

ZERO TRUST SECURITY FRAMEWORK FOR MICROSERVICE ARCHITECTURE DRIVEN WEB APPLICATIONS

A PROJECT REPORT

Submitted by

DINESH KUMAR. N

V. JAYARAJ

[Reg. No CH.EN.U4CYS21014]

[Reg. No CH.EN.U4CYS21026]

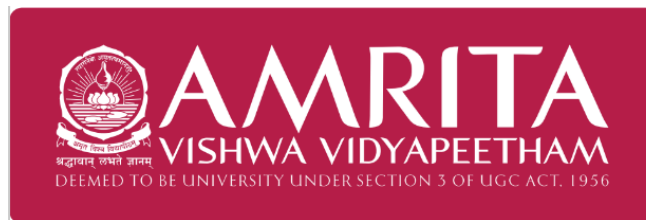
in partial fulfillment for the award of the degree of

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

Under the guidance of

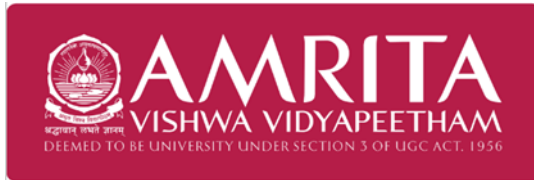
Dr. S. UDHAYA KUMAR

Submitted to



**AMRITA VISHWA VIDYAPEETHAM
AMRITA SCHOOL OF COMPUTING
CHENNAI – 601103**

April 2025



**SCHOOL OF
COMPUTING
CHENNAI**

BONAFIDE CERTIFICATE

This is to certify that this project report entitled “**ZERO TRUST SECURITY FRAMEWORK FOR MICROSERVICE ARCHITECTURE DRIVEN WEB APPLICATIONS**” 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

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

SUPERVISOR SIGNATURE

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

INTERNAL EXAMINER

EXTERNAL EXAMINER



**SCHOOL OF
COMPUTING
CHENNAI**

DECLARATION BY THE CANDIDATE

I declare that the report entitled " **ZERO TRUST SECURITY FRAMEWORK FOR MICROSERVICE ARCHITECTURE DRIVEN WEB APPLICATIONS**" 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

[Reg. No. CH.EN.U4CYS21014]

V. JAYARAJ

[Reg. No. CH.EN.U4CYS21026]

ABSTRACT

With the fast-evolving cybersecurity landscape, cyber threats have rendered the old perimeter-based models obsolete. With companies adopting cloud-native applications as well as microservices-based architecture, a robust and cutting-edge security framework came into being, at least as a risk mitigation factor. The paper explains the design and deployment of the Zero Trust Security Model protecting web applications deployed in extended Kubernetes Clusters to Android applications to enhance mobile security. For the web application, Zero Trust is implemented using JWT authentication, role-based access control (RBAC), continuous authentication, and data encryption, ensuring stringent identity verification and dynamic access control throughout the system. The model also uses fine-grained micro-segmentation and real-time threat detection, protecting critical resources from unauthorized access and preventing lateral movement attacks. In the ZeroSMS Android application, the Zero Trust model is applied to mobile apps by adding security tools such as JailMonkey for root access detection and debugging, code obfuscation using R8 and ProGuard for additional security. These ensure mobile applications are protected from unauthorized use and possible exploits. The architecture also includes mTLS and an unauthorized shell command interaction detection tool to protect containerized environments, providing strong security at application as well as infrastructure levels. Our analysis finds that this approach significantly reduces attack surfaces, provides enhanced protection against new threats, and provides an end-to-end security solution for web and mobile environments. The suggested framework provides a scalable, robust, and flexible security architecture, rendering it highly versatile to the needs of contemporary 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. Sountharajan**, 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	10
	3.1 Common Vulnerabilities and Zero Trust Protection	10
	3.2 Web Application Overview	11
	3.2.1 Admin and Auditor Interfaces	12
	3.2.2 Security and Fraud Detection	13
	3.3 Access Tokens Functionality and Implementation	14

3.4 SMS Parsing and Fraud Detection in Financial Transactions	16
3.4.1 Apache Spark for Machine Learning	16
3.4.2 Enhancing Model Training and Fraud Detection	18
3.4.3 Enhancing App Security in React Native	19
3.5 Policy Enforcement Point for Role Based Access Control	19
3.6 Secure Communication within the Kubernetes Clusters Using mTLS	20
4 SYSTEM DESIGN	21
4.1 System Requirements for Development	21
4.1.1 Web Application Frontend (React.js)	21
4.1.2 Python Backend API	22
4.1.3 React Native Mobile Application	22
4.1.4 Tools Used	22
4.2 System Requirements for Deployment in Docker and Kubernetes	22
4.2.1 Local Environment	22
4.2.2 Cloud Environment	23
4.3 Architecture of the Design	24
4.4 Attributes of the Input Data	25
4.5 Algorithm for Zero Trust Architecture Best Practices	26

4.6	Protocols and Standards	27
4.6.1	JWT (JSON Web Tokens)	27
4.6.2	mTLS (Mutual Transport Layer Security)	27
4.6.3	Policy Enforcement Point (PEP)	28
4.6.4	Falco: Real-Time Security Monitoring	28
4.6.5	Sentry: Real-Time Error Tracking and Debugging	28
4.6.6	Code Obfuscation	29
4.6.7	Root, Debug, and Developer Options Detection	29
5	IMPLEMENTATION AND RESULT ANALYSIS	30
5.1	Real-Time Implementation of The Prototype	30
5.1.1	Preparation of Environment in AWS	30
5.1.2	Deployment of Web Application to EKS Cluster	31
5.1.3	Deployment of Falco to EKS Cluster	31
5.2	Implementing Production Build for The Android App	31
5.3	ML Model for Spam Detection	32
5.3.1	Dataset	33
5.3.2	Preprocessing	33
5.4	Models Used for Training	33
5.4.1	Naive Bayes	34
5.4.2	Linear Support Vector Classifier	34
5.4.3	Logistic Regression	34

5.4.4 Random Forest	34
5.4.5 Gradient Boosting	35
5.4.6 XGBoost	35
5.5 Result Analysis	35
5.5.1 Deployed Web Application in EKS Cluster	35
5.5.2 Android Application Production Build Results	38
5.5.3 Comparing and Testing ZeroSMS Apk with Other Normal Apks	38
5.5.4 Comparing Various ML models for Spam Detection	39
6 CONCLUSION AND FUTURE WORK	42
References	43

LIST OF TABLES

TABLE NO	TITLE	PAGE NO.
5.1	Session Logs for Logged-In Users	37
5.2	Comparison of Model Performance Metrics	40

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	12
3.2	RBAC Implementation in The Web Application	12
3.3	Sequence Diagram Representing the Order of Events Taking Place	13
3.4	Overview Of the Android Application - ZeroSMS	14
3.5	JWT Life Cycle	15
3.6	JWT Architecture	16
3.7	Interaction Between the Android App and the Web App	16
3.8	Secure pipeline for real-time SMS parsing, model training, and anomaly prediction.	17
3.9	Mutual Transport Layer Security (mTLS) Architecture	20
4.1	Architecture Of the Developed System	24
5.1	List Of Pods Running in The EKS Cluster	30
5.2	Logs As Given by Flaco When Shell Is Accessed Inside a Container	31
5.3	Configuring Production Release with Keystore for Secure Signing and Code Obfuscation	32
5.4	Node Groups of EKS Cluster Deployed as EC2 Instances	36
5.5	User Dashboard for CRUD Operations	36
5.6	Share Data with Auditors	36
5.7	Auditor Dashboard with Tasks	36
5.8	Auditor's Read-Only View of Transactions	36
5.9	Admin Dashboard for Assigning Roles	36

5.10	Database Collections	36
5.11	AWS Lambda Alerts Triggered by Falco Tool Deployed in the Cluster	37
5.12	ZeroSMS In Production Release with Security Features	38
5.13	ZeroSMS Class Name Obfuscation	39
5.14	Reverse Engineered File Names	39
5.15	ZeroSMS Code Obfuscation	39
5.16	Normal Apk's Reverse Engineered Code	39
5.17	Comparison Between MobSF Scans of ZerSMS and Normal Apk file	39
5.18	Comparison Between Spam Detection Models	40

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 quick pick-up of agile development practices, cloud computing, and containerization technologies has broadened the attack surface exponentially, making it easy for cybercriminals to exploit vulnerabilities. Legacy perimeter security model of pre-configured trust zones could not match the sophistication of advanced distributed attacks. Phishing, ransomware, and insider attack vectors have been becoming more sophisticated. For instance, ransomware attacks increased by 150% in 2020 alone, costing organizations an estimated \$20 billion. Furthermore, Verizon 2020 Data Breach Investigations Report emphatically indicates that 86% of breaches were financially motivated, thus emphatically stating the deployment of stronger security controls. In response to such evolving threats, Zero Trust Security Model developed a groundbreaking approach that has shifted the security enforcement paradigm across networks, applications, and devices. Zero Trust Model operates on the premises that no entity should be trusted by default whether a user, a device or even an application—and continuous verification should be conducted for every attempt to access [1]. This remedies the vulnerability of

traditional security architectures by mandating continuous authentication and authorization whether the entity is internal or external to the network [2]. Its core concept, "never trust, always verify," requires access to be granted based on a solid evaluation of pre-configured security policies. This paradigm is highly applicable in microservice environments, where data and services are distributed across multiple containers and volumes. Employ technologies like JSON Web Token (JWT) authentication, mutual TLS (mTLS), and micro-segmentation, Zero Trust secures security postures by restricting access to only authenticated and authorized entities. In addition, the application of features such as root/debug detection and code obfuscation in ZeroSMS adds more security through the prevention of reverse engineering and unauthorized access. By the detection of the shell command injection tools, the Kubernetes containers are able to make the attacks stronger by detecting vulnerabilities as early as possible to prevent them. This paper is aiming at an investigation and trying to offer a better design and implementation of a Zero Trust framework in a modern web application in a microservice environment, overcoming the limitations faced 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, presenting new cybersecurity challenges. Traditional perimeter-based security models, referred to as "castle-and-moat" defenses, assume that threats exist primarily outside the network perimeter and resources within the perimeter are trustworthy. With remote work, cloud computing, segmentation, and data and application decoupling, such perimeters are now weak, and applications are exposed to insider threats, compromised resources, and highly sophisticated cyberattacks [3]. Zero Trust Security Model is a security paradigm. In contrast to other models, Zero Trust as shown in Fig.1.1 is not dependent on network location when it comes to establishing trust. Rather, it is requiring each entity—both internal and external—to authenticate and authorize several times before accessing sensitive resources. The model is utilizing technologies such as JWT for stateless user authentication and mTLS for secure, mutual identity authentication between containers or pods executing in Kubernetes clusters to ensure communication is 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.

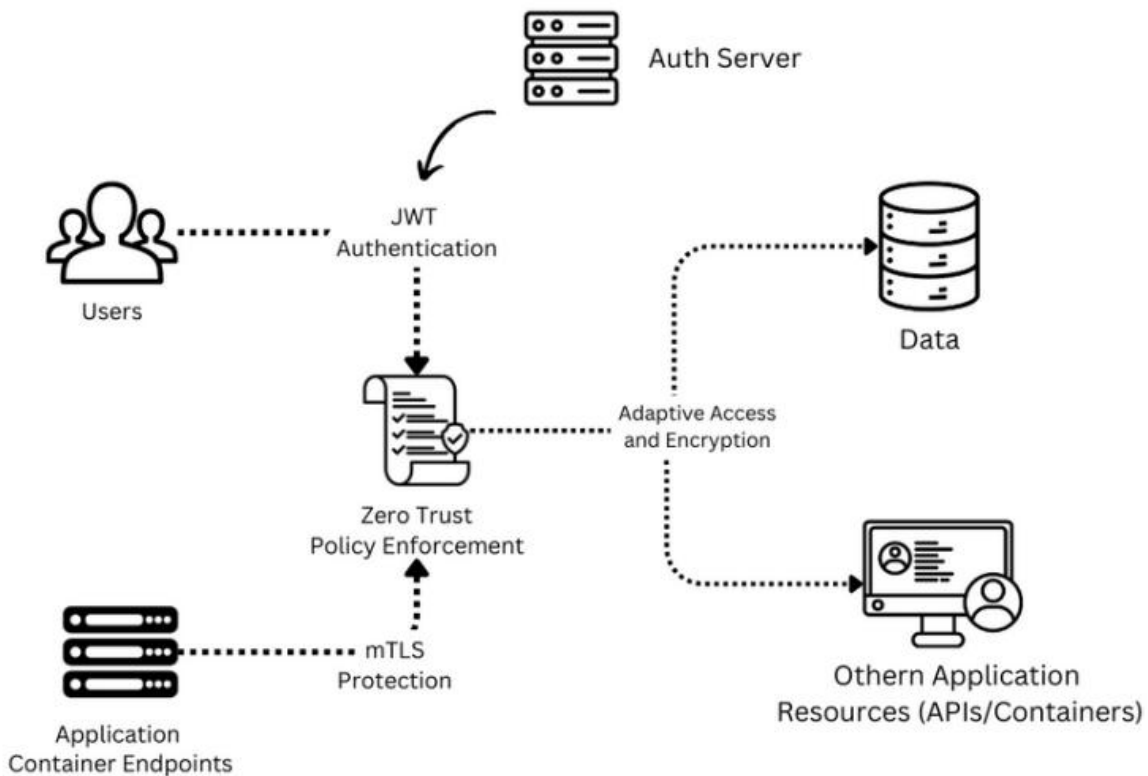


Fig. 1.1 Block diagram describing the design of ZTA and its basic architecture

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 issues need a much stronger, adaptive, and resilient security model immediately. Zero Trust Security Model addresses this issue by ensuring all the request for access is always authenticated, authorized, and validated irrespective of inter-service or external communication.

1.4 PROBLEM STATEMENT

The legacy perimeter-based security models cannot protect sensitive resources and information. Such environments, designed with a limited understanding of the threats to modern decentralized networks, offer lateral movement and poor identity authentication, both of which open Kubernetes clusters to advanced cyberattacks. New emerging requests or organizational requirements in such modern environments need a more advanced architecture, such as the one proposed by the Zero Trust Security Model, with continuous identity authentication, encrypted traffic, and security policy flexibility between microservices that helps to minimize the exposure footprints and block any unauthorized access.

1.5 SIGNIFICANCE AND MOTIVATION

The Zero Trust Security Model for modern web applications that are microservice-based has been of great importance over recent times. The model addresses several of the primary issues companies are facing when they are moving to cloud-native architectures, remote work patterns, and globally distributed operations on multiple cloud providers. The reason Zero Trust is implemented in Kubernetes clusters is that it can assist in preventing security threats in the clusters, enhance application security, reduce lateral movement, and aid regulatory compliance.

- **Internal Threat Mitigation:** Zero trust enforces strict access controls and ongoing identity verification. These minimize the risks of communicating containers by allowing authorized users to access information only

- 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. studied the implementation of Zero Trust Architecture (ZTA) in Kubernetes environments [7], highlighting its application in a cloud computing scenario. They highlighted how ZTA checked users, devices, and apps continuously, defying conventional trust models. The paper referenced the application of technologies like Kubernetes, Docker and RBAC/ABAC for richer access control. Further, they criticized conventional security models, pointing out the shortcomings of perimeter-based models within contemporary cloud infrastructure. Their suggested architecture enhanced security via continuous validation and logging, rendering it compatible with decentralized systems.

Varalakshmi et al. [8] investigated the improvement of JSON Web Token (JWT) authentication in Software Defined Networks (SDNs). They discovered vulnerabilities in standard JWT implementations and introduced a modified authentication scheme that enhances security without compromising performance. Their research emphasized the need to secure API endpoints and user sessions, which eventually leads to improving the overall integrity of network communications in SDN networks.

Bucko et al. [9] evaluated how the JWT authentication and authorization process can be enhanced in web applications through user behavior history. They examined how behavior-based measures can enhance the authenticity of authentication processes. They introduced a dynamic authentication mechanism that adjusts on the basis of user activity, lowering unauthorized access by a great margin and improving user security in web applications.

Jánoky et al. [10] also performed an analysis of JWT revocation mechanisms. They outlined the challenges in revoking tokens efficiently without impacting system performance. Efficient revocation schemes were highlighted as essential to maintaining applications' security based on JWTs, and their work provided significant insight into token lifecycle management.

Achary and Shelke [11] have examined fraud detection of banking transactions using some machine learning approaches. They suggested a framework to decrease the fraudulent transactions using various algorithms in real time. This indicated that machine learning was

effective for financial transaction security improvement, equipping banks with the same abilities to thwart fraud. Hence, the authors highlight the adaptive models that are trained on fraud patterns, which are dynamic, and hence, credit card security measures become even more effective.

Prusti et al. [12] investigates ensemble machine learning models for detecting fraudulent credit card transactions. Through the use of multiple algorithms, it is an attempt to improve precision, accuracy, and detection speed. Major findings indicate that ensemble methods perform better than single classifiers in detecting fraudulent transactions, minimizing false positives, and maximizing overall efficiency in real-time fraud detection systems.

Jaculine Priya and Saradha [13] surveyed machine learning algorithms and conducted thorough analyses including various techniques, evaluating the pros and cons of each in various setups. The article offered a critical overview of fraud detection systems, highlighting the ability of machine learning to increase predictive precision and working effectiveness in fraud management.

A. I. Weinberg et al. [14] examined Zero Trust Architecture (ZTA) for application and network security, underlining its importance in modern-day cybersecurity architectures. The study outlined how ZTA does away with implicit trust and implements rigorous access controls, giving a strong framework for the protection of sensitive information in dynamic settings.

Kang et al. [15] offered a concise overview of the Zero Trust theory and use of security. They explored numerous implementations and shared insight into how principles of Zero Trust would advance the overall security posture. Results displayed the usefulness of Zero Trust across different environments, highlighting the value in neutralizing contemporary cyber threats.

Ashfaq et al. [16] provided a critical review of the Zero Trust security model. This research examined prevailing literature with the objective of explaining basic principles and measures employable for Zero Trust under different circumstances. The authors indicated loopholes in preceding studies, hence offering suggestions for future research to enhance knowledge and usability of Zero Trust principles.

Liu et al. [17] examined the applicable literature to Zero Trust and viewed it with potentiality for IoT systems. They identified the significant issues along with paths toward Zero Trust concepts in the context of IoT networks. On that basis, they found it mandatory to implement strong access control measures with real-time verification in order to rectify security issues. The authors summarized existing trends of research very efficiently and underscored areas needed further exploration for maximizing IoT security through Zero Trust strategies.

Mehraj and Banday [18] suggested a Zero Trust model with cloud computing scenarios to counter security-related concerns like identity fraud, data breaches, and complexity in trust management. The proposed model by the authors highlights the requirement for robust access controls along with ongoing verification due to the dynamic and shared nature of cloud services. The authors described the inadequacy of conventional security mechanisms in the context of cloud.

Muddinagiri et al. [19] outlined a technique for local deployment of Kubernetes with Minikube for Docker container management, highlighting the benefits of containerization as a light-weight over virtual machines. The article highlights the need for local testing of Kubernetes in sectors such as finance and healthcare that need secure, scalable applications independent of cloud deployments. Their solution was illustrated through a Python web server constructed using a DockerFile.

Pace's thesis [20] explains the application of Zero Trust Networks with Istio in a Kubernetes setup. Istio is an open platform that functions as a service mesh where proxies are responsible for managing traffic, observability, security, and extensibility. Traffic shaping, canary releases, robust identity authentication through mutual TLS, and JWT-based authentication are some of its key features. The modular and extensible design of Istio makes it easily integrate with Kubernetes, making dynamic configurations possible without changing applications, providing an enterprise-level approach to microservices security and operational efficiency.

Kurbatov's thesis [21] discusses the Istio service mesh design and implementation of secure microservice communication. Istio allows secure service-to-service communication, load balancing, and traffic observation without changing application code. It installs Envoy proxy sidecars to handle traffic, encryption, authentication, and authorization, all with strong security. Istio architecture consists of Certificate Authority (CA) for key management, Policy

Enforcement Points (PEPs), and X.509 certificates for workload identity. This method adds security to communications throughout a microservice-based system, making it responsive to deployment requirements while upholding high-level security and management.

Yang et al. explained the security issues in container cloud environments [22], listing threats at the kernel, container, and orchestration layers. They include container escape, resource exhaustion, and insecure config. Major CVEs such as CVE-2019-5736 and CVE-2020-2023 illustrate how attackers use runtime vulnerabilities to escape the container. The paper also underscores poor network isolation in Kubernetes clusters, presenting threats such as ARP or DNS spoofing and BGP hijacking. The authors suggest strong kernel isolation features and enhanced configuration utilities for better security.

CHAPTER 3

METHODOLOGY

The containerized web application proposed in this paper combines an all-encompassing expense tracking system with compliance with the Zero Trust security model. Constructed with React.js as the frontend, Tailwind CSS styled, and backed by a MongoDB database with a Python API, the application enables users to effectively manage their financial transactions while maintaining secure access and data integrity at every level. This app utilizes Docker for containerization, Kubernetes for orchestration of the containers, and Helm for managing configurations of Kubernetes manifests dynamically. The entire app is then hosted on an AWS EKS cluster, remotely governed with tight 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 likely to be one of the most well-known types of attacks where an attacker has the ability to tamper with a SQL query by using ugly SQL code in input fields, which at times might lead to an attack that could result in an unauthorized access, data tampering, or even full database compromise. The above vulnerability is avoided by Zero Trust architecture by the enforcement of robust validated and sanitized processes. Through the use of parameterized queries and prepared statements, developers prevent input data from being treated as executable code.
- Cross-site scripting is an attack where harmful scripts are injected on a website used by another person. The threat with XSS attacks can vary from session hijacking, web page defacement to even malware diffusion. CSP in a Zero Trust environment limits XSS-related threats by defining allowed sources of content. Sanitizations as well as validations of user inputs as well as security headers in applications minimize the XSS attack surface.

- One of the biggest issues is unauthorized access to information, particularly in applications that handle users' sensitive data. The Zero Trust security models can solve this problem by imposing rigid controls on accessing resources so that users only get minimal permissions to perform the task assigned to them. MFA can thus provide a further level of security, where systems are capable of confirming the identity of a user using more than one method before they gain access. This still minimizes opportunities for unauthorized access even when credentials are compromised.

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

The core of this application is the user interface, where users can carry out CRUD operations (Create, Read, Update, Delete) on their transactions. Users can manually enter their expenditure, savings, and investments via a form-based system, where each entry is appended as a card component. Users can also see their transaction data graphically via graphical views achieved through react-chart-js, gaining insights into expenditure and financial health over time. One feature of this app is its integration with the ZeroSMS mobile app. The ZeroSMS app automatically parses SMS messages dealing with transactions, normally sent by payment services or banks, and synchronizes them with the web app. This is convenient for users who might forget to log some of their expenses or have no time to enter transaction details manually. The instant an SMS is parsed, the web app sends a notification to the user such that they can categorize and attend to such transactions at their convenience. This is time-saving and ensures small purchases or forgotten transactions end up in the expense tracker.

Another prominent feature is the provision for users to share their transaction information with a chosen list of auditors. The auditors, who are provided read-only access to user transactions, can see the information to generate financial reports. This feature is crucial for individuals who need external auditing services or wish to have transparency with financial advisors. The auditor's interface presents a list of users who have contributed their transaction information so that they can view the information without altering it, which is consistent with Zero Trust's concept of minimal privileges to authorized users.

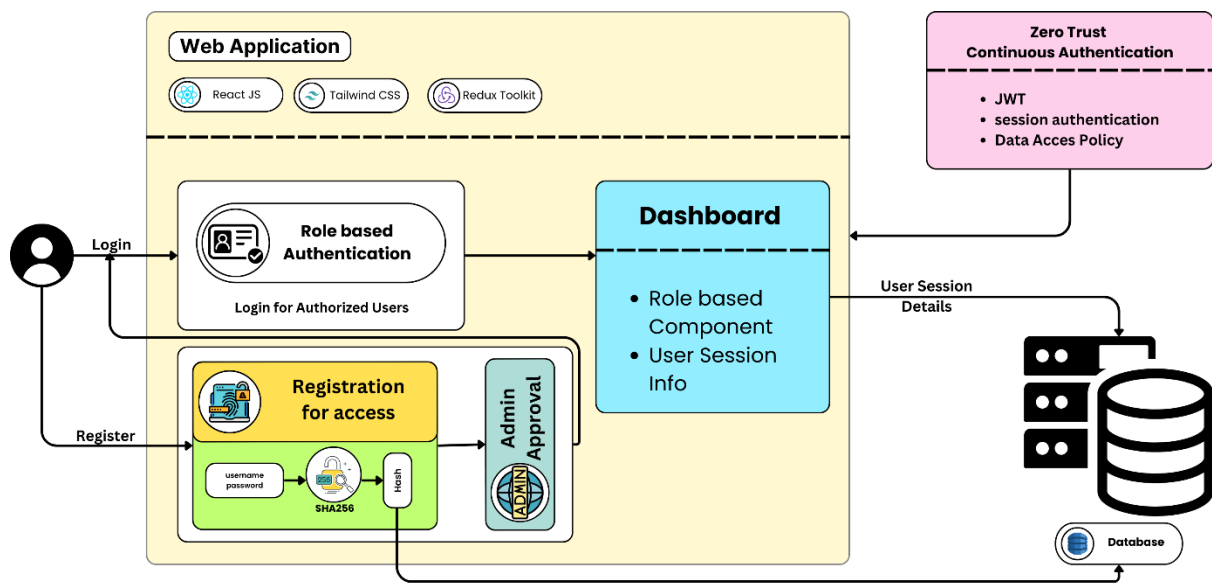


Fig. 3.1 Architecture diagram of the implementation of JWT in Web Application

3.2.1 Admin and Auditor Interfaces

The admin interface plays a vital role in controlling user roles and access within the application. New users have to register first via a registration page, but their access to the system is only authorized after approval by the admin. The admin can approve or deny users, as well as

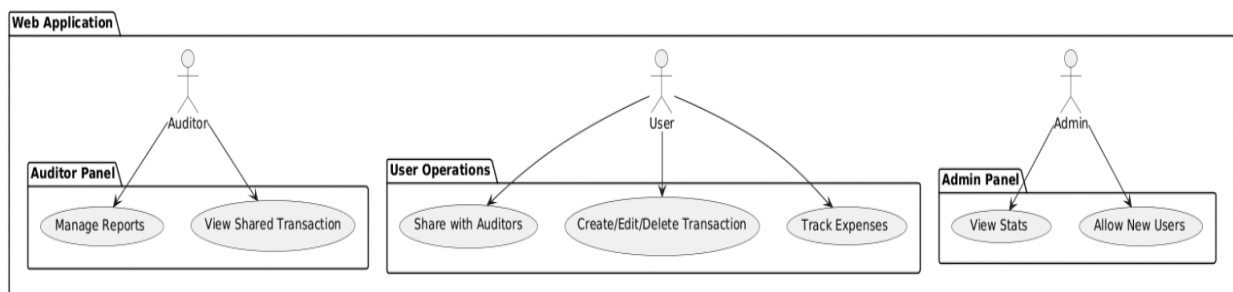


Fig 3.2. RBAC Implementation in the Web Application

designate particular roles like user, auditor, or admin. This role assignment feature is implemented as a table view, making it easy for the admin to handle user permissions. Role-based access control (RBAC) makes it such that the access to users' resources is restricted to only those required for the role. Once more, this enforces the Zero Trust model's principle of least privilege as seen in Figure 3.2. Aside from user management, the admin interface also has a user behavior analysis dashboard. This functionality allows the admin to monitor important logs like user login and logout time, system usage, and other activities that may represent suspicious behavior. Through constant monitoring of such activity, the application will be able to identify anomalies, which can be used to block unauthorized access or misuse of the system. This functionality integrates Zero Trust's "never trust, always verify" philosophy, where users

are continuously authenticated and monitored for suspicious activity so that they can be identified in the event of any threat.

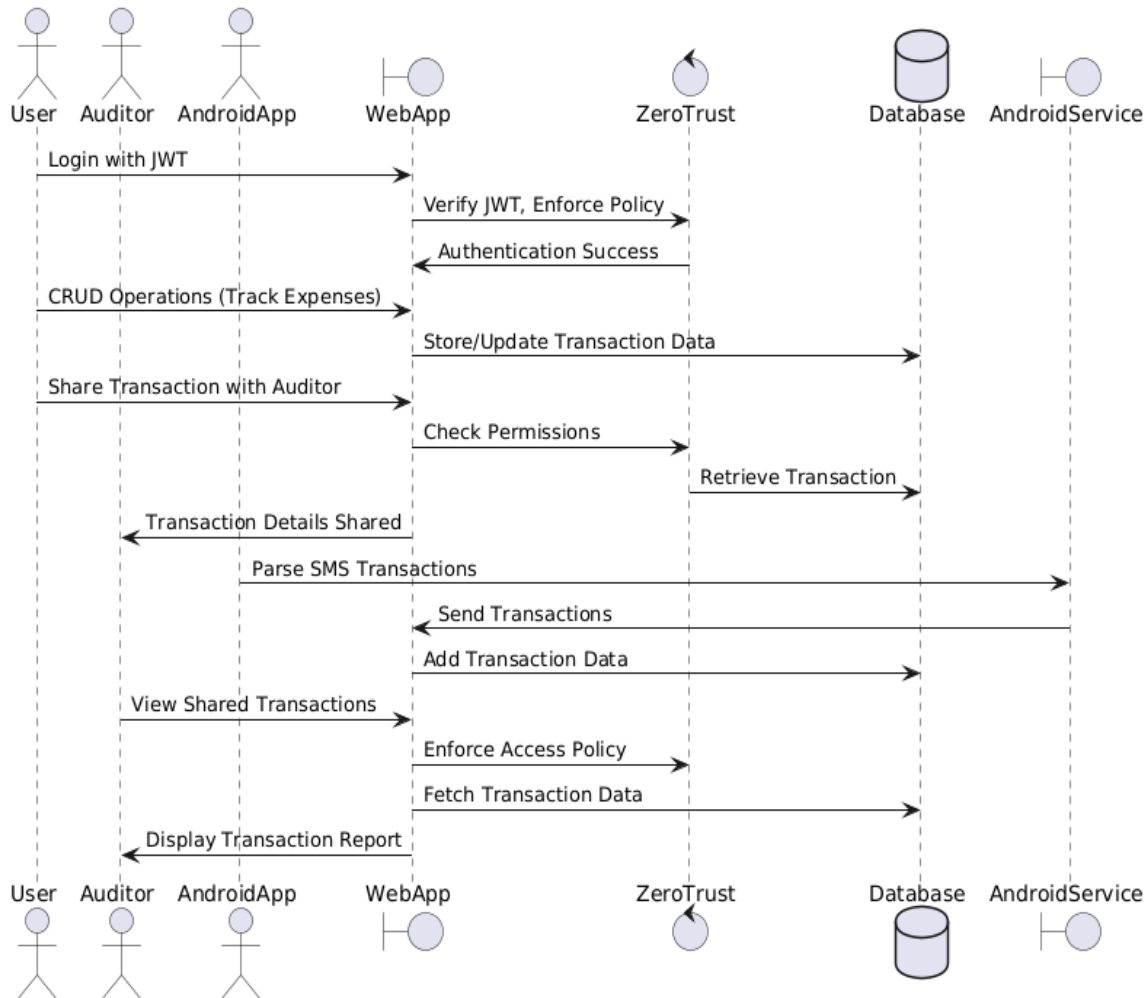


Fig. 3.3. Sequence Diagram representing the order of events taking place

3.2.2 Security and Fraud Detection

The application security architecture adheres to the Zero Trust paradigm, where all users and transactions are presumed to be untrusted until authenticated. Figure 3.4 illustrates how each access request for sensitive data is authenticated through JSON Web Tokens (JWT), granting secure, time-limited access tokens validating the user identity and role. When a user's JWT token is compromised or expired, access is denied on the spot, so no unauthorized requests are ever processed. The fraud detection model is the second important component of the system. Developed with machine learning (ML) algorithms, the model analyzes patterns of transactions to detect potentially fraudulent activity. It is trained on histories of transactions in MongoDB through algorithms such as Linear Support Vector Classifier (SVC) and to classify transactions as legitimate or suspicious.

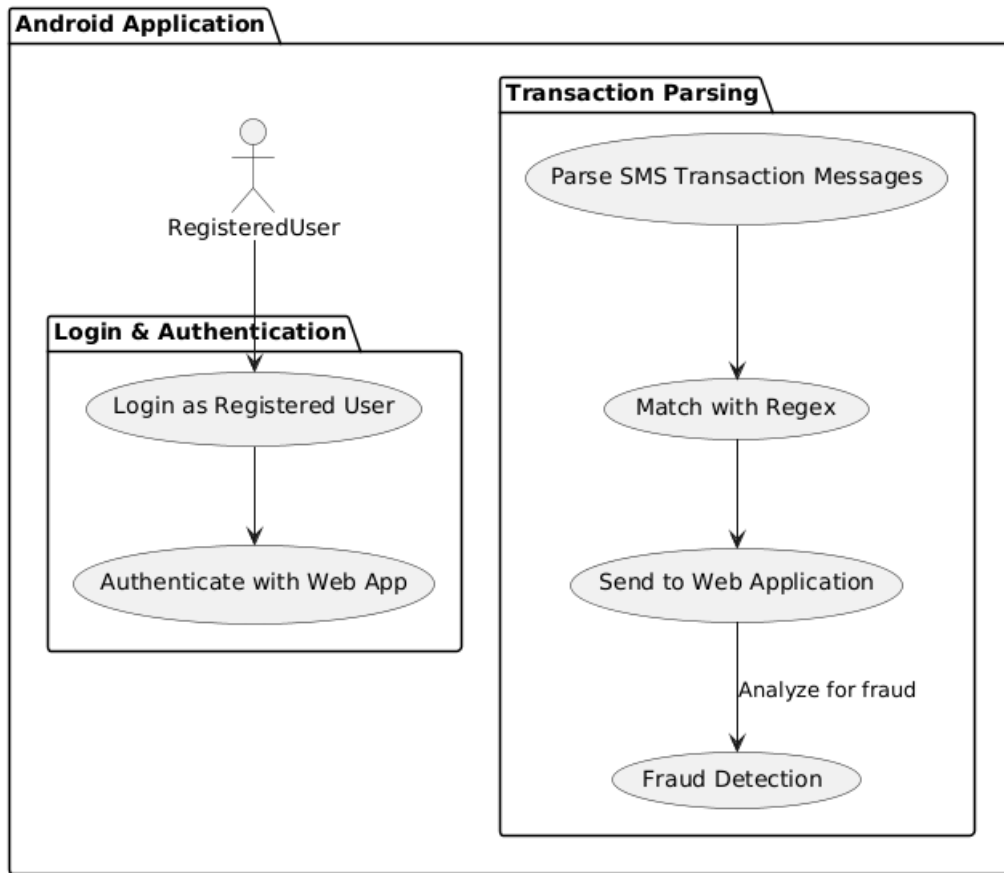


Fig. 3.4. Overview of the Android Application - ZeroSMS

This real-time analysis proves especially beneficial for auditors since they can see red flags or anomalies that need to be investigated. The fraud detection model supports the Zero Trust framework by offering ongoing monitoring and detection features so that even authorized users cannot behave fraudulently. The architecture of the application is depicted in Figure 3.2, where user and admin interactions flow, SMS parsing via ZeroSMS, and fraud detection are displayed. Figure 3.3 represents the dashboard of user behavior analysis, depicting how logs and user activity are monitored and tracked in real-time.

3.3 ACCESS TOKENS FUNCTIONALITY AND IMPLEMENTATION

In Zero Trust security architecture, the use of Access Tokens in the form of JSON Web Tokens (JWTs) is critical for the sake of strong authentication and authorization. JWTs are lightweight, URL-safe tokens that allow for secure exchange of information between entities. Made up of three parts—the header, which defines the signature algorithm; the payload, which contains user claims; and the signature, employed to verify the token—JWTs provide a stateless solution that advantages both scalability and security in distributed systems. The same is illustrated in Figure 3.6.

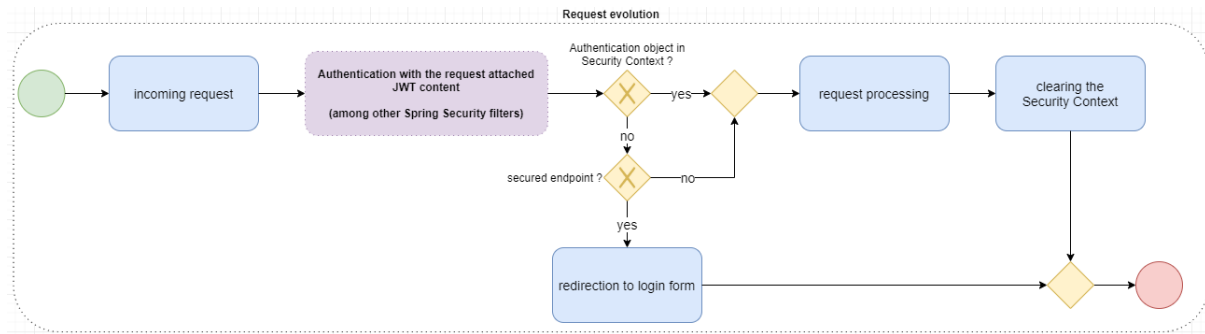


Fig. 3.5. JWT Life Cycle

This design supports smooth authentication processes, where the server can check the token without holding session details. As evident from Fig. 1.1, Python APIs for token generation and managing and strictly validating JWT are containerized and constructed to execute as an independent pod within the Kubernetes cluster, thus providing constant authentication of the client requests. When successfully authenticated, the system creates a JWT that has important user-specific claims like permissions and roles and sends them to the user. For all future requests, the user adds this token to the authorization header, and the application checks the token's integrity and validity before accessing protected resources. This deployment is not only founded on the least privilege principle since it restricts access based on a user's present permissions but also provides added security through the utilization of short-lived access tokens and refresh tokens to reduce the threat of token theft or abuse. Fig. 3.5 illustrates the life cycle of JWTs—from issuing, expiry, and renewal mechanisms.

In order to enhance the security of JWT usage, organizations implement and add on top additional security controls such as MFA, real-time monitoring, and assessing user behavior, geolocation, and device health together with JWT verification. There is a strong posture that is in line with the fundamental principles of Zero Trust. Additionally, efficient revocation of JWTs is essential to ensure that tokens that have been compromised are canceled or revoked in a timely manner. Their application, either by token blacklist or refresh properties of the ephemeral tokens, is of utmost importance in the delivery of secure environments within Zero Trust implementations.

Overall, the strategic integration of JWTs into Zero Trust frameworks not only simplifies access management but also offers an ongoing verification process that maintains the security integrity of the application.

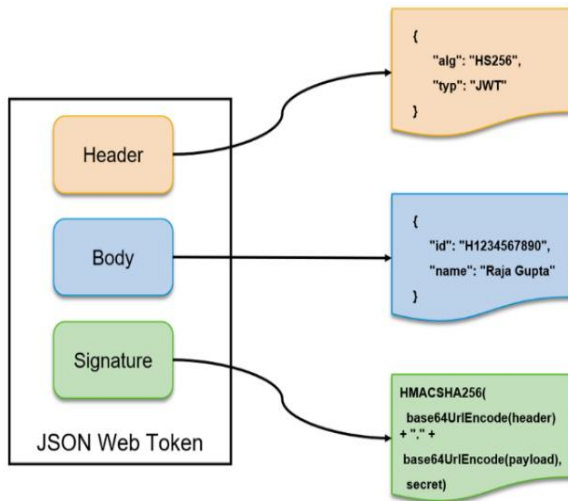


Fig. 3.6. JWT Architecture

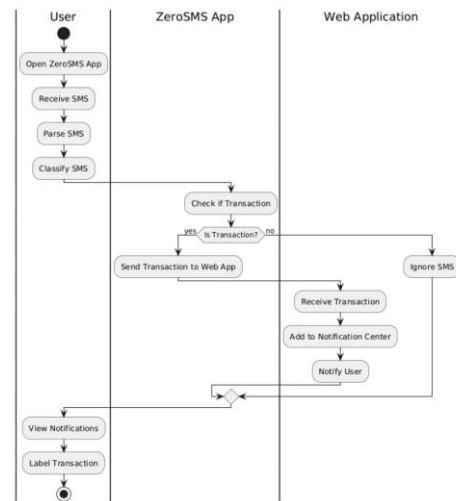


Fig. 3.7. Interaction between the Android App and Web App

3.4 SMS PARSING AND FRAUD DETECTION IN TRANSACTIONS

The growing use of electronic financial transactions has increased the demand for end-to-end expense management solutions that allow users to capture all transactions correctly. The ZeroSMS application meets this need by parsing SMS messages for financial transactions and flagging them as transaction-related or non-transaction-related. Not only does this make it easy for users to track their spending, but it also reduces the chances of missing small transactions like small spending and pocket money. Coupling the ZeroSMS application with a web application allows users to maintain their finances in one place while also ensuring that any missed transaction will be highlighted for investigation. In parsing the SMS messages, the application determines which messages are transaction-related and forwards them to the web application to display in a notification center. Customers can then go back to the notifications and mark the transactions as expenses, savings, or investments, and enter additional information as needed. This process optimizes the accuracy of expense tracking and ensures that all financial activity of interest is captured. As shown in Fig. 3.7, the process illustrates how the SMS parsing workflow increases the user's expense management, confirming the intuitive convergence of web and mobile 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. These

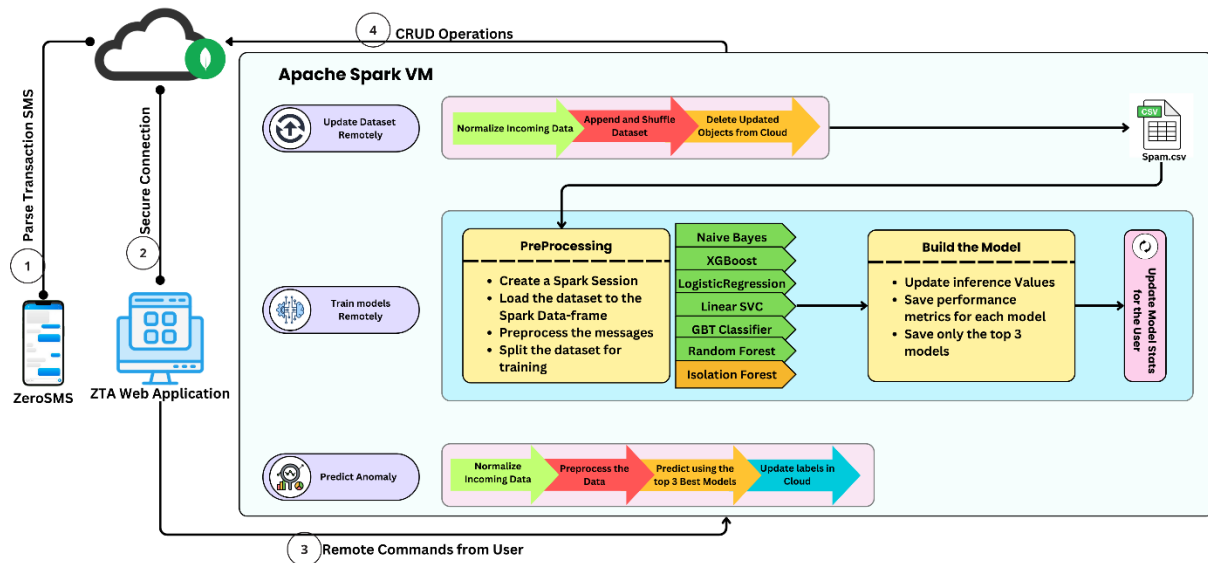


Fig. 3.8. Secure pipeline for real-time SMS parsing, model training, and anomaly prediction.

parsed messages are sent securely to the ZTA Web Application. In the application, these messages are normalized and stored in a MongoDB cloud database. In this Ubuntu VM, Apache Spark processes these messages, updating the dataset, shuffling data, and removing outdated records. Once the dataset is prepared, distributed computing in Spark is utilized to train several machine learning algorithms like Naive Bayes, Random Forest, Logistic Regression, Gradient Boosted Trees (GBT), and Support Vector Classifiers (SVC). For finding anomalies, Isolation Forest, which is tailored for the specific purpose of finding outliers, is incorporated by downloading its JAR file from Maven and then integrating it with Spark's environment. This allows the Isolation Forest to function on top of the MLlib algorithm, improving pipeline anomaly detection.

As noted in Fig. 3.8, The training process keeps only the top three models, based on measures like accuracy and precision. When new data is received, it is preprocessed and input into these top models for predictions of potential anomalies, such as fraudulent transactions. Results are updated back into the MongoDB cloud database to provide users with real-time access to the predictions. ZTA Web Application supports the flexibility and control for executing remote commands such as training models, updating databases, or prediction on the entire pipeline. It measures its own performance by ongoing monitoring of its models in the iterative learning phase. Accuracy, precision, and recall of inference are tracked and reported on the ZTA Web Application in a way that users receive actionable feedback on the performance of their machine learning models. The feedback mechanism ensures that only the

best-performing models make predictions, and the system becomes more reliable and accurate in the long run.

3.4.2 Enhancing Model Training and Fraud Detection

Other improvements have been incorporated in attempts to boost the SMS parsing and fraud detection system through improved model accuracy, scalability, and real-time inference. Among the notable improvements is the application of regular expressions (regex) to classify SMS messages as transaction-related or non-transaction-related to facilitate accurate filtering prior to the data being processed by the machine learning pipeline. Moreover, dynamic dataset refreshes have been incorporated to permit the system to refresh records on a routine basis, shuffle data, and eliminate stale entries. This ensures that the training dataset is free of bias and current to allow the models to keep up with changing transaction patterns. Machine learning training is performed using Apache Spark's MLlib, a scalable distributed computing platform that processes large-scale data with high efficiency. To maximize computational resources, remote model training has been given priority, offloading compute-intensive tasks to a specialized Spark-powered virtual machine, thus maximizing scalability and responsiveness.

The model evaluation strategy has been improved by retaining only the top three models on the basis of critical metrics such as accuracy, precision, recall, and F1-score, thus optimizing performance for fraud detection. Complete control of the machine learning pipeline management by the user is now provided by the ZeroSMS Web Application, with remote execution of activities such as training new models, dataset management, and calling fraud detection on incoming SMS transactions. Furthermore, inference accuracy, precision, and recall are logged and displayed on the web application, giving the user useful information regarding the model's performance and fraud detection efficiency. For better detection of anomalies, a hybrid modeling approach has been used through the integration of traditional classification models and outlier detection models. Specifically, Isolation Forest has been externally integrated using a Maven JAR file, making Spark's MLlib more precise in fraudulent transaction detection. All these changes combined improve the reliability, effectiveness, and responsiveness of the SMS parsing and fraud detection system so that users get real-time information with fewer false positives and false negatives.

3.4.3 Enhancing App Security in React Native

To improve the security of the React Native application, several libraries and techniques have been used. JailMonkey and RootBeer libraries were used to identify whether the application is run on a jailbroken or rooted device, which might be a security threat. Code obfuscation techniques were also used through obfuscator-io-metro-plugin and R8/ProGuard to make reverse engineering more challenging. The obfuscation process translates the application code to secure it against unauthorized access and tampering, improving overall security.

3.5 POLICY ENFORCEMENT POINT FOR ROLE BASED ACCESS CONTROL

A Policy Enforcement Point (PEP) is one of the most critical building blocks of web application security enforcement, particularly in a Zero Trust Architecture. The PEP is a guardian that examines all the incoming requests prior to being forwarded to the core services of the application, and only the authenticated services or users are permitted through. Once deployed in microservice based web applications, the PEP checks every request against preconfigured security policies. Policies determine who has access to what service and resources, when, and under what circumstances. The PEP usually works with the Policy Decision Point, which makes policy decisions like RBAC or ABAC. Based on a decision made by the PDP, the PEP permits or denies access.

In reality, the PEP is instantiated at strategic locations within the microservice topology, such as API gateways or middleware layers. All requests entering the application container service pass through the PEP, where they're authenticated and authorization policies enforced. If the user or service is policy-breaking or unauthorized, the PEP denies access. For example, in our app a, the PEP restricts the auditor from seeing anything other than the transactions and data of their client. In addition, the routes behind the services are restricted and are limited to the entities (user, admin, auditor) to see. It can also enforce least privilege access, where a user or service can do the minimum necessary for their role. In addition, PEP can be used as sidecar containers within microservices to ensure access controls are adhered to at all service endpoints. This is particularly necessary when services call other services internally within distributed systems. In general, PEP protects by ensuring uniform policy enforcement, offering fine-grained access control, and reducing potential attack surfaces.

3.6 SECURE COMMUNICATION WITHIN THE KUBERNETES CLUSTERS USING mTLS

Mutual TLS (mTLS) is among the key security elements of Zero Trust Architecture (ZTA) deployment of microservices. It offers solid authentication and safe communication among microservices by compelling the client and server to verify themselves with good certificates before communicating with each other. This two-way authentication is the focal point of the "never trust, always verify" philosophy of ZTA.

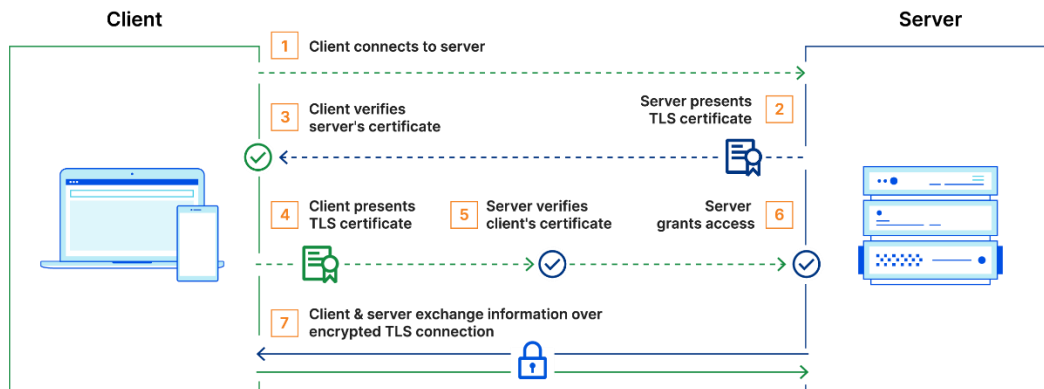


Fig. 3.9. Mutual Transport Layer Security (mTLS) Architecture

In mTLS, there is a single unique cryptographic certificate from a trusted Certificate Authority for every pod in the cluster. When any two services communicate with each other, it takes handshake time. Client and server both show their respective certificates. Then, one cross-verifies the others with the public key of CA to establish genuineness. After the authenticity of both certificates has been established, an encrypted communication channel is created using TLS, which stands for Transport Layer Security. This means that data being exchanged between services is authenticated as well as secured against eavesdropping and tampering. The mTLS architecture is shown in Figure 3.9. mTLS is being integrated into our application service meshes through a tool called Istio that automate the management of certificates and enforce mTLS at the network layer. In such environments, mTLS is enforced transparently on microservices, with no need for each service to have its own TLS logic. The service mesh manages certificate issuance, renewal, and rotation and provides seamless enforcement of mTLS. In contrast to conventional systems, which consider that once within the network, there will be trust, mTLS demands that all communication between services is identity-checked. This prevents unauthorized services from accessing resources since they can only communicate if they possess valid certificates. Coupled with Policy Enforcement Points (PEPs), mTLS enhances the general ZTA. The PEP checks that requests are coming from authorized services with valid certificates before they access resources.

CHAPTER 4

SYSTEM DESIGN

Here we move from theory to actual practice, discussing system requirements, tools, and technical infrastructure that we used to build and deploy the web and mobile application. We also briefly discuss local and cloud environment deployment with Docker and Kubernetes. This chapter is theory-dominant, with a focus on demonstrating actual application on the project side by discussing development environments and deployment approaches.

4.1 SYSTEM REQUIREMENTS FOR DEVELOPMENT

This section describes the software and hardware requirements 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 specification:

- 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)

Other frontend development tools include Webpack, which is used for bundling, and Babel for transpiling JavaScript.

- 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 describes the deployment plan of the application within local and cloud environments using Docker and Kubernetes. The software and hardware specifications of both environments, and cloud services utilized for deployment are highlighted below:

4.2.1 Local Environment

For local development and testing, the system can be containerized using Docker and deployed using Minikube for Kubernetes orchestration. The following are 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.

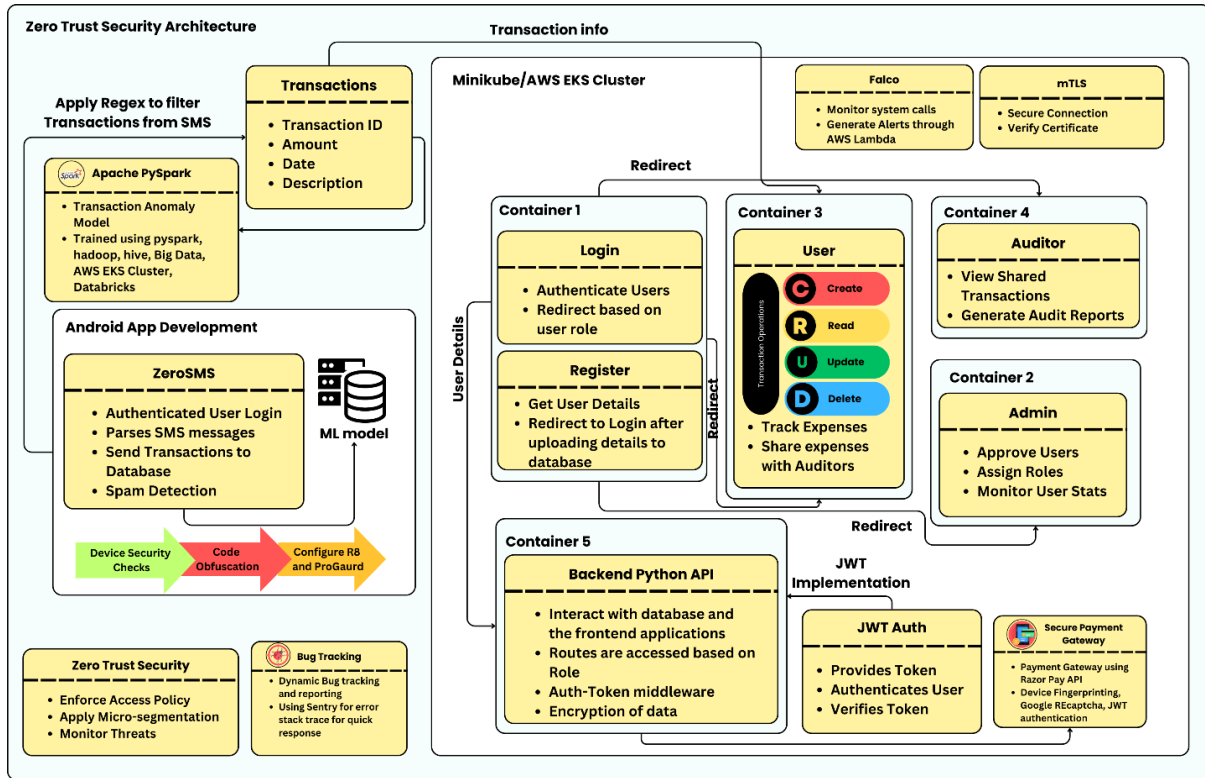


Fig. 4.1 Architecture design of the developed system

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

Require: Microservice set $\mathcal{M} = \{M_1, M_2, \dots, M_n\}$

Role-based access policies $R = \{R_{ad_{min}}, R_{aud_{itor}}, R_{user}\}$

JWT tokens T_{jwt} , mTLS certificates C_{tls}

User request U_{req} , Incoming transaction T_{sms}

1. procedure ENFORCE_ZERO_TRUST(U_{req})
2. $T_id \leftarrow \text{VERIFY_JWT}(U_{req}, T_{jwt})$
3. if $T_id = \emptyset$ then
4. Reject request with status 401 Unauthorized
5. $R_{eff_{active}} \leftarrow \text{EXTRACT_ROLE}(T_id)$
6. if POLICY_VIOLATION($R_{eff_{active}}, U_{req}$) then
7. Reject request with status 403 Forbidden
8. Forward request to M_{target} with secure mTLS channel
9. Procedure MUTUAL_TLS_COMMUNICATION(M_i, M_j)
10. $C_i, C_j \leftarrow \text{RETRIEVE_CERTIFICATES}(M_i, M_j)$
11. if $\neg \text{VERIFY_MUTUAL_TRUST}(C_i, C_j)$ then
12. Terminate connection with TLS Alert
13. else
14. Establish encrypted channel C_{enc}
15. procedure PROCESS_INCOMING_TRANSACTION(T_{sms})
16. $\mathcal{F}_{raw} \leftarrow \text{PARSE_TRANSACTION}(T_{sms})$
17. $\mathcal{F}_{clean} \leftarrow \text{SPAM_FILTER}(\mathcal{F}_{raw})$
18. if $\mathcal{F}_{clean} = \emptyset$ then
19. return Spam Detected — Transaction Ignored
20. STORE \mathcal{F}_{clean} in expense ledger \mathcal{L}
21. TRIGGER_AUDIT_NOTIFICATION if ANOMALY_SCORE(\mathcal{F}_{clean}) $> \epsilon$
22. procedure MODEL_UPDATE_PIPELINE
23. $D_{train} \leftarrow \text{REMOTE_FETCH_TRAININGDATA}()$
24. $M_{new} \leftarrow \text{TRAIN_MODEL}(D_{train})$
25. DEPLOYMODEL(M_{new}) via CI/CD Helm pipeline

4.6 PROTOCOLS AND STANDARDS

The system utilizes several security standards and protocols to safeguard the confidentiality, integrity, and authenticity of information being transferred between the application. Through the use of proven and tested security mechanisms, the system can safeguard against some of the common cybersecurity attacks like data breaches, Man-in-the-middle (MITM) attacks, and unauthorized access. These are necessary in a Zero Trust environment where no entity is by default trusted and all requests for access are needed to be authenticated.

4.6.1 JWT (JSON Web Tokens)

Employed for token-based session management and authentication, JWT is a lightweight, URL-safe way of expressing claims between two parties and is commonly employed for web application and API security. `authenticate(user)` and `verifyToken(token)` methods provide secure, token-based access to the system, keeping unauthorized users from accessing sensitive transaction information.

4.6.2 mTLS (Mutual Transport Layer Security)

In a microservices environment, mTLS is utilized to provide safe interservice communication. Because microservices exchange information via APIs and might possibly disclose sensitive information, mTLS introduces a necessary encryption and mutual authentication layer, with the assurance that only authenticated services talk to each other. In order to adopt mTLS, a specific certificate needs to be held by every microservice, issued by a trusted certificate authority (CA), which is validated upon service communication. The step 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 strengthens security by incorporating mTLS, providing encrypted and authenticated communication between services. Client and server authenticate one another, safeguarding against attacks such as impersonation. Istiod, the control plane of Istio, automates certificate management, providing each service with a unique identity, guaranteeing all traffic is encrypted and authenticated. Istiod, Istio's control plane, manages certificate lifecycles

automatically, giving each service a distinct identity, encrypting, and authenticating all traffic. Istio offers permissive and strict mTLS modes for gradual adoption. This aligns with Zero Trust principles, where no service is trusted by default. With sidecar proxies like Envoy, Istio enables secure communication without application-level change. It also manages certificate revocation and rotation automatically and provides efficient encryption, authentication, and access controls for distributed microservices.

4.6.3 Policy Enforcement Point (PEP)

Policy Enforcement Point (PEP) is a security aspect that makes real-time access decisions. It determines whether a given entity (service, device, or user) can access a protected resource or not based on a pre-established set of policies. PEPs discard every request and filter it against a security policy base to permit only authorized parties to access protected resources. PEPs play an important role in microservice architecture to manage service-to-service communications and incoming requests from the external world. PEPs impose access policies on different layers such as API gateways, service meshes, and individual services to limit processing to only authenticated and authorized requests.

4.6.4 Falco: Real-Time Security Monitoring

Falco is a cloud-native, open-source runtime security tool designed for systems based on microservices. Falco scans system calls in real-time to scan for malicious behavior or attacks. By intercepting system calls via eBPF, Falco observes activity such as file accesses and process starts against a known set of security rules. If a rule is broken, it produces alerts and can initiate actions, such as blocking processes or alerting security teams. With microservices, Falco achieves security by actively looking for undesirable access, unidentified process starts, and malicious network traffic, therefore offering the real-time picture of runtime behavior and security posture.

4.6.5 Sentry: Real-Time Error Tracking and Debugging

Sentry is an open-source application monitoring and error tracker that assists developers in locating, diagnosing, and fixing defects in real-time in development and production environments. Sentry tracks crashes, performance bottlenecks, and exceptions, along with detailed stack traces, contextual metadata, and impacted user insights. With seamless integration with a variety of programming languages and frameworks, Sentry facilitates proactive debugging by informing developers about errors, their root causes, and potential

solutions. This results in quicker resolution of critical issues, enhanced system stability, and automation of the overall development process by saving time spent on debugging and troubleshooting.

4.6.6 Code Obfuscation

Code obfuscation is one of the best practices that protects intellectual properties of an application, mainly of web and mobile apps that reverse-engineer an app's code. Obfuscation converts the source code to a very complex one that is impossible to read. It even makes it more challenging for an attacker to reverse-engineer or use its vulnerabilities.

- Proguard is a Java and Android app open-source utility that is very widely used for performing a variety of optimizations such as shrinking, optimizing, and obfuscating code. Proguard makes it difficult for the bad guys to get valuable information about structure by giving classes, fields, and methods very long, obfuscated names.
- R8 is the newer version of Proguard, designed specifically for use in Android development. R8 is better integrated with Android build systems and provides improved obfuscation and performance optimization. Apart from shrinking and renaming, R8 removes unused code, hence becoming leaner with more advanced optimization features compared to Proguard. Compilation speed and more aggressive code shrinking are some of the biggest benefits of R8, and therefore it is the most used among Android developers today.

4.6.6 Root, Debug, and Developer Options Detection

For additional security, the system leverages rooted device detection mechanisms, active debugging sessions, and enabled developer options. These are critical in defense against attacks based on modifying the operating system or analyzing the application in a controlled debugging environment. Rooted devices have the advantage of bypassing certain security controls, through which attackers can easily obtain access to sensitive application data. JailMonkey is an open-source library used mostly in React Native applications to ascertain if a device is rooted or modified in any other manner. Through the aid of these standards and protocols, the system provides secure, authenticated, and reliable financial transaction management, with strong security guaranteed through the Zero Trust model.

CHAPTER 5

IMPLEMENTATION AND RESULT ANALYSIS

This chapter addresses the deployment and real-time execution of the Android and web applications with care regarding Zero Trust security practices. The chapter details the deployment of the web application to a Minikube cluster followed by AWS EKS and secure deployment of the Android application. Major security controls such as data integrity processes and user access controls are detailed to emphasize the effectiveness of the proposed solutions in ensuring a secure system.

5.1 REAL-TIME IMPLEMENTATION OF THE PROTOTYPE

5.1.1 Preparation of Environment in AWS

To get the environment ready for implementing the Zero Trust Architecture, Terraform was utilized as the Infrastructure as Code (IaC) solution for uniform and automated AWS resource provisioning. A special Virtual Private Cloud (VPC) was setup to be used for hosting the Amazon EKS (Elastic Kubernetes Service) cluster that was highly available and scalable. In the same way, AWS Lambda is setup with AWS CloudWatch to trigger Falco alerts and generate the logs. A Network Load Balancer is used with Nginx-Ingress Controller to direct traffic from the internet to the cluster. Identity, and Access Management (IAM) roles were created was all the above configurations to allow secure communication and management of resources.

Context: arn:aws:eks:us-east-1:730335337356:cluster/ZTA-Cluster

Cluster: arn:aws:eks:us-east-1:730335337356:cluster/ZTA-Cluster

User: arn:aws:eks:us-east-1:730335337356:cluster/ZTA-Cluster

K9s Rev: v0.31.1 ⚡v0.40.10

K8s Rev: v1.32.2-eks-bc803b4

CPU: n/a

MEM: n/a

<0> all

<1> default

<ctrl-d> Delete

<d> Describe

<e> Edit

<?> Help

<ctrl-k> Kill

<l> Logs

<p> Port-Forward

<shift-f> Sanitize

<s> Shell

<n> Show Node

Logs

Logs Previous

Port-Forward

Sanitize

Shell

Show Node

K8S

Pods(all)[23]

NAMESPACE	NAME	PF	READY	STATUS	RESTARTS	IP	NODE	AGE
cert-manager	cert-manager-7b5cdf866f-zxk9k	●	1/1	Running	0	10.0.2.168	ip-10-0-2-110.ec2.internal	64m
cert-manager	cert-manager-cainjector-7c9788477c-tjzcv	●	1/1	Running	0	10.0.1.156	ip-10-0-1-76.ec2.internal	64m
cert-manager	cert-manager-webhook-764949f558-4n5tg	●	1/1	Running	0	10.0.1.215	ip-10-0-1-76.ec2.internal	64m
default	ingress-nginx-controller-b49d9c7b9-sfrl6	●	1/1	Running	0	10.0.1.83	ip-10-0-1-76.ec2.internal	93m
falco	falco-falcosidekick-6447d66899-dq7nm	●	1/1	Running	0	10.0.1.218	ip-10-0-1-76.ec2.internal	10m
falco	falco-falcosidekick-6447d66899-sv9qq	●	1/1	Running	0	10.0.2.230	ip-10-0-2-110.ec2.internal	10m
falco	falco-g8r5b	●	2/2	Running	0	10.0.1.36	ip-10-0-1-76.ec2.internal	10m
falco	falco-k8s-metacollector-67487764b-zn6kc	●	1/1	Running	0	10.0.1.117	ip-10-0-1-76.ec2.internal	10m
falco	falco-v7wj4	●	2/2	Running	0	10.0.2.141	ip-10-0-2-110.ec2.internal	10m
istio-system	istio-ingressgateway-67fb649dc8-srswd	●	1/1	Running	0	10.0.1.85	ip-10-0-1-76.ec2.internal	61m
istio-system	istiod-86d96548d6-6cjinl	●	1/1	Running	0	10.0.2.11	ip-10-0-2-110.ec2.internal	61m
kube-system	aws-node-swcx	●	2/2	Running	0	10.0.2.110	ip-10-0-2-110.ec2.internal	99m
kube-system	aws-node-vjzb8	●	2/2	Running	0	10.0.1.76	ip-10-0-1-76.ec2.internal	99m
kube-system	coredns-6b9575c64c-klwdq	●	1/1	Running	0	10.0.2.246	ip-10-0-2-110.ec2.internal	101m
kube-system	coredns-6b9575c64c-nh5fj	●	1/1	Running	0	10.0.2.147	ip-10-0-2-110.ec2.internal	101m
kube-system	kube-proxy-hfhpj	●	1/1	Running	0	10.0.2.110	ip-10-0-2-110.ec2.internal	99m
kube-system	kube-proxy-wqkcc	●	1/1	Running	0	10.0.1.76	ip-10-0-1-76.ec2.internal	99m
zta	admin-77bbd9bcb7-2r8cp	●	2/2	Running	0	10.0.1.45	ip-10-0-1-76.ec2.internal	25m
zta	auditor-74cfc78cc5-n6z5s	●	2/2	Running	0	10.0.2.121	ip-10-0-2-110.ec2.internal	25m
zta	backend-6448c765c5-bkckx	●	2/2	Running	0	10.0.1.103	ip-10-0-1-76.ec2.internal	25m
zta	pep-5d44ff996-c4x8h	●	2/2	Running	0	10.0.1.171	ip-10-0-1-76.ec2.internal	25m
zta	public-8676649576-z69w7	●	2/2	Running	0	10.0.2.205	ip-10-0-2-110.ec2.internal	25m
zta	user-84d9bb8d78-mf4tg	●	2/2	Running	0	10.0.1.54	ip-10-0-1-76.ec2.internal	14m


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 rolled out using Kubernetes manifests and a Helm chart. Docker images were created for every part of the application, specifying the required environment and dependencies. Helm chart was employed to coordinate the deployment of the web application into the EKS cluster. Every part of the application was deployed under their own prefix which is automatically managed by ingress controller running alongside the application. HTTPS protocol was mandated to provide secure communication between application and users. Figure 5.1. illustrates the Web Application and Falco tool pods running in EKS cluster.

5.1.3 Deployment of Falco to EKS Cluster

Falco, a cloud-native runtime security platform, was used to observe the activity of the containers and flag any suspicious activity. It was installed as a DaemonSet within Kubernetes so that every node in the EKS cluster had a Falco instance running for thorough monitoring. Specific policy rules were created to limit the anticipated behavior of the containers. Any violation of these policies. As can be seen from the Figure 5.2. the logs will be produced as standard output and also will be outputted to AWS cloud watch whenever any rules specified in Falco gets breached which enhances the Zero trust of the resources deployed within the cluster.



```
2025-03-25T17:00:13.048Z    {"hostname": "falco-4c38b", "output": "17:00:12.354062407: Alert Illegal read executed with file in shell (user=root container_id=41cb80504075 container_name=admin file=<NA>) c..."}
{"hostname": "falco-4c38b", "output": "17:00:12.354062407: Alert Illegal read executed with file in shell (user=root container_id=41cb80504075 container_name=admin file=<NA>) c..."}
{"hostname": "falco-4c38b", "output": "17:00:12.354062407: Alert Illegal read executed with file in shell (user=root container_id=41cb80504075 container_name=admin file=<NA>) c..."}
{"rule": "Illegal read of application file from Container", "source": "syscall", "tags": ["app_read"], "time": "2025-03-25T17:00:12.354062407Z", "output_fields": {"container_id": "41cb80504075", "container_image_repository": "docker.io/disciklean/zta-app-admin", "container_image_tag": "latest", "container_name": "admin", "evt.time": "1742922012354062407", "fd.name": "None", "k8s.ns.name": "zta", "k8s.pod.name": "admin-77bbd9bcb7-2r8cp", "user.name": "root"}}
```

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 application is built as a strong security tool with the primary aim of safeguarding sensitive data via secure messaging. Its development entails the adoption of strong mechanisms to identify rooting and developer options, especially utilizing JailMonkey, a library that assists in determining if the device is rooted or operating in an insecure environment. Through the utilization of JailMonkey's functionality, the application can guarantee that it runs only on secure devices, giving users assurance of 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

In order to make the ZeroSMS app production-ready, code optimization and obfuscation must be enabled. This is done by setting up the R8 and ProGuard tools in the android build system. As illustrated in figure 5.3, By making minifyEnabled true in the build.gradle file, R8 is told to eliminate unused code, resources, and obfuscate, thus improving the performance and security of the app. Furthermore, the app is set up to employ a secure keystore for signing the APK, so only official releases are propagated. This means creating a private signing key with the keytool command and keeping the keystore file in the correct directory. Lastly, the command `./gradlew assembleRelease` is run in order to compile the APK, utilizing all the given configurations, thereby producing a safe and optimized release build of the ZeroSMS application.

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 is an essential step in the preparation of the SMS messages for effective spam detection. The dataset is first imported into a DataFrame using the Pandas library. The columns that are duplicate are dropped in order to minimize the dataset to include only necessary columns: 'category' and 'msg.' The 'category' column is then transformed into binary form by adding a new column 'spam,' where the messages that have been labeled as "spam" are assigned a value of 1, while the "ham" messages are assigned a value of 0.

In order to facilitate the conversion of text messages into machine-readable form, the CountVectorizer from the Scikit-learn library is utilized. This transforms the text data into a token frequency matrix. The features (X) and labels (y) are then extracted from the DataFrame, with X being the transformed message data and y being the corresponding binary labels. The train_test_split function is then utilized to divide the dataset into training and test sets, with a ratio of 70% for training and 30% for testing. This structured approach allows the model to be trained on a representative sample of the data, allowing for effective testing and performance measurement.

5.4 MODELS USED FOR TRAINING

To boost the efficiency of the SMS fraud detection system, several machine learning models were employed, namely Logistic Regression, Random Forest, Gradient Boosted Trees

(GBTCClassifier), Naive Bayes, Linear Support Vector Classifier (LinearSVC), and XGBoost. The models were selected on the basis of their applicability in text classification, their capacity to deal with high-dimensional data, and their computational efficiency in identifying fraudulent transaction messages. Through the implementation of a combination of conventional and ensemble machine learning models, the system achieves a trade-off between computational efficiency and high prediction accuracy. The nature and function of each model during training are discussed below.

5.4.1 Naive Bayes

Multinomial Naive Bayes classifier, which utilizes Bayes' theorem, is found to be most appropriate for text classification tasks, i.e., identification of SMS fraud. It classifies features under the condition of conditional independence when the class is known, thus making it computationally efficient. Naive Bayes is best appropriate for high-dimensional data as it employs word frequencies in messages to infer class probabilities.

5.4.2 Linear Support Vector Classifier (LinearSVC)

Linear Support Vector Classifier (LinearSVC) is a strong supervised learning algorithm that finds the optimal hyperplane that optimally separates different classes with maximum distinction. It is computationally more efficient in handling large-scale data and high-dimensional feature spaces than the basic SVM. It was particularly useful in handling SMS-based fraud detection in this project as it could handle linearly separable data without sacrificing high generalization performance.

5.4.3 Logistic Regression

Logistic Regression is a strong yet straightforward classification algorithm that predicts the probability of a particular input belonging to a particular class. It is usually used in text classification problems due to its simplicity and interpretability. Logistic regression was configured with a high iteration limit in this project to ensure proper convergence. Its strength lies in its ability to model the relationship between text-based features and fraudulent transactions and hence it is a good fraud detection baseline.

5.4.4 Random Forest

Random Forest is an ensemble learning method that, during training, generates many decision trees and uses their predictions to improve predictive power. It is especially useful in preventing

overfitting, one of the major problems of a single decision tree. Random Forest's capability of dealing with big data having high feature dimension makes it applicable for SMS fraud detection, where multiple patterns of transactions exist. With voting on the outputs of many trees, it improves the stability of classification and the overall model quality.

5.4.5 Gradient Boosted Trees (GBClassifier)

Gradient Boosted Trees (GBClassifier) is an ensemble learning algorithm consisting of weak learners (decision trees) trained sequentially to learn from the errors of the previous models. The boosting technique enhances the predictive capability of the classifier so that it becomes highly efficient in detecting complex fraud patterns. The performance depends heavily on hyperparameter tuning, and in the project, it was subpar in the sense that its accuracy was worse than that of better-tuned ensemble algorithms such as XGBoost.

5.4.6 XGBoost

XGBoost (Extreme Gradient Boosting) was the best-performing model in this project, which worked best for all the most critical evaluation measures like accuracy, F1-score, and ROC-AUC score. As a more mature variant of gradient boosting. It improves decision trees with regard to parallel processing, regularization, and memory optimization. Its increased ability to learn intricate patterns in transactional SMS data made it the best-performing fraud detection model, significantly outperforming other classifiers.

5.5 RESULT ANALYSIS

This part recognizes the effectiveness of the security controls in code obfuscation, permission handling, and live threat detection utilized on both systems. For the web application, the assessment is from the successful deployment in an EKS cluster, with AWS EC2 instances for the backend and frontend. Performance metrics such as session logs and security alerts triggered by AWS Lambda with Falco reflect the strength and robustness of the system against threats.

5.5.1 Web Application Deployment in EKS Cluster

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

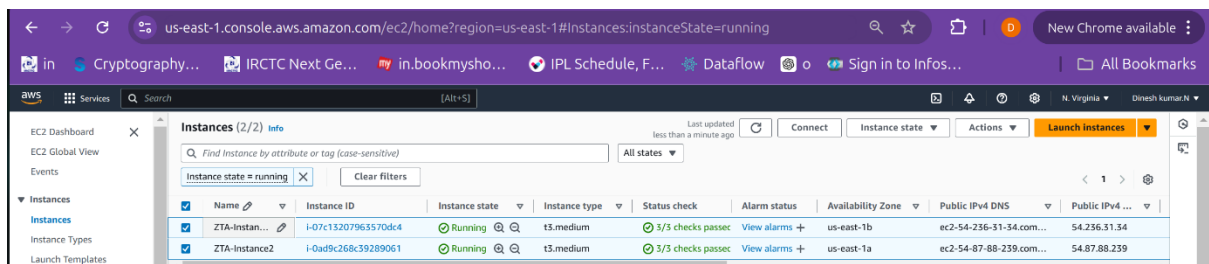


Fig 5.4. Node Groups of EKS Cluster deployed as EC2 instances

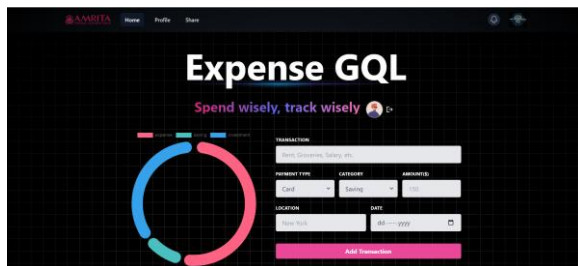


Fig 5.5. User dashboard for CRUD operations

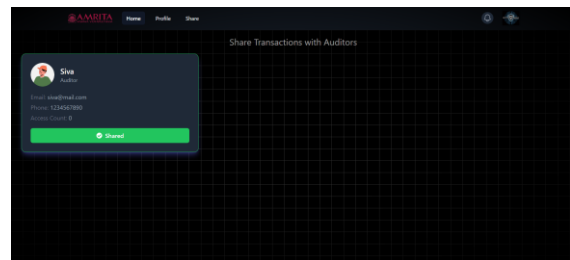


Fig 5.6. Share data with auditors

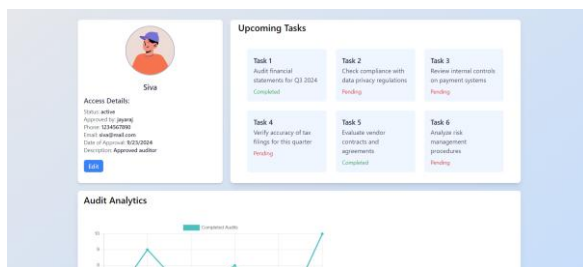


Fig 5.7. Auditor dashboard with tasks

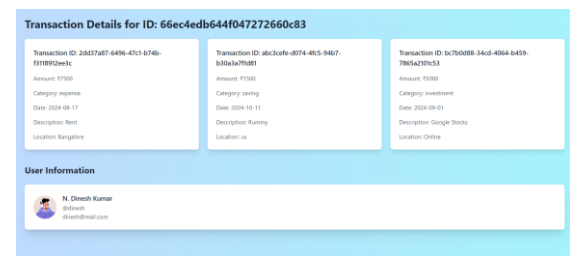


Fig 5.8. Auditor's read-only view of transactions



Fig 5.9. Admin dashboard for assigning roles

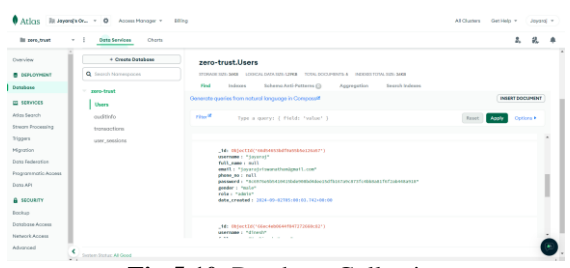


Fig 5.10. Database Collections

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 in the application is enhanced by AWS Lambda and the use of Falco tool, which keeps watch on the cluster for any anomalies and sends alerts for real-time threat detection, as depicted in Figure 5.11. Fig. 5.11 presents the alerts from Lambda as logs trapped in AWS Cloudwatch. These logs hold such valuable data as the breached rule, container name, container ID, pod name, and namespace under which the attack was carried out, as well as the system call used to carry out the attack. All this cumulative data highlights the application of Zero Trust strategy and facilitates the identification of the source cause of malicious activity. In addition, session logs also offer a complete audit trail, monitoring user activity and detecting malicious behavior, as illustrated in Table 5.1. These logs maintain Zero Trust compliance by ensuring all user activity is constantly monitored and logged.

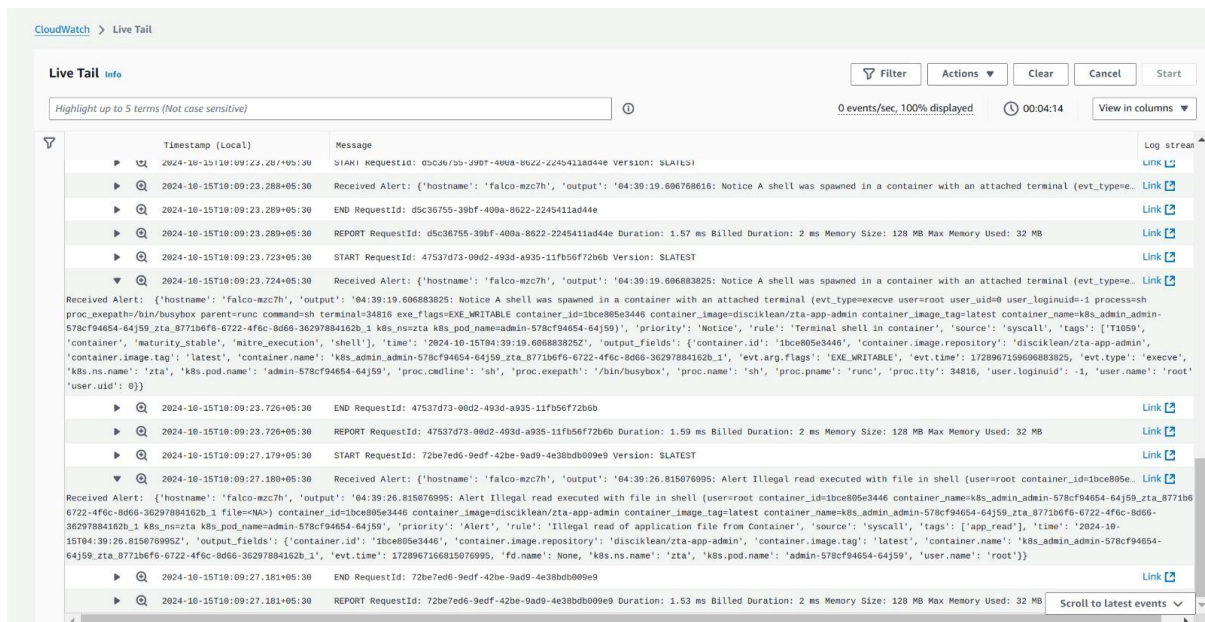


Fig 5.11. AWS Lambda Alerts triggered by Falco tool deployed in the cluster

Table 5.1 Session logs for logged in users

Field	User 1	User 2	User 3
_id	66e9072851dccc0a150dbe0	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

5.5.2 Android Application Production Build Results

The Android application was developed. with extensive security features in mind, as demonstrated in Figure 5.3, which depicts an alert dialog stating that developer options are active. The application also has provisions to avoid screenshots during the process of logging in, further complementing its security features.

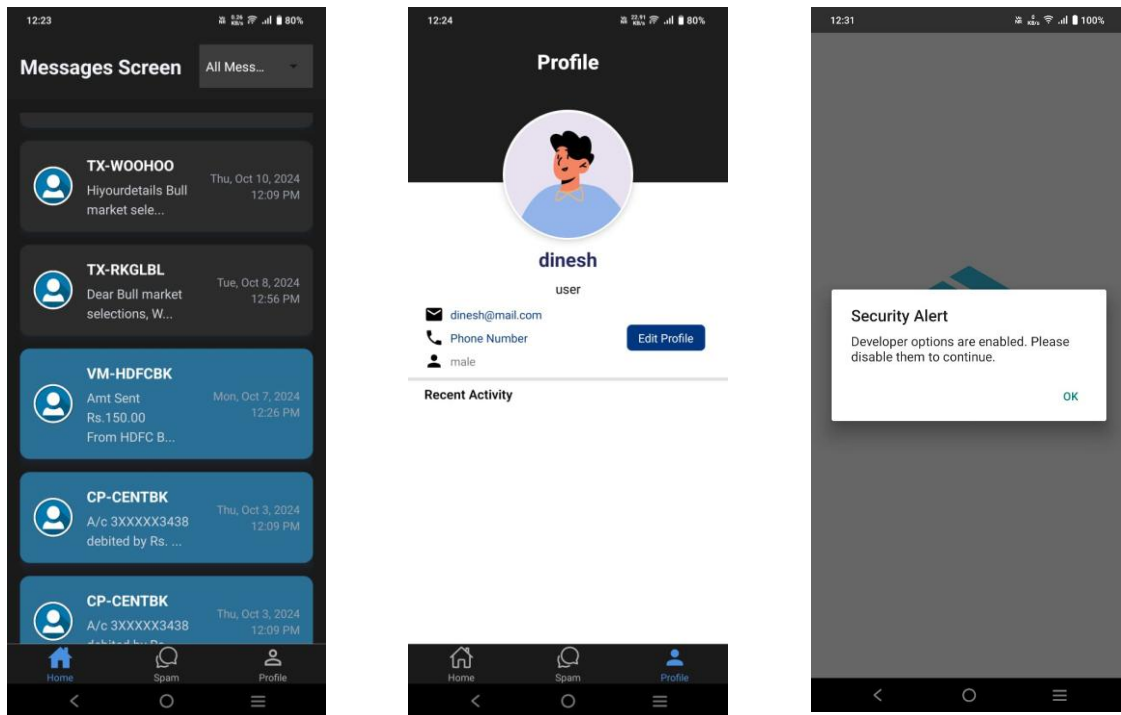


Fig 5.12. ZeroSMS in production release with security features

5.5.3 Comparing and Testing ZeroSMS Apks with Other Normal Apks

As a result of the use of code obfuscation tools, the reverse engineering of the ZeroSMS APK is dramatically different from that of normal APK files, as shown in Figures 5.4 and 5.5. The names of classes in the application are obfuscated, making it harder for an attacker to follow the folders and files. The use of R8 and ProGuard additionally improved the process of obfuscation, as shown in Figure 5.6. By comparison, Figure 5.7 shows a normal APK where the code is still well visible and easy to comprehend.

Figure 5.8 shows a comparison of MobSF scans for the ZeroSMS APK and a normal APK. ZeroSMS is given a security score of 57, grading 'B', whereas the regular APK is given a score of 31, grading 'C'. Note that ZeroSMS SMS read and write permissions are activated, and "http" traffic is allowed in its AndroidManifest.xml file by the script: `android:usesCleartextTraffic="true"`. This setup was required for communicating with the

Mean Squared Error and Mean Absolute Error were minimum at 0.01, reflecting low errors in predictions. With an ROC AUC Score of 0.98 and Specificity of 0.99, XGBoost also showed great discriminative power for separating fraud from non-fraud messages and, therefore, topped the performance across this project. Naive Bayes too came out very efficient, particularly computationally as well as in terms of generalizability. It achieved an F1 Score of 0.96, Recall of 0.95, and Precision of 0.97, with an Accuracy of 0.94. The MSE and MAE of the model were 0.06, and ROC AUC Score was 0.95, which indicates its ability to deal with high-dimensional text data. The Specificity was also high at 0.97, reflecting its ability to identify non-fraudulent messages correctly.

Table 5.2. Comparison of Model Performance Metrics

Model	F1 Score	Recall	Precision	Accuracy	Mean Squared Error	ROC AUC Score	Mean Absolute Error	Specificity
XGBoost	0.99	0.99	0.99	0.99	0.01	0.98	0.01	0.99
Naive Bayes	0.96	0.95	0.97	0.94	0.06	0.95	0.06	0.97
LinearSVC	0.92	0.9	0.94	0.89	0.11	0.92	0.11	0.94
Logistic Regression	0.87	0.85	0.89	0.87	0.13	0.88	0.13	0.89
Random Forest	0.8	0.78	0.82	0.85	0.2	0.82	0.2	0.82
GBTClassifier	0.69	0.68	0.7	0.69	0.31	0.72	0.31	0.7

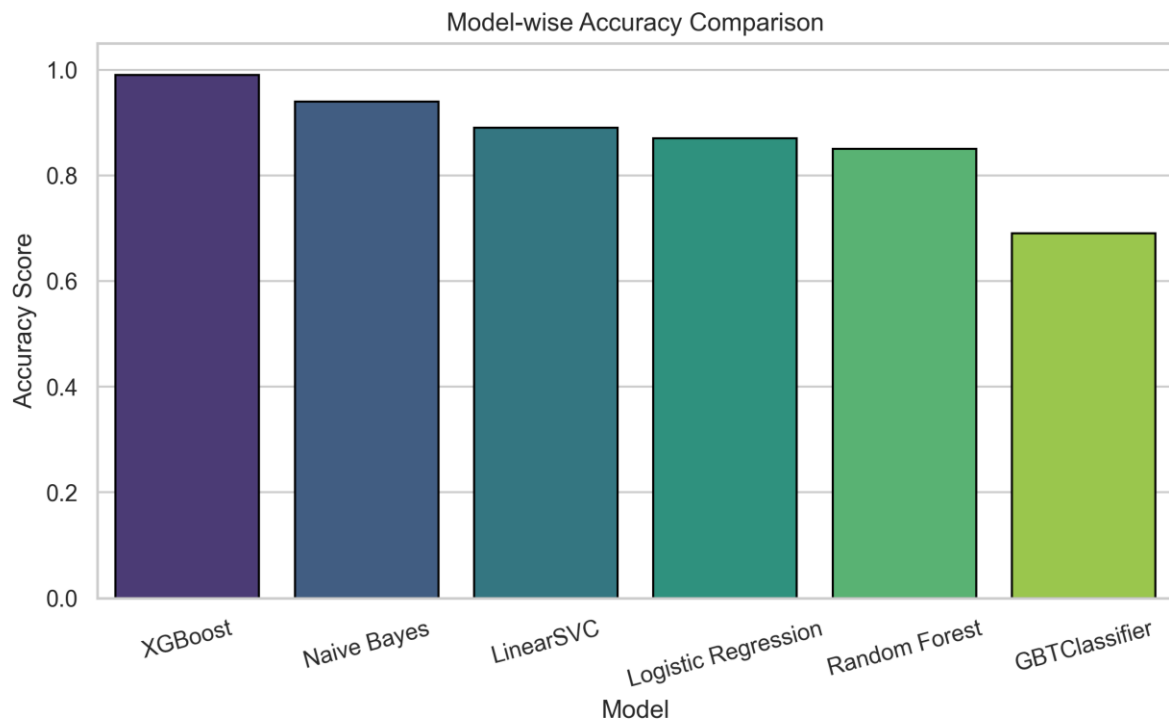


Fig 5.18. Comparison between Spam Detection Models

Linear Support Vector Classifier (LinearSVC) followed, with an F1 Score of 0.92, Recall of 0.90, and Precision of 0.94, indicating a balanced classification model. Accuracy was 0.89, and MSE and MAE were 0.11, while the ROC AUC Score was 0.92. Specificity was 0.94, indicating good identification of true negatives. Logistic Regression indicated moderate performance, with an F1 Score of 0.87, Recall of 0.85, and Precision of 0.89. Accuracy was 0.87, while MSE and MAE were 0.13, and the ROC AUC Score was 0.88. Its Specificity was 0.89, indicating a good baseline model but less efficient than others. The Random Forest classifier indicated rather low performance, with an F1 Score of 0.80, Recall of 0.78, and Precision of 0.82. Accuracy was 0.85, while MSE and MAE were 0.20, and the ROC AUC Score was 0.82. Specificity was 0.82, indicating high trade-off between performance and simplicity in ensemble methods without deep optimization.

Lastly, Gradient Boosted Trees (GBClassifier) performed the worst with an F1 Score of 0.69, Recall of 0.68, and Precision of 0.70. It only scored 69% Accuracy with MSE and MAE scores of 0.31 and a ROC AUC Score of 0.72. Its Specificity was also relatively lower at 0.70, which means that it has not much ability to detect complex fraud patterns in transactional SMS messages. In general, the comparison clearly shows that XGBoost performed much better than all the other models and therefore is the best to use in real-world fraud detection systems.

CHAPTER 6

CONCLUSION AND FUTURE WORK

In an era where cybersecurity threats are becoming more and more sophisticated, the security of containerized environments is of paramount importance. In this project, we investigated a number of methods and best practices to enhance security through the application of the Zero Trust methodology. Through the development of a proof of concept (PoC) with a secure web application, a Python API, and an Android application, we illustrated how the theory of Zero Trust can be applied in an efficient manner. Key technologies like JWT authentication, continuous monitoring, code obfuscation, root detection, mutual TLS (mTLS) in containers, and efficient enforcement of policy for Role-Based Access Control (RBAC) have been utilized in the development process. The result of this project reiterates the imperativeness of having an active security concept in containerized environments. Through vigilant monitoring at every level to identify trust and ongoing monitoring for vulnerabilities, organizations can make their defenses stronger more. There are several areas of future research and security development in containerized environments.

Future research can be conducted for monitoring trends and anomalies in containerized applications based on machine learning techniques and further development of continuous monitoring processes. Security auditing processes can be automated for continuous monitoring of the security stance of containerized applications and vulnerability scanning along with industry standards compliance checks, timely and based on industry standards. Baking security processes into CI/CD pipelines will incorporate security as a consideration and implementation right from the earliest phases of application development. Also, investment in resources and educating developers and operators about Zero Trust principles will imbue a culture of security-first in development teams. Realistic field deployment of the developed applications in multiple organizational environments would also offer a vehicle for collection of data regarding security performance and feedback from users for continuous improvements and adjustments in the envisioned approach. With these fields established, organizations can provide more support for their security measures and maintain their containerized environments safe from continuously changing threats without ever deviating from the Zero Trust model.

REFERENCES

- [1] O. E. Imokhai, T. E. Emmanuel, A. A. Joshua, E. S. Emakhu, and S. Adebisi, "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](https://doi.org/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.