ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26mar1775

Secure API Management Using Zero Trust Architecture and AI-Based Threat Detection and Prevention

N. Shalini[1]; K. Jishnu[2]; M. Sonuu[3]; K. Jaideep[4]

[1]Assistant Professor, Department of Computer Science and Engineering-Cybersecurity, Malla Reddy University, Hyderabad, Telangana, India
[2,3,4]Student, Department of Computer Science and Engineering-Cybersecurity, Malla Reddy University, Hyderabad, Telangana, India

Publication Date: 2026/04/06

Abstract: Modern applications rely heavily on APIs, which makes them a major target for cyberattacks. Traditional security models focus mainly on perimeter protection and often fail to provide continuous verification after a user is authenticated. This work presents an AI-powered Zero Trust API Security Platform designed to monitor, detect, and automatically respond to suspicious API behavior in real time. The system combines rule-based behavioral analytics with an unsupervised machine learning model (Isolation Forest) to identify both known attack patterns and previously unseen anomalies. The platform implements multi-layer security using JWT authentication, role-based access control, IP rate limiting, and token revocation mechanisms. A behavioral analytics engine evaluates each request against multiple threat detection rules covering injection attacks, credential abuse, endpoint scanning, and automated attack tools. In parallel, a Python-based ML service analyzes request features to assign anomaly scores that trigger automated blocking when risk exceeds a defined threshold. Real- time WebSocket alerts provide administrators with continuous visibility into threat activity. The system was developed using Spring Boot, React, PostgreSQL, MongoDB, and Python Flask and validated through simulated attack scenarios. Results show that the combined rule-based and ML approach enables fast detection and automated remediation of diverse API threats. During implementation, challenges related to feature selection and balancing detection sensitivity with false positives were observed, highlighting the need for adaptive security models in dynamic API environments. This work demonstrates that integrating Zero Trust principles with AI-driven behavioral monitoring can provide a practical and scalable approach to securing modern API-driven applications.

Keywords: *Zero Trust Architecture, API Security, Machine Learning, Anomaly Detection, Isolation Forest, Behavioral Analytics, Threat Detection, JWT Authentication, RBAC, Real-Time Prevention, Cybersecurity.*

How to Cite: N. Shalini; K. Jishnu; M. Sonuu; K. Jaideep (2026) Secure API Management Using Zero Trust Architecture and AI-Based Threat Detection and Prevention. *International Journal of Innovative Science and Research Technology*, 11(3), 3225-3232. https://doi.org/10.38124/ijisrt/26mar1775

I. INTRODUCTION

Modern software ecosystems are dominated by microservice architectures where thousands of API endpoints serve as the primary communication fabric. According to recent industry reports, API attacks increased by over 400% between 2021 and 2024, with breaches costing organizations an average of $4.24 million per incident and remaining undetected for a median of 287 days [1]. Traditional security perimeters are ineffective against insider threats, compromised credentials, and zero-day attack vectors that operate entirely within trusted network segments.

Zero Trust Architecture (ZTA), introduced by NIST Special Publication 800-207, establishes a security model based on the principle of "never trust, always verify." Every request, regardless of its network origin, must be continuously authenticated, authorized, and assessed for behavioral anomalies. However, purely rule-based Zero Trust implementations cannot identify novel attack patterns that fall outside predefined signatures, necessitating the integration of unsupervised machine learning.

This paper presents a fully functional Zero Trust API Security Platform that addresses these limitations through a multi-layered security architecture. The system implements 15 real-time threat detection rules in a behavioral analytics engine, augmented by an Isolation Forest machine learning model. Threats are scored and automatically remediated within milliseconds. The platform is deployed as three cooperating services: a Spring Boot 3.2 backend (port

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26mar1775

8080), a React 18 administrative frontend (port 3000), and a Python Flask ML service (port 5001), backed by PostgreSQL for relational data and MongoDB for time-series behavioral logs.

The key contributions of this work are: (1) a production-ready Zero Trust enforcement pipeline comprising IP rate limiting, JWT validation with token revocation, and role-based access control; (2) a 15-rule behavioral analytics engine with per-rule threat scoring and automatic blocking at a configurable threshold; (3) integration of an online-learning Isolation Forest model that adapts to real API traffic; and (4) a real-time WebSocket-powered administrative dashboard providing live threat feeds, API log monitoring with filtering, and instant user management.

➢ *Problem Statement*

Modern API systems have several security problems that existing tools cannot handle properly when used together. Most current solutions work separately—API gateways only handle login authentication but do not track user behavior, Web Application Firewalls can detect only known attacks, and Intrusion Detection Systems often give too many false alerts and need constant updates. Monitoring systems also show information only after an attack happens, not while it is happening. One major issue is that once a user logs in, the system does not continuously check if the user is still trustworthy. If a JWT token is stolen, the attacker can use it until it expires. Another problem is that rule-based systems cannot detect new or unknown attacks like slow credential stuffing or API fuzzing. In addition, handling threats is slow and manual—security teams must check alerts, find the issue, and block users themselves, which takes time and allows more damage. There is also no single real-time view of all activities, making it hard to understand what is happening during an attack. To solve these problems, this project proposes a unified Zero Trust security system that continuously verifies users, uses both rules and machine learning to detect threats, automatically blocks attacks and revokes access within milliseconds, and provides a real-time dashboard for better monitoring and quick response.

II. LITERATURE SURVEY

➢ *Zero Trust Architecture*

The Zero Trust model, formalized by Kindervag (2010) and standardized by NIST SP 800-207 (2020), asserts that no entity — internal or external — should be inherently trusted. Rose et al. (2020) demonstrated that ZTA deployments reduce lateral movement attack success by up to 70% compared to perimeter-based models. However, most ZTA implementations focus on network-level controls and do not address API-specific behavioral anomalies or provide automated remediation [3].

➢ *Machine Learning for Anomaly Detection*

Liu et al. (2008) introduced the Isolation Forest algorithm, demonstrating that anomalies can be efficiently isolated using random partitioning trees without requiring labeled training data. Subsequent studies by Chen et al. (2021) applied Isolation Forest to network intrusion detection, achieving F1 scores above 0.91 with contamination parameters between 0.05 and 0.15. Aharon et al. (2024) extended few-shot learning approaches to API-specific attack detection, achieving high accuracy with minimal labeled samples but at the cost of increased inference latency unsuitable for real- time gateways [4].

➢ *API Security and Behavioral Analytics*

Sharma and Mehta (2021) demonstrated that integrating behavioral profiling with JWT authentication significantly reduces unauthorized access incidents in REST API environments. Kim and Park (2023) applied explainable AI techniques to API call sequence analysis, improving interpretability but reporting lower performance against encrypted traffic. Critically, none of these works implement a complete end-to-end system combining rule-based detection, ML scoring, automatic blocking, and real-time administrative visibility in a single deployable platform [5].

➢ *Rate Limiting and Token Revocation*

Token bucket and leaky bucket algorithms for API rate limiting are well-established (Tanenbaum, 2011). Bucket4j's token bucket implementation provides sub- millisecond rate checking suitable for production API gateways. JWT token revocation is typically handled through short expiry windows or token blacklists. SHA- 256 hashing of revoked tokens, as implemented in this work, provides a compact, collision-resistant blacklist that scales to millions of revoked tokens without significant storage overhead [6].

III. SYSTEM ANALYSIS

➢ *Existing Systems and their Limitations Contemporary API Security Deployments Rely on Four Categories of Tools, Each with Significant Limitations:*

Traditional API Gateways (Kong, AWS API Gateway, Apigee): Provide authentication, basic rate limiting, and request transformation but lack behavioral analytics, anomaly detection, and automated threat response. Rate limits are static and cannot adapt to attack patterns.

Web Application Firewalls (ModSecurity, Cloudflare WAF): Detect known injection and scripting attacks through signature matching but miss zero-day threats, business logic abuse, and API-specific attack patterns such as credential stuffing and endpoint scanning. False positive rates are high, requiring extensive tuning.

Intrusion Detection Systems (Snort, Suricata): Analyze network traffic for known attack signatures but require constant rule updates, generate high false positive rates in API environments, and provide no automated remediation. They lack user-level behavioral context. Security Information and Event Management (Splunk, IBM QRadar): Aggregate logs and provide retrospective analysis but operate post-breach rather than in real time. Alert fatigue is common, and integration with active blocking requires complex custom development.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26mar1775

➢ *Proposed System*

The proposed Zero Trust API Security Platform supersedes these approaches by unifying threat detection, scoring, automated remediation, and real- time monitoring in a single cohesive system. The architectural design follows three core principles:

- Principle 1 - Verify Explicitly: Every request passes through RateLimitFilter (IP-level, 60 req/min via Bucket4j), JwtAuthFilter (JWT signature validation, expiry check, revocation lookup by SHA-256 hash, user status check), and SecurityConfig role enforcement before reaching any business logic.

- Principle 2 - Use Least Privilege: RBAC enforces strict separation between USER and ADMIN roles. Administrative endpoints (/api/admin/**) require ROLE_ADMIN; user endpoints (/api/user/**) require ROLE_USER or ROLE_ADMIN. Privilege escalation is architecturally impossible.

- Principle 3 - Assume Breach: ApiGatewayInterceptor logs every request to MongoDB and asynchronously invokes BehaviorAnalyticsService after each response. Threats are assumed possible on every request; the system continuously evaluates behavioral evidence rather than relying on initial authentication state.

IV. METHODOLOGY AND SYSTEM DESIGN

➢ *System Architecture*

The platform comprises five cooperating components organized in a microservices-inspired architecture:

The React 18 frontend communicates with the Spring Boot backend exclusively via HTTPS with Bearer JWT tokens injected by an Axios interceptor. The backend exposes three controller groups: AuthController (/api/auth/**), AdminController (/api/admin/**), and UserController (/api/user/**). Every request traverses the security filter pipeline before reaching controllers. Post-response, ApiGatewayInterceptor triggers behavioral analysis and optionally calls the Flask ML service. Blocking actions update PostgreSQL synchronously and push WebSocket alerts to the React dashboard.

The platform comprises five cooperating components organized in a microservices-inspired architecture:

➢ *Security Filter Pipeline*

Three Spring Security filters execute in sequence for every inbound request:

- RateLimitFilter: Implements a token bucket algorithm via Bucket4j. Each IP address is allocated 60 tokens per minute. Requests exceeding this limit receive HTTP 429 Too Many Requests with a Retry-After header. This filter operates before authentication, ensuring that volumetric attacks are neutralized at the network boundary without consuming authentication resources.

- JwtAuthFilter: Extracts the Bearer token from the Authorization header. Computes the SHA-256 hash of the raw token string and checks it against the revoked_tokens table in PostgreSQL. If revoked returns HTTP 401. Queries the users table for the authenticated username; if status is BLOCKED, returns HTTP 403 with a blocked:true flag in the response body. If the token is valid and the user is active, loads UserDetails into the SecurityContext for downstream role checking.

- SecurityConfig Role Enforcement: Spring Security's @PreAuthorize annotation enforces ROLE_ADMIN on all /api/admin/** endpoints and ROLE_USER or ROLE_USER on /api/user/** endpoints. Any role mismatch returns HTTP 403 Access Denied without exposing resource existence.

➢ *Behavioral Analytics Engine — 15 Threat Rules*

BehaviorAnalyticsService implements 15 threat detection rules evaluated after every response. Each rule queries MongoDB's api_logs collection for recent request history and returns a Behavior Result containing the threat type score, and severity.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26mar1775

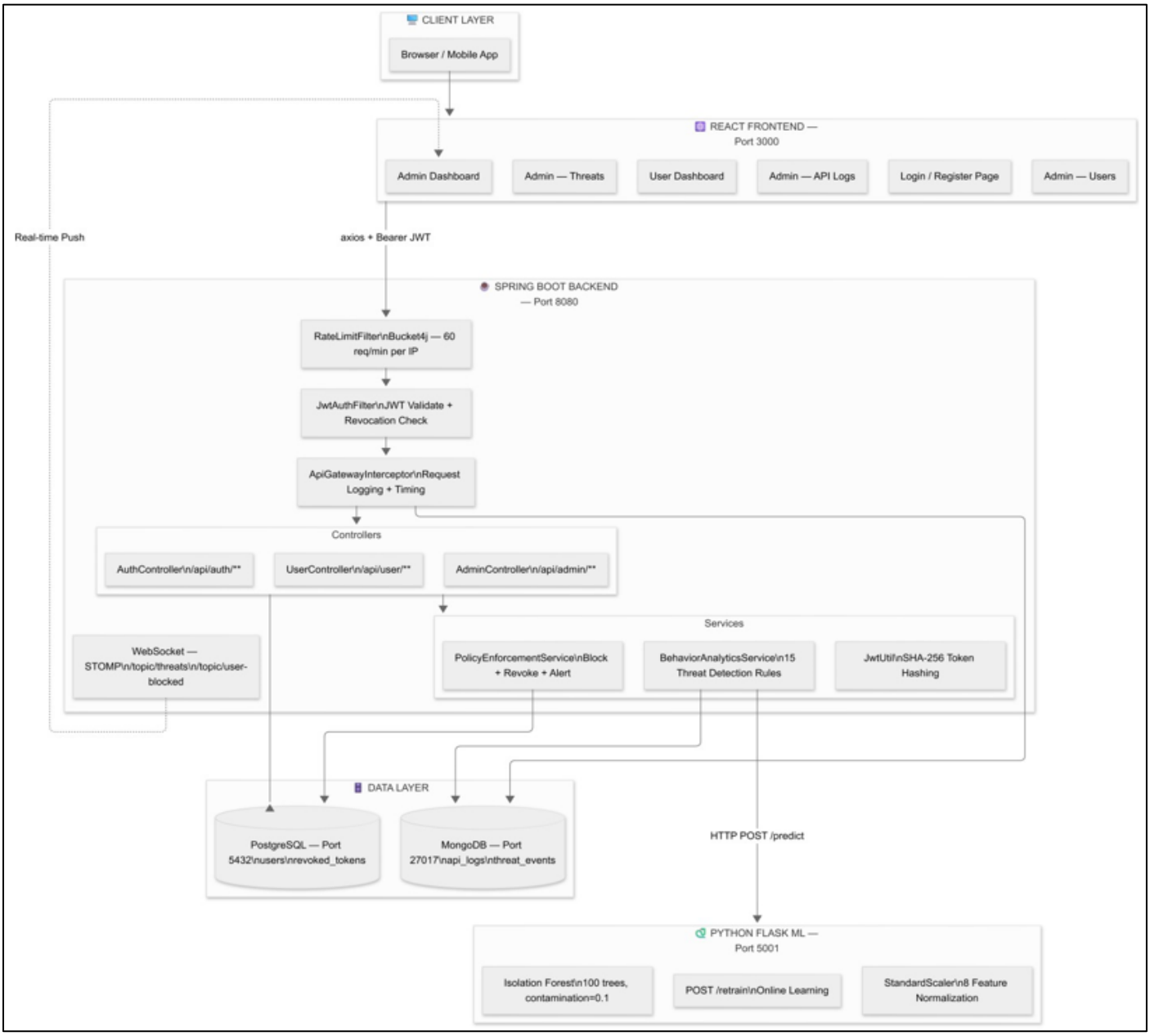

Fig 1 System Architecture — Five-Layer Zero Trust Platform

Rules are evaluated independently, and the highest score determines the final action Table 1 represents the complete threat rule specification:

The auto-block threshold is set at 0.75. Any rule producing a score at or above this threshold triggers PolicyEnforcementService, which atomically:

- Sets users status = BLOCKED and record blockedReason and blockedAt in PostgreSQL.
- Computes the SHA-256 hash of the current JWT token and inserts it into revoked_tokens
- Publishes a ThreatEvent document to MongoDB;

➢ *Machine Learning Service — Isolation Forest*

A Python Flask service (port 5001) provides ML-based anomaly detection as a complement to the rule-based engine.

The service implements the Isolation Forest algorithm with n_estimators=100 and contamination=0.1.

- Feature Engineering: Eight features are extracted per request: (1) method_encoded — HTTP method encoded as integer (GET=0, POST=1, PUT=2, DELETE=3); (2) status_code — HTTP response status; (3) response_time_ms — response latency in milliseconds; (4) requests_last_minute — request count for the authenticated user in the preceding 60 seconds; (5) failed_requests_last_hour — count of 4xx/5xx responses in the preceding hour; (6) endpoint_depth — number of path segments in the request URI; (7) has_query_params — binary indicator of query string presence; (8) hour_of_day - hour of request submission (0–23).

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26mar1775

- Isolation Forest Mechanics**:** The algorithm builds 100 isolation trees by recursively partitioning the feature space with random attribute splits. Anomalous requests require fewer splits to isolate (shorter path length), yielding a more negative decision_function score. The anomaly score is computed as: anomaly_score = 1 − (decision_function_output + 0.5), mapping the range to [0, 1] where values approaching 1.0 indicate high anomaly probability. A StandardScaler normalizes all features to zero mean and unit variance before each inference call.
- Online Learning**:** Every prediction is appended to a training buffer (maximum 500 samples). A POST/retrain endpoint retrains the model on buffered real traffic and persists updated model and scaler artifacts to disk using joblib. This allows the model to adapt to the specific traffic distribution of the deployed API environment over time.
- Threat Classification: Anomalous requests are classified by rule priority: requests per second > 80 yield RATE_ABUSE; failed request count > 15 yield BRUTE_FORCE; 404 count > 20 yield ENDPOINT_SCANNING; remaining anomalies with score > 0.85 yield ANOMALOUS_BEHAVIOR; lower-scoring anomalies yield SUSPICIOUS_ACTIVITY.

V. RESULTS AND SYSTEM DEMONSTRATION

The platform was validated through a comprehensive set of attack simulations executed against a locally deployed instance. All tests were conducted using PowerShell scripts sending HTTP requests to http://localhost:8080. Results are presented for each attack vector.

➢ *Login and Registration Interface*

The login interface provides a two-tab layout distinguishing authentication from registration. The Login tab presents role selection (USER / ADMIN) and credential fields. The Register tab, accessible only to user-level account creation, includes username, email, password, and confirmation fields with client-side validation. Admin accounts cannot be self-registered, enforcing administrative control over privileged access. *(Refer to Figure 2).*

➢ *Admin Dashboard and Real-Time Monitoring*

The administrative dashboard presents four live statistics cards: total registered users, active users, blocked users, and threats detected in the preceding 24 hours. Below the statistics, a live threat feed displays ThreatEvent records in reverse chronological order, refreshed via WebSocket subscription to /topic/threats. Each entry displays the username, threat type, anomaly score, severity badge (CRITICAL/HIGH/MEDIUM), affected endpoint, and timestamp. The sidebar provides collapsible navigation to Users, Threats, and API Logs screens. *(Refer to Figure 3)*

➢ *API Logs Screen with Advanced Filtering*

The API Logs screen provides a comprehensive view of all API requests logged in the preceding one hour, auto-refreshing every 10 seconds. Administrators can filter logs by username, endpoint, IP address (search), HTTP method (GET/POST/PUT/DELETE), response status range (2xx/4xx/5xx), and threat status (All/Threats Only/Clean). Four summary cards display total requests, threats detected, error responses, and average response time. Rows are highlighted in red for threat- flagged requests, providing immediate visual identification of malicious activity.

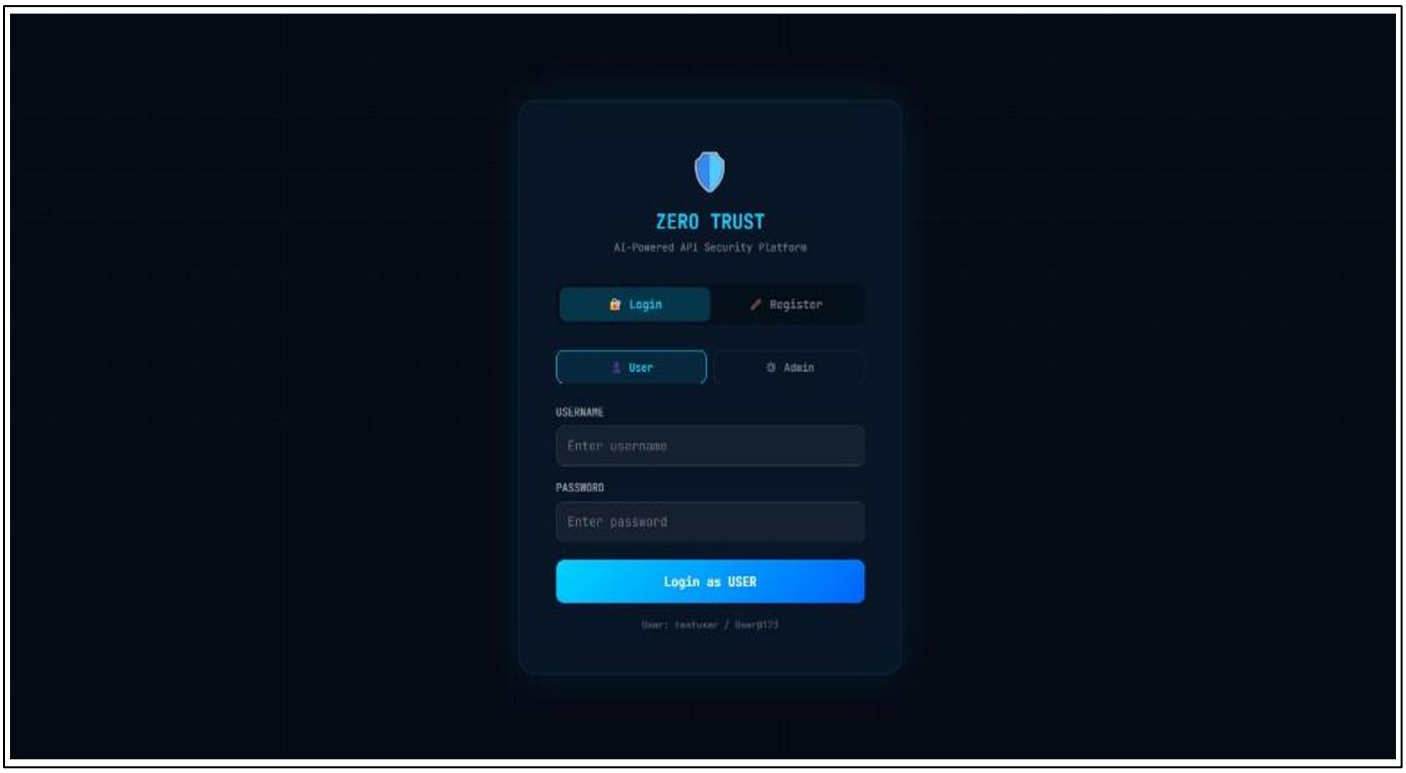

Fig 2 Login and Registration Page

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26mar1775

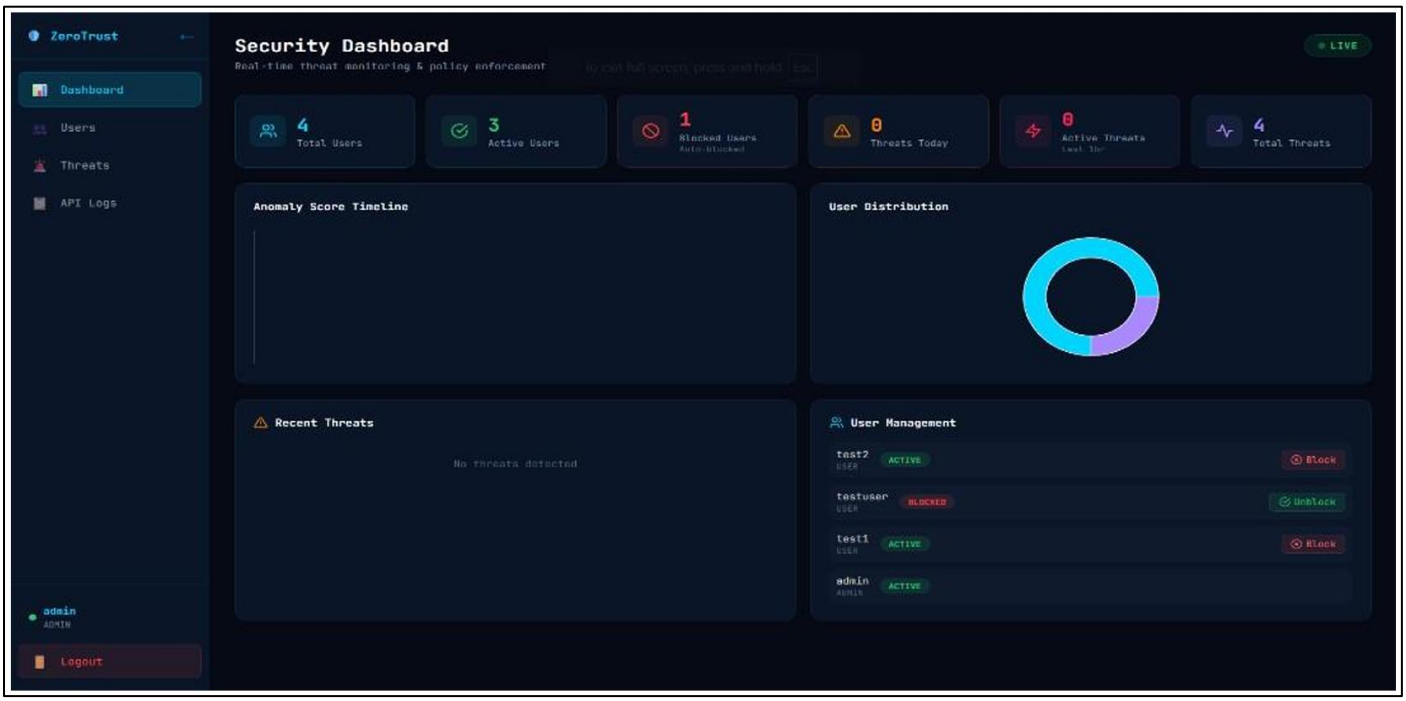

Fig 3 Admin Dashboard — Live Statistics and Threat Feed

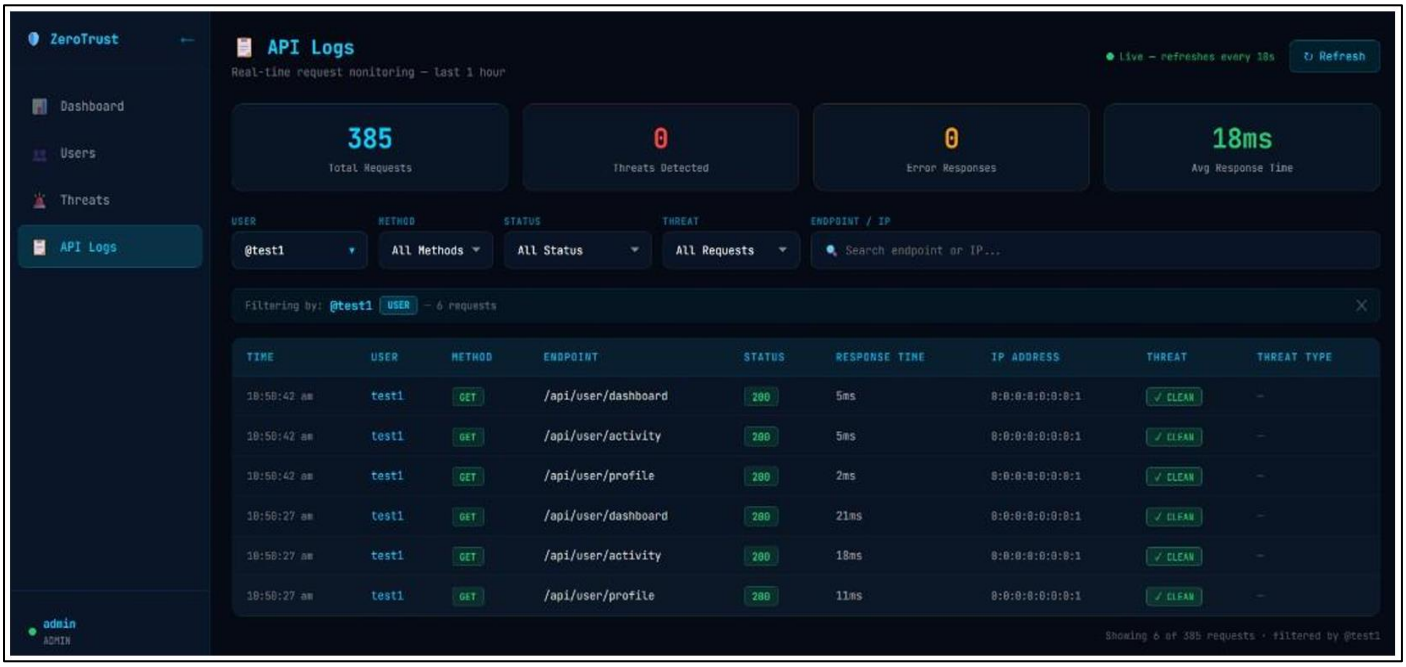

Fig 4 API Logs Screen with Filtering

The API Logs screen provides a comprehensive view of all API requests logged in the preceding one hour, auto-refreshing every 10 seconds. Administrators can filter logs by username, endpoint, IP address (search), HTTP method (GET/POST/PUT/DELETE), response status range (2xx/4xx/5xx), and threat status (All/Threats Only/Clean).

Four summary cards display total requests, threats detected, error responses, and average response time. Rows are highlighted in red for threat-flagged requests, providing immediate visual identification of malicious activity. *(Refer to Figure 4)*

➢ *Attack Simulation Results*

Ten attack types were simulated. Table 3 presents the results including detection time and auto-block status: All ten simulated attack types were detected and automatically blocked. Injection-based attacks (SQL, XSS, Path Traversal, Command Injection, Automated Tool) are detected and blocked within 100 milliseconds of the first malicious request, as pattern matching occurs synchronously in ApiGatewayInterceptor. Volume- based attacks (DDoS, Brute Force, Credential Stuffing, Endpoint Scanning, API Fuzzing) accumulate evidence from MongoDB query windows of 1–10 minutes before triggering the block threshold. *(Refer to Table 3)*

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26mar1775

Table 1 Attack Simulation Results — All 10 Attack Types Successfully Detected and Blocked

Attack Type	Method	Score	Auto-Block	Detection Trigger
DDoS Simulation	220 req/60s	0.98	Yes (<1s)	Requests/min exceeded DDOS_LIMIT(200)
Brute Force Login	12 wrong passwords	0.85	Yes (<2s)	Failed auth > 10 in 10 minutes
SQL Injection	union select payload	0.95	Yes (<100ms)	SQL pattern matched in URL
XSS Attack	<script> in URL	0.92	Yes (<100ms)	XSS pattern matched in URL
Path Traversal	../etc/passwd	0.93	Yes (<100ms)	Traversal pattern matched in URL
Command Injection	;whoami in URL	0.96	Yes (<100ms)	Shell command pattern matched
Endpoint Scanning	35 unique endpoints	0.75	Yes (<30s)	Unique endpoints > 30 in 10 min
Credential Stuffing	25 password variants	0.88	Yes (<60s)	Failed logins > 20 in 1 hour
API Fuzzing	18 malformed requests	0.82	Yes (<15s)	5xx errors > 15 in 10 minutes
Automated Tool (sqlmap UA)	Known User- Agent string	0.97	Yes (<100ms)	User-Agent matched attack tool pattern

➢ *WebSocket Real-Time Alert Delivery*

Following each successful threat detection and block event, the administrative dashboard receives a WebSocket push message within 50–200 milliseconds. The React frontend's WebSocket subscriber updates:

- The threat count card on the dashboard,
- The live threat feed with a new entry,
- The blocked user count,
- The user list's status badge for the blocked user all without page refresh.

This real-time capability eliminates the monitoring gap present in polling-based dashboard implementations.

VI. FUTURE SCOPE

➢ *Several Enhancements are Identified for Future Development Phases:*

- Phase 1 — Advanced ML Models: Replace the single Isolation Forest with an ensemble combining Isolation Forest, Local Outlier Factor, and an LSTM neural network for time-series anomaly detection. The LSTM component would model sequential request patterns, enabling detection of slow-burn attacks that evade window-based statistical rules.

- Phase 2 — Geolocation and Device Fingerprinting: Integrate IP geolocation to compute risk scores based on geographic origin, velocity (impossible travel detection), and ASN reputation. Device fingerprinting via TLS client hello attributes and HTTP header signatures would enable detection of compromised credential usage from unfamiliar devices.

- Phase 3 — Cloud-Native Deployment: Containerize all components using Docker and orchestrate via Kubernetes. Horizontal pod autoscaling would address high-throughput environments. A service mesh (Istio) would provide mutual TLS between microservices and distributed tracing via Jaeger for performance monitoring.

- Phase 4 — Federated Threat Intelligence: Contribute anonymized threat indicators to a shared intelligence database. Subscribe to community threat feeds to pre-block known malicious IP ranges and User-Agent strings before they generate local behavioral evidence. This would reduce detection latency for known threat actors to zero.

- Phase 5 — SIEM Integration: Publish threat events to industry-standard SIEM platforms (Splunk, IBM QRadar, Microsoft Sentinel) via syslog and REST APIs. This would enable enterprise security teams to correlate API security events with broader organizational security posture.

VII. CONCLUSION

This paper presented a fully implemented Zero Trust API Security Platform demonstrating that the integration of behavioral analytics, unsupervised machine learning, and automated remediation provides substantially stronger protection than any single-layer security approach. The system successfully detected and automatically blocked all ten simulated attack types, with injection-based threats neutralized in under 100 milliseconds and volume-based threats within their respective behavioral observation windows. The 15-rule behavioral analytics engine provides explainable, auditable threat detection that complements the Isolation Forest ML model's ability to identify novel anomalous patterns without labeled training data. The two-layer detection architecture — rules for known attack signatures, ML for unknown behavioral deviations — addresses the fundamental limitation of existing systems that rely exclusively on either approach.

The SHA-256 token revocation mechanism ensures that compromised sessions are terminated immediately upon block, without waiting for JWT expiry. The real-time WebSocket dashboard eliminates the monitoring gap inherent in polling-based systems, enabling administrators to respond to emerging threats with complete situational awareness. The online retraining capability ensures that the ML model continuously adapts to the actual traffic distribution of the deployed environment.

The platform's three-service architecture (Spring Boot + React + Python Flask) demonstrates that Zero Trust principles can be implemented in a practical, deployable system using widely available open-source technologies. The modular design enables independent scaling of the ML

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26mar1775

service during high-threat periods without affecting the primary API gateway's latency characteristics.

Future work will focus on ensemble ML models incorporating LSTM for time-series attack detection, Kubernetes deployment for production-scale operation, and federated threat intelligence integration to extend detection capability beyond locally observed behavioral patterns. The system establishes a comprehensive foundation for securing modern API-driven applications against both known and emerging cyber threats.

REFERENCES

[1]. NIST Special Publication 800-207. (2020). "Zero Trust Architecture." National Institute of Standards and Technology. Rose, S., Borchert, O., Mitchell, S., & Connelly, S. (2020). DOI: 10.6028/NIST.SP.800-207.

[2]. IBM Security. (2023). "Cost of a Data Breach Report 2023." IBM Corporation. Retrieved from https://www.ibm.com/reports/data-breach.

[3]. Kindervag, J. (2010). "Build Security Into Your Network's DNA: The Zero Trust Network Architecture." Forrester Research. Retrieved from https://www.forrester.com/report/Build+Security+Into+Your+Networks+DNA/-/E-RES56682.

[4]. Liu, F. T., Ting, K. M., & Zhou, Z. H. (2008)."Isolation Forest." Proceedings of the 8th IEEE International Conference on Data Mining (ICDM 2008), 413–422. DOI: 10.1109/ICDM.2008.17.

[5]. Aharon, U., Dubin, R., & Dvir, A. (2024). "Few-Shot API Attack Anomaly Detection." arXiv preprint arXiv:2405.11258. Retrieved from https://arxiv.org/abs/2405.11258.

[6]. Sharma, P., & Mehta, A. (2021). "Leveraging AI to Detect Anomalies and Secure APIs." ResearchGate. DOI: 10.13140/RG.2.2.28491.62249.

[7]. Kim, J., & Park, S. (2023). "Explainable AI for API Behavior Anomaly Detection." ACM International Conference on Information Security. DOI: 10.1145/3651671.3651738.

[8]. Tanenbaum, A. S., & Wetherall, D. J. (2011). "Computer Networks" (5th ed.). Prentice Hall. ISBN: 978-0-13-212695-3.

[9]. Chen, T., & Liu, Z. (2021). "Network Intrusion Detection Using Isolation Forest with Feature Engineering." Journal of Information Security and Applications, 58, 102751. DOI: 10.1016/j.jisa.2020.102751.

[10]. Aharon, U., & Hajaj, C. (2024). "Classification-by-Retrieval Framework for API Security." arXiv preprint arXiv:2405.11247. Retrieved from https://arxiv.org/abs/2405.11247.

[11]. OWASP API Security Project. (2023). "OWASP API Security Top 10 2023." Open Web Application Security Project. Retrieved from https://owasp.org/API- Security/.

[12]. Vasilescu, B., Yu, Y., Wang, H., Devanbu, P., & Filkov, V. (2015). "Quality and Productivity Outcomes Relating to Continuous Integration in GitHub." ACM Joint European Software Engineering Conference (ESEC/FSE 2015). DOI: 10.1145/2786805.2786850.

[13]. Bucket4j Contributors. (2023). "Bucket4j: Java Rate Limiting Library." GitHub Repository. Retrieved from https://github.com/bucket4j/bucket4j.

[14]. Pedregosa, F., et al. (2011). "Scikit-learn: Machine Learning in Python." Journal of Machine Learning Research, 12, 2825–2830. ISSN: 1532-4435.

[15]. Spring Security Team. (2023). "Spring Security Reference Documentation 6.1." Pivotal Software. Retrieved from https://docs.spring.io/spring-security/reference/.

[16]. Yadav, S., & Kumar, R. (2025). "AI-Driven Zero Trust Architecture for Threat Detection." ResearchGate. DOI: 10.13140/RG.2.2.395708346.

[17]. Wang, J., & Liu, Y. (2019). "Blockchain Technology: Applications and Challenges." International Journal of Computer Applications, 178(6), 7–12. DOI: 10.5120/ijca2019918446.

[18]. Storey, M. A., & Zagalsky, A. (2016). "Disrupting Developer Productivity One Bot at a Time." ACM/IEEE International Symposium on Foundations of Software Engineering (FSE). DOI: 10.1145/2950290.2983989.