Zero Trust Security for Web Applications in Microservice-Based Environments

Jayaraj Viswanathan Department of Computer Science and Engineering,

Amrita Vishwa Vidyapeetham,

Chennai, India,[*ch.en.u4cys21026@ch.students.amrita.edu*](mailto:ch.en.u4cys21026@ch.students.amrita.edu)

Dinesh Kumar NDepartment of Computer Science and Engineering,

Amrita Vishwa Vidyapeetham,

Chennai, India,

[*ch.en.u4cys21014@ch.students.amrita.edu*](mailto:ch.en.u4cys21014@ch.students.amrita.edu)

Udhaya Kumar SDepartment of Computer Science and Engineering,

Amrita Vishwa Vidyapeetham,

Chennai, India,

[*ch.en.u4cys21014@ch.students.amrita.edu*](mailto:ch.en.u4cys21014@ch.students.amrita.edu)

***Abstract*—With the rapidly shifting landscape in cybersecurity, cyber threats have made the old perimeter-based models obscurate. With the increasing proliferation of cloud-native applications and microservice architectures into organizations, a sound security model shall be a basic necessity that shall withstand these threats. This paper introduces a Zero Trust Security Model for web applications in Kubernetes clusters, utilizing key technologies such as JSON Web Token (JWT) authentication and mutual Transport Layer Security (mTLS). Unlike traditional models that rely on predefined trust zones, this approach enforces continuous identity verification, dynamic access control, and fine-grained micro-segmentation to secure sensitive data. The model incorporates Zero Trust principles, including encrypted communications, strict user and device authentication, and real-time threat detection. It provides robust policy enforcement to protect APIs, containers, databases, and other critical resources in a microservices environment. The persistent observation of user activity, along with the implementation of machine learning algorithms for the identification of threats, improves the system's capability to recognize and address potential incidents in real-time. Our experimental validation shows that this model effectively minimizes lateral movement and unauthorized access, significantly reducing attack surfaces. The proposed Zero Trust architecture offers a scalable, secure, and resilient framework suited to modern containerized applications, ensuring data integrity and security.**

***Keywords—*** ***Zero Trust Security, JWT Authentication, Mutual TLS, Cybersecurity, Web Application Security, Adaptive Access Control, Micro-segmentation, Data Protection, Cloud-Based Architecture, Kubernetes Clusters.***

# **Introduction**

Organizations are facing a wide range of cyber threats in the modern digital environment that reflect the weaknesses of traditional perimeter-based security protocols. These kinds of frameworks - often called "castle-and-moat" defenses - operate on the assumption that threats come from outside a network, thus leaving internal resources open to vulnerabilities. At present, this oversimplification approach has led to serious violations, such as the 2017 Equifax hack, which exposed the personal information of approximately 147 million people and had an estimated cost of $4 billion in remediation and settlements. The rapid adoption of cloud computing, agile methodologies, and containerization has significantly expanded the attack surface, making organizations more susceptible to sophisticated cyber threats. Incidents associated with phishing, ransomware, and insider threats have increased, with ransomware attacks having risen by 150% in 2020, with an estimated damage of about $20 billion. Further, the Verizon 2020 Data Breach Investigations Report indicated that 86% of breaches were financially motivated, highlighting the critical need for adaptive and proactive security measures that can evolve alongside emerging threats. To counter these challenges, the Zero Trust Security Model is emerging as the new paradigm shift in the more comprehensive approach to cybersecurity. Here, unlike the traditional models wherein trust zones are predefined and based on network locations, Zero Trust requires no inherent trust for any entity-be it a user, device, or application. All access requests are screened through continuous authentication and authorization irrespective of origin. In the philosophy of "never trust, always verify" which has a very significant relevance in environments that are characterized with microservice architectures with dispersed data and services on many containers, accessible to users from diversified locations. This paper reports on the conceptualization and implementation of a Zero Trust framework in a modern web application with a microservice architecture. By employing key technologies such as JSON Web Tokens (JWT) for stateless authentication, mutual TLS (mTLS) for secure communication, and micro-segmentation to isolate services, the proposed framework enhances security by continuously verifying all entities and enforcing strict access controls.

This significantly mitigates risks associated with lateral movement attacks, potentially leading to widespread data breaches. The motivation for adopting Zero Trust in Kubernetes clusters arises from the complexities of securing modern distributed systems. Traditional models often grant access based on network location, which is inadequate in environments where applications are highly decoupled and operate across various cloud providers. In these settings, vulnerabilities can be exploited, necessitating a model that safeguards distributed resources and sensitive data effectively. It presents noteworthy contributions toward the domain of cybersecurity by dealing with the constraints inherent in conventional security models and adapting to the fluid characteristics of cloud-native applications. The proposed framework provides a scalable and flexible approach to safeguarding contemporary, containerized settings to ensure the integrity and confidentiality of sensitive information at all places across an organization's application ecosystem.

# **Related** **Works**

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

Varalakshmi et al. [8] explored the enhancement of JSON Web Token (JWT) authentication within Software Defined Networks (SDNs). They identified vulnerabilities in traditional JWT implementations and proposed a modified authentication approach that improves security without sacrificing performance. Their analysis highlighted the importance of securing API endpoints and user sessions, ultimately enhancing the overall integrity of network communications in SDN environments. Bucko et al. [9] assessed how the JWT authentication and authorization process could be improved in web applications using user behavior history. Their study analyzed how behavior-based metrics could improve the reliability of authentication processes. They proposed a dynamic authentication mechanism that adapts based on user activity, significantly reducing unauthorized access and enhancing user security in web applications.

Jánoky et al. [10] conducted an analysis of revocation mechanisms for JSON Web Tokens (JWTs). They identified the challenges associated with effectively revoking tokens without degrading system performance. Their research emphasized the need for efficient revocation strategies to ensure the security of applications relying on JWTs, contributing valuable insights into the management of token lifecycles. Achary and Shelke [11] have analyzed fraud detection in banking transactions by applying some machine learning methodologies. They proposed a framework in order to reduce the fraudulent activities by using multiple algorithms in real time. This showed the effectiveness of machine learning for the enhancement of financial transaction security, providing banks with the same skills to prevent fraud. Thus, the authors underscore the adaptive models that learn from fraud patterns, which are dynamic in nature, making credit card security protocols much more effective.

Fraudulent Transaction Detection in Credit Card by Applying Ensemble Machine Learning Techniques by Prusti et al. [12]: Researches ensemble machine learning models for fraudulent credit card transaction detection. By combining multiple algorithms, it is an effort to enhance precision, accuracy, and speed of detection. Key findings show that ensemble methods outperform individual classifiers in identifying fraudulent transactions, reducing false positives, and increasing overall efficiency in real-time fraud detection systems. Jaculine Priya and Saradha [13] reviewed machine learning algorithms for fraud detection and prevention. Their comprehensive analysis encompassed various techniques, assessing their strengths and weaknesses in different contexts. The paper provided a critical overview of the current state of fraud detection systems, emphasizing the potential of machine learning in enhancing predictive accuracy and operational efficiency in fraud management.

A.I. Weinberg and others [14] discussed Zero Trust Architecture (ZTA) in relation to application and network security, outlining relevance in modern paradigms of cybersecurity. The research detailed how ZTA eliminates implicit trust and enforces strict access controls, providing a robust model for securing sensitive data in dynamic environments. Kang and others [15] provided an abridged version of the Zero Trust security, from theoretical perspectives to practice. They discussed several implementations and looked into the possibility that Zero Trust principles can have an augmenting effect on the overall security posture. Their findings illustrated the applicability of Zero Trust in diverse settings, emphasizing its importance in mitigating modern cybersecurity threats.

Ashfaq et al. [16] conducted an in-depth review of the Zero Trust security framework. This study investigated existing literature while outlining fundamental principles and strategies applicable for Zero Trust in various scenarios. The authors pointed out gaps in related studies, thus providing ideas for future research to deepen understanding and applicability of Zero Trust principles. Liu et al. [17] evaluated the relevant literature for Zero Trust, seeing its potential in IoT systems. They have identified major challenges as well as approaches towards Zero Trust principles in IoT environments. Consequently, they concluded that strict access controls with continuous verification must be in place to solve security issues. The authors provided a comprehensive overview of current research trends and highlighted areas requiring further investigation to enhance IoT security through Zero Trust frameworks.

Mehraj and Banday [18] proposed a Zero Trust framework with cloud computing contexts to overcome security-related issues such as identity theft, data breaches, and complications in trust management. The model proposed by the authors emphasizes the necessity of strong access controls coupled with continuous verification owing to the dynamic and shared nature of cloud services. The authors explained the inadequacy of traditional security approaches within the context of cloud. Muddinagiri et al. [19] presented a method for deploying Kubernetes locally using Minikube to manage Docker containers, emphasizing the advantages of containerization as a lightweight alternative to virtual machines. The paper highlights the importance of local Kubernetes testing for industries like finance and healthcare, which require secure, scalable applications without relying on cloud-based deployments. Their approach was demonstrated using a Python-based web server built with a DockerFile. Pace's thesis [20] focuses on the implementation of Zero Trust Networks using Istio in Kubernetes environments. Istio, as a service mesh, manages traffic, observability, and security, offering features like mutual TLS, JWT-based authentication, and seamless integration with Kubernetes, enhancing microservices security and operational efficiency.

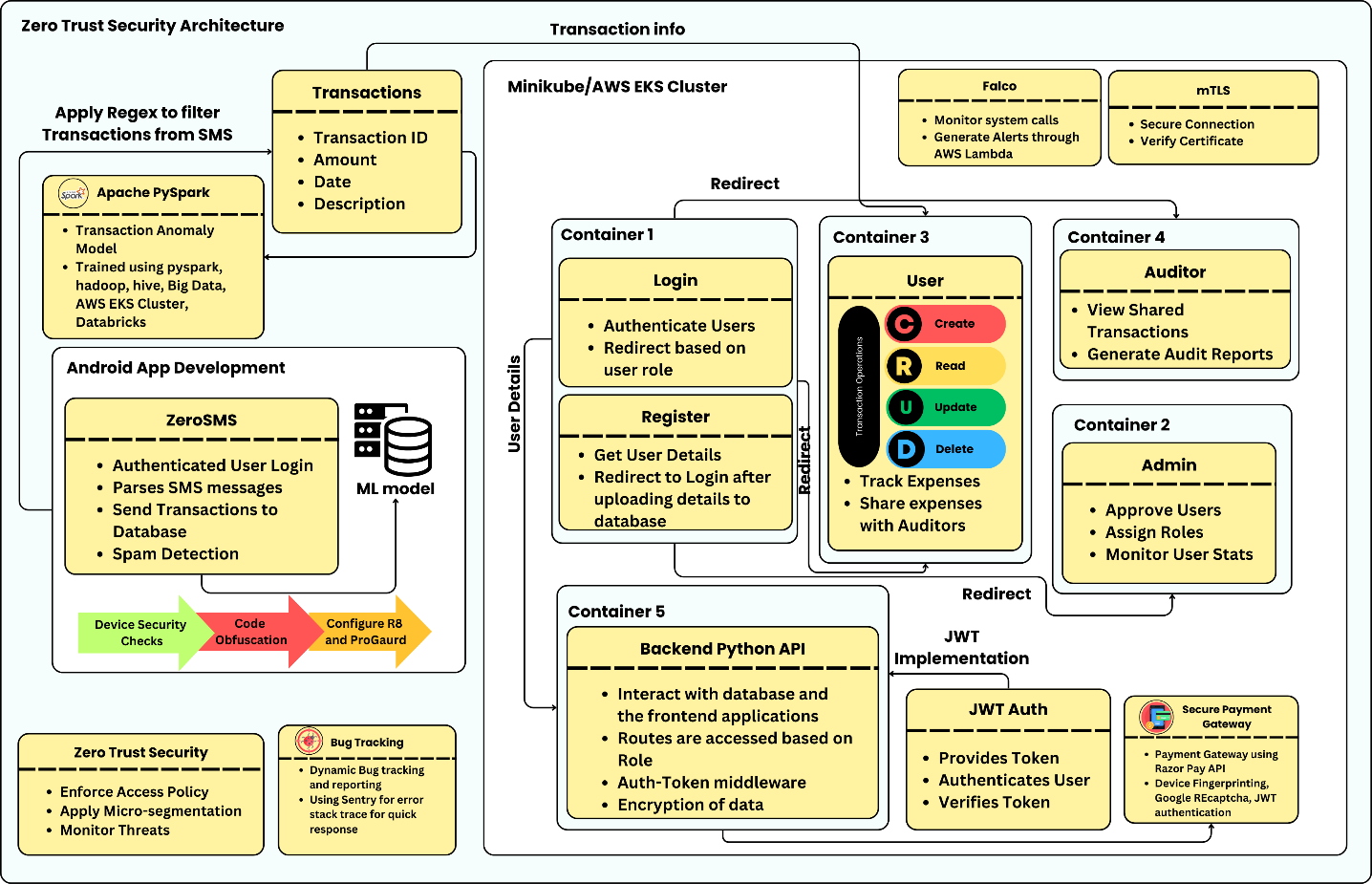


Fig. 1. Architecture diagram of the implementation of Zero Trust in the Web Application

1. **System Architecture**

As illustrated in Figure 1 and 2, the architecture encapsulates several integrated layers—each engineered to uphold the core tenets of Zero Trust Security: least privilege, continuous verification, and strict segmentation of access. At the foundational layer lies the Kubernetes Cluster, which orchestrates containerized microservices using Docker. The application follows a distributed model, where each core functionality—user management, transaction handling, authentication, SMS ingestion, and auditor interfaces—is deployed as an independent service within isolated pods. This decoupled architecture facilitates horizontal scalability, fault isolation, and secure deployment pipelines through Helm charts and CI/CD workflows. Above this layer resides a robust Network Security Framework, comprising service meshes (implemented via Istio) that enforce mutual Transport Layer Security (mTLS) for intra-cluster communication. Each service interaction is verified through cryptographic certificates, ensuring that only authenticated services can communicate within the cluster. This measure prevents lateral movement attacks and enforces strict access control between microservices. The application integrates with AWS Elastic Kubernetes Service (EKS), which serves as the remote hosting environment. The deployment pipeline is governed by Infrastructure-as-Code (IaC) configurations, ensuring that each infrastructure component is provisioned with strict IAM policies, Security Groups, and Virtual Private Cloud (VPC) segmentation. Role-Based Access Control (RBAC) is enforced at multiple levels—both within the Kubernetes cluster and at the application layer—through dynamic policy bindings and contextual identity validation. The system is composed of three core interfaces: User Interface, Admin Interface, and Auditor Interface, each served through dedicated microservices with well-defined access scopes. The User Interface allows end-users to log expenses and visualize their financial data, while the Admin Interface controls role assignments, behavioral analytics, and system policy management. The Auditor Interface, designed with read-only privileges, provides a controlled environment for authorized external reviewers to access shared user data without compromising data integrity.

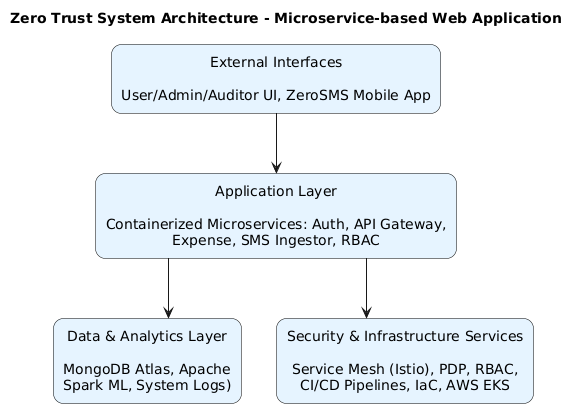


Fig. 2. Various layers in the architecture

The ZeroSMS mobile application acts as an auxiliary data acquisition component in the architecture. It parses SMS-based financial notifications and securely transmits the data to the backend via RESTful APIs. These APIs are containerized and deployed within the cluster, with authentication strictly enforced through JWT-based Access Tokens. Each token is generated, validated, and expired in accordance with pre-set authorization policies, ensuring

that every request undergoes rigorous verification before accessing any internal resource. The system architecture also incorporates a Remote Analytics and Spam Detection Layer, where transaction data from the mobile app is

uploaded to a MongoDB cloud database and subsequently processed by an Apache Spark-powered virtual machine. This layer is responsible for distributed preprocessing, spam classification, and model training using machine learning algorithms. The design promotes high-volume, low-latency processing capabilities, allowing continuous learning and dynamic dataset refreshes without affecting frontend responsiveness. Finally, a dedicated Policy Enforcement Point (PEP) sits at the application gateway and internal service boundaries. The PEP evaluates every request against contextual access policies by interacting with a centralized Policy Decision Point (PDP), enforcing granular access restrictions at route and service levels. This control mechanism ensures that Zero Trust principles are embedded throughout the system—across users, services, and data flows. By integrating container orchestration, fine-grained access control, continuous monitoring, and secure data pipelines, the architecture provides a resilient and future-ready platform for Zero Trust web application development. Each architectural decision is deliberately aligned with the objectives of enhancing user experience, system performance, and security posture, thereby enabling a secure and scalable expense tracking ecosystem.

1. **Methodology**

The proposed containerized web application is designed to provide users with a comprehensive expense tracking system that adheres to Zero Trust security principles as illustrated in Figure 1. This application uses Docker for containerization, Kubernetes for Container Orchestration and Helm to dynamically handle configurations of Kubernetes manifests. The complete application is then deployed to AWS EKS cluster remotely enforced with strict IAM policies and VPC.

A. Web Application Overview

At the core of this application is the user interface, which enables users to easily perform CRUD operations (Create, Read, Update, Delete) on their financial transactions. Users can manually input their expenses, savings, and investments through a form-based system that adds each entry as a card component. A distinguishing feature of this application is its integration with the ZeroSMS mobile app. The ZeroSMS app automatically parses SMS messages related to transactions, which are often sent by banks or payment services, and synchronizes them with the web application. This functionality is especially beneficial for users who may forget to log some expenses or are too busy to manually enter transaction details Another key feature is to the ability for users to share their transaction data with a selected list of auditors. The auditors, who are granted read-only access to user transactions, can view the data to prepare financial reports. The auditor’s interface displays

list of users who have shared their transaction data, allowing them to access the data without modifying it, which aligns with Zero Trust's principle of minimal privileges for authorized users.

1. Admin and Auditor Interfaces

The admin interface is critical for managing user roles and access within the application. New users must first register through a registration page, but their access to the system is only granted upon approval by the admin. Role-based access control guarantees that only the resources required for completing a task in an assigned role are accessible to a given user, which supports the core principle of Zero Trust: least privilege. The admin panel has as well behavior analytics dashboard, along with user management functionality. This feature enables the admin to monitor important logs, such as user login and logout times, system activity, and other behaviors that may indicate suspicious activity. Since the application continuously monitors these behaviors, it will be possible to detect anomalies that can be useful in preventing unauthorized access or wrong usage of the system. This functionality enhances the Zero Trust principle of "never trust, always verify" because the users will have continuous authentication and their behavior examined to reveal possible threats.

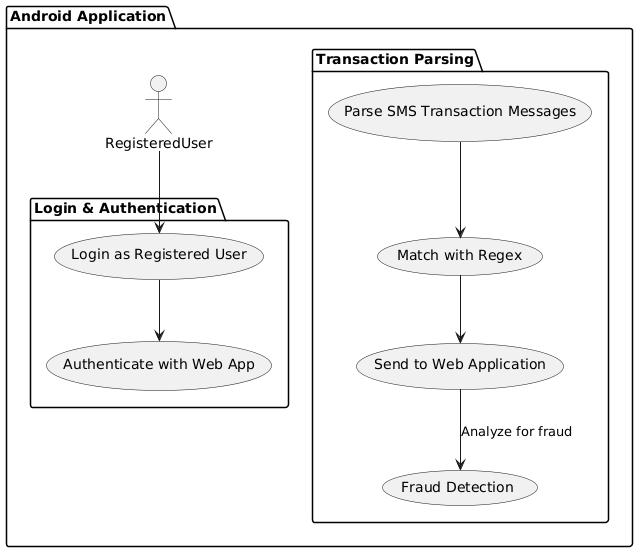


Fig. 3. Activity Diagram for the ZeroSMS Android Application

1. Access Tokens functionality and Implementation

Each request to access sensitive data is validated JSON Web Tokens (JWT), which provide secure, time-bound access tokens that validate the user’s identity and role. If a user’s JWT token is invalid or expired, access is immediately denied, ensuring that no unauthorized requests are processed. JWTs are compact, URL-safe tokens that enable secure information exchange between parties, comprising three elements as illustrated in Figure 5, a header, payload or payload with user claims, and a signature to validate the authenticity. This structure allows for seamless authentication flows, where the server can validate the token without maintaining session information. The Python APIs to generate and manage and rigorously validate JWT are containerized and built to run as a separate pod inside the Kubernetes cluster which ensures continuous authentication of requests from the clients. Upon successful authentication, the system generates a JWT containing essential user-specific claims, such as roles and permissions, which is then transmitted to the user. For subsequent requests, the user includes this token in the authorization header, allowing the application to verify the token's integrity and validity before granting access to protected resources. This implementation does not only follow the least privilege principle by limiting access based on existing permissions granted to the user but also adds security layers as it uses transient access tokens, thus decreasing threats against token theft or misuse.

1. Spam Detection in Financial Transactions

The ZeroSMS app parses the SMS messages and uses reges to classify them as either transaction-related or non-transaction-related. This functionality not only helps users track their expenses more effectively but also mitigates the risk of overlooking minor transactions such as small purchases and pocket money. As, illustrated in Figure. 3, the workflow demonstrates how SMS parsing feeds into the user’s expense management process, emphasizing the seamless integration between the mobile and web platforms.

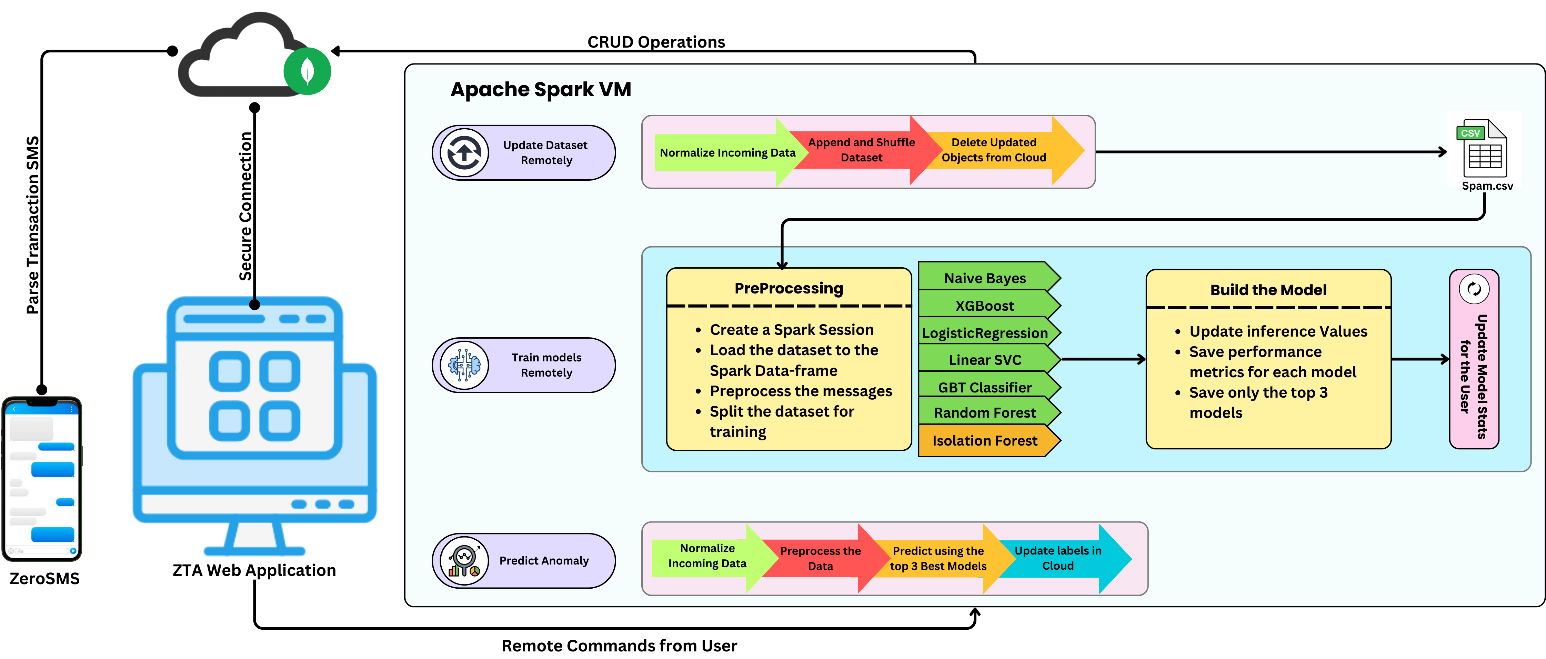


Fig. 4. Remote Mode processing andl training using Apache Spark

To further enhance the functionality of the ZeroSMS app, a Machine Learning (ML) model was developed to detect spam messages. The project was intended to be a strong, large-scale Apache Spark-powered pipeline and remote model training as shown in Figure. 4. Essentially, the project was meant to process SMS transaction messages for analysis, identify potential fraud or anomalies, and dynamically learn new patterns in the data. The basic approach was based on distributed data processing using Spark, remote machine learning model training, and dynamic dataset refreshes to provide scalability and responsiveness.

The process had commenced with SMS data extraction from the ZeroSMS mobile application, which parsed transaction-related messages. These messages were securely transmitted to a MongoDB cloud database, from which Apache Spark, deployed on a virtual machine (VM), ingested and processed the data. Spark played a pivotal role in this methodology as it enabled distributed computing, allowing large-scale data processing with efficiency and speed. The data consumed were subjected to a series of preprocessing operations, ranging from normalization to dataset shuffling and removal of old records to ensure that clean, updated, and unbiased data was provided to the training pipeline. Once the data was preprocessed, it was utilized for remote model training on the Spark-powered VM. Spark’s MLlib library served as the primary machine learning toolkit, enabling the construction and evaluation of multiple models concurrently. Algorithms such as Naive Bayes, Logistic Regression, Random Forest, Gradient Boosted Trees (GBT), Linear Support Vector Classifier (SVC), and XGBoost were employed directly from MLlib. Furthermore, Isolation Forest, another popular anomaly detection algorithm, was externally integrated by adding its JAR file from Maven to the Spark environment during runtime. This enabled a hybrid modeling strategy by blending both classic classifiers and outlier models, thereby guaranteeing a complete assessment of SMS transaction behaviors.

The training phase was performed remotely, thus offloading the computationally expensive process of model building from the mobile device or local system to a VM running Spark. This helped ensure the system scaled well with increasing data volumes, bypassing computational limits that could be posed by the use of mobile devices or local systems. The trained models were therefore tested against performance metrics such as accuracy, precision, recall, and F1-score values; however, only the top three performing models were kept for future inference. This process of selection ensured that the most accurate models were used for detecting fraud. For real-time prediction, incoming SMS data was normalized and preprocessed similarly. The processed data was then passed to the top three trained models, which collectively evaluated the likelihood of the message being fraudulent. The prediction results were uploaded back into the MongoDB cloud database, enabling the ZeroSMS application to retrieve and display the classifications to the user.

An integral part of the methodology was dynamic data updating and continuous model evaluation. As transactional patterns in SMS messages evolved over time, the system was designed to support on-demand remote training whenever required. This allowed the models to be refreshed periodically, ensuring that the detection capabilities adapted to emerging fraud patterns. The ZeroSMS web application provided users with the functionality to trigger remote model training, update datasets, and initiate predictions, granting full control over the learning and inference pipeline. This methodology, driven by Spark’s distributed computing, remote model training, and dynamic model updates, ensured that the ZeroSMS system was not only scalable and efficient but also resilient to the ever-changing nature of transactional fraud in SMS-based communications. As shown Table 1, the performance metrics highlight the model's precision and recall rates, emphasizing its capability to minimize false positives and negatives in spam detection.

1. Policy Enforcement Point for RBAC

A Policy Enforcement Point (PEP) is a critical security component in modern web applications, particularly within a Zero Trust Architecture (ZTA). It plays a foundational role in safeguarding access to sensitive resources and enforcing security policies across the application ecosystem. In essence, the PEP acts as a gatekeeper that intercepts every incoming request before it reaches the core services or application logic. This ensures that only authorized users, applications, or services are permitted to interact with the system. The primary responsibility of the PEP is to validate, authenticate, and authorize requests in

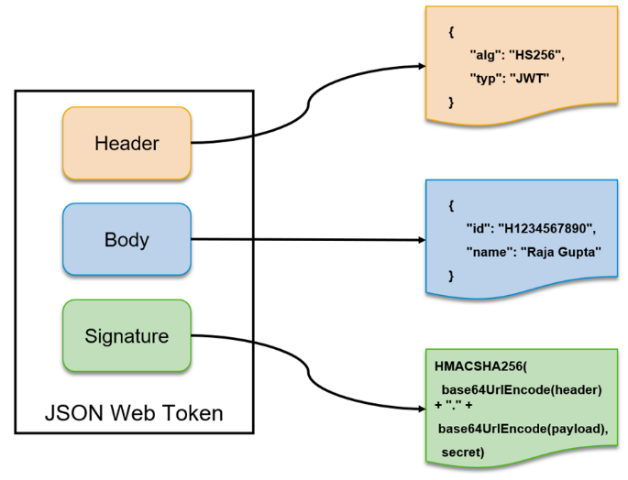


Fig. 5. JWT Architecture

real-time according to pre-defined security policies, typically driven by Role-Based Access Control (RBAC) or Attribute-Based Access Control (ABAC) models. When used in microservice-based web applications, the PEP is even more critical. Microservices inherently divide the application into small, standalone services that communicate with one another across a network. This distributed nature expands the attack surface, and therefore, uniform access control and policy enforcement across services is a requirement. In such environments, the PEP is often integrated into API gateways, middleware layers, or deployed as sidecar containers alongside each microservice. This placement ensures that every incoming request—whether from an external client or an internal microservice—passes through the PEP before gaining access to any resource. The PEP evaluates each request against the security policies and collaborates with a Policy Decision Point (PDP). The PDP is responsible for making the actual access control decision based on the user’s role, attributes, context, or other security policies. Once the PDP returns its decision, the PEP either permits or denies the request accordingly.

For instance, in our microservice-based application, the PEP ensures that an auditor is allowed to access only transaction data related to their specific clients, while other data remains restricted. Additionally, route-level restrictions are enforced based on user roles (e.g., User, Admin, Auditor). Each role is granted access only to the endpoints necessary for performing their duties. This is an example of the least privilege principle, where users and services are granted, the minimum permissions required to complete their tasks, reducing the risk of unauthorized data exposure or malicious activity. Moreover, in containerized environments, PEPs can be deployed as sidecar containers alongside each microservice. This pattern is commonly used in service mesh architectures, where access control policies are enforced at each individual service endpoint, ensuring that service-to-service communications within the microservice environment are also tightly controlled. This internal security is particularly important when microservices are sharing sensitive information, since any compromised service can be used to gain access to other areas of the system if controls are not implemented.

In addition to access control, PEPs can also enforce security policies related to rate limiting, IP whitelisting, authentication protocols, and logging. Every request is meticulously evaluated before being routed to the intended microservice, ensuring that security is embedded at the network and application layers. Furthermore, audit trails and logging mechanisms integrated with PEPs provide visibility into access patterns, helping detect unauthorized access attempts or potential security breaches in real-time. Ultimately, the Policy Enforcement Point strengthens the overall security posture of a Zero Trust microservice-based system by enforcing uniform policies, limiting access on a need-to-know basis, securing service-to-service interactions, and mitigating attack vectors. This results in a granular, dynamic, and context-aware access control mechanism, aligning perfectly with the core principles of Zero Trust Security — “Never trust, always verify”.

1. mTLS within the Kubernetes Clusters

Mutual TLS (mTLS) is a foundational security mechanism in implementing Zero Trust Architecture (ZTA) for microservices-based applications [21]. It plays a crucial role in establishing secure, authenticated, and encrypted communication channels between microservices within a Kubernetes cluster, aligning with the “never trust, always verify” principle of ZTA. Unlike traditional network security models, which often assume that traffic within a network is inherently trusted, mTLS requires each service-to-service interaction to undergo mutual authentication before any data exchange takes place. This ensures that both the client and the server involved in the communication verify each other's identity and establish a secure TLS (Transport Layer Security) connection, protecting the data from eavesdropping, interception, or tampering. In an mTLS setup within Kubernetes, each pod in the cluster is assigned a unique cryptographic certificate, issued by a trusted Certificate Authority (CA). These certificates serve as identity proofs for services within the cluster. When two microservices attempt to communicate, the TLS handshake process involves both the client and the server presenting their respective certificates. During this process, each party validates the other's certificate by using the public key of the trusted CA. Once both certificates are verified, an encrypted communication channel is established using TLS, securing the data exchanged between the services.

This mutual authentication mechanism prevents unauthorized or malicious services from impersonating legitimate services within the cluster. Without a valid certificate issued by the trusted CA, a rogue service cannot participate in the communication, thereby eliminating the risk of unauthorized access to internal resources. Additionally, mTLS guarantees data integrity and confidentiality, ensuring that data-in-transit cannot be intercepted, altered, or leaked by malicious actors, even if they gain network access within the cluster. In our microservice-based application, mTLS is integrated seamlessly using a service mesh tool called Istio, which automates certificate issuance, renewal, and rotation processes, while enforcing mTLS at the network layer. This eliminates the burden on individual microservices to implement TLS logic manually, as the service mesh handles all security aspects transparently. When microservices within the cluster interact, Istio ensures that every request is authenticated through mTLS, and secure connections are maintained consistently across the system.

A significant advantage of this approach is that mTLS functions independently of the application logic, as the service mesh operates at the infrastructure level. This allows developers to focus on business logic, while security concerns like authentication, encryption, and certificate lifecycle management are offloaded to the mesh. Furthermore, mTLS combined with Policy Enforcement Points (PEPs) enhances the Zero Trust security posture. While mTLS validates the identity of services and secures the communication channel, PEPs ensure that the authenticated service has the necessary authorization to access a resource. Together, these mechanisms enforce both authentication and fine-grained authorization controls, reducing the attack surface within the microservice environment. Unlike traditional perimeter-based security models, mTLS ensures that every microservice communication is treated as untrusted until verified. This aligns perfectly with ZTA principles, preventing lateral movement attacks within the cluster, even if an attacker breaches a single microservice or gains access to the internal network. With mTLS enabled across the Kubernetes cluster, every microservice interaction is identity-verified and encrypted, minimizing the risk of spoofing attacks, man-in-the-middle attacks, and unauthorized access [22].

Ultimately, mTLS within Kubernetes clusters, supported by service meshes like Istio, represents a robust security control in Zero Trust environments. It establishes trust at the communication level, protects service-to-service interactions, and, when integrated with PEPs and RBAC/ABAC policies, ensures that access to sensitive resources is both authenticated and authorized, thereby fortifying the security of microservice architectures [21][22].

1. **RESULTS**

With the Zero Trust application, some of the fundamental principles of Zero Trust Architecture were fully implemented: strict access control, continuous authentication, verification of end-users and devices, and proper security protection of sensitive resources by integrating role-based access control and least-privilege policies. The integration of Kubernetes for container orchestration further enhanced scalability and resource management, allowing the system to adapt to the changing security requirements in real-time. Overall, the implementation exhibited robust security postures with minimal performance overhead. The web application was deployed in an EKS cluster, utilizing node groups belonging to the EKS Cluster. As illustrated in Figure 6., the architecture ensures scalability and fault tolerance, with seamless communication between services for a smooth user experience.

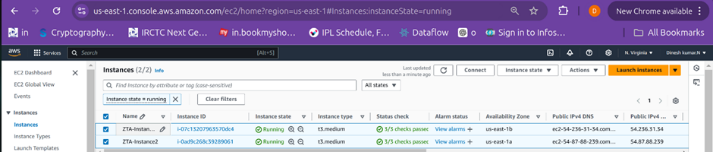


Fig 6. Node Groups of EKS Cluster deployed as EC2 instances

Security within the application is strengthened through AWS Lambda and the Falco tool, which monitors the cluster for anomalies and triggers alerts for real-time threat detection. It monitors system calls in real-time to detect abnormal behavior or potential threats. By intercepting system calls using eBPF, Falco analyzes actions like file access and process creation against a set of predefined security rules. When a rule is violated, it generates alerts and can trigger responses, such as blocking processes or notifying security teams. In microservices, Falco enhances security by continuously monitoring for unauthorized access, unexpected process launches, and unusual network activity, thus providing vital insights into runtime behavior and security posture. Figure 7. showcases the alerts from Lambda as logs captured in AWS Cloudwatch. These logs contain valuable information such as the violated rule, container name, container ID, pod name, and namespace where the attack occurred, along with the system call used to perform the attack. This collected data shows the deployment of the Zero Trust strategy and assists in identifying the cause of suspicious activity.

Additionally, session logs provide a thorough audit trail, tracking user activities and identifying suspicious behavior. These logs ensure compliance with Zero Trust principles by ensuring all user actions are continuously monitored and recorded.

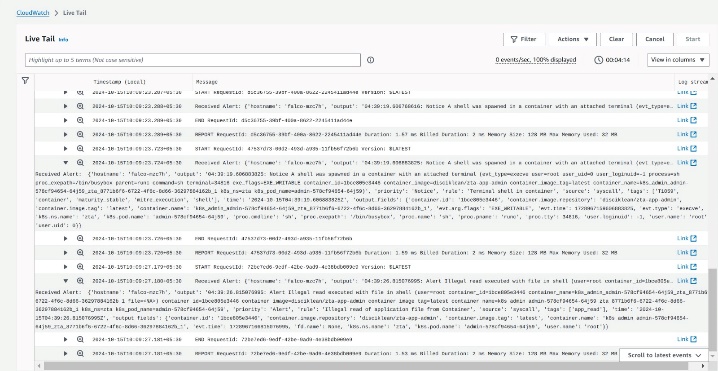


Fig 7. AWS Lambda Alerts triggered by Falco tool deployed in the cluster

The comparison of different machine learning models as shown in Figure 8, for classifying SMS messages as ham or spam was done on a balanced dataset consisting of 83 ham messages (49.7%) and 84 spam messages (50.3%), which made the results unbiased and fair to both classes. The models assessed during this phase included Logistic Regression, Random Forest, GBTClassifier (Gradient Boosted Trees), Naive Bayes, LinearSVC, and XGBoost. The performance of each model was measured based on three key metrics: Accuracy, F1-score, and ROC-AUC score. Each metric provided unique insights into the model's effectiveness. Accuracy measured the proportion of correctly classified messages. F1-score, being the harmonic mean of precision and recall, highlighted the model’s ability to balance false positives and false negatives. ROC-AUC score assessed the model’s capability to distinguish between spam and ham, with higher values indicating stronger discriminative power.

After examining the outcomes, XGBoost was the best-performing model in all the evaluation metrics. It recorded an accuracy of 99%, an F1-score of 99%, and a ROC-AUC score of 0.98, reflecting its outstanding performance in both accurately classifying messages and having a near-perfect false positive to false negative ratio. Following XGBoost, Naive Bayes also demonstrated remarkable performance with an accuracy of 94% and an F1-score of 96%, showcasing its strength in text classification tasks, especially in scenarios involving word frequency-based feature extraction. LinearSVC stood out as another reliable choice, achieving an accuracy of 89% and a F1-score of 92%, proving its efficiency in handling linearly separable text classification tasks. Conversely, Logistic Regression yielded a strong and stable baseline performance at 87% accuracy, and hence is a reliable option for generic classification problems. Random Forest was fairly effective at 80% accuracy, implying it picked up on some data trends but fell behind more sophisticated ensemble algorithms such as XGBoost. The worst performance came from the GBTClassifier at 69% accuracy, perhaps due to the risk of overfitting or poorly optimized hyperparameters.

TABLE 1. 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 |

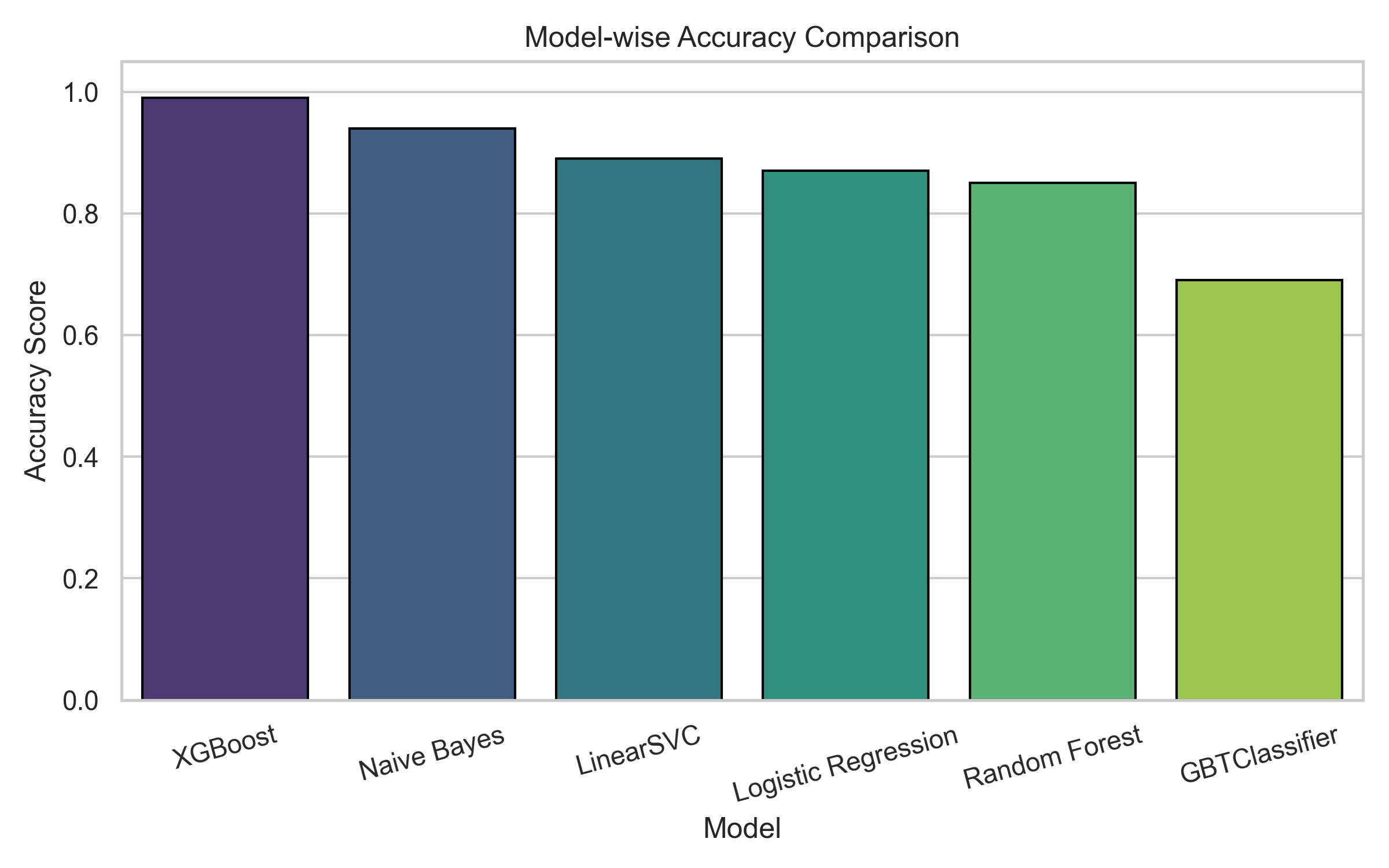


Fig 8. Model’s Accuracy Comparison in chart view

The evaluation process followed a structured workflow. Initially, the SMS data underwent preprocessing, including text normalization, tokenization, and feature extraction. Subsequently, each model was trained in a distributed computing environment using Spark, ensuring scalability and efficiency. After training, the models were tested on unseen data, and their performance metrics were calculated. Finally, the evaluation results were visualized using bar graphs for clarity, enabling quick identification of the top-performing models. Based on the evaluation, XGBoost, Naive Bayes, and LinearSVC were identified as the most suitable models for deployment in the production environment. To prepare for production release, the app leverages the JailMonkey library to detect rooting and unsafe environments, while utilizing R8 and ProGuard tools for code optimization and obfuscation to enhance performance and security as shown in Figure 9. The APK is securely signed with a private key from a trusted keystore, ensuring authorized distribution. Due to R8 and ProGuard obfuscation, reverse engineering ZeroSMS is significantly more challenging as illustrated in Figure 10.

To prepare the ZeroSMS application for a secure production release, a multi-layered security approach was employed, focusing on code obfuscation, runtime environment detection, and secure application signing. These measures collectively protect the application’s intellectual property, enhance performance, and mitigate potential reverse engineering or tampering attempts. Obfuscation transforms the application’s source code into a complex and unintelligible format, making it significantly harder for attackers to reverse-engineer, analyze vulnerabilities, or extract sensitive logic. This technique guards the intellectual property of the application and adds a layer of defense against code tampering and unauthorized modifications. ProGuard and R8 are two widely used tools for code obfuscation and optimization in Android applications. ProGuard is an open-source tool designed for Java and Android applications, offering a combination of shrinking, optimizing, and obfuscating the code. It minimizes the application’s size by removing unused classes and methods while renaming classes, fields, and methods with non-descriptive names, making it challenging for adversaries to analyze the code structure. This reduces the attack surface and protects against static analysis-based attacks. R8, on the other hand, is a more advanced and modern successor to ProGuard, specifically integrated into Android’s build system to improve both code optimization and obfuscation. R8 performs aggressive code shrinking, dead code elimination, and method inlining, resulting in smaller APK sizes and improved runtime performance. It renames identifiers to obscure names, similar to ProGuard, while achieving faster build times and more sophisticated optimization. Given its superior performance and tighter integration with Android Studio, R8 is now the default compiler in Android development and is favored by modern development teams.

|  |  |  |
| --- | --- | --- |
|  |  |  |

Fig. 9. ZeroSMS in production release with security features

Combining R8 and ProGuard obfuscation techniques, the ZeroSMS app's code is transformed into a compact and difficult-to-reverse-engineer format, raising the complexity for attackers attempting to decompile and analyze the APK, as depicted in Figure 10. Additionally, root detection, debugging detection, and developer options detection mechanisms are integrated into the ZeroSMS application to prevent execution in insecure environments. These runtime checks are crucial in a Zero Trust security model, as rooted devices, active debugging and enabled developer options can expose applications to tampering, data extraction, and runtime modifications. Rooting grants users elevated privileges, enabling them to bypass Android’s security mechanisms. This increases the risk of data theft, unauthorized API access, and application tampering. Attackers can exploit rooted environments to extract encryption keys, intercept API communications, or alter application behavior, posing severe security threats to financial applications like ZeroSMS.

TABLE 2. METRICS SECURITY COMPARISON OF MOBILE APPLICATIONS USING MOBSF

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **S.No** | **Application Name** | **Security Score (/100)** | **Grade** | **High Severity** | **Medium Severity** | **Info** | **Secure Flags** | **Trackers** | **Privacy Risk** | **Cleartext HTTP** | **Notable Vulnerabilities** |
| 1 | Zero-SMS (HTTP Disabled) | 64 | A | 0 | 8 | 1 | 2 | 0 | 0 | No | Weak RNG, SHA-1/MD5, SQL Injection Risk |
| 2 | Zero-SMS (HTTP Enabled) | 57 | B | 2 | 9 | 1 | - | - | - | Yes | SMS Permissions, Cleartext HTTP, SQL Injection Risk |
| 3 | Proton Password Manager | 51 | B | 2 | 20 | 2 | 2 | 1 | 1 | Yes | Padding Oracle Attack, Insecure WebView |
| 4 | Google Play Services | 53 | B | 2 | 31 | 3 | 3 | 2 | 2 | Yes | Padding Oracle, Janus Vulnerability |
| 5 | Instagram | 40 | B | 18 | 62 | 2 | 2 | 1 | 1 | Yes | StrandHogg 2.0, WebView, Janus, Task Hijacking |
| 6 | Celcom Digi | 56 | B | 1 | 12 | 1 | 2 | 2 | 2 | Yes | Padding Oracle, Insecure Activities |
| 7 | Capmocracy Social Network | 55 | B | 1 | 2 | 0 | 1 | 0 | 0 | Yes | Janus, Backup Enabled, Unpatched SDK |

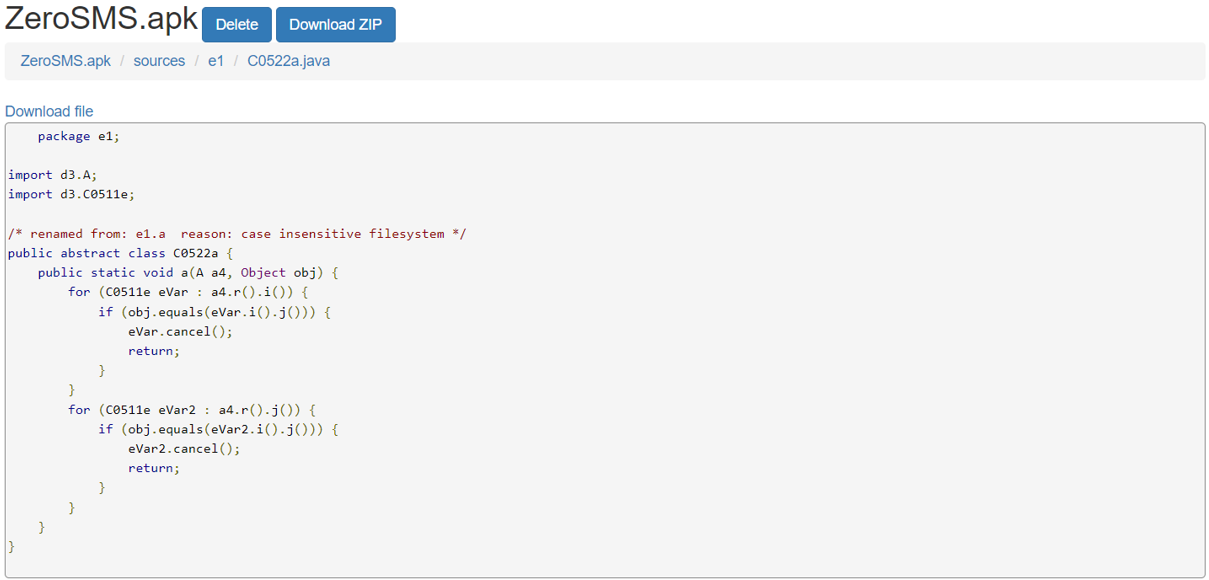


Fig. 10. ZeroSMS with class name and code Obfuscation

tools allow attackers to analyze application behavior, trace API calls, and identify vulnerabilities in real-time. Attackers can use debuggers to bypass security checks, intercept sensitive data, or manipulate transactions. Detecting and blocking active debugging sessions prevents these forms of dynamic analysis. Enabling developer options can introduce security risks, such as allowing USB debugging, OEM unlocking, or mock location data, which facilitates application tampering or unauthorized testing. Detecting and restricting application functionality in such environments mitigates potential exploitation. To implement these runtime environment checks, the ZeroSMS application leverages JailMonkey, an open-source security library widely used in React Native applications. JailMonkey can detect rooted devices, active debuggers, and developer options by analyzing system properties, file system paths, and process behaviors. When any unsafe condition is detected, the application can respond by restricting functionality or terminating execution, thereby minimizing the risk of data leakage or security breaches. Lastly, to ensure the authenticity and integrity of the APK package, the ZeroSMS application is signed with a private key from a trusted keystore before distribution. This cryptographic signing process verifies that the APK has not been tampered with after release, ensuring only the officially authorized version of the app can be installed on devices. Users receive a warning if an unauthorized or modified APK is detected, adding another safeguard against malicious repackaging attacks.

The MobSF scan, as illustrated in Figures 11 a) and b), evaluates the security posture of the ZeroSMS application compared to a standard APK. ZeroSMS achieves a security score of 57, earning a Grade B, whereas the normal APK scores 31, receiving a Grade C. This reflects that ZeroSMS’s security score is 83.87% higher than that of the standard APK. The difference in scores is largely attributed to the permissions and network configurations required by ZeroSMS. Specifically, ZeroSMS has SMS read and write permissions enabled, which increases its risk exposure compared to a simple application. Additionally, ZeroSMS allows "http" traffic for backend connectivity, which is defined in its AndroidManifest.xml file using the configuration android:usesCleartextTraffic="true". This setting was necessary to facilitate communication with the backend service; however, it negatively impacts the security score because, by default, Android applications are designed to connect exclusively via secure "https" protocols. In contrast, the standard APK is a basic “Hello World” application created using Android Studio, with no special permissions or network configurations, resulting in a lower attack surface. While ZeroSMS’s security score special permissions or network configurations, resulting in a lower attack surface. While ZeroSMS’s security score was affected by the use of SMS permissions and cleartext traffic, its overall score still reflects a robust security posture, significantly outperforming the standard APK.

To further put the security stance of ZeroSMS into perspective with other popular production-grade applications, a detailed MobSF-based security scorecard was created. Table 2. shows a comparative study of ZeroSMS (both HTTP-allowed and HTTP-blocked) with some of the most popular mobile applications such as Proton Password Manager, Instagram, and Google Play Services. The table shows important metrics like security scores, risk ratings, number of critical vulnerabilities, and privacy trackers. Notably, ZeroSMS (HTTP-blocked) registers a better security score than a lot of commercially deployed applications and gets a Grade A rating with very few high-severity results and no privacy trackers. The comparative analysis reveals that ZeroSMS has a similar or

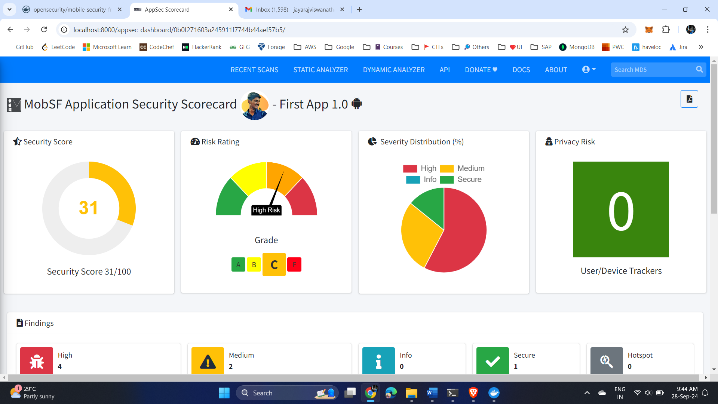


Fig. 11. a) MobSF scan result of standard APK

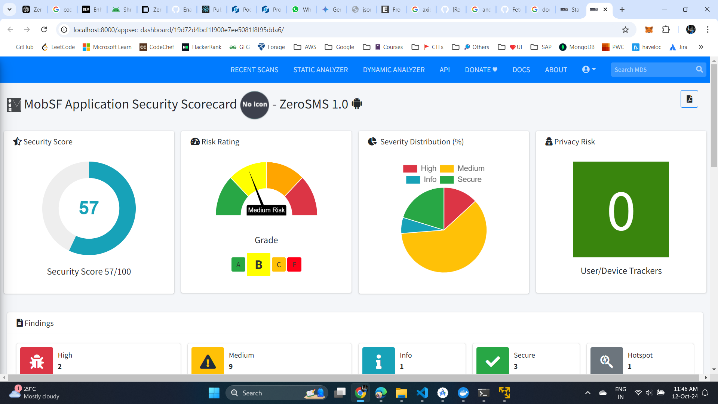


Fig. 11. b) MobSF scan result of ZeroSMS

even better security posture to some commercially deployed applications, testifying to the efficacy of runtime environment checks employed, application hardening practices used, and cryptography integrity verification features implemented. The HTTP-enabled version of ZeroSMS, although slightly lower in score because of its cleartext traffic configuration, remains competitive, affirming that focused improvements in network security policies could further improve its application security grade.

1. **CONCLUSION AND FUTURE WORK**

In today’s landscape, where cyber threats are growing in sophistication, securing containerized environments is crucial. This project has explored various strategies and best practices to bolster security through the implementation of Zero Trust principles. By developing a proof of concept (PoC) that includes a secure web application, a Python API, and an Android app, we have effectively demonstrated how to adopt Zero Trust principles. Essential strategies such as JWT authentication, continuous monitoring, code obfuscation, root detection, mutual TLS (mTLS) within containers, and robust policy enforcement for Role-Based Access Control (RBAC) have been applied throughout the development process. The results highlight the importance of a dynamic, layered approach to security, ensuring that trust is verified at every stage and vulnerabilities are monitored consistently, significantly reducing risks.

Looking ahead, there are several opportunities to advance security in containerized environments. Future work could include leveraging machine learning algorithms to detect anomalous behavior and patterns in containerized applications, further enhancing continuous monitoring. Automating security auditing processes would allow for real-time vulnerability scanning and compliance checks, ensuring adherence to industry standards. Integrating security practices into CI/CD pipelines would guarantee that security is considered at every phase of application development. Additionally, offering resources and training on Zero Trust principles to developers and operators will foster a security-conscious culture across teams. Conducting real-world deployments of the applications in various organizational settings will provide valuable insights into security performance and user experiences, driving further refinements. By expanding on these areas, organizations can strengthen their security measures and protect their containerized environments against evolving threats while staying aligned with the Zero Trust model.

1. **REFERENCES**
2. O. E. Imokhai, T. E. Emmanuel, A. A. Joshua, E. S. Emakhu, and S. Adebiyi, "Zero Trust Architecture: Trend and Impact on Information Security," International Journal of Emerging Technology and Advanced Engineering, vol. 12, no. 7, pp. 87-92, July 2022.
3. 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.
4. 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.
5. 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.
6. B. Pranoto, "Threat Mitigation in Containerized Environments," Applied Research in Artificial Intelligence and Cloud Computing, vol. 6, no. 8, pp. 142-151, 2023.
7. 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.
8. 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.
9. 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.
10. 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.
11. 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.
12. 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.
13. D. Prusti and S. K. Rath, "Fraudulent Transaction Detection in Credit Card by Applying Ensemble Machine Learning Techniques," in Proc. Jun 2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT),
14. 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.
15. 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.
16. 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.
17. 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.
18. 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.
19. 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.
20. 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).
21. M. Pace, Zero Trust Networks with Istio, Master’s thesis, Politecnico di Torino, Turin, Italy, 2020.
22. A. Kurbatov, Design and Implementation of Secure Communication Between Microservices, Master’s thesis, Aalto University, Espoo, Finland, 2020.
23. 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