ZERO TRUST SECURITY FOR WEB APPLICATIONS IN MICROSERVICE-BASED ENVIRONMENTS

A PROJECT REPORT

Submitted by

DINESH KUMAR. N
[Reg. No CH.EN.U4CYS21014]

V. JAYARAJ
[Reg. No CH.EN.U4CYS21026]

in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE AND ENGINEERING (CYBER SECURITY)

Under the guidance of

Dr. S. UDHAYA KUMAR

Submitted to

AMRITA VISHWA VIDYAPEETHAM

AMRITA SCHOOL OF COMPUTING

CHENNAI – 601103

January 2025

BONAFIDE CERTIFICATE

This is to certify that this project report entitled **"ZERO TRUST SECURITY FOR WEB APPLICATIONS IN MICROSERVICE-BASED ENVIRONMENTS"** is the bonafide work of **N. Dinesh Kumar [Reg. No CH.EN.U4CYS21014] and V. Jayaraj [Reg. No CH.EN.U4CYS21026]** who carried out the project work under my supervision.

CHAIRPERSON SIGNATURE | **SUPERVISOR SIGNATURE**

Dr. SOUNTHARRAJAN S
Associate Professor
Department of CSE
Amrita Vishwa Vidyapeetham
Amrita School of Computing
Chennai

Dr. S. Udhaya Kumar
Associate Professor
Department of CSE
Amrita Vishwa Vidyapeetham
Amrita School of Computing
Chennai

INTERNAL EXAMINER | **EXTERNAL EXAMINER**

DECLARATION BY THE CANDIDATE

I declare that the report entitled "**ZERO TRUST SECURITY FOR WEB APPLICATIONS IN MICROSERVICE-BASED ENVIRONMENTS**" submitted by me for the degree of Bachelor of Technology in Computer Science and Engineering (Cyber Security) is the record of the project work carried out by us under the guidance of **Dr. UDHAYAKUMAR S,** Associate Professor, Department of Computer Science and Engineering and this work has not formed the basis for the award of any degree, diploma, associateship, fellowship, titled in this or any other University or other similar institution of higher learning.

DINESH KUMAR. N	**V. JAYARAJ**
[Reg. No. CH.EN.U4CYS21014]	**[Reg. No. CH.EN.U4CYS21026]**

ABSTRACT

With the rapidly shifting landscape in cybersecurity, cyber threats have made the old perimeter-based models obscurate. As enterprises are embracing cloud-native applications as well as microservices-based architectures, a strong and nuanced security framework emerged, at the very least as a mitigative risk factor. The paper discusses the design and implementation of the Zero Trust Security Model securing web applications running in extended Kubernetes Clusters to Android applications to improve mobile security. For the web application, Zero Trust is implemented using JWT authentication, role-based access control (RBAC), continuous authentication, and data encryption, ensuring strict identity verification and dynamic access control across the system. Additionally, the model enforces fine-grained micro-segmentation and real-time threat detection, protecting critical resources from unauthorized access and mitigating lateral movement attacks. In the ZeroSMS Android application, the Zero Trust model is extended to mobile apps by integrating security tools like JailMonkey for detecting root access and debugging, along with code obfuscation using R8 and ProGuard for enhanced protection. These measures ensure that mobile applications are safeguarded from unauthorized use and potential exploits. The architecture further incorporates mTLS and a tool for detecting unauthorized shell command interactions to secure containerized environments, ensuring robust security at both application and infrastructure levels. Our evaluation demonstrates that this approach effectively reduces attack surfaces, enhances protection against modern threats, and provides a comprehensive security solution for both web and mobile contexts. The proposed framework offers a scalable, resilient, and adaptable security architecture, making it well-suited to the demands of modern cybersecurity challenges.

Keywords: Zero Trust Security, JWT Authentication, Mutual TLS, Cybersecurity, Web Application Security, Android App Security, Micro-segmentation, Kubernetes Clusters, Role-Based Access Control.

ACKNOWLEDGEMENT

This project work would not have been possible without the contribution of many people. It gives me immense pleasure to express my profound gratitude to our honorable Chancellor **Sri Mata Amritanandamayi Devi**, for her blessings and for being a source of inspiration. I am indebted to extend my gratitude to our Director, **Mr. I B Manikantan** Amrita School of Computing and Engineering, for facilitating us with all the facilities and extended support to gain valuable education and learning experience.

I register my special thanks to **Dr. V. Jayakumar**, Principal Amrita School of Computing and Engineering, **Dr. Sountharrajan**, Chairperson of Department of Computer Science and Engineering and **Dr. S. Udhaya Kumar**, Program Chair Department of CSE(CYS) for the support given to me in the successful conduct of this project and for his inspiring guidance, personal involvement and constant encouragement during the entire course of this work.

I am grateful to the Project Coordinator, Review Panel Members and the entire faculty of the Department of Computer Science & Engineering, for their constructive criticisms and valuable suggestions which have been a rich source to improve the quality of this work.

DINESH KUMAR. N [CH.EN.U4CYS21014]

V. JAYARAJ [CH.EN.U4CYS21026]

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
	Abstract	**iv**
	List of Tables	**x**
	List of Figures	**xi**
	List of Symbols and Abbreviations	**xiii**
1	**INTRODUCTION**	**1**
	1.1 Background Study	1
	1.2 Analyzing Cybersecurity Dynamics within the Zero Trust Framework	2
	1.3 Problem Identification	3
	1.4 Problem Statement	4
	1.5 Significance and Motivation	4
	1.6 Objective and Scope of the Project	5
2	**LITERATURE REVIEW**	**6**
3	**METHODOLOGY**	**13**
	3.1 Common Vulnerabilities and Zero Trust Protection	13
	3.2 Web Application Overview	14
	3.2.1 Admin and Auditor Interfaces	15
	3.2.2 Security and Fraud Detection	16
	3.3 Access Tokens Functionality and Implementation	17

	3.4 SMS Parsing and Fraud Detection in Financial Transactions	19
	3.4.1 Apache Spark for Machine Learning	19
	3.4.2 ML Model for SMS Spam Detection	21
	3.4.3 Enhancing App Security in React Native	22
	3.5 Policy Enforcement Point for Role Based Access Control	22
	3.6 Secure Communication within the Kubernetes Clusters Using mTLS	23
4	**SYSTEM DESIGN**	**24**
	4.1 System Requirements for Development	24
	4.1.1 Web Application Frontend (React.js)	24
	4.1.2 Python Backend API	25
	4.1.3 React Native Mobile Application	25
	4.1.4 Tools Used	25
	4.2 System Requirements for Deployment in Docker and Kubernetes	25
	4.2.1 Local Environment	25
	4.2.2 Cloud Environment	26
	4.3 Architecture of the Design	27
	4.4 Attributes of the Input Data	28
	4.5 Algorithm for Zero Trust Architecture Best Practices	29

4.6 Protocols and Standards 30

4.6.1 JWT (JSON Web Tokens) 30

4.6.2 mTLS (Mutual Transport Layer Security) 30

4.6.3 Policy Enforcement Point (PEP) 31

4.6.4 Falco: Real-Time Security Monitoring 31

4.6.5 Code Obfuscation 31

4.6.6 Root, Debug, and Developer Options Detection 32

5 IMPLEMENTATION AND RESULT ANALYSIS 33

5.1 Real-Time Implementation of The Prototype 33

5.1.1 Preparation of Environment in AWS 33

5.1.2 Deployment of Web Application to EKS Cluster 34

5.1.3 Deployment of Falco to EKS Cluster 34

5.2 Implementing Production Build for The Android App 34

5.3 Ml Model for Spam Detection 35

5.3.1 Dataset 36

5.3.2 Preprocessing 36

5.4 Models Used for Training 36

5.4.1 Naive Bayes 37

5.4.2 Support Vector Machine (SVM) 37

5.4.3 Logistic Regression 37

5.4.4 Random Forest 38

5.4.5 Gradient Boosting 38

5.5 Result Analysis 38

5.5.1 Deployed Web Application in EKS Cluster 38

5.5.2 Android Application Production Build Results 40

5.5.3 Comparing and Testing ZeroSMS Apk with Other Normal Apks 41

5.5.3 Comparing Various ML models for Spam Detection 42

6 CONCLUSION AND FUTURE WORK 45

References 46

LIST OF TABLES

TABLE NO	TITLE	PAGE NO.
2.1	Literature Summary Table	9
3.1	Comparison of Different Classifiers based on Accuracy	21
5.1	Session Logs for Logged-In Users	40
5.2	Comparison of Model Performance Metrics	43

LIST OF FIGURES

FIGURE NO.	TITLE	PAGE NO.
1.1	Block Diagram representing the Design of ZTA and its Basic Architecture	3
3.1	Architecture Diagram of Implementing JWT in Web Applications	15
3.2	RBAC Implementation in The Web Application	15
3.3	Sequence Diagram Representing the Order of Events Taking Place	16
3.4	Overview Of The Android Application - ZeroSMS	17
3.5	JWT Life Cycle	18
3.6	JWT Architecture	19
3.7	Interaction Between the Android App and the Web App	19
3.8	Secure pipeline for real-time SMS parsing, model training, and anomaly prediction.	20
3.9	Mutual Transport Layer Security (mTLS) Architecture	23
4.1	Architecture Of the Developed System	27
5.1	List Of Pods Running in The EKS Cluster	33
5.2	Logs As Given by Flaco When Shell Is Accessed Inside a Container	34
5.3	Configuring Production Release with Keystore for Secure Signing and Code Obfuscation	38
5.4	Node Groups of EKS Cluster Deployed as EC2 Instances	39
5.5	User Dashboard for CRUD Operations	39
5.6	Share Data with Auditors	39
5.7	Auditor Dashboard with Tasks	39
5.8	Auditor’s Read-Only View of Transactions	39
5.9	Admin Dashboard for Assigning Roles	39
5.10	Database Collections	39

5.11	AWS Lambda Alerts Triggered by Falco Tool Deployed in the Cluster	40
5.12	ZeroSMS In Production Release with Security Features	41
5.13	ZeroSMS Class Name Obfuscation	42
5.14	Reverse Engineered File Names	42
5.15	ZeroSMS Code Obfuscation	42
5.16	Normal Apk’s Reverse Engineered Code	42
5.17	Comparison Between MobSF Scans of ZerSMS and Normal Apk file	42
5.18	Comparison Between Spam Detection Models	44

LIST OF SYMBOLS AND ABBREVIATIONS

ZTNA	-	Zero Trust Network Access
JWT	-	JSON Web Token
SDN	-	Software Defined Networking
IoT	-	Internet of Things
mTLS	-	Mutual Transport Layer Security
ML	-	Machine Learning
AI	-	Artificial Intelligence
IDP	-	Identity Provider
SIEM	-	Security Information and Event Management
OTP	-	One-Time Password
IAM	-	Identity and Access Management
K8s	-	Kubernetes
CI/CD	-	Continuous Integration/Continuous Deployment
YAML	-	YAML Ain't Markup Language
ACL	-	Access Control List
APK	-	Android Application Package
RBAC	-	Role Based Access Control
ABAC	-	Attribute based Access Control
PEP	-	Policy Enforcement Point
PDP	-	Policy Decision Point
CA	-	Certificate Authority

CHAPTER 1

INTRODUCTION

1.1 BACKGROUND STUDY

Modern organizations operating in a digital ecosystem encounter numerous cyber threats, which mark the failure of existing perimeter-based security. Premised on previously determined trust zones, traditional perimeter-based security models have not kept up with the evolving sophistication of contemporary distributed cyberattacks. These models operate under the assumption that threats exist primarily outside the network, which has proven dangerously simplistic in a world where attacks often originate from within. The vulnerabilities inherent in traditional security architectures include limited visibility into network traffic, inadequate user and device verification, and the inability to monitor behaviors in real-time. One of the significant effects of such vulnerabilities is lateral movement in a network, which usually exposes attackers to the access of restricted data following a perimeter breach. For instance, the Equifax data breach in 2017 exposed the personal details of nearly 147 million people. Estimated remediation and damages are said to be around $4 billion. Similarly, in the case of the 2020 SolarWinds cyberattack, third-party software vulnerabilities had been exploited to breach numerous organizations, including those of the US government, with some showing the cost of recovery and potential losses counted in billions of dollars.

The rapid adoption of agile methodologies, cloud computing, and containerization technologies has significantly expanded the attack surface, enabling cybercriminals to exploit vulnerabilities more effectively. Traditional perimeter-based security model, based on predefined trust zones, could not hold a candle to the complexity of advanced distributed attacks. Attack vectors such as phishing, ransomware, and insider threats have grown increasingly sophisticated. For example, ransomware attacks surged by 150% in 2020 alone, costing businesses an estimated $20 billion. Moreover, the Verizon 2020 Data Breach Investigations Report clearly points out that 86% of breaches were financially motivated, which clearly highlights the necessity to adopt stronger measures of security. To counter these changing threats, Zero Trust Security Model came up with a revolutionary approach which has drastically changed the way security is implemented across networks, applications, and devices. Zero Trust Model works under the assumptions that no entity be trusted by default whether a user, a device or even an application—and constant verification must be conducted for every access attempt [1]. This addresses the weaknesses of traditional security architectures by enforcing perpetual authentication and authorization, regardless of whether the entity is from within or outside the network [2]. Its fundamental principle, "never trust, always verify," mandates that access be granted based on a rigorous evaluation of predefined security policies. This approach is particularly relevant in microservice environments, where data and services are distributed across multiple containers and volumes. By utilizing technologies such as JSON Web Token (JWT) authentication, mutual TLS (mTLS), and micro-segmentation, Zero Trust strengthens security postures by limiting access to only authenticated and authorized entities. Additionally, the implementation of features such as root/debug detection and code obfuscation in ZeroSMS enhances security by safeguarding against reverse engineering and unauthorized access. By detecting the shell command injection tools, the Kubernetes containers can make the attacks more resilient by identifying vulnerabilities as soon as possible to prevent them. This paper aims at an investigation and trying to present a better design and implementation of a Zero Trust framework in a modern web application running in a microservice environment, overcoming limitations found with traditional security models.

1.2 ANALYZING CYBERSECURITY DYNAMICS WITHIN THE ZERO TRUST FRAMEWORK

The emergence of decentralized data, distributed systems, and cloud computing has transformed how organizations operate, creating new challenges for cybersecurity. Traditional perimeter-based security approaches, often referred to as "castle-and-moat" defenses, assume that threats exist primarily outside the network perimeter, with internal resources inherently trusted. However, the shift to remote work, cloud services, segmentation, and decoupling of applications and data has made these boundaries porous, leaving applications vulnerable to insider threats, compromised resources, and increasingly sophisticated cyberattacks [3].

The Zero Trust Security Model represents a paradigm shift in how security is conceptualized. Unlike traditional models, Zero Trust as shown in Fig.1.1 does not rely on network location to establish trust. Instead, it mandates that every entity—whether internal or external—be continuously authenticated and authorized before being granted access to sensitive resources. The model leverages technologies like JWT for stateless authentication of users and mTLS for secure, mutual verification of identity between containers or pods running in Kubernetes clusters, ensuring that communication remains encrypted and secure [4]. This approach aligns with the decoupled nature of modern applications, which are segmented across multiple containers in Kubernetes clusters of various cloud providers like AWS and accessed by users and devices from varied locations.

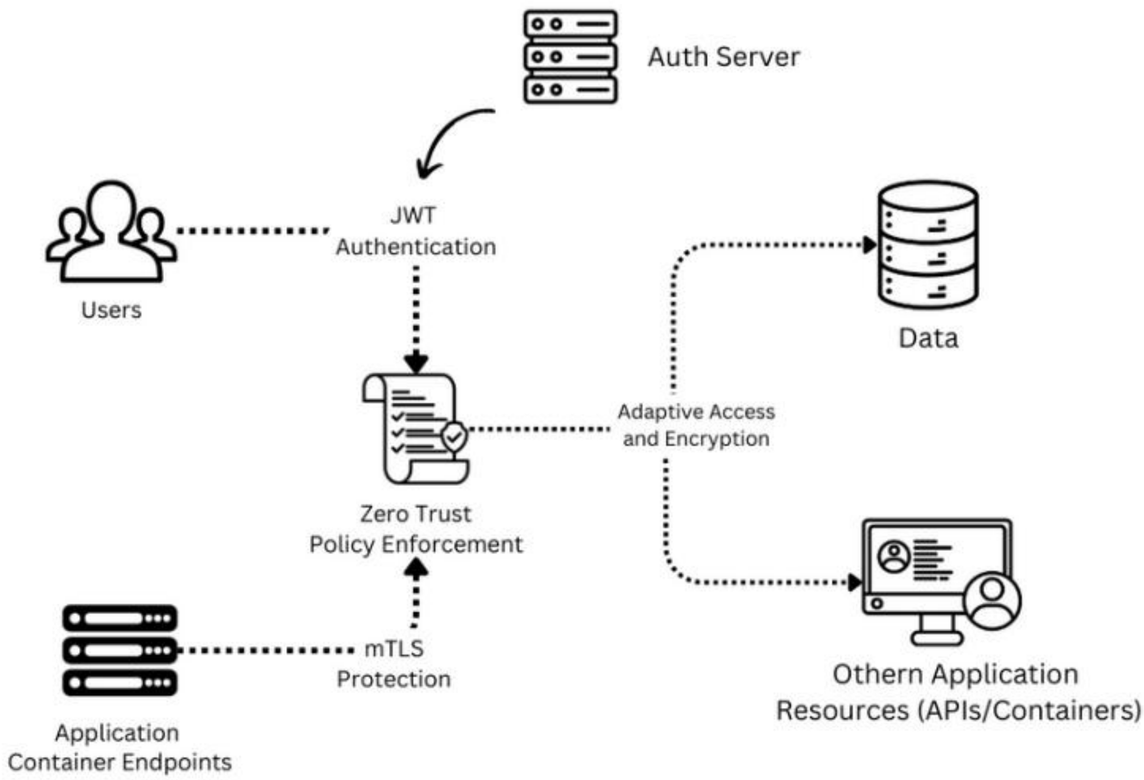

Fig. 1.1 Block diagram describing the design of ZTA and its basic architecture

1.3 PROBLEM IDENTIFICATION

Several critical issues in protecting modern microservice environments push the shift from perimeter-based security to Zero Trust. The issues in traditional security models involve the assumption of threats as primarily external, with which internal systems and data are exposed once the perimeter is breached. The moving parts in this model involve the following critical problems:

- Expanded Attack Surface: The adoption of agile, cloud computing, remote work, and decoupling practices has increased the usage of microservices, making it difficult to monitor and secure all potential entry points. This would allow hackers to hijack the access, use a weak containerized application, or even use the credential from the user to unauthorized access.
- Lateral Movement: Once an attacker gains access to one pod in a Kubernetes cluster, they can move laterally across other containers without encountering significant restrictions, allowing for extensive damage of other pods and data breaches, as evidenced by several high-profile incidents.

- Insufficient Identity Verification: Traditional security models grant users and devices access based on network location, with minimal continuous verification. This lack of ongoing validation leaves Kubernetes clusters vulnerable to unauthorized access.
- Distributed Systems Complexity: As an application's features and data are increasingly distributed across cloud platforms, implementing consistent security policies to restrict and manage access control becomes more challenging.
- Those problems create an urgent need for a much more robust, adaptive, and resilient security model. Zero Trust Security Model resolves this issue by ensuring that all access request is continuously authenticated, authorized, and validated regardless of inter-service or external communication.

1.4 PROBLEM STATEMENT

Traditional perimeter-based security models fail to keep sensitive data and resources secure. These environments, designed with a limited understanding of the threats facing today's decentralized networks, allow for lateral movement and insufficient identity verification, both of which leave Kubernetes clusters vulnerable to sophisticated cyberattacks. Emerging new requests or needs of organizations operating in such modern environments demand a more advanced architecture, such as that presented by the Zero Trust Security Model, providing continuous identity verification, encrypted communication, and adaptability of security policies across microservices that helps in minimizing the exposure footprints and averting any unauthorized access.

1.5 SIGNIFICANCE AND MOTIVATION

The Zero Trust Security Model in modern web applications using microservice architecture has highly been of importance. The model addresses several critical issues facing organizations as they transition to cloud-native architectures, remote work models, and globally distributed operations across various cloud providers. The motivation for implementing Zero Trust in Kubernetes clusters lies in its ability to mitigate security threats inside the clusters, enhance protection for applications, reduce lateral movement, and meet regulatory compliance requirements.

- Mitigation of Internal Threats: Zero trust introduces strict access control and continues to authenticate identities. These reduce the risks associated with communicating containers by ensuring only authorized individuals are able to access the information

- Cloud Application Security: Zero Trust’s dynamic approach to securing containerized environments ensures that access requests are evaluated in real time, protecting data across various platforms.
- Reduction of Lateral Movement: By using micro-segmentation and enforcing stringent access control policies, Zero Trust limits an attacker's ability to move laterally within a compromised cluster [5].
- Compliance and Regulation: In the light of highly increasing regulatory demands, Organizations must prove their capacity for monitoring and controlling access to sensitive data, with zero trust offering a framework that proves to meet these demands through continuous authentication and authorization of all access requests.

1.6 OBJECTIVE AND SCOPE OF THE PROJECT

This project has been designed to create a zero-trust secure web application that integrates the concept of Zero Trust Security Model over microservices architecture. This application would involve JWT authentication, mTLS, and some control policies for micro-segmentation to ensure that the security framework is quite robust. The following were the objectives:

- Continuous Identity Verification: Implement mechanisms to authenticate users and devices with every request, using JWT for stateless, token-based authentication and mTLS for mutual verification.
- Adaptive Access Control: Develop dynamic access control policies that adjust based on user roles and request context, ensuring minimal privilege for users.
- Encrypted Communication: Secure all communications within the application using mTLS to protect against interception and tampering.
- Prevention of Lateral Movement: Implement micro-segmentation to isolate components and prevent attackers from moving laterally across the system.
- Real-Time Threat Detection: Machine learning algorithms would be integrated to detect anomaly user behavior allowing proactive security actions to be taken [6].

The scope of this project encompasses the design and implementation of the Zero Trust Security Model within a microservice architecture, focusing on integration of advanced security features such as root/debug detection and code obfuscation to enhance application security, deployment of a tool for detecting shell command injection in Kubernetes containers to safeguard against specific vulnerabilities. And continuous evaluation and adjustment of security measures to adapt to the evolving threat landscape.

CHAPTER 2

LITERATURE REVIEW

2.1 LITERATURE REVIEW BASED ON PREVIOUS RESEARCH PAPERS

D'Silva et al. researched the adoption of Zero Trust Architecture (ZTA) in Kubernetes environments [7], emphasizing its relevance in a cloud computing context. They emphasized how ZTA continuously verified users, devices, and applications, challenging traditional trust models. The study discussed the use of technologies such as Kubernetes, Docker and RBAC/ABAC for enhanced access control. Additionally, they critiqued traditional security frameworks, noting the inadequacies of perimeter-based models in modern cloud infrastructures. Their proposed architecture improved security through continuous verification and logging, making it adaptable to decentralized systems.

Varalakshmi et al. [8] explored the enhancement of JSON Web Token (JWT) authentication within Software Defined Networks (SDNs). They identified vulnerabilities in traditional JWT implementations and proposed a modified authentication approach that improves security without sacrificing performance. Their analysis highlighted the importance of securing API endpoints and user sessions, ultimately enhancing the overall integrity of network communications in SDN environments.

Bucko et al. [9] assessed how the JWT authentication and authorization process could be improved in web applications using user behavior history. Their study analyzed how behavior-based metrics could improve the reliability of authentication processes. They proposed a dynamic authentication mechanism that adapts based on user activity, significantly reducing unauthorized access and enhancing user security in web applications.

Jánoky et al. [10] conducted an analysis of revocation mechanisms for JSON Web Tokens (JWTs). They identified the challenges associated with effectively revoking tokens without degrading system performance. Their research emphasized the need for efficient revocation strategies to ensure the security of applications relying on JWTs, contributing valuable insights into the management of token lifecycles.

Achary and Shelke [11] have analyzed fraud detection in banking transactions by applying some machine learning methodologies. They proposed a framework in order to reduce the fraudulent activities by using multiple algorithms in real time. This showed the effectiveness of machine learning for the enhancement of financial transaction security, providing banks with the same skills to prevent fraud. Thus, the authors underscore the adaptive models that learn from fraud patterns, which are dynamic in nature, making credit card security protocols much more effective.

Prusti et al. [12] explores ensemble machine learning models for fraudulent credit card transaction detection. By combining multiple algorithms, it is an effort to enhance precision, accuracy, and speed of detection. Key findings show that ensemble methods outperform individual classifiers in identifying fraudulent transactions, reducing false positives, and increasing overall efficiency in real-time fraud detection systems.

Jaculine Priya and Saradha [13] reviewed machine learning algorithms and did comprehensive analysis encompassing various techniques, assessing their strengths and weaknesses in different contexts. The paper provided a critical overview fraud detection systems, emphasizing the potential of machine learning in enhancing predictive accuracy and operational efficiency in fraud management.

A. I. Weinberg et al. [14] studied Zero Trust Architecture (ZTA) for applications and network security, highlighting its relevance in contemporary cybersecurity frameworks. The research detailed how ZTA eliminates implicit trust and enforces strict access controls, providing a robust model for securing sensitive data in dynamic environments.

Kang et al. [15] provided a brief survey of the theory and application of Zero Trust security. They examined various implementations and discussed how Zero Trust principles could enhance overall security posture. Their findings illustrated the applicability of Zero Trust in diverse settings, emphasizing its importance in mitigating modern cybersecurity threats.

Ashfaq et al. [16] conducted an in-depth review of the Zero Trust security framework. This study investigated existing literature while outlining fundamental principles and strategies applicable for Zero Trust in various scenarios. The authors pointed out gaps in related studies, thus providing ideas for future research to deepen understanding and applicability of Zero Trust principles.

Liu et al. [17] evaluated the relevant literature for Zero Trust, seeing its potential in IoT systems. They have identified major challenges as well as approaches towards Zero Trust principles in IoT environments. Consequently, they concluded that strict access controls with continuous verification must be in place to solve security issues. The authors provided a comprehensive overview of current research trends and highlighted areas requiring further investigation to enhance IoT security through Zero Trust frameworks.

Mehraj and Banday [18] proposed a Zero Trust framework with cloud computing contexts to overcome security-related issues such as identity theft, data breaches, and complications in trust management. The model proposed by the authors emphasizes the necessity of strong access controls coupled with continuous verification owing to the dynamic and shared nature of cloud services. The authors explained the inadequacy of traditional security approaches within the context of cloud.

Muddinagiri et al. [19] presented a method for deploying Kubernetes locally using Minikube to manage Docker containers, emphasizing the advantages of containerization as a lightweight alternative to virtual machines. The paper highlights the importance of local Kubernetes testing for industries like finance and healthcare, which require secure, scalable applications without relying on cloud-based deployments. Their approach was demonstrated using a Python-based web server built with a DockerFile.

Pace's thesis [20] explores the implementation of Zero Trust Networks using Istio within a Kubernetes environment. Istio, an open platform, operates as a service mesh where proxies handle traffic management, observability, security, and extensibility. Key features include traffic shaping, canary deployments, strong identity verification via mutual TLS, and JWT-based authentication. Istio's modular and extensible architecture integrates seamlessly with Kubernetes, allowing dynamic configurations without modifying applications, offering a robust approach to microservices security and operational efficiency.

Kurbatov's thesis [21] presents the design and implementation of secure communication between microservices using the Istio service mesh. Istio enables secure service-to-service communication, load balancing, and traffic monitoring without altering application code. It deploys Envoy proxy sidecars to manage traffic, encryption, authentication, and authorization, ensuring robust security. Istio's architecture includes Certificate Authority (CA) for key management, Policy Enforcement Points (PEPs), and X.509 certificates for workload identity. This approach enhances communication security across a microservice-based system, making it adaptable to deployment demands while maintaining high-level security and control.

Yang et al. discussed the security challenges in container cloud environments [22], identifying vulnerabilities across the kernel, container, and orchestration layers. These include container escape, resource exhaustion, and insecure configurations. Critical CVEs like CVE-2019-5736 and CVE-2020-2023 highlight how attackers exploit runtime vulnerabilities to escape containers. The paper also emphasizes weak network isolation in Kubernetes clusters, posing risks like ARP or DNS spoofing and BGP hijacking. The authors recommend robust kernel isolation mechanisms and improved configuration tools for enhanced security.

2.2 LITERATURE SUMMARY TABLE

S.NO	Author(s)	Title	Objective	Methodology	Conclusion
1	D. D'Silva, D. D. Ambawade	"Building a Zero Trust Architecture Using Kubernetes" [7]	To explore how Kubernetes can be used to implement a Zero Trust architecture	Discusses the use of Kubernetes for Zero Trust implementation	Zero Trust principles can be efficiently implemented using Kubernetes infrastructure
2	P. Varalakshmi, B. Guhan, P. V. Siva, T. Dhanush, K. Saktheeswaran	"Improvising JSON Web Token Authentication in SDN" [8]	To enhance JWT authentication in SDN environments	Uses modified JWT authentication techniques for Software-Defined Networks (SDN)	The JWT enhancements provide improved security and efficiency in SDN environments
3	A. Bucko, K. Vishi, B. Krasniqi, B. Rexha	"Enhancing JWT Authentication and Authorization in Web Applications Based on User Behavior" [9]	To enhance JWT authentication and authorization based on user behavior history	Applies behavior-based analysis for JWT authentication improvements	Behavior-based JWT offers better security against attacks like token theft
4	L. V. Jánoky, J. Levendovszky, P. Ekler	"An Analysis on the Revoking Mechanisms for JSON Web Tokens" [10]	To analyze and improve revoking mechanisms in JWT	Analyzes current JWT revocation mechanisms and suggests enhancements	Proposed mechanisms for revoking JWT improve security in token-based

					authentication systems
5	R. Achary, C. J. Shelke	“Fraud Detection in Banking Transactions Using Machine Learning” [11]	To apply machine learning algorithms for fraud detection in banking transactions	Uses machine learning models for detecting anomalous patterns in banking transactions	Machine learning proves effective in detecting fraudulent transactions in real-time
6	D. Prusti, S. K. Rath	“Fraudulent Transaction Detection in Credit Card by Applying Ensemble Machine Learning Techniques” [12]	To develop ensemble machine learning techniques for credit card fraud detection	Applies a combination of machine learning algorithms to detect credit card fraud	Ensemble techniques significantly improve the accuracy of fraud detection in credit card transactions
7	G. J. Priya, S. Saradha	“Fraud Detection and Prevention Using Machine Learning Algorithms: A Review” [13]	To review the application of machine learning algorithms in fraud detection and prevention	Provides a systematic review of various machine learning algorithms used for fraud detection	Machine learning offers innovative and efficient solutions for fraud detection and prevention
8	A. I. Weinberg, K. Cohen	“Zero Trust Implementation in the Emerging Technologies Era: A Survey” [14]	To survey the current trends in Zero Trust implementation across emerging technologies	Surveys recent developments and research in Zero Trust implementation for various technologies	Zero Trust is crucial for improving security in modern, emerging technology landscapes
9	H. Kang, G. Liu, Q. Wang, L. Meng, J. Liu	“Theory and Application of Zero Trust Security: A Brief Survey” [15]	To present a brief overview of the theoretical framework and applications of Zero Trust security	Surveys the key theoretical aspects and applications of Zero Trust in modern IT environments	Zero Trust security is a growing necessity in securing critical infrastructure and IT systems

10	S. Ashfaq, S. A. Patil, S. Borde, P. Chandre, P. M. Shafi	"Zero Trust Security Paradigm: A Comprehensive Survey and Research Analysis" [16]	To provide a comprehensive survey of the Zero Trust security paradigm	Conducts a comprehensive survey of Zero Trust models and security practices	Zero Trust represents a paradigm shift in security, emphasizing identity verification at every access point
11	C. Liu, R. Tan, Y. Wu, Y. Feng, Z. Jin, F. Zhang, Y. Liu, Q. Liu	"Dissecting Zero Trust: Research Landscape and Its Implementation in IoT" [17]	To dissect the Zero Trust security model and explore its implementation in IoT environments	Provides an analysis of the research landscape and specific implementation strategies for Zero Trust in IoT	Zero Trust offers promising solutions for securing IoT environments, but challenges such as scalability remain
12	S. Mehraj, M. T. Banday	"Establishing a Zero Trust Strategy in Cloud Computing Environment" [18]	To establish a Zero Trust strategy tailored for cloud computing environments	Proposes a cloud-specific Zero Trust strategy and architecture	Zero Trust enhances security in cloud environments by reducing the risk of unauthorized access to cloud resources
13	R. Muddinagiri, S. Ambavane, S. Bayas	"Self-Hosted Kubernetes: Deploying Docker Containers Locally with Minikube" [19]	To demonstrate the process of deploying Docker containers locally using Minikube	Uses Minikube for local containerized Kubernetes deployment	Minikube allows for an efficient local setup of containerized applications for testing and development
14	M. Pace	"Zero Trust Networks with Istio" [20]	To explore the integration of Zero Trust principles in networks using Istio	Implements Istio for securing microservices in a Zero Trust network	Istio is a powerful tool for implementing Zero Trust in cloud-native applications

15	A. Kurbatov	"Design and Implementation of Secure Communication Between Microservices" [21]	To design secure communication protocols between microservices	Develops secure communication protocols using encryption and mutual authentication between microservices	Secure communication is essential for Zero Trust in microservice-based architectures
16	Y. Yang, W. Shen, B. Ruan, W. Liu, K. Ren	"Security Challenges in the Container Cloud" [22]	To address the security challenges specific to containerized cloud environments	Analyzes various security challenges in container-based cloud deployments	Container-based cloud environments require robust security strategies to mitigate threats such as isolation failures

Table 2.1. Literature summary table

The above Table provides a summary of the literature survey on research papers related to Zero Trust architecture, JSON Web Token (JWT) security enhancements, and fraud detection using machine learning algorithms. It highlights the objectives, methodologies, and conclusions drawn by various authors, focusing on advancements in security frameworks, especially in cloud computing, web applications, and software-defined networks. The table offers insights into the application of Zero Trust principles, JWT mechanisms, and machine learning techniques for improving authentication, authorization, and fraud prevention across different domains.

CHAPTER 3

METHODOLOGY

The containerized web application proposed in this paper integrates a comprehensive expense tracking system along with adherence to the Zero Trust security principles.Built using React.js for the frontend, styled with Tailwind CSS, and supported by a MongoDB backend with a Python API, the application allows users to manage their financial transactions efficiently while ensuring secure access and data integrity at all levels. This application uses Docker for containerization, Kubernetes for container orchestration, and Helm to dynamically handle configurations of Kubernetes manifests. The complete application is then deployed to an AWS EKS cluster, which is remotely enforced with strict IAM policies and VPC.

3.1 COMMON VULNERABILITIES AND ZERO TRUST PROTECTION

Web application developments commonly suffer common mis-handling and vulnerabilities that lead to attacks such as SQL injection, cross-site scripting, and unauthorized data access. These weaknesses put confidential data at risk for malicious activities on integrity, availability, or confidentiality, hence causing the organization great losses in terms of finance and reputation.

- SQL Injection is probably one of the most popular varieties of attacks where the attacker can manipulate a SQL query through nasty SQL code in input fields, which sometimes could result in an attack that could lead to an unauthorized access, data manipulation, or even complete compromise of databases. The vulnerability above is nullified by Zero Trust architecture through enforcing of strong validated and sanitized processes. By the employment of parameterized queries and prepared statements, developers avoid allowing input data to be treated as executable code.
- Cross-site scripting is an attack whereby malicious scripts are injected on a website visited by someone else. The risks with XSS attacks can range from session hijacking, web page defacement to even malware diffusion. CSP in a Zero Trust framework reduces XSS-related risks by specifying permissible sources of content. Validations as well as sanitizations of user inputs as well as security headers within applications reduce the XSS attack surface.

- One major concern is unauthorized access to data, especially in applications dealing with users' sensitive information. The Zero Trust security models are able to address this by forcing strict controls over the accessing of resources, ensuring that users have only minimal permissions required for the assigned tasks. MFA is therefore able to add another layer of security, whereby systems are in a position to verify the identity of a user through more than one method before they can access it. This further reduces chances of unauthorized access even when credentials are broken.

Moreover, Role-Based Access Control (RBAC) ensures that users only have access to the information and resources necessary for their role. This principle of least privilege limits the potential impact of compromised accounts and reduces the risk of insider threats.

3.2 WEB APPLICATION OVERVIEW

At the core of this application is the user interface, which enables users to perform CRUD operations (Create, Read, Update, Delete) on their financial transactions. Users can manually input their expenses, savings, and investments through a form-based system that adds each entry as a card component. Additionally, users can visualize their transaction data through graphical views implemented using react-chart-js, providing insights into spending patterns and financial health over time. A distinguishing feature of this application is its integration with the ZeroSMS mobile app. The ZeroSMS app automatically parses SMS messages related to transactions, which are often sent by banks or payment services, and synchronizes them with the web application. This functionality is especially beneficial for users who may forget to log some expenses or are too busy to manually enter transaction details. Once an SMS is parsed, the web application notifies the user, allowing them to classify and manage these transactions at their convenience. This automated process helps ensure that even small purchases or overlooked transactions are accounted for within the expense tracker.

Another key feature is the ability for users to share their transaction data with a selected list of auditors. The auditors, who are granted read-only access to user transactions, can view the data to prepare financial reports. This feature is essential for those who require external auditing services or want to maintain transparency with financial advisors. The auditor’s interface displays a list of users who have shared their transaction data, allowing them to access the data without modifying it, which aligns with Zero Trust's principle of minimal privileges for authorized users.

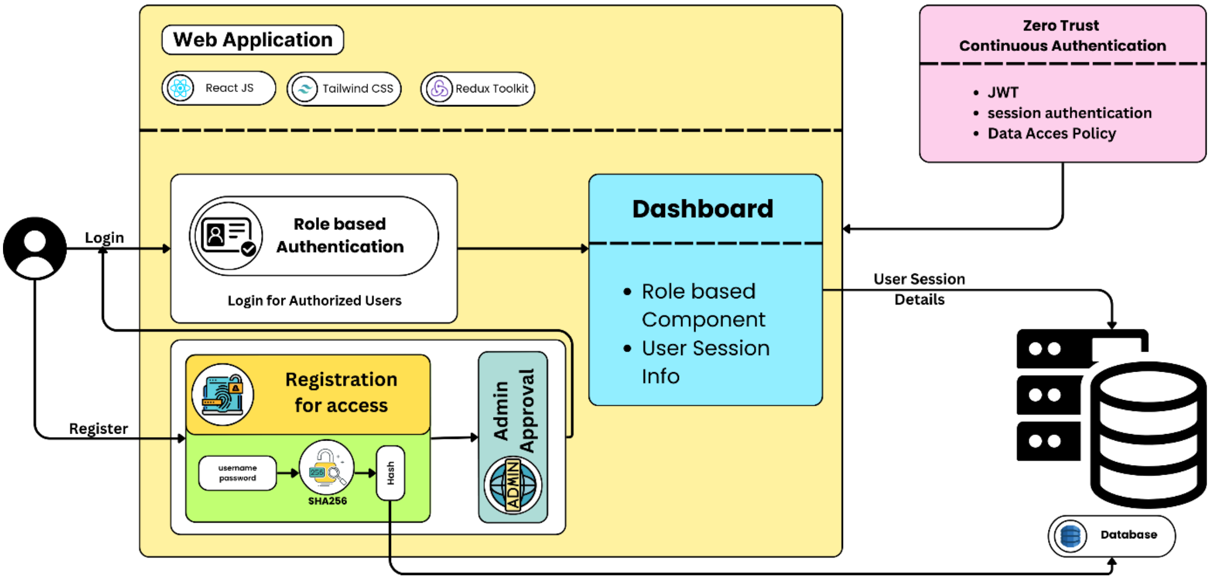

Fig. 3.1 Architecture diagram of the implementation of JWT in Web Application

3.2.1 Admin and Auditor Interfaces

The admin interface is critical for managing user roles and access within the application. New users must first register through a registration page, but their access to the system is only granted upon approval by the admin. The admin has the authority to approve or reject users,

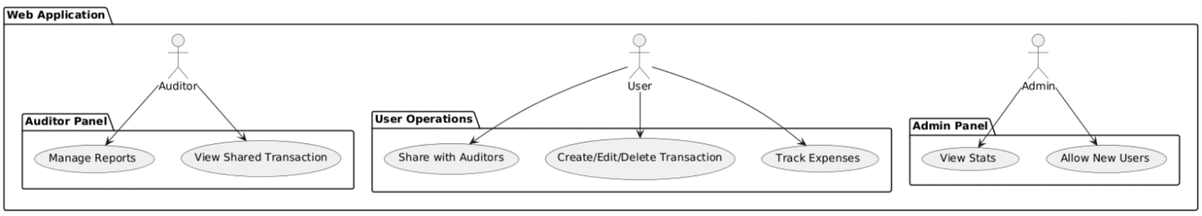

Fig 3.2. RBAC Implementation in the Web Application

as well as assign specific roles such as user, auditor, or admin. This role assignment functionality is presented as a table view, where the admin can easily manage user permissions. Role-based access control (RBAC) ensures that access to the resources of users are limited to only the ones necessary for the role. Again, this supports the principle of least privilege of the Zero Trust model as highlighted in Figure 3.2. In addition to user management, the admin interface provides a user behavior analysis dashboard. This feature enables the admin to monitor important logs, such as user login and logout times, system activity, and other behaviors that may indicate suspicious activity. By continuously monitoring these behaviors, the application can detect anomalies, helping to prevent unauthorized access or potential misuse of the system. This feature reinforces Zero Trust's "never trust, always verify" approach, where users are continuously authenticated and their behavior is scrutinized to detect potential threats.

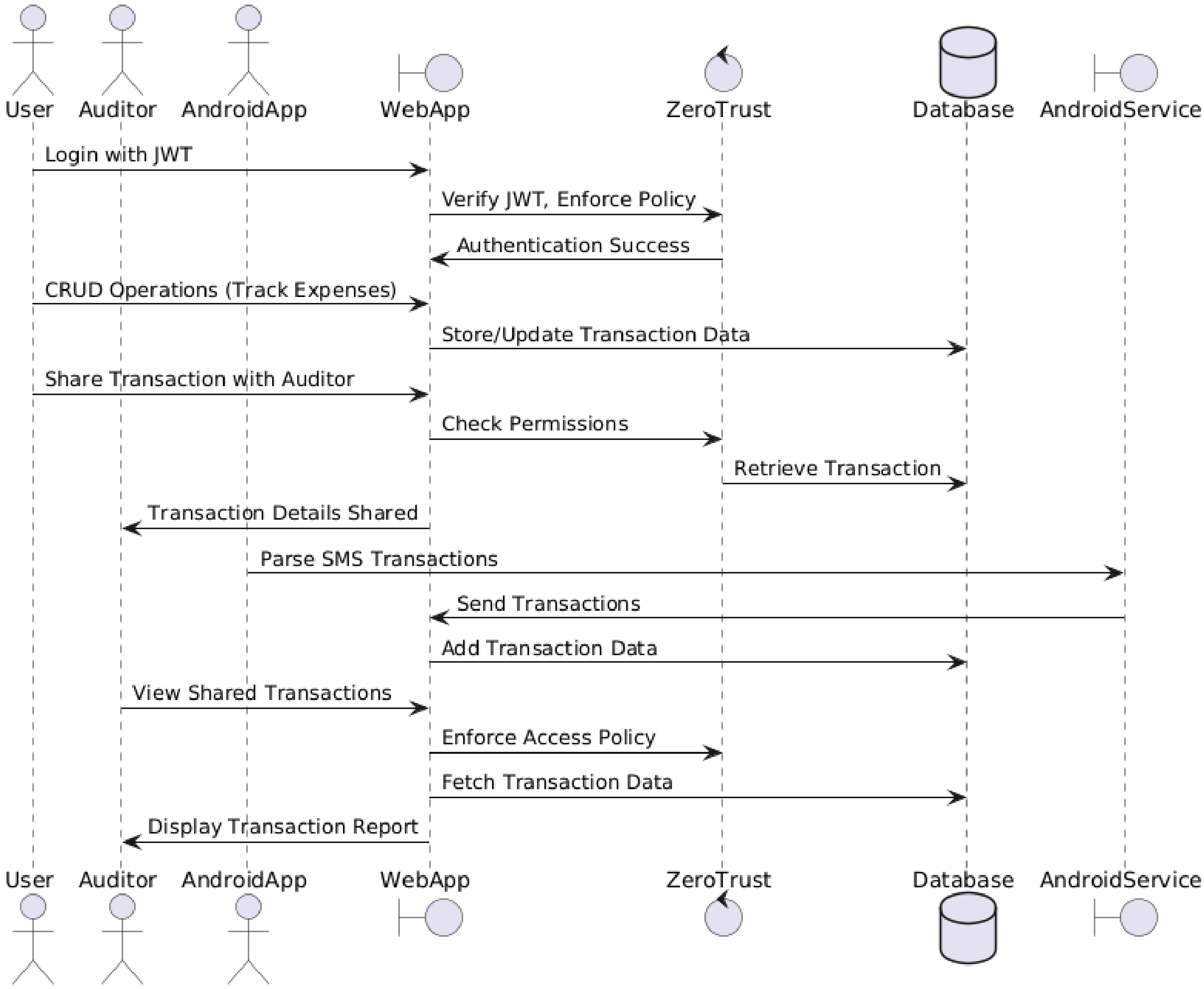

Fig. 3.3. Sequence Diagram representing the order of events taking place

3.2.2 Security and Fraud Detection

The security architecture of the application depends on the Zero Trust framework, which ensures that each user and each transaction is treated as potentially untrusted until verified. Figure 3.4 depicts how each request to access sensitive data is validated through JSON Web Tokens (JWT), which provide secure, time-bound access tokens that validate the user's identity and role. If a user's JWT token is invalid or expired, access is immediately denied, ensuring that no unauthorized requests are processed. The fraud detection model is another critical component of the system. Developed using machine learning (ML) algorithms, the model analyzes transaction patterns to detect potentially fraudulent activity. It is trained on historical transaction data stored in MongoDB, leveraging algorithms such as random forest and support vector machines (SVM) to classify transactions as legitimate or suspicious.

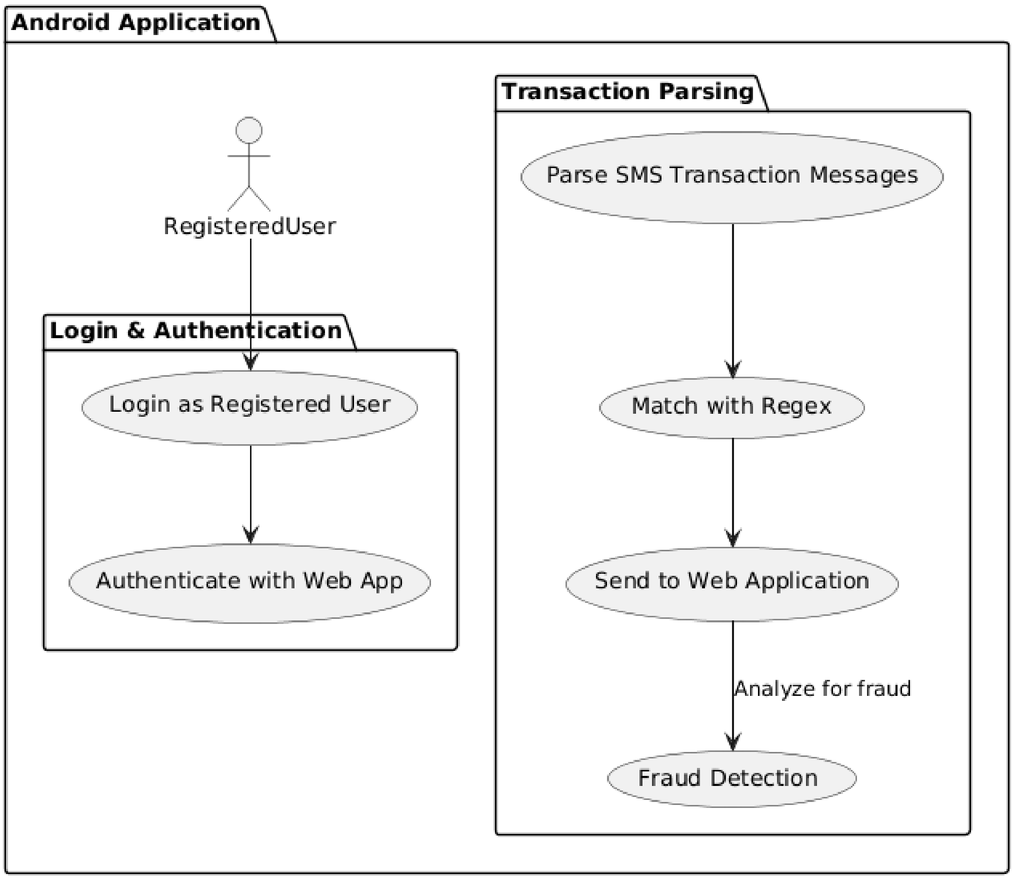

Fig. 3.4. Overview of the Android Application - ZeroSMS

This real-time analysis is particularly valuable for auditors, as they can identify anomalies or red flags that may require further investigation. The fraud detection model complements the Zero Trust approach by providing continuous monitoring and detection capabilities, ensuring that even authorized users do not engage in fraudulent behavior. The application architecture is illustrated in Figure 3.2, which shows the flow of user and admin interactions, SMS parsing through ZeroSMS, and fraud detection. Figure 3.3 illustrates the user behavior analysis dashboard, showing how logs and user activity are tracked and monitored in real time.

3.3 ACCESS TOKENS FUNCTIONALITY AND IMPLEMENTATION

In the context of Zero Trust security architecture, the utilization of Access Tokens, specifically JSON Web Tokens (JWTs), is essential for maintaining robust authentication and authorization mechanisms. JWTs are compact, URL-safe tokens that enable secure information exchange between parties. Composed of three components—the header, which specifies the signature algorithm; the payload, which includes user claims; and the signature, used to authenticate the token—JWTs offer a stateless alternative that benefits both scalability and security in distributed systems. The same is depicted in Figure 3.6.

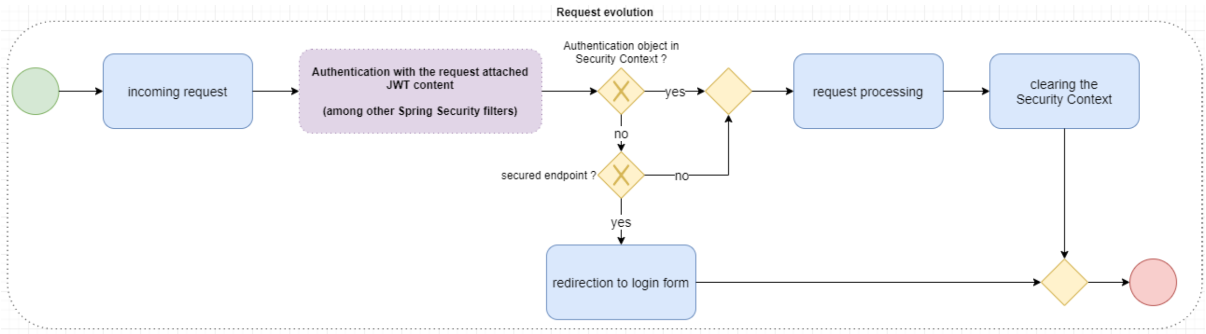

Fig. 3.5. JWT Life Cycle

This structure allows for seamless authentication flows, where the server can validate the token without maintaining session information. As shown in Fig. 1.1, the Python APIs to generate and manage and rigorously validate JWT are containerized and built to run as a separate pod inside the Kubernetes cluster, ensuring continuous authentication of requests from the clients. Upon successful authentication, the system generates a JWT containing essential user-specific claims, such as roles and permissions, which are then transmitted to the user. For subsequent requests, the user includes this token in the authorization header, allowing the application to verify the token's integrity and validity before granting access to protected resources. This implementation is not only based on the principle of least privilege as it limits access based on a user's current permissions but also enhances security by using short-lived access tokens and refresh tokens to mitigate the risk of token theft or misuse. Fig. 3.5 displays the lifecycle of JWTs-from issuing, expiry, and renewal mechanisms.

To improve the security of JWT implementations, organizations can implement and incorporate supplementary security measures including MFA, continuous monitoring, and evaluating user behavior, geolocation, and device health in combination with JWT validation. A robust posture is achieved that is aligned to the basic tenets of Zero Trust. Furthermore, effective revocation mechanisms for JWTs are crucial to ensure that compromised tokens are invalidated promptly. The implementation of these mechanisms, such as utilizing a token blacklist or short-lived tokens with refresh capabilities, is vital for maintaining a secure environment in Zero Trust applications.

Overall, the strategic integration of JWTs within Zero Trust frameworks not only streamlines access management but also ensures a continuous verification process that upholds the security integrity of the application.

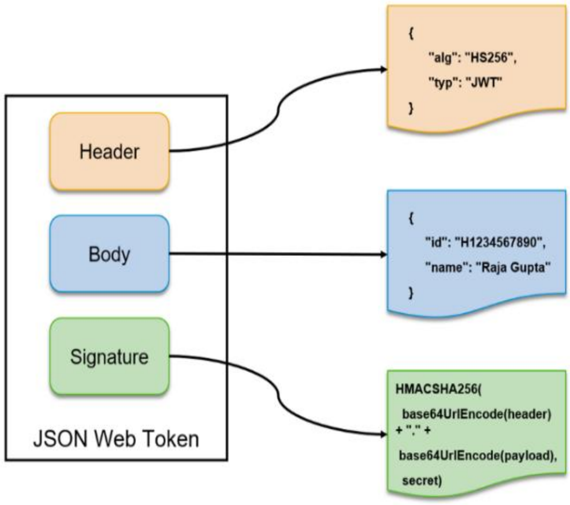

Fig. 3.6. JWT Architecture

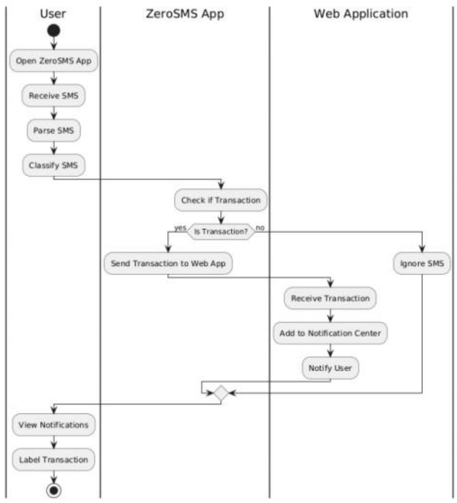

Fig. 3.7. Interaction between the Android App and Web App

3.4 SMS PARSING AND FRAUD DETECTION IN TRANSACTIONS

The increasing reliance on digital financial transactions has highlighted the need for comprehensive expense management solutions that ensure users capture all transactions accurately. The ZeroSMS app addresses this challenge by parsing SMS messages related to financial transactions and classifying them as either transaction-related or non-transaction-related. This functionality not only helps users track their expenses more effectively but also mitigates the risk of overlooking minor transactions such as small purchases and pocket money. By integrating the ZeroSMS app with a web application, users can manage their finances in a centralized location while also ensuring that any missed transactions are flagged for review.

Once the SMS messages are parsed, the app identifies those that pertain to transactions and sends them to the web application, where they appear in a notification center. Users can then revisit these notifications to categorize the transactions as expenses, savings, or investments, and to provide additional information as needed. This approach enhances the accuracy of expense tracking and ensures that all relevant financial activities are recorded. As illustrated in Fig. 3.7, the workflow demonstrates how SMS parsing feeds into the user's expense management process, emphasizing the seamless integration between the mobile and web platforms.

3.4.1 Apache Spark for Machine Learning

The process starts from the ZeroSMS Mobile App that parses messages received as SMS messages for transaction information including debits, credits, and UPI activities. Such

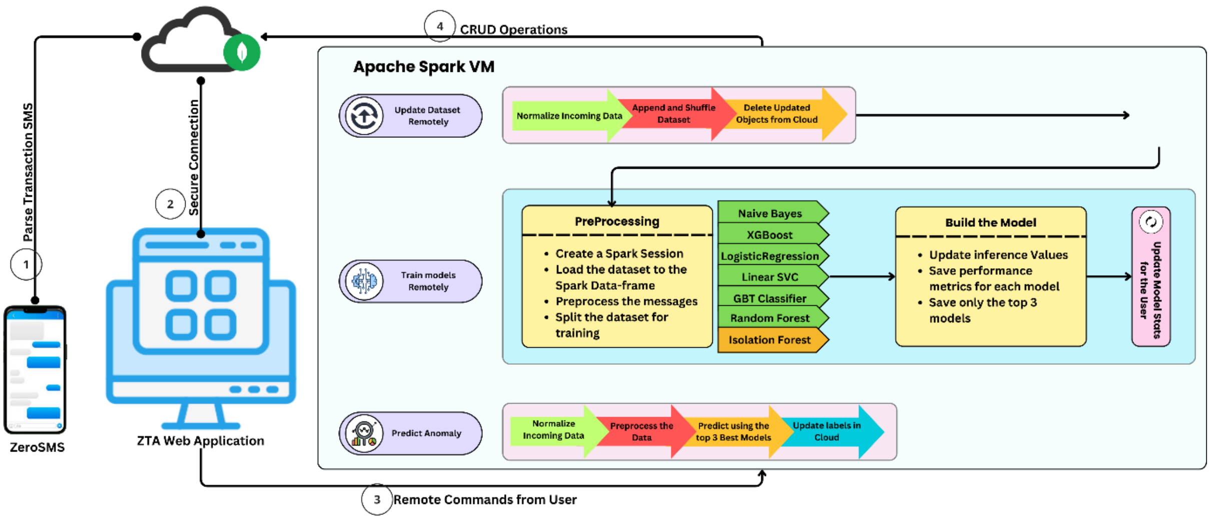

Fig. 3.8. Secure pipeline for real-time SMS parsing, model training, and anomaly prediction.

parsed messages are transmitted securely to the ZTA Web Application. In the application, these messages are normalized, then stored in a MongoDB cloud database. Here in the Ubuntu VM, Apache Spark processes these messages, updating the dataset, shuffling data, and deleting outmoded records. With the dataset ready, distributed computing in Spark is leveraged to train various machine learning algorithms such as Naive Bayes, Random Forest, Logistic Regression, Gradient Boosted Trees (GBT), and Support Vector Classifiers (SVC). For anomaly detection, Isolation Forest, specifically designed for the task of discovering outliers, is added by downloading its JAR file from Maven and then linking it with Spark's framework. This makes the Isolation Forest work along the MLlib algorithm, enhancing anomaly detection by pipeline.

As mentioned in Fig. 3.8, The training process retains only the top three models, according to metrics such as accuracy and precision. When new data arrives, it is preprocessed and fed into these top models for predictions of possible anomalies, like fraudulent transactions. Results are updated back into the MongoDB cloud database to ensure that users have real-time access to the predictions. The ZTA Web Application provides the flexibility and control for the users to execute remote commands like training new models, updating datasets, or running predictions on the whole pipeline. The system evaluates its performance by continually monitoring its models in this iterative learning process. The inference accuracy, precision, and recall are logged and displayed on the ZTA Web Application, giving users actionable insights into the effectiveness of their machine learning models. This feedback loop ensures that only the best-performing models are used for predictions, improving the system's reliability and precision over time.

3.4.2 ML Model for SMS Spam Detection

To further enhance the functionality of the ZeroSMS app, a Machine Learning (ML) model was developed to detect fraudulent transactions. The developed model was built using a combination of supervised learning techniques and ensemble methods, specifically leveraging algorithms such as Random Forest, Gradient Boosting, and Support Vector Machines (SVM). A diverse dataset comprising historical transaction data, including both legitimate and fraudulent transactions, was utilized to train the model. Feature engineering played a critical role in this process, where features such as transaction amount, frequency, and user behavior patterns were extracted from the data to improve classification accuracy.

Classifier Model	Accuracy
Naive Bayes	0.9784688995215312
Support Vector Machine (SVM)	0.9772727272727273
Logistic Regression	0.9778708133971292
Random Forest	0.9742822966507177
Gradient Boosting	0.9694976076555024

Table 3.1. Comparisons of Different classifiers based on Accuracy

The ML model underwent several phases of development, including data preprocessing, model training, and validation. During preprocessing, the data was cleaned and normalized, addressing any inconsistencies or missing values. A cross-validation technique was employed for the assessment of the model’s fit, which actually generalized very well to the unseen data. Thus, after thorough testing, the model achieved an accuracy of over 96%, thus proving it to be effective at pointing out potential fraud cases. As seen in Table 3.1, the performance metrics point to the credibility of classifier models applied, which depicts the capability of the model in terms of offering reduced false positives as well as the false negatives in the course of fraud detection. The integration of this ML model within the ZeroSMS app allows for real-time monitoring of transactions, alerting users to any suspicious activities based on predefined thresholds and user behavior patterns. By continuously learning from new data, the model can adapt to evolving fraudulent strategies, ensuring that users are protected against financial fraud. Ultimately, the ZeroSMS app not only aids in managing expenses but also provides a robust layer of security against fraudulent transactions, enhancing users' overall financial management experience.

3.4.3 Enhancing App Security in React Native

To bolster the security of the React Native application, several libraries and techniques have been integrated. The JailMonkey and RootBeer libraries were utilized to detect if the application is running on a jailbroken or rooted device, which could pose security risks. Additionally, code obfuscation techniques were employed using obfuscator-io-metro-plugin and R8/ProGuard to make reverse engineering of the application more difficult. This obfuscation process modifies the application code to protect it from unauthorized access and manipulation, enhancing overall security.

3.5 POLICY ENFORCEMENT POINT FOR ROLE BASED ACCESS CONTROL

A Policy Enforcement Point (PEP) is a crucial component in enforcing security within web applications, especially in the context of a Zero Trust Architecture. The PEP acts as a gatekeeper that intercepts all incoming requests before they reach the application's core services, ensuring that only authorized users or services gain access When implemented in microservice based web applications, the PEP evaluates each request based on predefined security policies. These policies determine who can access which service and resources, at what time, and under what conditions. The PEP typically collaborates with the Policy Decision Point, which decides access according to such policies as RBAC or ABAC. Once a decision is made by the PDP, the PEP either grants or denies access.

In practical terms, the PEP is embedded at critical points within the microservice architecture, such as API gateways or middleware layers. Every incoming request to the service of application containers passes through the PEP, where it is authenticated, and authorization policies are applied. If the user or service is unauthorized or violates policies, the PEP blocks access. For an instance, in our application a, the PEP ensures that an auditor is allowed to access only the transactions and data related to their client. Also, the routes behind the services are restricted and are specific to the entities (user, admin, auditor) to access. It could also enforce least privilege access, allowing users or services to only perform the actions necessary for their role. Additionally, PEPs can be deployed as sidecar containers in microservices, ensuring that access controls are enforced at each service endpoint. This is especially important when services communicate internally within distributed environments. Overall, PEP strengthens security by ensuring consistent policy enforcement, providing granular access control, and mitigating potential attack vectors.

3.6 SECURE COMMUNICATION WITHIN THE KUBERNETES CLUSTERS USING mTLS

Mutual TLS (mTLS) is a foundational security mechanism in implementing Zero Trust Architecture (ZTA) for microservices. It provides robust authentication and encrypted communication between microservices by requiring both the client and server to present valid certificates, ensuring mutual trust before communication occurs. This mutual verification process is key to the "never trust, always verify" principle of ZTA.

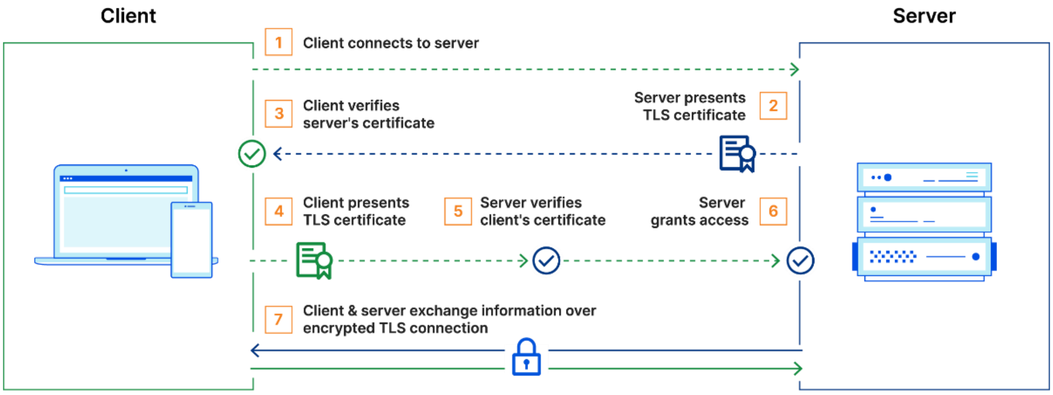

Fig. 3.9. Mutual Transport Layer Security (mTLS) Architecture

In mTLS, there is one distinct cryptographic certificate issued by a trusted Certificate Authority for each pod in the cluster. When any two services have a conversation with each other, it involves handshake time. Both client and server present their respective certificates. Then, one cross-checks the others against the public key of CA to prove genuineness. Once the authenticity of both certificates has been proved, an encrypted communication channel is established through TLS, abbreviated as Transport Layer Security. This implies that the information exchanged between services is authenticated as well as protected from eavesdropping and tampering. The architecture of mTLS is illustrated in Figure 3.9. mTLS is integrated into our application service meshes using a tool named Istio which automate certificate management and enforce mTLS at the network layer. In these environments, mTLS is applied transparently to microservices, without requiring each service to implement its own TLS logic. The service mesh handles certificate issuance, renewal, and rotation, ensuring seamless mTLS enforcement. Unlike traditional systems that assume trust once inside the network, mTLS requires that every interaction between services involves identity verification. This eliminates the risk of unauthorized services accessing resources, as only those with valid certificates are allowed to communicate. When combined with Policy Enforcement Points (PEPs), mTLS strengthens the overall ZTA. The PEP verifies that requests originate from authorized services with valid certificates before granting access to resources.

CHAPTER 4

SYSTEM DESIGN

In this section, we transition from theoretical concepts to practical implementation, detailing system requirements, tools, and the technical infrastructure used to develop and deploy the web and mobile application. We also describe both the deployment process in local and cloud environments using Docker and Kubernetes. This chapter is more theoretically based, with emphasis on illustrating real-world applicability on the project side by focusing on development environments and deployment strategies.

4.1 SYSTEM REQUIREMENTS FOR DEVELOPMENT

This section outlines the software and hardware requirements necessary for building and running the web application frontend, Python backend API, and React Native mobile application. The application was developed on Windows 11 with the following system specifications:

- OS: Windows 11 10.0.22631 (64 bit)
- Processor: 11th Gen Intel(R) Core (TM) i7-11370H @ 3.30GHz
- Cores/Threads: 4 Cores, 8 Logical Processors
- Memory: minimum 8GB RAM
- Secondary Storage: minimum 128GB SSD or HDD

4.1.1 Web Application Frontend (React.js)

The frontend of the application is built using React.js, a popular JavaScript library for building user interfaces.

- Node.js Version: v20.11.0
- NPM Version: 10.8.1 or Yarn Version: 1.22.21
- React Version: 18.3.1

Other necessary tools for the frontend development include Webpack for bundling and Babel for transpiling JavaScript code.

4.1.2 Python Backend API

The backend API, which provides the necessary server-side logic for the application, is developed using Python.

- Python Version: 3.10.6 (> 3.10 preferred)

4.1.3 React Native Mobile Application

The mobile application is developed using React Native, enabling cross-platform compatibility for iOS and Android.

- React Native Version: 0.75.3
- Java Version: 17.0.8 (LTS)
- Gradle Version: 8.8
- Kotlin Version: 1.9.22
- Groovy Version: 3.0.21
- Ant Version: 1.10.13
- JVM: 17.0.8 (Oracle Corporation 17.0.8+9-LTS-211)

4.1.4 Tools Used

- Visual Studio Community 2022: Version 17.11.35327.3 for code editing and debugging
- Android Studio: Android Studio Koala Feature Drop | 2024.1.2 for mobile development and Android emulation

4.2 SYSTEM REQUIREMENTS FOR DEPLOYMENT IN DOCKER AND KUBERNETES

This section details the deployment strategy for the application in both local and cloud environments using Docker and Kubernetes. The hardware and software requirements for each environment, as well as the cloud services used for deployment are mentioned below:

4.2.1 Local Environment

For local development and testing, the system can be containerized using Docker and deployed with Minikube for Kubernetes orchestration. Below are the system requirements for local deployment.

- Operating System Requirements:

1. Windows: 8 and above (64-bit)
2. Linux: Debian or RedHat-based distributions
3. MacOS: Sequoia or later

- Software Requirements:
  1. Docker: Containerization platform
  2. Minikube: For Kubernetes local cluster
  3. Helm: Kubernetes package manager for managing applications
  4. K9s: Terminal-based UI to interact with Kubernetes clusters
- Hardware Requirements:
  1. Processor: Intel® Core™ i7-1165G7 (2.8 GHz up to 3.9 GHz) or AMD equivalent
  2. Memory: Minimum 8GB RAM
  3. Storage: Minimum 128GB SSD or HDD
  4. Network: 10-100 Mbps for downloading dependencies and containers

4.2.2 Cloud Environment

The cloud deployment leverages Amazon Web Services (AWS) to deploy the containerized application with Elastic Kubernetes Service (EKS) and other cloud-native services.

- Software Requirements for the Deployment System:
  1. AWS CLI: For interacting with AWS services
  2. Terraform: Infrastructure-as-code (IaC) tool for provisioning AWS resources
  3. Helm: To deploy Kubernetes resources on EKS
  4. K9s: For Kubernetes cluster management
- AWS Services Used:
  1. Elastic Compute Cloud (EC2): Virtual machines for running services
  2. Elastic Kubernetes Service (EKS): Managed Kubernetes service
  3. Virtual Private Cloud (VPC): For network isolation
  4. Identity and Access Management (IAM): For securing access to AWS services
  5. Elastic Load Balancing (ELB): For balancing incoming traffic across instances
  6. Lambda: For running serverless functions, if required
- Requirements for EKS Nodegroups:
  1. Operating System: Amazon Linux 2 or Ubuntu
  2. EC2 Instance Type: t3.large (2 vCPUs, 8GB RAM, 5 Gbps bandwidth)
  3. Storage: Minimum 50GB Elastic Block Store (EBS)

4.3 ARCHITECTURE OF THE DESIGN

Fig 4.1 depicts architecture of secure transaction and monitoring system integrating multiple components such as a mobile Android app, a web application, and a security module leveraging Zero Trust principles.

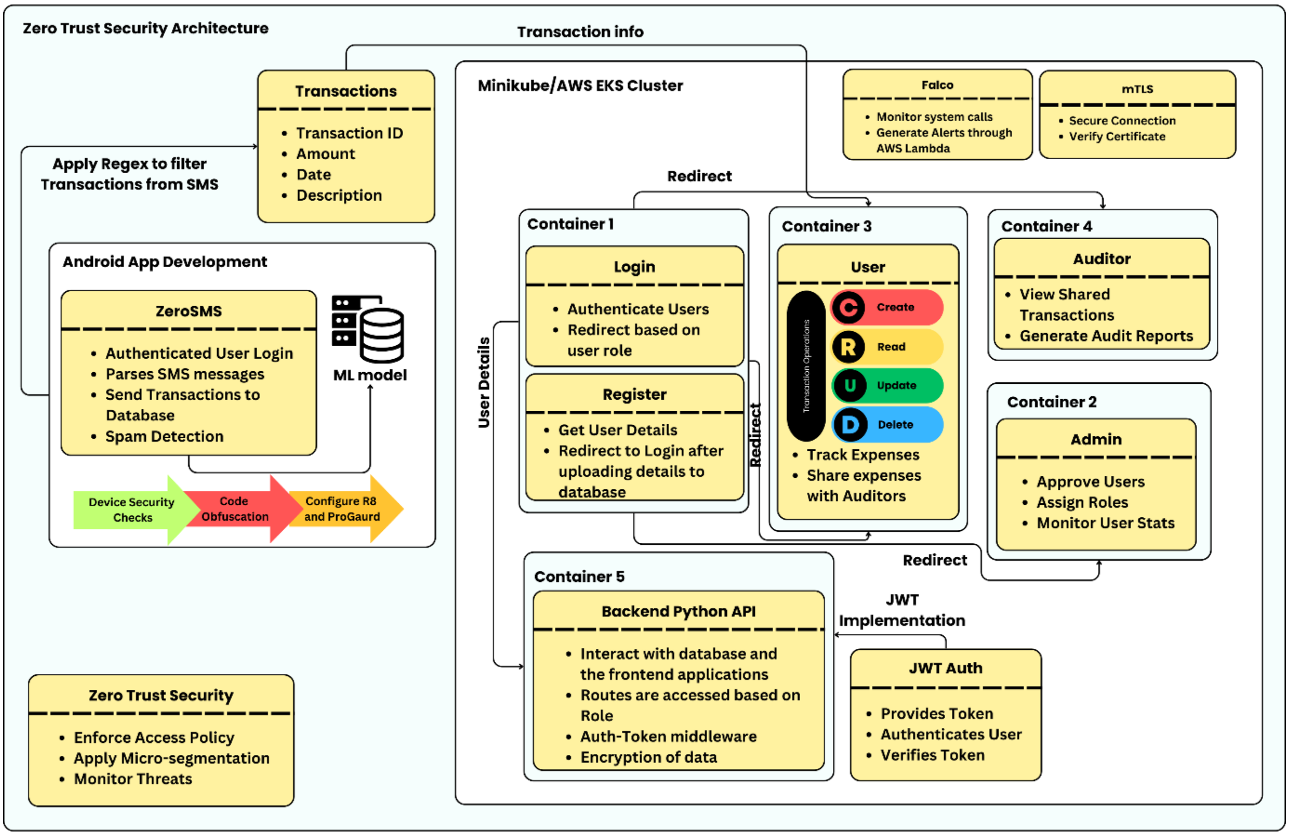

Fig. 4.1 Architecture design of the developed system

1. Android App: The AndroidApp class parses SMS data to extract transaction information, which is then sent to the WebApp for further processing. It also includes methods for user login and spam detection.
2. Web Application: The WebApp class acts as the central hub, managing users, transactions, and applying Zero Trust security policies. The app is responsible for adding transactions, retrieving transaction details for specific users, and applying security policies for authentication.
3. Authentication Layer: JWTAuth (JSON Web Token Authentication) enables authentication of users and validate tokens during login and transaction operations. The authenticate() and verifyToken() methods prevents unauthorized users from accessing the system.

4. mTLS (Mutual Transport Layer Security): This protocol ensures secure communication between the Android app, web app, and users by verifying certificates and securing connections. This ensures that data is encrypted and transmitted over secure channels.
5. Zero Trust Security: The ZeroTrustSecurity class continuously enforces security policies. This includes:
   - enforceAccessPolicy():Ensures that sensitive data can be accessed only by authorized users.
   - applyMicrosegmentation(): Implements microsegmentation to control network traffic and reduce subsequent movements laterally in case of a security breach.
   - monitorThreats(): Monitors potential security threats.
6. Admin, Auditor, and User Roles:
   - Admin: Manages users, approves new users, and views system statistics.
   - Auditor: Generates reports and views detailed transaction reports.
   - User: Can log in to the system, track expenses, and share transaction details with auditors if needed.

4.4 ATTRIBUTES OF THE INPUT DATA

The input data for this system primarily revolves around transaction details and user authentication. Here are the key attributes:

1. Transaction Data (Transaction Class):
   - transactionID (int): A unique identifier for each transaction.
   - amount (float): The monetary value of the transaction.
   - date (Date): The date when the transaction took place.
   - description (string): A brief description or note about the transaction.

Methods like getTransactionDetails() are used to retrieve transaction-specific information, essential for tracking and reporting purposes.

2. User Data (User Class):
   - userID (int): A unique identifier assigned to each user.
   - name (string): The full name of the user.
   - email (string): The email address used for authentication and communication.

The user class has methods like login() for accessing the system and trackExpenses() for viewing and managing personal transactions.

3. Authentication Tokens (JWTAuth Class):
    - token (string): A JWT token generated upon successful authentication. This token is used for session management and verifying the user’s identity across multiple requests.
4. Parsed SMS Data (AndroidApp Class):
    - transactionData (string): SMS content is parsed by the Android app to extract transaction information, which is then sent to the web application for processing.

4.5 ALGORITHM FOR ZERO TRUST ARCHITECTURE BEST PRACTICES

```
ZeroTrustArchitecture main
Initialize ()
if UserLogin then
user ← AuthenticateUser()
if user.authorized then EnforceLeastPrivilege(user)
else DenyAccess()
end if
end if
if RequestAccessToResource then
if VerifyAccess(user) then LogAccess(user)
else DenyAccess()
end if
end if
if ManageContainers then
InitializeAndSecureKubernetes()
DeployContainersWithmTLS()
end if
if DockerizeApp then BuildDeployAndSecureApp()
end if
if MonitorActivity then AuditAndDetectThreats(user)
end if
end procedure
procedure AuthenticateUser
return ValidateCredentialsAndAssignRole()
end procedure
procedure VerifyAccess(user)
return ValidateTokenAndAccess(user)
end procedure
```

4.6 PROTOCOLS AND STANDARDS

The system incorporates several security protocols and standards to protect the confidentiality, integrity, and authenticity of data exchanged within the application. By adopting well-established security measures, the system can defend against common threats in cybersecurity such as data breaches, Main-in-the-middle (MITM) attacks, and unauthorized access. These protocols are essential to maintaining a Zero Trust environment, where no entity is inherently trusted, and every access request must be verified.

4.6.1 JWT (JSON Web Tokens)

Used for authentication and token-based session management, JWT is a light, URL-safe means of representing claims between two parties and is widely used for securing APIs and web applications. authenticate(user) and verifyToken(token) methods ensure secure, token-based access to the system, preventing unauthorized users from accessing sensitive transaction data.

4.6.2 mTLS (Mutual Transport Layer Security)

In a microservices architecture, mTLS is employed to ensure secure interservice communication. Since microservices communicate over APIs and can potentially expose sensitive data, mTLS adds an essential layer of encryption and mutual authentication, ensuring that only authorized services interact with each other. To implement mTLS, a unique certificate must be possessed by each microservice which is issued by a trusted certificate authority (CA), which is verified during service communication. The process involves:

- Certificate Authority (CA): A central authority that issues certificates to each service, which can be internal or external.
- Service Meshes: Tools like Istio or Linkerd that simplify mTLS management in large microservices environments by handling certificate issuance, renewal, and validation.

Istio enhances security by integrating mTLS, ensuring encrypted and authenticated communication between services. Both client and server authenticate each other, protecting against threats like impersonation. Istiod, Istio’s control plane, automates certificate management, issuing unique identities to each service, ensuring all traffic is encrypted and verified. Istio supports permissive and strict mTLS modes for gradual adoption. This approach aligns with Zero Trust principles, where no service is trusted by default. By using sidecar proxies like Envoy, Istio simplifies secure communication without requiring application-level changes. It also automates certificate rotation and revocation, providing effective encryption, authentication, and access controls in distributed microservices environments.

4.6.3 Policy Enforcement Point (PEP)

A Policy Enforcement Point (PEP) refers to security feature responsible for making real-time access control decisions. It evaluates whether an entity (user, device, or service) should be granted access to a resource based on predefined policies. PEPs intercept each request and validate it against a set of security policies, ensuring that only authorized parties have access to protected resources. In microservice architectures, PEPs are crucial for controlling service-to-service interactions and external requests. PEPs enforce access policies at multiple levels, including API gateways, service meshes, and individual services, to ensure that only verified and authorized requests are processed.

4.6.4 Falco: Real-Time Security Monitoring

Falco is an open-source cloud-native runtime security tool developed for microservice based environments. It monitors system calls in real-time to detect abnormal behavior or potential threats. By intercepting system calls using eBPF, Falco analyzes actions like file access and process creation against a set of predefined security rules. When a rule is violated, it generates alerts and can trigger responses, such as blocking processes or notifying security teams. In microservices, Falco enhances security by continuously monitoring for unauthorized access, unexpected process launches, and unusual network activity, thus providing vital insights into runtime behavior and security posture.

4.6.5 Code Obfuscation

Code obfuscation is another important practice that guards the intellectual properties of an application, more so for mobile and web environments that could reverse-engineer the code of an application. Obfuscation translates source code into a very complex version that cannot be easily understood. This makes it difficult for an attacker to reverse-engineer or exploit its vulnerabilities.

- Proguard: Proguard is a popular open-source tool for Java and Android applications that performs a variety of optimization tasks, including shrinking, optimizing, and obfuscating the code. Proguard makes it harder for adversaries to gain insights into application structure by renaming classes, fields, and methods with obscure names.

- R8: R8 is the more modern successor to Proguard, specifically designed for Android development. R8 integrates more tightly with Android build systems and improves both obfuscation and performance optimization. In addition to shrinking and renaming, R8 removes unused code, making it leaner while providing more advanced optimization features than Proguard. One key advantage of R8 is its faster compilation times and more aggressive code shrinking, making it a preferred choice for Android developers today.

4.6.6 Root, Debug, and Developer Options Detection

To further enhance security, the system implements mechanisms to detect rooted devices, active debugging sessions, and whether developer options are enabled. These checks are essential for preventing attacks that rely on modifying the operating system or analyzing the application in a controlled debugging environment. Rooted devices allow users to bypass certain security controls, through which attackers can easily access sensitive data of the application. JailMonkey is an open-source library used primarily in React Native applications to detect if a device has been rooted or compromised in other ways.

By leveraging these standards and protocols, the system ensures secure, authenticated, and reliable management of financial transactions, maintaining robust security through the Zero Trust framework.

CHAPTER 5

IMPLEMENTATION AND RESULT ANALYSIS

This chapter presents the real-time implementation and deployment of the web and Android applications, centered around Zero Trust security practices. It outlines the deployment of the web application on a Minikube cluster and subsequently on AWS EKS, along with the secure implementation of the Android app. Key security measures, including user access controls and data integrity protocols, are discussed to highlight the effectiveness of the proposed solutions in maintaining a secure environment.

5.1 REAL-TIME IMPLEMENTATION OF THE PROTOTYPE

5.1.1 Preparation of Environment in AWS

To prepare the environment for deploying the Zero Trust Architecture, Terraform was used as the Infrastructure as Code (IaC) tool to ensure consistent and automated provisioning of AWS resources. A dedicated Virtual Private Cloud (VPC) was created to host the Amazon EKS (Elastic Kubernetes Service) cluster with high availability and scalability. Similarly, AWS Lambda is configured with AWS CloudWatch to capture alerts from Falco and produce the logs. A Network Load Balancer is deployed along with Nginx-Ingress Controller to route internet traffic to the cluster. Identity, and Access Management (IAM) roles were defined was all the above configurations to enable secure communication and management of resources.

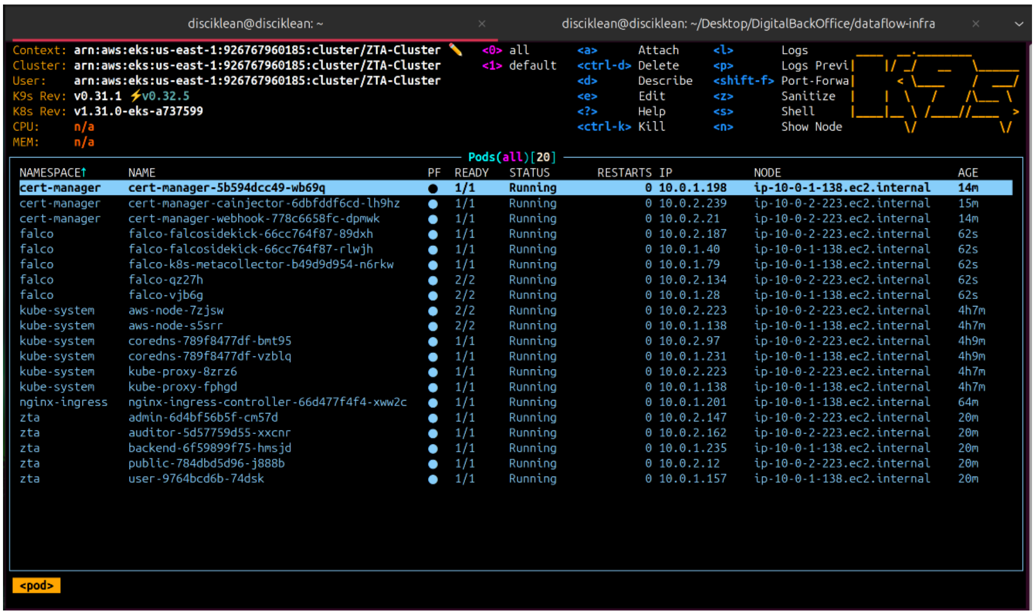

Fig 5.1. List of pods running in the EKS cluster

5.1.2 Deployment of Web Application to EKS Cluster

The web application was containerized and deployed using Kubernetes manifests and a Helm chart. Docker images were created for each component of the application, defining the necessary environment and dependencies. Helm chart was used to manage the deployment of the web application to the EKS cluster. Each component of the application was deployed under their own prefix which is seamlessly managed by ingress controller deployed along with the application. HTTPS protocol was enforced to ensure secure communication between the users and application. Figure 5.1. showcases the pods of Web Application and Falco tool running in EKS cluster.

5.1.3 Deployment of Falco to EKS Cluster

Falco, a cloud-native runtime security tool, was deployed to monitor container activity and detect any anomalous behavior. It was deployed as a DaemonSet in Kubernetes, ensuring that each node in the EKS cluster had Falco instance running for comprehensive monitoring. Custom policy rules were defined to restrict the expected behavior of the containers. Any deviation from these policies. As illustrated in the Figure 5.2. the logs will be given as standard output and also will be exported to AWS cloud watch when any rules defined in Falco gets violated which improves the Zero trust of resources deployed in the cluster.

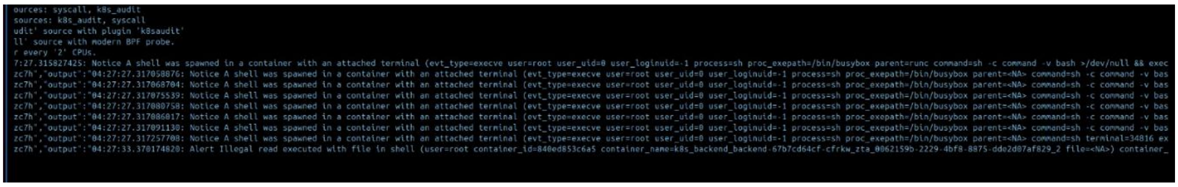

Fig 5.2. Logs as given by Flaco when shell is accessed inside a container

5.2 IMPLEMENTING PRODUCTION BUILD FOR THE ANDROID APP

The ZeroSMS app is designed as a robust security application focused on protecting sensitive information through secure messaging. Its development process involves implementing effective measures to detect rooting and developer options, particularly using JailMonkey, a library that helps identify whether the device is compromised or running in an unsafe environment. By leveraging JailMonkey's capabilities, the app can ensure that it operates only on secure devices, providing users with confidence in the integrity of their communications.

```
    signingConfigs {
        debug {
            storeFile file('debug.keystore')
            storePassword 'android'
            keyAlias 'androiddebugkey'
            keyPassword 'android'
        }
        release {
            if (project.hasProperty('MYAPP_UPLOAD_STORE_FILE')) {
                storeFile file(MYAPP_UPLOAD_STORE_FILE)
                storePassword MYAPP_UPLOAD_STORE_PASSWORD
                keyAlias MYAPP_UPLOAD_KEY_ALIAS
                keyPassword MYAPP_UPLOAD_KEY_PASSWORD
            }
        }
    }
    buildTypes {
        debug {
            signingConfig signingConfigs.debug
        }
        release {
            // Caution! In production, you need to generate your own keystore file.
            // see https://reactnative.dev/docs/signed-apk-android.
            signingConfig signingConfigs.release
            debuggable false
            shrinkResources true
            minifyEnabled true
            proguardFiles getDefaultProguardFile("proguard-android.txt"), "proguard-rules.pro"
        }
    }
}
```

Fig 5.3. Configuring production release with keystore for secure signing and code obfuscation

To prepare the ZeroSMS app for production release, enabling code optimization and obfuscation is essential. This is achieved by configuring the R8 and ProGuard tools within the build system of android. As shown in figure 5.3, By setting minifyEnabled to true in the build.gradle file, R8 is instructed to remove unused code, resources, and perform obfuscation, thereby enhancing the app’s performance and security. Additionally, the app is configured to use a secure keystore for signing the APK, ensuring that only authorized releases are distributed. This involves generating a private signing key with the keytool command and placing the keystore file in the appropriate directory. Finally, the command “./gradlew assembleRelease” is executed to build the APK, applying all the specified configurations, thus resulting in a secure and optimized release version of the ZeroSMS app.

5.3 ML MODEL FOR SPAM DETECTION

The ZeroSMS app is designed to streamline SMS management, particularly for transaction-related messages that help users track their finances. It automatically sends important transaction notifications to a secure database for easy reference. However, spam messages can clutter the inbox, making it difficult for users to identify essential information. To address this issue, incorporating spam detection capabilities became crucial, ensuring that users can focus on legitimate messages and enhancing the app's overall usability.

5.3.1 Dataset

The SMS Spam Collection dataset, accessible on Kaggle, consists of 5,574 tagged SMS messages in English, providing a robust foundation for SMS spam detection research. Each entry in the dataset is structured with two columns: One column, v1, consists of the label, either "ham," meaning it is legitimate, or "spam," and the raw text of the message in the other, v2. The dataset was drawn from the following reliable sources: it contains 425 manually collected spam messages gathered from Grumbletext, a UK forum that provides a posting service for messages about SMS spam; and the other half is comprised of 3,375 randomly selected ham messages from the NUS SMS Corpus, which were collected from volunteers at the National University of Singapore. Other contributions to this dataset include 450 ham messages from a PhD thesis and 1,002 ham messages besides 322 spam messages from the SMS Spam Corpus v.0.1 Big application. Dataset for reference: https://rb.gy/7vbfxx

5.3.2 Preprocessing

The preprocessing stage is crucial for preparing the SMS messages for effective spam detection. Initially, the dataset is loaded into a DataFrame using Pandas library. Unnecessary columns are then dropped to streamline the dataset, retaining only the relevant columns: 'category' and 'msg.' Subsequently, the 'category' column is transformed into a binary format, creating a new column 'spam,' where messages categorized as "spam" are assigned a value of 1, and "ham" messages are assigned a value of 0.

To facilitate the conversion of text messages into a format which is more readable by machine learning models, the CountVectorizer from the Scikit-learn library is employed. This tool transforms the text data into a matrix of token counts. The features (X) and labels (y) are extracted from the DataFrame, with X representing the transformed message data and y representing the corresponding binary labels. Finally, the train_test_split function is used to split the dataset into training and testing sets, in the ratio of 70% for training and 30% for testing. This structured approach guarantees the training of model on a representative sample of the data, allowing for effective evaluation and performance assessment.

5.4 MODELS USED FOR TRAINING

To enhance the performance of the spam detection system, different machine learning models were used. These include Gradient Boosting, Naive Bayes, Logistic Regression, Random

Forest and Support Vector Machine (SVM). Those models were selected from their suitability for specific kinds of classification problems, especially high-dimensional and complex datasets. By leveraging both simple and advanced algorithms, the models offer a balance between computational efficiency and predictive accuracy, making them well-suited for the spam detection problem. Each model's unique characteristics are explored in the following sections.

5.4.1 Naive Bayes

The Multinomial Naive Bayes Classifier is a Bayes theorem-based probability model. It will be a proper classifier in the case of tasks like text classification such as, spam detection-since it follows the feature independence assumption conditioned on the class label. The computations involved in computing the probabilities of each class depend on the extracted features from the SMS message making the model computationally efficient and easy to implement. Its simplicity and effectiveness in handling of high-dimensional data contribute to its popularity in natural language processing tasks.

5.4.2 Support Vector Machine (SVM)

The Support Vector Machine is one of the incredibly powerful algorithms for supervised learning in both classification and regression models. SVM classifies data points correctly due to the best hyperplane with maximum distance between classes. In the following project, the SVM is used with probability=True, so that it is capable of computing the ROC AUC score, which is calculated based on probability estimates. SVM is an algorithm that can deal with non-linear relationships as it uses kernel functions, making it versatile for different kinds of data, including text data.

5.4.3 Logistic Regression

Logistic Regression is a linear model that attempts to do the important task of making a binary classification of given inputs. It estimates the probability that a given input belongs to a particular class using a logistic function. For this project, the model was configured with a maximum of 1000 iterations to ensure convergence. Surprisingly, logistic regression performs very well at any kind of text classification task and extremely well at problems where the features and the target variable are approximately linear, where it is very interpretable and easy to work with.

5.4.4 Random Forest

Random Forest is a technique of ensemble learning where training produces multiple decision trees that then pool their output for enhanced predictive precision and robustness. This model reduces the problem of overfitting common to the algorithms used for decision trees. With the averaging predictions of several trees, Random Forest is applied to enhance classification performance in favor of being a more general solution. So, suitability in handling huge amounts of data with many features makes this kind of classifier particularly suitable for spam-detection tasks where the input space may well be enormous.

5.4.5 Gradient Boosting

Another approach of ensemble learning using which models are built in sequence is called gradient boosting. New models are created by correcting errors committed by earlier models. This approach generates an effective prediction model by aggregation of many weak learners, which are often a decision tree. The Gradient Boosting Classifier in this project leverages this approach to improve accuracy and handle complex relationships within the data. It is particularly effective for tasks like spam detection, as it can capture intricate patterns in the dataset, leading to enhanced classification performance

5.5 RESULT ANALYSIS

This section highlights the effectiveness of the security measures pertaining to code obfuscation, permission management, and real-time threat detection deployed across both platforms. For the web application, the analysis focuses on the successful deployment within an EKS cluster, featuring AWS EC2 instances for both the frontend and backend. Performance metrics such as session logs and security alerts triggered by AWS Lambda with Falco showcase the system’s robustness and resilience against vulnerabilities.

5.5.1 Web Application Deployment in EKS Cluster

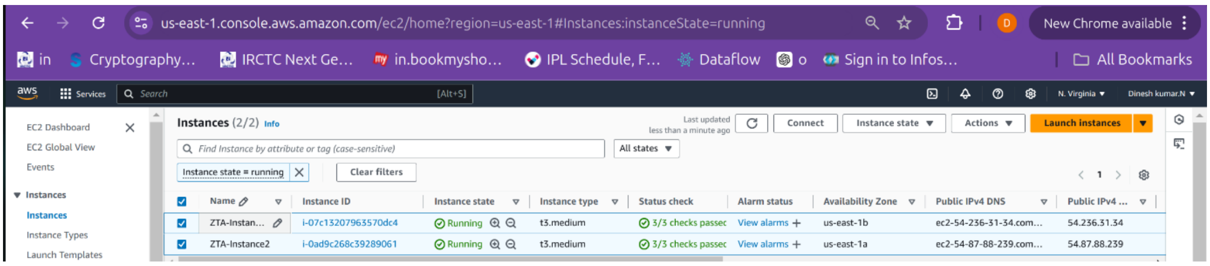

Fig 5.4. Node Groups of EKS Cluster deployed as EC2 instances

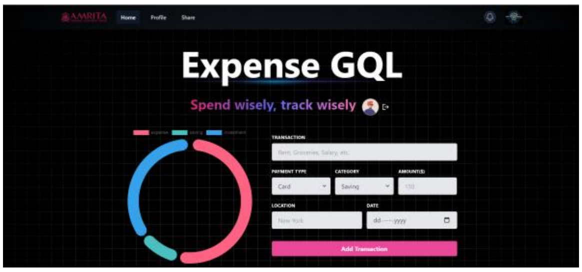

Fig 5.5. User dashboard for CRUD operations

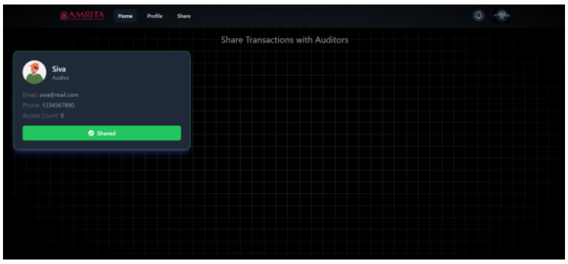

Fig 5.6. Share data with auditors

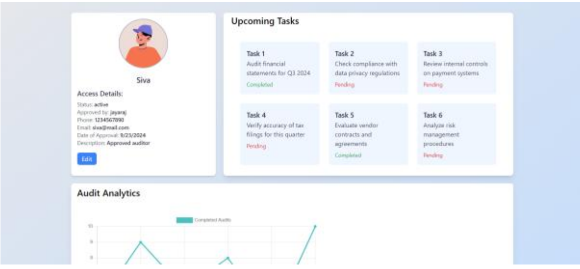

Fig 5.7. Auditor dashboard with tasks

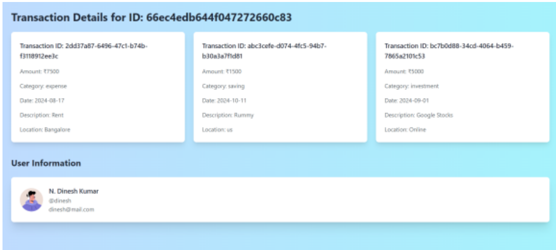

Fig 5.8. Auditor’s read-only view of transactions

Fig 5.9. Admin dashboard for assigning roles

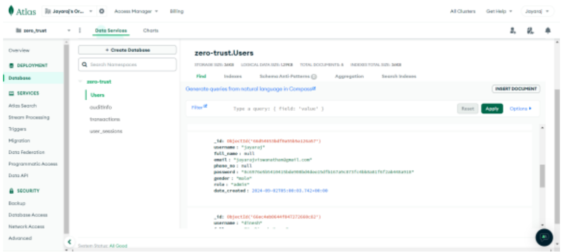

Fig 5.10. Database Collections

The web application was deployed in an EKS cluster, utilizing node groups belonging to the EKS Cluster. As illustrated in Figure 5.4, the architecture ensures scalability and fault tolerance, with seamless communication between services for a smooth user experience. The user dashboard enables CRUD operations, allowing users to manage data securely, as illustrated in Figure 5.5. Data sharing with auditors is seamlessly integrated, ensuring selective access to sensitive information, as shown in Figure 5.6. The auditor dashboard provides a clear overview of tasks, and auditors have read-only access to transactions, as depicted in Figures 5.7 and 5.8. The admin dashboard facilitates dynamic role assignment and access control, as illustrated in Figure 5.9. Additionally, the database collections efficiently store role-based access and transaction data, as shown in Figure 5.10.

Security within the application is strengthened through AWS Lambda and the Falco tool, which monitors the cluster for anomalies and triggers alerts for real-time threat detection, as illustrated in Figure 5.11. Fig. 5.11 showcases the alerts from Lambda as logs captured in AWS Cloudwatch. These logs contain valuable information such as the violated rule, container name, container ID, pod name, and namespace where the attack occurred, along with the system call used to perform the attack. This collective data emphasizes the enforcement of Zero Trust strategy and aids in identifying the root cause of suspicious activity. Additionally, session logs provide a thorough audit trail, tracking user activities and identifying suspicious behavior, as shown in Table 5.1. These logs ensure compliance with Zero Trust principles by ensuring all user actions are continuously monitored and recorded.

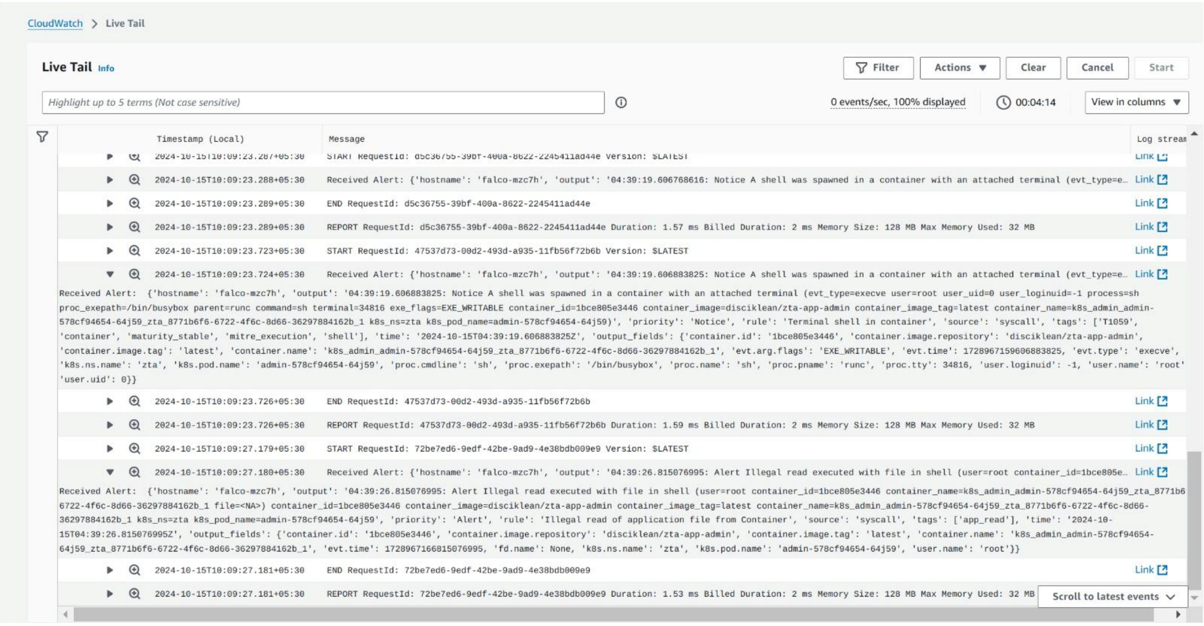

Fig 5.11. AWS Lambda Alerts triggered by Falco tool deployed in the cluster

Field	User 1	User 2	User 3
_id	66e9072851dcced0a150dbe0	66e936584582a105b70ee0a9	66f1340bafc75d1c676baebe
user_id	66d54653bdf0a55b5e126a57	66e934f44582a105b70ee0a8	66f133e9afc75d1c676baebb
username	jayaraj	dinesh	siva
role	admin	user	auditor
sessions	1	1	1
Date	17-09-24	03-10-24	23-09-24
Login Time	2024-09-17T04:35:52.286+00:00	2024-10-03T07:57:12.037+00:00	2024-09-23T09:25:30.920+00:00

Table 5.1 Session logs for logged in users

5.5.2 Android Application Production Build Results

The Android application was developed with comprehensive security measures in place, as illustrated in Figure 5.3, which shows an alert dialog indicating that developer options are enabled. Additionally, the application includes safeguards to prevent screenshots during the login process, further enhancing its security protocols.

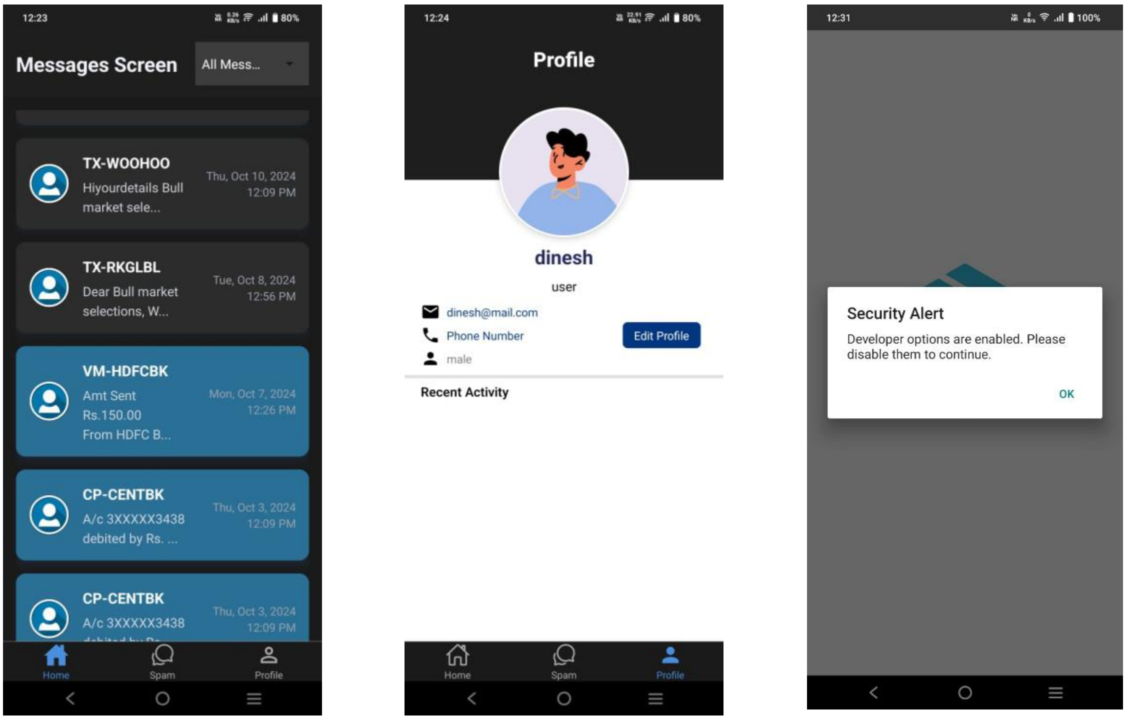

Fig 5.12. ZeroSMS in production release with security features

5.5.3 Comparing and Testing ZeroSMS Apks with Other Normal Apks

Due to the implementation of code obfuscation tools, the reverse engineering of the ZeroSMS APK significantly differs from that of standard APK files, as depicted in Figures 5.4 and 5.5. The class names within the application have been obfuscated, complicating an attacker's ability to navigate through the folders and files. The utilization of R8 and ProGuard further enhanced the obfuscation process, as illustrated in Figure 5.6. In contrast, Figure 5.7 presents a typical APK where the code remains clearly visible and easily understandable.

Figure 5.8 presents a comparison of the MobSF scans for both the ZeroSMS APK and a standard APK. The security score for ZeroSMS is 57, earning a ‘B’ grade, while the normal APK scores 31, receiving a ‘C’ grade. It is important to note that ZeroSMS has SMS read and write permissions enabled, and “http” traffic is permitted in its AndroidManifest.xml file through the script: *android:usesCleartextTraffic="true"*. This configuration was necessary for connecting to the backend service and significantly influenced the security score as android application by default connects only to ‘https’. In contrast, the normal APK is merely a "Hello World" application built in Android Studio, lacking any additional privileges. Despite this, ZeroSMS achieved a security score that is 83.87% higher than that of the standard APK.

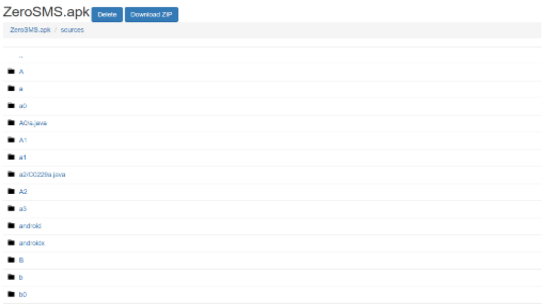

Fig 5.13. ZeroSMS class name Obfuscation

Fig 5.14. Reverse engineered file names

Fig 5.15. ZeroSMS code Obfuscation

Fig 5.16. Normal Apk's reverse engineered code

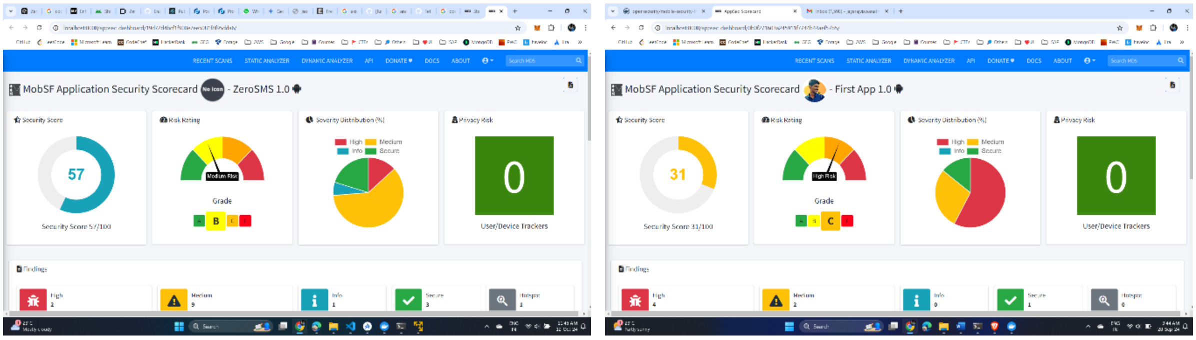

Fig 5.17. Comparison between MobSF scans of ZeroSMS and Normal Apk file

5.5.3 Comparing Various ML models for Spam Detection

In evaluating the performance of the classifiers employed in this project, several key metrics were analyzed as in Table 5.3 to understand which model is best suited for the given dataset. Naive Bayes emerged as the most-effective model, attaining a high F1 score of 0.918, which indicates a well-balanced measure of precision and recall. The recall rate for Naive Bayes stood at 0.918, signifying its robust capability in correctly identifying a substantial proportion of actual positive cases. Moreover, the model demonstrated an impressive accuracy of 0.978 see fig 5.8, underscoring its predictive reliability. The mean squared error (MSE) for Naive Bayes was 0.022, indicating minimal deviation between predicted and actual values.

Additionally, the ROC AUC score was calculated at 0.975, reflecting the model's strong discriminative power between the positive and negative classes. The mean absolute error (MAE) also corroborated these findings with a value of 0.022. Support Vector Machine (SVM), while exhibiting a perfect precision score of 1.000, recorded an F1 score of 0.905 due to a slight decrease in recall at 0.826. The accuracy of the SVM model was 0.977, and it attained the highest ROC AUC score of 0.984, indicating its exceptional capacity to distinguish between classes.

Model	F1 Score	Recall	Precision	Accuracy	Mean Squared Error	ROC AUC Score	Mean Absolute Error	Specificity
Naive Bayes	0.918	0.918	0.918	0.978	0.022	0.975	0.022	0.988
SVM	0.9505	0.826	1.0	0.977	0.023	0.984	0.023	1.0
Logistic Regression	0.909	0.845	0.984	0.978	0.022	0.983	0.022	0.998
Random Forest	0.905	0.826	1.0	0.977	0.023	0.981	0.023	1.0
Gradient Boosting	0.87	0.781	0.983	0.969	0.031	0.973	0.031	0.998

Table 5.2. Comparison of Model Performance Metrics

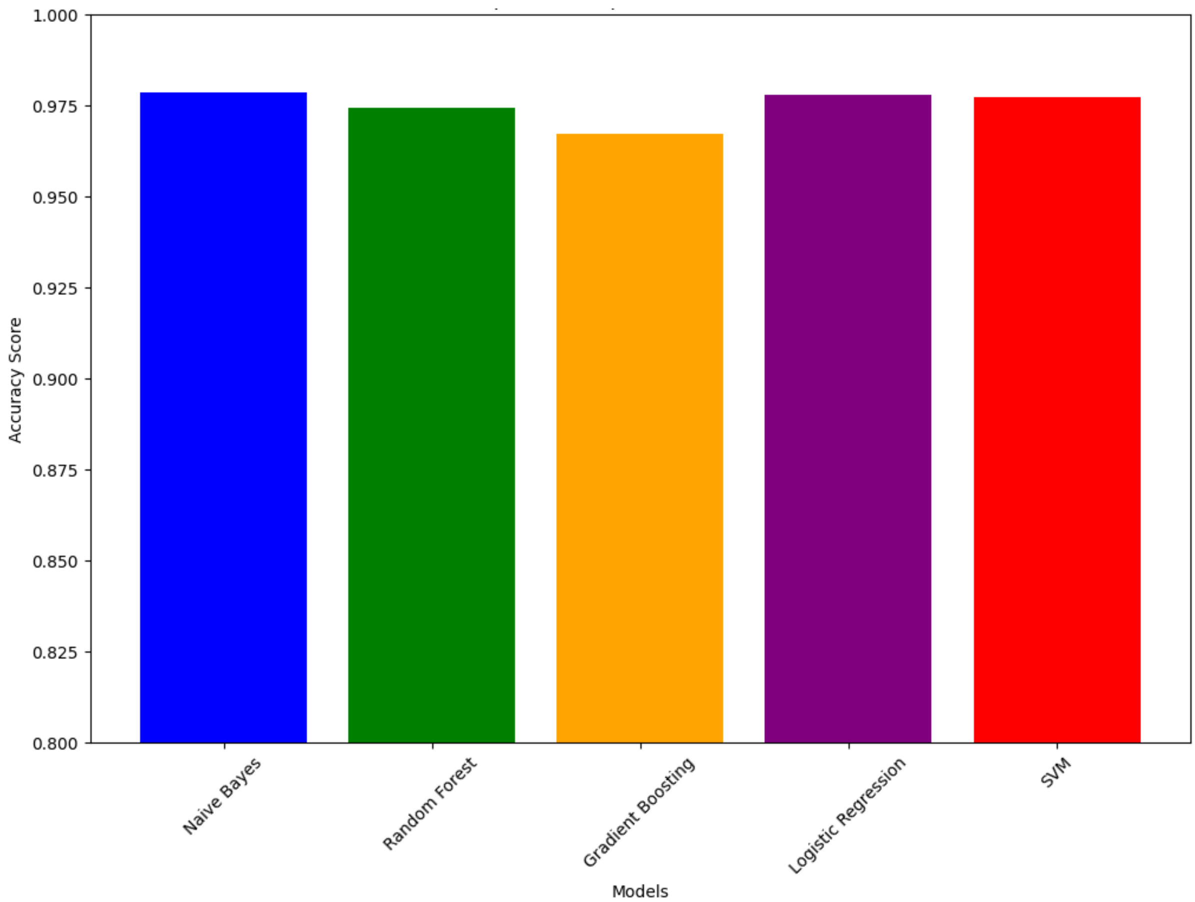

Fig 5.18. Comparison between Spam Detection Models

The MSE for SVM was 0.023, and the MAE was 0.023. Logistic Regression followed closely behind, with an F1 score of 0.909, recall of 0.845, and with an accuracy of 0.978. The model's MSE was recorded at 0.022, with a ROC AUC score of 0.983, reinforcing its effectiveness in classification tasks. Random Forest performance is slightly lower, with an F1 score of 0.891, a recall of 0.804, and an accuracy of 0.974. Its MSE was 0.026, while the ROC AUC score was 0.981. The specificity for Random Forest was perfect at 1.000, indicating its efficacy in identifying true negatives. Gradient Boosting presented the lowest overall performance among the models evaluated, achieving an F1 score of 0.870, recall of 0.781, and an accuracy of 0.969. Its MSE was 0.031, and the ROC AUC score was 0.974, suggesting some limitations in predictive capabilities compared to the other models.

CHAPTER 6

CONCLUSION AND FUTURE WORK

In an era where threats in cybersecurity are increasingly sophisticated, guaranteeing the security of containerized environments is paramount. This project has explored various strategies and best practices to enhance security through the implementation of the Zero Trust strategy. By developing a proof of concept (PoC) which includes a secure web application, a Python API, and an Android application, we have demonstrated how to adopt Zero Trust principles effectively. Key strategies such as JWT authentication, continuous monitoring, code obfuscation, root detection, mutual TLS (mTLS) within containers, and robust policy enforcement for Role-Based Access Control (RBAC) have been applied throughout the development process. The outcomes of this project underscore the importance of adopting a dynamic approach to security in containerized environments. By prioritizing trust verification at every layer and consistently monitoring for potential vulnerabilities, organizations can significantly mitigate risks.

Looking ahead, there are several avenues for further exploration and enhancement of security in containerized environments. Future work could be implemented for analyzing patterns and detecting anomalies in containerized applications through application of machine learning algorithms and further enhancing the process of continuous monitoring. Automating security auditing processes can provide continuous assessments of the security posture of containerized applications, allowing for timely vulnerability scanning and compliance checks against industry standards. Integrating security practices into CI/CD pipelines will ensure that security measures are considered and implemented from the earliest stages of application development. Furthermore, providing resources and training for developers and operators to understand Zero Trust principles will foster a security-first mindset across development teams. Conducting real-world deployments of the developed applications within various organizational contexts would also enable the gathering of data on security performance and user experiences, facilitating ongoing improvements and adaptations of the proposed strategies. By advancing these areas, organizations can strengthen their security frameworks and effectively protect their containerized environments against evolving threats, all while adhering to the Zero Trust model.

REFERENCES

[1] O. E. Imokhai, T. E. Emmanuel, A. A. Joshua, E. S. Emakhu, and S. Adebiyi, "Zero Trust Architecture: Trend and Impact on Information Security," International Journal of Emerging Technology and Advanced Engineering, vol. 12, no. 7, pp. 87-92, July 2022.

[2] S. Rose, O. Borchert, S. Mitchell, and S. Connelly, NIST Special Publication 800-207 Zero Trust Architecture. Gaithersburg, MD: National Institute of Standards and Technology (NIST), 2020.

[3] Y. He, D. Huang, L. Chen, Y. Ni, and X. Ma, "A Survey on Zero Trust Architecture: Challenges and Future Trends," Wireless Communications and Mobile Computing, vol. 2022, Article ID 6476274, 13 pages, 2022. DOI: https://doi.org/10.1155/2022/6476274.

[4] A. Mustyala and S. Tatineni, "Advanced Security Mechanisms in Kubernetes: Isolation and Access Control Strategies," ESP Journal of Engineering & Technology Advancements, vol. 1, no. 2, pp. 45-52, 2021.

[5] B. Pranoto, "Threat Mitigation in Containerized Environments," Applied Research in Artificial Intelligence and Cloud Computing, vol. 6, no. 8, pp. 142-151, 2023.

[6] S. Bagheri, H. Kermabon-Bobinnec, S. Majumdar, Y. Jarraya, L. Wang, and M. Pourzandi, "Warping the Defence Timeline: Non-disruptive Proactive Attack Mitigation for Kubernetes Clusters," in Proc. IEEE International Conference on Communications (ICC 2023), Rome, Italy, 28 May - 01 June, 2023, pp. 567-574.

[7] D. D'Silva and D. D. Ambawade, "Building a Zero Trust Architecture Using Kubernetes," in Proc. 2021 6th International Conference for Convergence in Technology (I2CT), Pune, India, Apr. 2021, pp. 1-5.

[8] P. Varalakshmi, B. Guhan, P. V. Siva, T. Dhanush, and K. Saktheeswaran, "Improvising JSON Web Token Authentication in SDN," in Proc. 2022 International Conference on Communication, Computing and Internet of Things (IC3IoT), Mar. 2022. DOI: 10.1109/IC3IOT53935.2022.9767873.

[9] A. Bucko, K. Vishi, B. Krasniqi, and B. Rexha, "Enhancing JWT Authentication and Authorization in Web Applications Based on User Behavior History," Computers, vol. 12, no. 4, pp. 78-85, 2023. DOI: https://doi.org/10.3390/computers12040078.

[10] L. V. Jánoky, J. Levendovszky, and P. Ekler, "An Analysis on the Revoking Mechanisms for JSON Web Tokens," International Journal of Distributed Sensor Networks, vol. 14, no. 9, Aug. 2018. DOI: 10.1177/1550147718801535.

[11] R. Achary and C. J. Shelke, "Fraud Detection in Banking Transactions Using Machine Learning," in Proc. 2023 International Conference on Intelligent and Innovative Technologies in Computing, Electrical and Electronics (IITCEE), Jan. 2023. DOI: 10.1109/IITCEE57236.2023.10091067.

[12] D. Prusti and S. K. Rath, "Fraudulent Transaction Detection in Credit Card by Applying Ensemble Machine Learning Techniques," in Proc. 2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT), Jun. 2019. DOI: 10.1109/ICCCNT45670.2019.8944867.

[13] G. J. Priya and S. Saradha, "Fraud Detection and Prevention Using Machine Learning Algorithms: A Review," in Proc. 2021 7th International Conference on Electrical Energy Systems (ICEES), Feb. 2021. DOI: 10.1109/ICEES51510.2021.9383631.

[14] A. I. Weinberg and K. Cohen, "Zero Trust Implementation in the Emerging Technologies Era: A Survey," Complex Engineering Systems, vol. 4, pp. 16-28, 2024. DOI: http://dx.doi.org/10.20517/ces.2024.41.

[15] H. Kang, G. Liu, Q. Wang, L. Meng, and J. Liu, "Theory and Application of Zero Trust Security: A Brief Survey," Entropy, vol. 25, no. 12, 2023. DOI: https://doi.org/10.3390/e25121595.

[16] S. Ashfaq, S. A. Patil, S. Borde, P. Chandre, and P. M. Shafi, "Zero Trust Security Paradigm: A Comprehensive Survey and Research Analysis," Journal of Electrical Systems, vol. 19, no. 2, pp. 28-37, 2023.

[17] C. Liu, R. Tan, Y. Wu, Y. Feng, Z. Jin, F. Zhang, Y. Liu, and Q. Liu, "Dissecting Zero Trust: Research Landscape and Its Implementation in IoT," Cybersecurity, vol. 7, no. 20, pp. 1-10, 2024. DOI: https://doi.org/10.1186/s42400-024-00212-0.

[18] S. Mehraj and M. T. Banday, "Establishing a Zero Trust Strategy in Cloud Computing Environment," in Proc. 2020 International Conference on Computer Communication and Informatics (ICCCI 2020), Coimbatore, India, Jan. 2020.

[19] R. Muddinagiri, S. Ambavane, and S. Bayas, "Self-Hosted Kubernetes: Deploying Docker Containers Locally with Minikube," in Proc. 2019 International Conference on Innovative Trends and Advances in Engineering and Technology (ICITAET).

[20] M. Pace, “Zero Trust Networks with Istio”, Master’s thesis, Politecnico di Torino, Turin, Italy, 2020.

[21] A. Kurbatov, “Design and Implementation of Secure Communication Between Microservices”, Master’s thesis, Aalto University, Espoo, Finland, 2020.

[22] Y. Yang, W. Shen, B. Ruan, W. Liu, and K. Ren, "Security Challenges in the Container Cloud," in Proc. 2021 Third IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA), Dec. 2021. DOI: 10.1109/TPS-ISA52974.2021.