup	Team onboarding, tool configuration, baseline assessments, training	Sept 10 – Sept 30, 2024
Phase 2: Core Implementation	Access controls, network segmentation, CI/CD security, microservices optimization	Oct 1 – Nov 20, 2024
Phase 3: Monitoring and Optimization	Advanced monitoring, security testing, system documentation	Nov 21 – Dec 10, 2024

Table 3.1: Integrated Action Plan Timeline

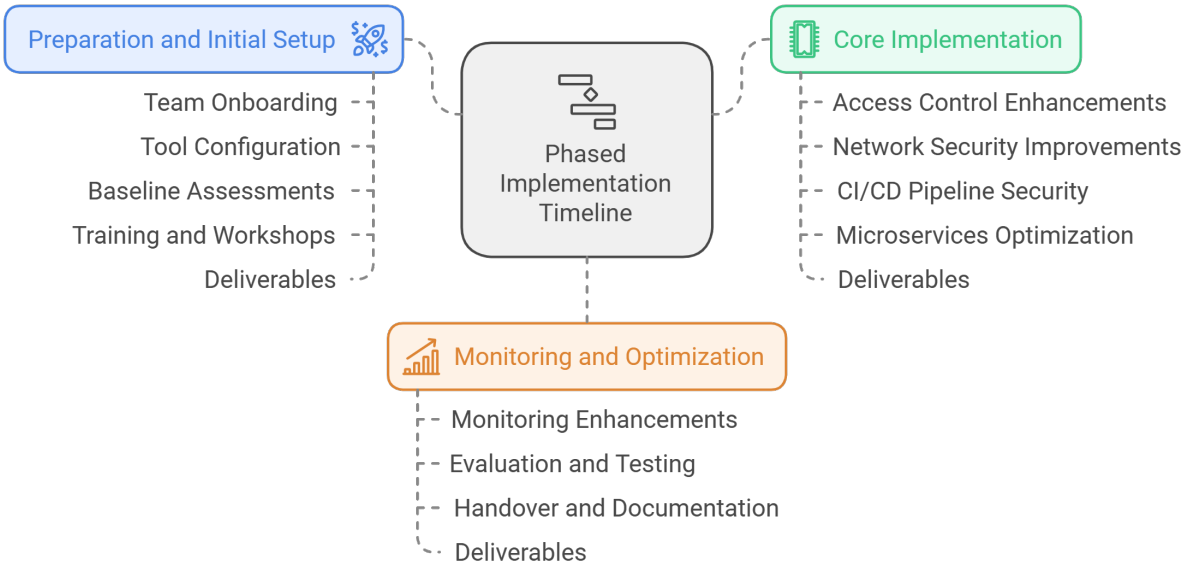

Figure 3.1: Integrated Action Plan Timeline

3.3.2 Team Responsibilities

- **Security Team:** Implement access control, secrets management, and vulnerability scanning.
- **DevOps Team:** Handle network segmentation, service mesh, and Kubernetes security.
- **Data Protection Team:** Ensure encryption, audit logging, and compliance with GDPR.

3.4 Conclusion

This chapter outlines a detailed set of recommendations and an actionable plan to mitigate the risks identified in Chapter 2. The proposed measures will enhance the security, scalability, and reliability of the Firma'IQ microservices architecture. The next chapter will evaluate the progress and metrics for monitoring the system post-implementation.

Chapter

4 Monitoring, Evaluation, and Continuous Improvement

Contents

4.1 Introduction

The deployment of recommended measures outlined in Chapter 3 addresses identified vulnerabilities and strengthens the Firma'IQ microservices architecture. However, a static approach to security, performance, and scalability is insufficient in a dynamic technology landscape. Continuous monitoring, detailed evaluation of system behavior, and a commitment to iterative improvement are essential to ensure the architecture evolves with the growing demands of users and the ever-changing threat environment.

This chapter outlines a comprehensive framework for monitoring, evaluation, and continuous improvement. The proposed strategies include real-time tracking of system metrics, defining meaningful Key Performance Indicators (KPIs), and implementing mechanisms for ongoing assessment and enhancement. By adopting these practices, the architecture will remain secure, resilient, and efficient over time.

4.2 Post-Implementation Monitoring

Effective monitoring allows proactive identification and resolution of potential issues before they escalate into critical problems. This section provides detailed guidelines for establishing a robust monitoring framework.

4.2.1 Real-Time Monitoring Framework

a) Importance of Real-Time Monitoring

Real-time monitoring is crucial for maintaining operational stability, detecting threats, and ensuring compliance with service-level agreements (SLAs). It enables:

- Early detection of performance bottlenecks and system failures.
- Continuous oversight of resource utilization and availability.

- Immediate alerts in the event of anomalies or suspicious activities.

b) Key Monitoring Tools and Their Roles

1. **Prometheus and Grafana:**
    - **Prometheus:** Collects time-series data on metrics such as CPU usage, memory consumption, and network throughput from Kubernetes nodes, containers, and microservices.
    - **Grafana:** Provides interactive dashboards to visualize Prometheus metrics and define alert thresholds for critical parameters.
2. **ELK Stack (Elasticsearch, Logstash, Kibana):**
    - **Logstash:** Aggregates logs from multiple sources, such as application servers, ingress controllers, and databases.
    - **Elasticsearch:** Indexes logs for efficient querying and analysis.
    - **Kibana:** Enables custom dashboards for log visualization and pattern identification.
3. **Jaeger for Distributed Tracing:**
    - Traces request flows across microservices, identifying latency bottlenecks and service dependencies.
    - Helps pinpoint failures in complex workflows spanning multiple services.
4. **Falco for Runtime Security:**
    - Monitors Kubernetes environments for runtime anomalies, such as unauthorized file access or privilege escalations.
    - Provides integration with notification tools like Slack and PagerDuty.

c) Step-by-Step Implementation

1. Deploy Prometheus to monitor cluster-level metrics and microservice performance.
2. Configure Grafana dashboards to visualize metrics and create alerts for conditions such as high latency or resource exhaustion.

3. Set up Logstash pipelines to aggregate logs from applications, network devices, and databases.
4. Implement Jaeger in all microservices by including tracing libraries in service codebases.
5. Deploy Falco on Kubernetes nodes and define policies to monitor privileged activities and suspicious behavior.

4.2.2 Incident Detection and Management

a) Alerting Framework

An effective alerting system enables timely response to issues by notifying the appropriate teams. Alerts should be:

- **Severity-Based:** Categorize alerts into critical, high, medium, and low based on their potential impact.
- **Actionable:** Provide detailed information about the issue, including affected components and recommended actions.
- **Integrated:** Use tools like PagerDuty, Opsgenie, or Slack for automated notifications.

b) Defining Alert Thresholds

Thresholds should be based on historical trends and operational SLAs:

- CPU usage exceeding 85% for more than 5 minutes triggers a medium-priority alert.
- Service response times greater than 300ms for over 10 minutes trigger a high-priority alert.
- Database connection failures generate a critical alert immediately.

c) Incident Response Workflow

1. **Detection:** An alert is triggered by monitoring tools, notifying the incident response team.
2. **Notification:** The team receives detailed information about the incident via integrated tools.
3. **Triage:** The team assesses the severity of the issue and prioritizes it based on potential impact.
4. **Investigation:** Logs, traces, and metrics are analyzed to identify the root cause.
5. **Resolution:** Corrective actions are implemented, tested, and deployed.
6. **Post-Incident Review:** A detailed report is generated, documenting the incident and preventive measures.

4.2.3 Continuous Security Monitoring

a) Components of Security Monitoring

- **Vulnerability Management:**
  - Use tools like Trivy and Clair to scan Docker images, dependencies, and configurations for known vulnerabilities.
  - Automate vulnerability scanning within CI/CD pipelines to block insecure builds.
- **Configuration Compliance:**
  - Validate Kubernetes configurations against CIS benchmarks using tools like kube-bench.
  - Ensure compliance with internal security policies through automated audits.
- **Threat Detection:**
  - Monitor runtime activities for deviations from expected behavior using Falco.
  - Analyze logs for unusual patterns, such as repeated failed login attempts or unauthorized access to sensitive endpoints.

b) Security Incident Automation

- Integrate vulnerability scanning tools with CI/CD pipelines to ensure secure code delivery.
- Automate remediation of common misconfigurations using Kubernetes admission controllers or policy-as-code tools like Open Policy Agent (OPA).

4.3 Evaluation Metrics and KPIs

Defining metrics is essential for measuring the success of implemented measures and identifying areas for improvement.

4.3.1 Performance Metrics

- **API Latency:** Average response time should not exceed 200ms, with 99th percentile latency remaining below 300ms.
- **Resource Utilization:** CPU and memory usage should not exceed 75% of allocated limits under peak loads.
- **System Uptime:** Ensure 99.9% availability for critical microservices, measured monthly.

4.3.2 Security Metrics

- **Incident Response Time:** Reduce the mean time to detect and resolve critical incidents to less than 4 hours.
- **Vulnerability Reduction Rate:** Target a 90% reduction in critical vulnerabilities within the first three months of implementation.
- **Audit Log Coverage:** Ensure 100% logging coverage for all database interactions and administrative actions.

4.3.3 Operational Metrics

- **Deployment Frequency:** Maintain at least three successful deployments per week across all microservices.
- **Mean Time to Recovery (MTTR):** Ensure service disruptions are resolved within 2 hours on average.
- **Failure Rate:** Keep failed deployments below 2% of total deployment attempts.

4.4 Continuous Improvement Framework

4.4.1 Feedback Loops

a) End-User Feedback

- Regularly collect feedback through user satisfaction surveys and performance reviews.
- Monitor feature usage patterns to identify bottlenecks or underutilized functionalities.

b) Internal Feedback

- Conduct retrospective meetings post-deployment to discuss challenges, successes, and lessons learned.
- Use internal performance reports to measure team productivity and resource allocation.

4.4.2 Automated Updates and Scaling

- Use dependency management tools like Renovate to automate updates for libraries and packages.
- Implement horizontal and vertical pod autoscaling in Kubernetes to adapt resource allocations dynamically.

4.4.3 Evolving Threat Landscape

- Regularly review threat intelligence feeds and security advisories to stay updated on emerging risks.
- Expand penetration testing to include sophisticated attack vectors, such as supply chain attacks and insider threats.

4.5 Conclusion

This chapter provided a detailed framework for monitoring, evaluating, and continuously improving the Firma'IQ microservices architecture. By adopting robust real-time monitoring systems, defining actionable KPIs, and fostering a culture of feedback and adaptation, the organization can maintain a secure, efficient, and future-ready architecture. Continuous improvement ensures that the system evolves alongside technological advancements and user demands, safeguarding its long-term success.

GENERAL CONCLUSION

The advent of microservices architecture has revolutionized the design and deployment of modern software systems, offering unprecedented scalability, flexibility, and resilience. Unlike monolithic architectures, microservices break applications into smaller, independent services that can be developed, deployed, and scaled individually. This shift has enabled organizations to respond more effectively to dynamic business needs and technological advancements. However, with these advantages come significant challenges in terms of security, monitoring, and operational efficiency, requiring innovative strategies and tools to address them effectively.

This report aims to tackle these challenges through a comprehensive audit of a microservices-based system. The audit focuses on identifying critical vulnerabilities, such as unsecured inter-service communication, weak access controls, and insufficient monitoring frameworks. It also evaluates the existing CI/CD pipelines, network configurations, and Kubernetes cluster management to ensure operational resilience and compliance with industry standards. Additionally, the audit incorporates advanced tools like Prometheus, Grafana, and Falco to implement robust monitoring and alerting mechanisms while maintaining system performance under varying loads.

The primary goal of this project is to design and implement a secure, automated, and scalable architecture that adheres to industry best practices and anticipates emerging threats. The findings of this report include detailed risk assessments, prioritized recommendations, and an integrated action plan to address existing vulnerabilities and enhance the system's resilience. By adopting these measures, the organization can build a future-ready system architecture that not only meets current operational requirements but also supports sustainable growth and innovation. This work seeks to contribute to the broader discourse on best practices for microservices security and operational excellence.

BIBLIOGRAPHY

[1] **Kubernetes Documentation**. Kubernetes API Concepts **[Online]**. Available: https://kubernetes.io/docs/reference/

[2] **Red Hat**. (2021). *An Introduction to Microservices Architecture*. Available: https://www.redhat.com/en/topics/microservices/what-are-microservices

[3] **Lamport, Leslie**. (1994). *LaTeX: A Document Preparation System*. Addison Wesley, Massachusetts, 2nd ed.

[4] **Docker Documentation**. (2023). Best Practices for Writing Dockerfiles **[Online]**. Available: https://docs.docker.com/develop/dev-best-practices/

[5] **HashiCorp Vault**. (2023). Secrets Management for Modern Applications **[Online]**. Available: https://www.hashicorp.com/products/vault

[6] **Prometheus Documentation**. (2023). Monitoring and Alerting Toolkit **[Online]**. Available: https://prometheus.io/docs/introduction/overview/

[7] **Falco Documentation**. (2023). Open Source Runtime Security for Kubernetes **[Online]**. Available: https://falco.org/docs/

[8] **Wang, Y., Yao, Q., Kwok, J.T., and Ni, L.M.** (2020). Generalizing from a Few Examples: A Survey on Few-Shot Learning. *ACM Computing Surveys (CSUR)*, 53(3), pp. 1-34.

[9] **SonarQube Documentation**. (2023). Continuous Inspection for Code Quality and Security **[Online]**. Available: https://docs.sonarqube.org/latest/

[10] **ENET'Com**. National School of Electronics and Telecommunications **[Online]**. Available: https://enetcom.rnu.tn/fra/pages/260/PrÃl'sentation-de-lâĂŹENETâĂŹCOM

APPENDICES

A.1 Architecture Diagram

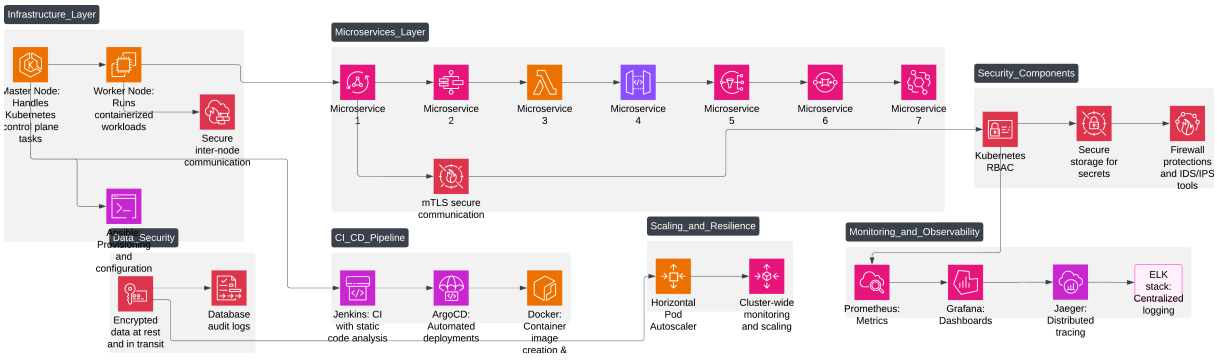

Figure A.1: System Architecture Diagram for Firma'IQ.

Description:

The architecture diagram in Figure A.1 presents the design of the Firma'IQ system, integrating multiple components to ensure a secure, scalable, and efficient operational framework. Key highlights include:

- **Infrastructure Layer:** Kubernetes-based orchestration, with a Master Node for control plane tasks and Worker Nodes for running containerized workloads.
- **Microservices Layer:** Independent microservices communicate securely using mTLS.
- **Security Components:** Kubernetes RBAC, secure storage for secrets, and IDS/IPS tools for advanced security.
- **CI/CD Pipeline:** Tools such as Jenkins, ArgoCD, and Docker streamline the deployment process.

- **Monitoring and Observability:** Real-time monitoring with Prometheus and Grafana, and distributed tracing with Jaeger.
- **Scaling and Resilience:** Horizontal Pod Autoscaler ensures the system adapts to dynamic workloads.

This architecture embodies cutting-edge methodologies for robust performance and security.

A.2 Audit Results Screenshots

Lynis Audit Overview

```
elk@elk:~$ sudo lynis audit system

[ Lynis 2.6.2 ]

################################################################################
#######
  Lynis comes with ABSOLUTELY NO WARRANTY. This is free software, and you
 are
  welcome to redistribute it under the terms of the GNU General Public Li
cense.
  See the LICENSE file for details about using this software.

  2007-2018, CISOfy - https://cisofy.com/lynis/
  Enterprise support available (compliance, plugins, interface and tools)
################################################################################
#######


[+] Initializing program
------------------------------------
  - Detecting OS...                                           [ DONE ]
  - Checking profiles...                                      [ DONE ]

  ---------------------------------------------------
  Program version:           2.6.2
  Operating system:          Linux
  Operating system name:     Ubuntu Linux
  Operating system version:  20.04
  Kernel version:            5.15.0
  Hardware platform:         x86_64
  Hostname:                  elk
  ---------------------------------------------------
  Profiles:                  /etc/lynis/default.prf
  Log file:                  /var/log/lynis.log
  Report file:               /var/log/lynis-report.dat
  Report version:            1.0
  Plugin directory:          /etc/lynis/plugins
  ---------------------------------------------------
  Auditor:                   [Not Specified]
  Language:                  en
  Test category:             all
  Test group:                all
  ---------------------------------------------------
  - Program update status...                                  [ WARNING ]

      ===============================================================
```

Figure A.2: Lynis Audit: Initialization and Overview

Description:

Figure A.2 captures the initialization phase of the Lynis audit, showing detected operating system details, kernel version, and audit profiles.

```
      ================================================================
============
        Lynis update available
      ================================================================
============

        Current version is more than 4 months old

        Current version : 262   Latest version : 312

        Please update to the latest version.
        New releases include additional features, bug fixes, tests, and b
aselines.

        Download the latest version:

        Packages (DEB/RPM) -  https://packages.cisofy.com
        Website (TAR)      -  https://cisofy.com/downloads/
        GitHub (source)    -  https://github.com/CISOfy/lynis

      ================================================================
============


[+] System Tools
------------------------------------
  - Scanning available tools...
  - Checking system binaries...

[+] Plugins (phase 1)
------------------------------------
 Note: plugins have more extensive tests and may take several minutes to
complete

  - Plugin: debian
    [
[+] Debian Tests
------------------------------------
  - Checking for system binaries that are required by Debian Tests...
    - Checking /bin...                                         [ FOUND ]
    - Checking /sbin...                                        [ FOUND ]
    - Checking /usr/bin...                                     [ FOUND ]
    - Checking /usr/sbin...                                    [ FOUND ]
    - Checking /usr/local/bin...                               [ FOUND ]
    - Checking /usr/local/sbin...                              [ FOUND ]
  - Authentication:
```

Figure A.3: Lynis Audit: Update Recommendations and System Tools

Description:

Figure A.3 highlights outdated packages detected by Lynis, along with recommendations for upgrades to improve system security.

Debian Tests and Boot Services

```
  - Authentication:
    - PAM (Pluggable Authentication Modules):
      - libpam-tmpdir                                         [ Not Insta
lled ]
      - libpam-usb                                            [ Not Insta
lled ]
  - File System Checks:
    - DM-Crypt, Cryptsetup & Cryptmount:
  - Software:
    - apt-listbugs                                            [ Not Insta
lled ]
    - apt-listchanges                                         [ Not Insta
lled ]
    - checkrestart                                            [ Not Insta
lled ]
    - needrestart                                             [ Not Insta
lled ]
    - debsecan                                                [ Not Insta
lled ]
    - debsums                                                 [ Not Insta
lled ]
    - fail2ban                                                [ Not Insta
lled ]
]

[+] Boot and services
------------------------------------
  - Service Manager                                           [ SysV Init
 ]
  - Checking UEFI boot                                        [ DISABLED
]
  - Checking presence GRUB2                                   [ FOUND ]
    - Checking for password protection                        [ WARNING ]
  - Check running services (systemctl)                        [ DONE ]
        Result: found 34 running services
  - Check enabled services at boot (systemctl)                [ DONE ]
        Result: found 65 enabled services
  - Check startup files (permissions)                         [ OK ]
  - Checking sulogin in rescue.service                        [ NOT FOUND
 ]

[+] Kernel
------------------------------------
  - Checking default run level                                [ RUNLEVEL
5 ]
  - Checking CPU support (NX/PAE)
```

Figure A.4: Lynis Audit: Debian Tests and Boot Services

Description:

Figure A.4 displays the results of Debian-specific tests and active boot services, including warnings and missing configurations.

Kernel and Memory Processes

```
]
  - Checking presence GRUB2                                  [ FOUND ]
    - Checking for password protection                       [ WARNING ]
  - Check running services (systemctl)                       [ DONE ]
        Result: found 34 running services
  - Check enabled services at boot (systemctl)               [ DONE ]
        Result: found 65 enabled services
  - Check startup files (permissions)                        [ OK ]
  - Checking sulogin in rescue.service                       [ NOT FOUND
 ]

[+] Kernel
------------------------------------
  - Checking default run level                               [ RUNLEVEL
5 ]
  - Checking CPU support (NX/PAE)
    CPU support: PAE and/or NoeXecute supported              [ FOUND ]
  - Checking kernel version and release                      [ DONE ]
  - Checking kernel type                                     [ DONE ]
  - Checking loaded kernel modules                           [ DONE ]
      Found 85 active modules
  - Checking Linux kernel configuration file                 [ FOUND ]
  - Checking default I/O kernel scheduler                    [ NOT FOUND
 ]
  - Checking for available kernel update                     [ UNKNOWN ]
  - Checking core dumps configuration                        [ DISABLED
]
    - Checking setuid core dumps configuration               [ PROTECTED
 ]
  - Check if reboot is needed                                [ NO ]

[+] Memory and Processes
------------------------------------
  - Checking /proc/meminfo                                   [ FOUND ]
  - Searching for dead/zombie processes                      [ OK ]
  - Searching for IO waiting processes                       [ OK ]

[+] Users, Groups and Authentication
------------------------------------
  - Administrator accounts                                   [ OK ]
  - Unique UIDs                                              [ OK ]
  - Consistency of group files (grpck)                       [ OK ]
  - Unique group IDs                                         [ OK ]
  - Unique group names                                       [ OK ]
  - Password file consistency                                [ OK ]
  - Query system users (non daemons)                         [ DONE ]
```

Figure A.5: Lynis Audit: Kernel and Memory Processes

Description:

Figure A.5 outlines the kernel configuration checks and memory usage processes, offering suggestions to enhance kernel security.

Users, Groups, and Authentication

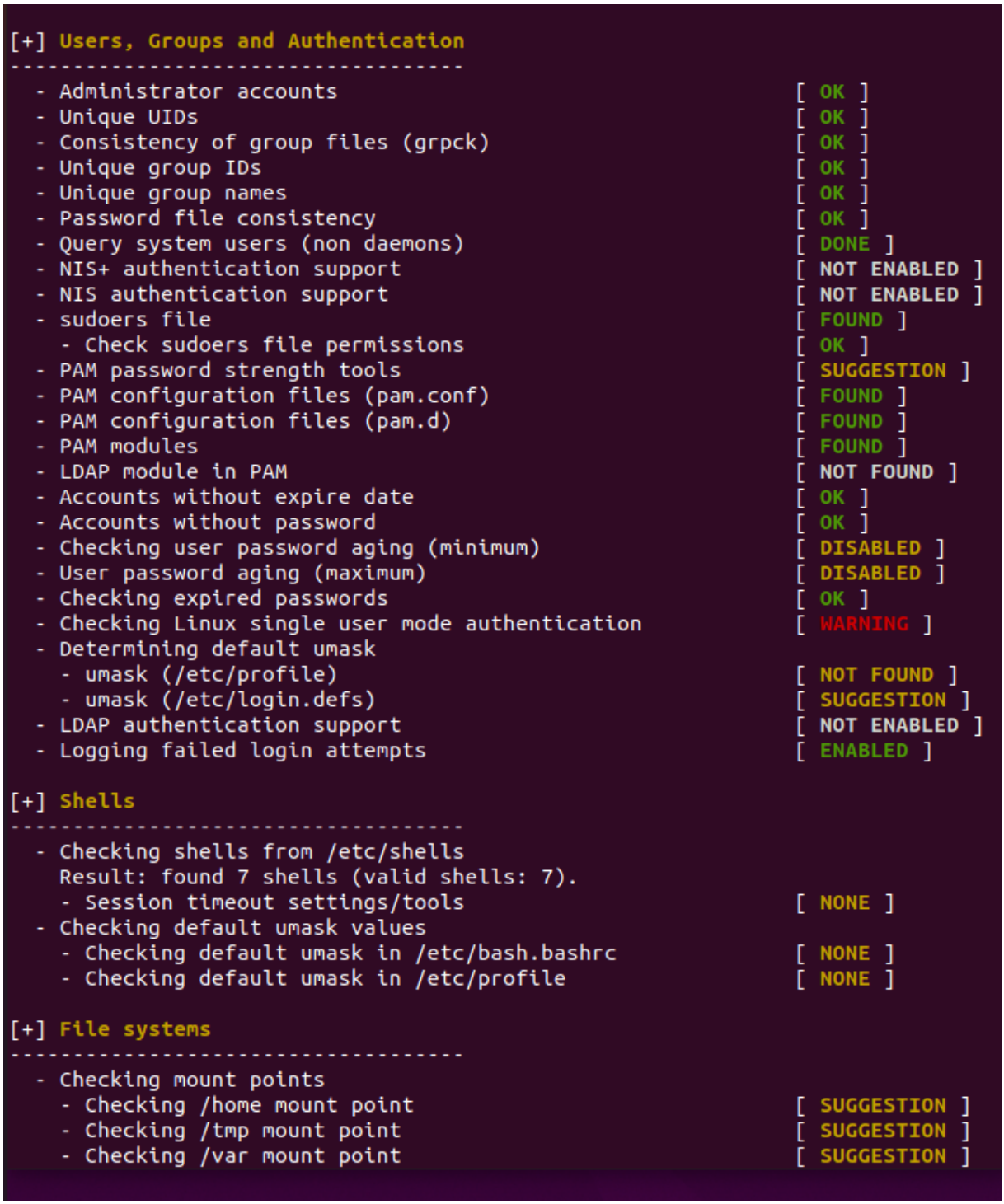

Figure A.6: Lynis Audit: Users, Groups, and Authentication

Description:

Figure A.6 reviews user account security, password policies, and authentication mechanisms.

File Systems and USB Devices

```
    - Checking /var mount point                                [ SUGGESTION ]
  - Query swap partitions (fstab)                              [ OK ]
  - Testing swap partitions                                    [ OK ]
  - Testing /proc mount (hidepid)                              [ SUGGESTION ]
  - Checking for old files in /tmp                             [ OK ]
  - Checking /tmp sticky bit                                   [ OK ]
  - Checking /var/tmp sticky bit                               [ OK ]
  - ACL support root file system                               [ ENABLED ]
  - Mount options of /                                         [ NON DEFAULT ]
  - Disable kernel support of some filesystems
    - Discovered kernel modules: cramfs freevxfs hfs hfsplus jffs2 udf

[+] USB Devices
------------------------------------
  - Checking usb-storage driver (modprobe config)              [ NOT DISABLED ]
  - Checking USB devices authorization                         [ ENABLED ]
  - Checking USBGuard                                          [ NOT FOUND ]

[+] Storage
------------------------------------
  - Checking firewire ohci driver (modprobe config)            [ DISABLED ]

[+] NFS
------------------------------------
  - Check running NFS daemon                                   [ NOT FOUND ]

[+] Name services
------------------------------------
  - Checking /etc/resolv.conf options                          [ FOUND ]
  - Searching DNS domain name                                  [ FOUND ]
      Domain name: myguest.virtualbox.org
  - Checking /etc/hosts
    - Checking /etc/hosts (duplicates)                         [ OK ]
    - Checking /etc/hosts (hostname)                           [ OK ]
    - Checking /etc/hosts (localhost)                          [ OK ]
    - Checking /etc/hosts (localhost to IP)                    [ OK ]

[+] Ports and packages
------------------------------------
  - Searching package managers
    - Searching dpkg package manager                           [ FOUND ]
      - Querying package manager
    - Query unpurged packages                                  [ FOUND ]
  - Checking security repository in sources.list file          [ OK ]
  - Checking APT package database                              [ OK ]
  - Checking vulnerable packages                               [ OK ]
```

Figure A.7: Lynis Audit: File Systems and USB Devices

Description:

Figure A.7 examines file system configurations and controls for unauthorized USB device access.

Ports and Packages

```
[+] Ports and packages
------------------------------------
  - Searching package managers
    - Searching dpkg package manager                    [ FOUND ]
      - Querying package manager
    - Query unpurged packages                           [ FOUND ]
  - Checking security repository in sources.list file   [ OK ]
  - Checking APT package database                       [ OK ]
  - Checking vulnerable packages                        [ OK ]
  - Checking upgradeable packages                       [ SKIPPED ]
  - Checking package audit tool                         [ INSTALLED ]
    Found: apt-check

[+] Networking
------------------------------------
  - Checking IPv6 configuration                         [ ENABLED ]
      Configuration method                              [ AUTO ]
      IPv6 only                                         [ NO ]
  - Checking configured nameservers
    - Testing nameservers
        Nameserver: 127.0.0.53                          [ OK ]
    - Minimal of 2 responsive nameservers               [ WARNING ]
  - Checking default gateway                            [ DONE ]
  - Getting listening ports (TCP/UDP)                   [ DONE ]
      * Found 11 ports
  - Checking promiscuous interfaces                     [ OK ]
  - Checking waiting connections                        [ OK ]
  - Checking status DHCP client                         [ NOT ACTIVE ]
  - Checking for ARP monitoring software                [ NOT FOUND ]

[+] Printers and Spools
------------------------------------
  - Checking cups daemon                                [ RUNNING ]
  - Checking CUPS configuration file                    [ OK ]
    - File permissions                                  [ WARNING ]
  - Checking CUPS addresses/sockets                     [ FOUND ]
  - Checking lp daemon                                  [ NOT RUNNING ]

[+] Software: e-mail and messaging
------------------------------------

[+] Software: firewalls
------------------------------------
  - Checking iptables kernel module                     [ FOUND ]
    - Checking iptables policies of chains              [ FOUND ]
    - Checking for empty ruleset                        [ OK ]
```

Figure A.8: Lynis Audit: Ports and Packages

Description:

Figure A.8 shows open port scans and installed package audits to identify potential vulnerabilities.

Networking Configuration

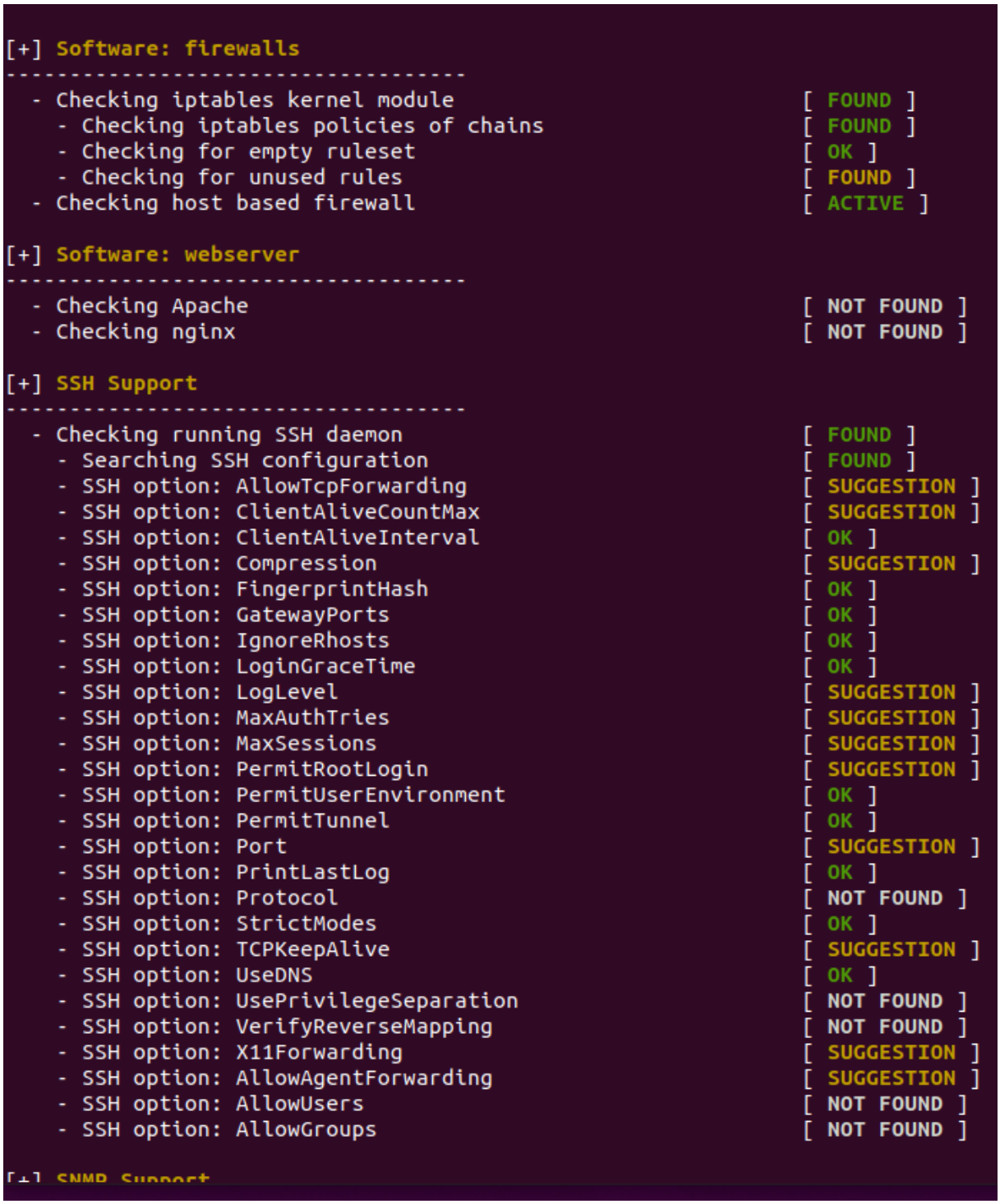

Figure A.9: Lynis Audit: Networking Configuration

Description:

Figure A.9 reviews network configurations, DNS settings, and firewall policies for improvement.

Software and Firewalls

Figure A.10: Lynis Audit: Software and Firewalls

Description:

Figure A.10 captures software vulnerabilities and firewall configuration statuses.

Final Recommendations and Warnings

```
[+] Banners and identification
------------------------------------
  - /etc/issue                                              [ FOUND ]
    - /etc/issue contents                                   [ WEAK ]
  - /etc/issue.net                                          [ FOUND ]
    - /etc/issue.net contents                               [ WEAK ]

[+] Scheduled tasks
------------------------------------
  - Checking crontab/cronjob                                [ DONE ]

[+] Accounting
------------------------------------
  - Checking accounting information                         [ NOT FOUND ]
  - Checking sysstat accounting data                        [ NOT FOUND ]
  - Checking auditd                                         [ NOT FOUND ]

[+] Time and Synchronization
------------------------------------
  - NTP daemon found: systemd (timesyncd)                   [ FOUND ]
  - Checking for a running NTP daemon or client             [ OK ]

[+] Cryptography
------------------------------------
  - Checking for expired SSL certificates [0/4]             [ NONE ]

[+] Virtualization
------------------------------------

[+] Containers
------------------------------------
    - Docker
      - Docker daemon                                       [ RUNNING ]
        - Docker info output (warnings)                     [ NONE ]
      - Containers
        - Total containers                                  [ UNKNOWN ]
    - File permissions                                      [ OK ]

[+] Security frameworks
------------------------------------
  - Checking presence AppArmor                              [ FOUND ]
    - Checking AppArmor status                              [ ENABLED ]
  - Checking presence SELinux                               [ NOT FOUND ]
  - Checking presence grsecurity                            [ NOT FOUND ]
  - Checking for implemented MAC framework                  [ OK ]
```

Figure A.11: Lynis Audit: Final Recommendations and Warnings

Description:

Figure A.11 summarizes the audit's recommendations and highlights critical areas needing immediate action.

COMPREHENSIVE SECURITY AND PERFORMANCE AUDIT OF A MICROSERVICES-BASED ARCHITECTURE

Résumé:

Ce rapport présente une analyse approfondie d'une architecture basée sur des microservices, en mettant l'accent sur la sécurité, les performances et la résilience. Le projet vise à identifier les vulnérabilités critiques, optimiser les processus CI/CD, renforcer les politiques de contrôle d'accès et améliorer la surveillance en temps réel. Les recommandations incluent des outils avancés comme Prometheus, Grafana et Falco, ainsi qu'une migration vers des pratiques sécurisées, afin de garantir une infrastructure évolutive et automatisée. Ce travail contribue à établir une architecture alignée sur les meilleures pratiques de l'industrie et prête à faire face aux menaces émergentes.

Mots clés: Microservices, Sécurité, CI/CD, Kubernetes, Résilience, Optimisation des performances.

Abstract:

This report presents a comprehensive analysis of a microservices-based architecture, focusing on security, performance, and resilience. The project aims to identify critical vulnerabilities, optimize CI/CD processes, strengthen access control policies, and improve real-time monitoring. Recommendations include advanced tools such as Prometheus, Grafana, and Falco, alongside a migration to secure practices to ensure a scalable and automated infrastructure. This work contributes to establishing an architecture aligned with industry best practices and prepared to address emerging threats.

Key-words: Microservices, Security, CI/CD, Kubernetes, Resilience, Performance Optimization.

التقرير الحالي يقدم دراسة شاملة للبنية المعمارية القائمة على الخدمات المصغّرة (Microservices)، مع التركيز على تعزيز الأمان، تحسين الأداء، وضمان المرونة. يهدف المشروع إلى الكشف عن الثغرات الحرجة، تطوير عمليات التكامل والتسليم المستمر (CI/CD)، تقوية سياسات إدارة الوصول، وتطوير آليات المراقبة في الوقت الحقيقي. تتضمن التوصيات استخدام أدوات متقدمة مثل Prometheus وGrafana وFalco، إضافةً إلى تبني ممارسات آمنة لضمان بنية تحتية مؤتمتة وقابلة للتوسع. يسعى هذا العمل إلى بناء بنية متوافقة مع معايير الصناعة الحديثة وقادرة على التصدي للتهديدات المستقبلية.

الكلمات المفتاحية: البنية المصغّرة، الأمن السيبراني، التكامل والتسليم المستمر (CI/CD)، Kubernetes، المرونة، تحسين الأداء.