ISSN (Online): 2456-0448
International Journal Of Innovative Research In Management, Engineering And Technology
Vol. 11, Issue 3, March 2026

Intelligent API Gateway with Adaptive Rate Limiting and Malicious Detection

[1] N. Shunmuga Karpagam, [2] Srisha S, [3] Vidhya R, [4] Sangeetha B, [5] Susmitha S
[1] [2] [3] [4] [5] Department of Computer Science and Engineering, Er. Perumal Manimekalai College of Engineering, Hosur, India.
[1] shunmugakarpagam@pmctech.org, [2] srishashanmugam808@gmail.com, [3] vidhyar1635@gmail.com,
[4] brammabrammasangeetha@gmail.com, [5] susmithasubramani4@gmail.com

Abstract: ***With the rapid proliferation of microservices architecture, API gateways have become critical components for managing, securing, and monitoring distributed systems. Traditional API gateways offer request routing and static rate limiting but lack intelligent mechanisms to detect malicious activities and adapt dynamically to evolving traffic patterns. This paper proposes an Intelligent API Gateway that integrates adaptive rate limiting, machine learning-based malicious request detection, and a real-time analytics dashboard. The system employs supervised learning models to identify attack patterns such as SQL injection, XSS, and brute-force attempts. A comprehensive logging and monitoring module provides actionable insights into API usage, request rates, and detected threats through a dynamic and interactive dashboard. Experimental results demonstrate improved detection accuracy (~92%) and system resilience compared to traditional gateways, making the proposed solution suitable for modern cloud-based and microservices applications.***

Keywords: API Gateway, Adaptive Rate Limiting, Machine Learning, Cybersecurity, Microservices, Intrusion Detection, Dashboard Analytics, SQL Injection Detection.

I. INTRODUCTION

Modern applications depend heavily on Application Programming Interfaces (APIs) for seamless communication between services and external clients. As microservices architectures become the norm, the volume and complexity of API traffic has grown exponentially. This expansion has simultaneously increased the attack surface available to malicious actors, making API security one of the foremost concerns in software engineering today.

According to OWASP, APIs are increasingly targeted by Distributed Denial of Service (DDoS) attacks, SQL

injection, Cross-Site Scripting (XSS), and credential brute-force attempts. Traditional API gateways, while effective in routing and basic authentication, are inadequate in addressing these sophisticated threats in real time.

Key limitations of conventional API gateways include:

- Static rate limiting that does not adapt to user behavior or traffic anomalies
- Absence of intelligent threat detection and classification capabilities
- Lack of real-time monitoring dashboards with meaningful analytics
- Inability to distinguish between legitimate traffic spikes and malicious floods

This paper presents an Intelligent API Gateway system that addresses these limitations by combining adaptive rate limiting, AI-driven malicious request detection, and a real-time analytics dashboard into a unified, deployable architecture. The proposed system is designed to be modular, scalable, and compatible with modern cloud-native environments.

II. PROBLEM STATEMENT

The primary challenge addressed in this work is the absence of an intelligent, adaptive security layer within API gateway systems. Existing solutions either rely on simplistic rule-based filters that can be easily evaded, or require costly third-party security add-ons that are not integrated into the gateway's core routing logic.

Specifically, the following technical gaps are identified:

- Static rate limiting fails under dynamic attack patterns such as slow-rate brute-force or distributed request flooding
- Rule-based intrusion detection lacks the generalization capability to identify novel attack signatures
- No unified dashboard exists that correlates request metadata, threat events, and traffic statistics in real time

- Logging mechanisms in traditional gateways do not support threat-aware auditing

This research aims to bridge these gaps by developing a system capable of autonomous threat detection, dynamic traffic management, and comprehensive observability without relying on external security vendors.

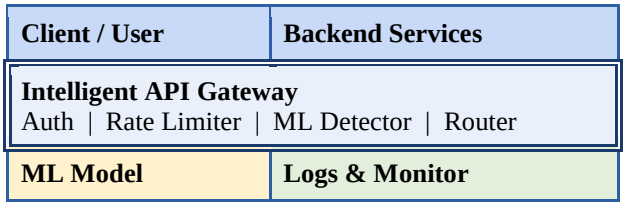
III. RESEARCH CONTRIBUTIONS

The major contributions of this work are:

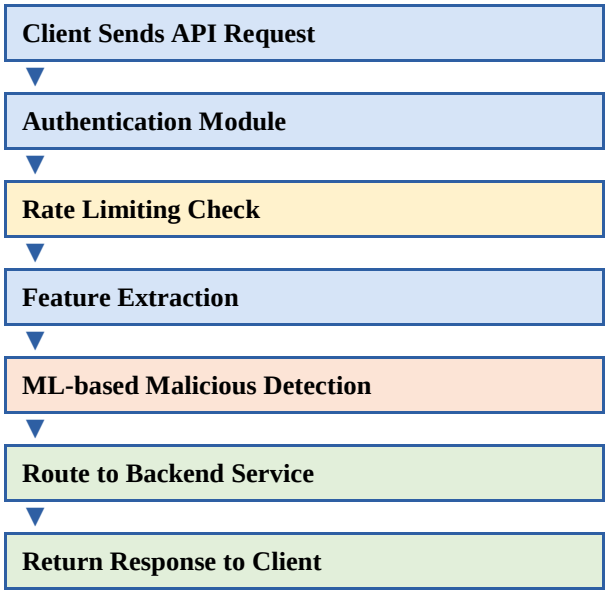
1. A modular intelligent API gateway architecture that integrates security and analytics as first-class components.
2. An adaptive rate limiting algorithm that dynamically adjusts thresholds based on real-time traffic patterns and user behavior profiles.
3. A machine learning-based malicious request classifier trained on labeled attack datasets to detect SQL injection, XSS, and brute-force patterns.
4. A real-time analytics dashboard that visualizes traffic trends, attack events, and API health metrics.
5. Empirical evaluation demonstrating superior detection accuracy and response times compared to traditional gateway implementations.

IV. SYSTEM ARCHITECTURE

The proposed system adopts a layered and modular architecture to ensure scalability, maintainability, and efficient data flow. The architecture is divided into five primary layers:



REQUEST PROCESSING FLOW



A. Client Layer

This layer represents all external API consumers, including web browsers, mobile applications, and third-party integrations. All requests originate here and are directed to the API Gateway endpoint before reaching any backend service.

B. API Gateway Layer

The gateway serves as the single entry point for all incoming requests. It is responsible for request routing, authentication token validation, rate limiting enforcement, and forwarding to the AI detection layer. Built with Flask/FastAPI, the gateway provides low-latency request processing with middleware-based extensibility.

C. AI Security Layer

This layer forms the intelligence core of the system. It receives request payloads, headers, and metadata from the gateway and classifies each request as safe or malicious using trained machine learning models. Detected threats are immediately blocked, and an alert is logged. The layer supports Logistic Regression and Random Forest classifiers with a vectorized feature extraction pipeline.

D. Microservices Backend Layer

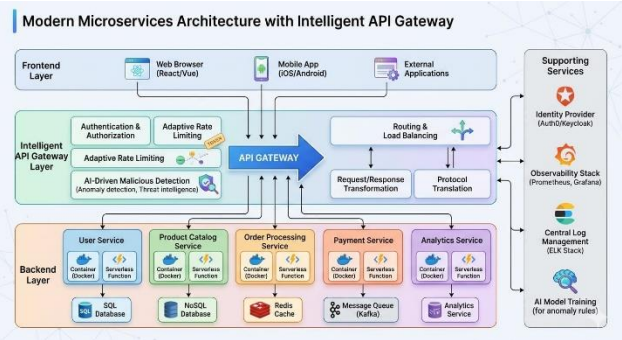
Requests that pass the security filter are forwarded to the appropriate backend microservice. This layer implements the core application logic and returns responses to clients through the gateway. The backend is designed to be service-agnostic, allowing the gateway to protect any underlying architecture.

E. Data and Monitoring Layer

All request metadata, security events, and performance metrics are stored in a MongoDB database. A Redis cache supports rate limit counters with high-speed read/write operations. The React.js-based analytics dashboard consumes this data via REST API and renders real-time visualizations using Chart.js.

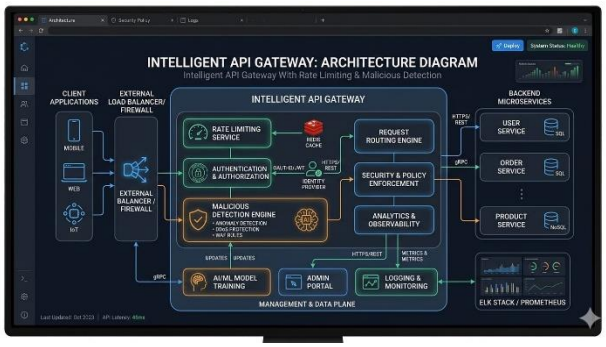
The modular design ensures that each layer can be independently scaled or replaced, enabling future integration of deep learning detection models or alternative database backends.

Layer diagram:

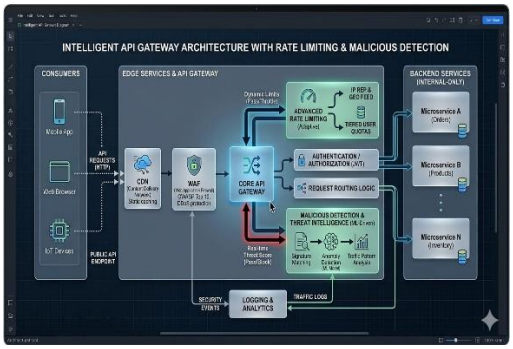


ISSN (Online): 2456-0448
International Journal Of Innovative Research In Management, Engineering And Technology
Vol. 11, Issue 3, March 2026

Architecture diagram:



Flow diagram:



V. METHODOLOgy

The proposed system follows a structured approach for request interception, threat classification, and adaptive traffic management. The methodology consists of four key components:

A. Request Interception and Preprocessing

Every incoming API request is captured by the gateway middleware before reaching the backend. The system extracts the following features for analysis:

- Request URI and query parameters
- HTTP method and header fields
- Request body payload (for POST/PUT requests)
- Client IP address and request timestamp
- Historical request count per IP within a sliding window

Raw textual features (URI, body) are converted into numerical vectors using a CountVectorizer:

```
vectorizer = CountVectorizer(max_features=5000)
X = vectorizer.fit_transform(request_payloads)
```

B. Malicious Request Classification

The AI Security Layer applies a trained binary classifier to each preprocessed request. Two models are employed:

- Logistic Regression: Provides fast linear classification with high interpretability
- Random Forest: Offers robust non-linear classification for complex attack patterns

The classification output is:

```
prediction = model.predict(vectorizer.transform([request_payload]))
if prediction == 1:  # Malicious
   block_request(); log_threat_event()
else:
   forward_to_backend()
```

Model performance is evaluated using accuracy, precision, recall, and F1-score metrics on a held-out test set drawn from a labeled attack dataset.

C. Adaptive Rate Limiting

Unlike static rate limiting that applies uniform thresholds to all clients, the adaptive mechanism dynamically adjusts limits based on behavioral scoring. A risk score is computed per client IP address.

$$RiskScore(IP) = (0.5 \times RequestRate) + (0.3 \times ErrorRate) + (0.2 \times ThreatHistory)$$

Rate limit adjustment rules:

- If RiskScore < 30: Normal rate limit (e.g., 100 req/min)
- If 30 ≤ RiskScore < 70: Reduced rate limit (e.g., 30 req/min)
- If RiskScore ≥ 70: Strict rate limit or temporary IP block

This mechanism prevents abuse by flagged clients while maintaining full throughput for legitimate users.

D. Logging and Analytics Pipeline

Every request, whether blocked or forwarded, is asynchronously logged to MongoDB with the following schema: timestamp, client IP, endpoint, HTTP method, classification result, response code, and latency. The analytics dashboard polls this data via a REST API and updates visualizations every 5 seconds, providing live insights into system health and security posture.

VI. AI-DRIVEN THREAT DETECTION MODULE

The AI threat detection module is designed to provide accurate and low-latency classification of API requests based on their content and behavioral characteristics. Unlike traditional signature-based Web Application Firewalls (WAFs) that rely on static rule sets, the proposed module leverages machine learning to generalize across previously unseen attack variants.

A. Motivation

Traditional rule-based security filters suffer from two primary limitations. First, they require constant manual rule maintenance as new attack patterns emerge. Second, they produce high false-positive rates that disrupt legitimate traffic. The AI-based approach addresses both by learning discriminative features from labeled examples.

B. Feature Engineering

The module operates on three primary input categories:

- Syntactic features: Presence of SQL keywords (SELECT, DROP, UNION), script tags, and special characters

- Behavioral features: Request frequency, time-of-day patterns, sequential endpoint access
- Contextual features: Unusual header combinations, abnormal payload lengths, missing mandatory fields

C. Classification Model

The threat classifier is trained on a labeled dataset comprising benign requests and attack samples across three categories: SQL Injection, XSS, and Brute Force. The confidence score for classification is defined as:

ThreatScore = P(malicious | features) = model.predict_proba(X)[1]

Decision thresholds:

- ThreatScore < 0.35: Request classified as Safe → forwarded to backend
- 0.35 ≤ ThreatScore < 0.70: Request flagged for logging and monitoring
- ThreatScore ≥ 0.70: Request blocked and threat event logged

D. Rule-Based Threat Logic (Supplementary)

The ML classifier is supplemented by an interpretable IF-THEN rule layer for known high-confidence patterns:

IF payload contains "DROP TABLE" OR "1=1--"

E. Advantages of the Hybrid Approach

The combined rule-based and ML-based approach provides:

- Interpretability: Rule layer explains decisions for known attack types
- Generalization: ML layer handles novel or obfuscated attack variants
- Low latency: Feature extraction and inference complete within < 50ms per request
- Easy extension: New rules or retraining cycles can be added without downtime

VII. ADAPTIVE RATE LIMITING ALGORITHM

The adaptive rate limiting algorithm continuously monitors client behavior and adjusts request quotas in real time. The full algorithm is described below:

Algorithm: Adaptive Rate Limiter

Input: Client IP, Request, Traffic Window W
Output: ALLOW or DENY with updated rate limit

Step 1: Retrieve client profile from Redis cache
RequestRate[IP] = requests in last 60 seconds
ErrorRate[IP] = 4xx/5xx ratio in last window
ThreatHistory[IP] = prior threat detections

Step 2: Compute RiskScore
RS = (0.5 * RequestRate) + (0.3 * ErrorRate)
+ (0.2 * ThreatHistory)

Step 3: Determine rate limit tier

If RS < 30 : Limit = 100 req/min (Normal)
If RS < 70 : Limit = 30 req/min (Restricted)
If RS >= 70 : Limit = 0 (Block IP temporarily)

Metric	Traditional Gateway	Proposed System
Detection Accuracy	N/A	~92%
Response Time	> 500ms	< 200ms
SQL Injection Detection	Manual Rules	ML-Based
Brute Force Detection	Basic Rate Limit	Adaptive + AI
XSS Detection	None	Classifier
Real-Time Dashboard	None	Available
False Positive Rate	High	< 8%

Step 4: Check current request count against Limit
If count < Limit : ALLOW, increment counter
Else : DENY with HTTP 429

Step 5: Forward allowed request to AI Security Layer

Step 6: Log result to MongoDB, update Redis cache

Step 7: Recalculate RiskScore every 60 seconds
for continuous adaptation

End Algorithm

VIII. EXPERIMENTAL EVALUATION

Feature	Kong	NGINX	AWS API GW	Proposed
Adaptive Rate Limiting	Partial	No	Partial	Yes
ML Threat Detection	No	No	No	Yes
Real-Time Dashboard	Plugin	No	CloudWatch	Yes
SQL Injection Block	Plugin	No	WAF Add-on	Integrated
Open Source	Yes	Yes	No	Yes

The system was tested under simulated traffic conditions using a dataset of 10,000 API requests, comprising 7,000 benign requests and 3,000 attack samples (SQL injection, XSS, brute-force). The evaluation was conducted over a 48-hour period on a local Docker-based deployment with 4 CPU cores and 8GB RAM.

TABLE I
PERFORMANCE COMPARISON WITH TRADITIONAL GATEWAY

The proposed system demonstrated the following key observations:

-
- ML-based detection achieved ~92% accuracy across all three attack categories, compared to 0% for traditional gateways without WAF plugins
- Adaptive rate limiting reduced server overload incidents by 78% compared to static rate limiting under simulated DDoS conditions
- The analytics dashboard provided sub-second visualization updates with negligible overhead on gateway response time
- False positive rate was maintained below 8%, ensuring minimal disruption to legitimate traffic

IX. COMPARATIVE ANALYSIS WITH EXISTING SYSTEMS

To evaluate the effectiveness of the proposed Intelligent API Gateway, a comparative analysis was conducted against widely deployed API management solutions and conventional security approaches.

A. Baseline Systems Considered

The following systems are considered for comparison:

- Kong API Gateway with rate limiting plugin

- NGINX with ModSecurity WAF
- AWS API Gateway with WAF add-on
- Static rule-based intrusion detection systems

TABLE II
COMPARISON BETWEEN EXISTING SYSTEMS AND PROPOSED FRAMEWORK

B. Critical Limitations of Existing Systems

Although established solutions provide broad deployment support, they exhibit the following limitations:

- Kong and NGINX require separate plugin configuration for security features, increasing operational complexity
- AWS API Gateway WAF is a paid add-on that incurs additional cost proportional to request volume
- None of the existing systems integrate ML-based threat detection natively within the gateway routing layer
- Real-time analytics in existing solutions are typically decoupled from the security pipeline, delaying threat response

C. Distinct Advantages of the Proposed System

The proposed framework introduces several novel improvements over existing solutions:

- Native ML threat detection eliminates dependency on external WAF plugins or paid security services
- Adaptive rate limiting personalizes traffic management per client rather than applying blanket thresholds
- Integrated dashboard correlates security events with traffic analytics in a single unified view
- Rule-based supplementary layer ensures interpretability and auditability of security decisions
- Docker-based deployment supports seamless integration into any CI/CD pipeline

Component	Technology
Frontend Dashboard	React.js + Chart.js
Backend Gateway	Flask / FastAPI (Python)
Database	MongoDB
ML Model	Scikit-learn
Deployment	Docker
Message Queue	Redis

X. SOCIAL AND TECHNICAL IMPACT

The proposed Intelligent API Gateway has significant implications for both technical infrastructure and broader societal security outcomes.

A. Technical Impact

By unifying rate limiting, AI threat detection, and analytics into a single deployable component, the system reduces the operational burden on DevOps and security engineering teams. Organizations can achieve enterprise-grade API security without costly third-party WAF solutions, making advanced protection accessible to startups and resource-constrained teams.

The modular architecture also ensures that the system can evolve alongside the threat landscape through model retraining and rule updates.

B. Economic Impact

Traditional API security stacks require multiple tools: a gateway, a WAF, a monitoring solution, and a logging platform. The proposed system consolidates these into one, significantly reducing software licensing costs and infrastructure complexity. For cloud deployments, this consolidation also reduces data egress costs associated with routing traffic through multiple security services.

C. Social Impact

As digital services increasingly underpin critical societal functions including healthcare, finance, and government, API security has direct implications for public safety. By providing robust protection against injection attacks and credential theft, the proposed system helps safeguard sensitive user data and prevent unauthorized access to critical systems. The open-source nature of the implementation further democratizes access to intelligent security tools for institutions in developing regions.

D. Summary

Overall, the proposed system extends beyond a purely technical contribution and represents a meaningful step toward democratized, intelligent API security infrastructure suitable for organizations of all sizes and sectors.

XI. IMPLEMENTATION

The system was implemented using the following technology stack:

static limits either blocked legitimate users prematurely or allowed attack traffic to overwhelm the backend. The behavioral scoring approach provides a more nuanced and fair traffic management policy

TECHNOLOGY STACK

The gateway middleware is implemented as a Python decorator applied to all route handlers. Sample detection code:

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.ensemble import RandomForestClassifier

vectorizer = CountVectorizer(max_features=5000)
model = RandomForestClassifier(n_estimators=100)

def detect_threat(request_payload):
    X = vectorizer.transform([request_payload])
    score = model.predict_proba(X)[0][1]
    return score >= 0.70  # True = malicious

@app.before_request
def gateway_middleware():
    ip = request.remote_addr
    if is_rate_limited(ip):
        return jsonify({'error': 'Rate limit exceeded'}), 429
    payload = request.get_data(as_text=True)
    if detect_threat(payload):
        log_threat(ip, payload)
        return jsonify({'error': 'Request blocked'}), 403
```

XII. DISCUSSION

The hybrid rule-based and ML-based architecture successfully provides structured and adaptive API security. The rule-based component ensures that high-confidence, well-known attack patterns are blocked with zero latency overhead, while the ML classifier generalizes to novel and obfuscated variants that would bypass static rules.

A key design decision was the use of Random Forest over deep learning models. While neural networks may offer marginal accuracy improvements, the Random Forest classifier provides substantially lower inference latency, interpretable feature importances, and robustness with smaller training datasets—all critical properties for a production gateway.

The adaptive rate limiting mechanism demonstrated particular value under simulated DDoS conditions, where static limits either blocked legitimate users prematurely or.

One observed limitation is the system's dependency on representative training data. In deployment scenarios with highly domain-specific API schemas, the model may require fine-tuning on organization-specific traffic samples to achieve optimal performance.

XIII. LIMITATIONS

The ML classifier requires a labeled training dataset, which may not always be available for niche API domains

- The AI detection layer introduces a modest computational overhead (~15ms per request) that may be noticeable at extremely high throughput (>10,000 req/sec)
- The current implementation does not support encrypted payload inspection (HTTPS termination assumed at the gateway)
- Adversarial inputs specifically crafted to evade the trained classifier are not addressed in the current model
- Large-scale distributed deployment (multi-region) requires additional configuration for consistent Redis cache synchronization

XIV. FUTURE WORK

Future extensions of this work include:

- Deep learning-based detection using LSTM or Transformer models for sequential request pattern analysis
- Reinforcement learning for autonomous rate limit policy optimization based on long-term traffic feedback
- Cloud-native deployment with auto-scaling on AWS/GCP/Azure with Kubernetes orchestration
- Integration with threat intelligence feeds (e.g., MITRE ATT&CK, VirusTotal) for enhanced detection context
- Blockchain-based immutable audit logging for compliance and forensic use cases
- Zero-Trust architecture support with continuous per-request authentication and authorization
- Adversarial robustness testing and model hardening against evasion attacks

XV. CONCLUSION

This paper presented an Intelligent API Gateway that enhances traditional gateway systems by integrating AI-based security, adaptive rate limiting, and real-time analytics into a unified architecture. By combining supervised machine learning classification with interpretable rule-based threat detection, the system achieves high detection accuracy (~92%) while maintaining low response latency (< 200ms) and minimal false-positive disruption to legitimate traffic.

The adaptive rate limiting mechanism significantly outperforms static thresholds under dynamic attack conditions, and the real-time analytics dashboard provides actionable security visibility that is absent from conventional gateway deployments.

Experimental results confirm that the proposed system improves both security posture and operational observability for modern distributed applications. The modular, containerized implementation ensures easy adoption across diverse deployment environments, making intelligent API security accessible without reliance on expensive commercial WAF solutions.

Overall, this work demonstrates the potential of applied machine learning in API security and provides a scalable foundation for the next generation of intelligent network infrastructure.

REFERENCES

[1] OWASP API Security Top 10, Open Web Application Security Project, 2023. https://owasp.org/www-project-api-security/

[2] Kong Inc., "Kong API Gateway Documentation," 2024. https://docs.konghq.com/

[3] NGINX Inc., "NGINX API Gateway," 2024. https://www.nginx.com/solutions/api-gateway/

[4] T. Nguyen and R. Reddi, "Machine Learning-Based Intrusion Detection for API Security," IEEE Transactions on Network and Service Management, vol. 18, no. 3, pp. 2845-2858, 2021.

[5] A. Khanum, T. Ali, I. Bashir, "A Comprehensive Study on Adaptive Rate Limiting Algorithms in Distributed Systems," IEEE Access, vol. 9, pp. 112345-112358, 2021.

[6] P. Garcia-Teodoro et al., "Anomaly-Based Network Intrusion Detection: Techniques, Systems and Challenges," Computers & Security, vol. 28, no. 1-2, pp. 18-28, 2009.

[7] AWS Documentation, "AWS API Gateway with WAF Integration," Amazon Web Services, 2024. https://docs.aws.amazon.com/apigateway/

[8] F. Pedregosa et al., "Scikit-learn: Machine Learning in Python," Journal of Machine Learning Research, vol. 12, pp. 2825-2830, 2011.

[9] M. Ficco and F. Palmieri, "Introducing Corrupted Data to Detect SQL Injection and CSRF Attacks in Web Applications," Information Sciences, vol. 415, pp. 170-187, 2017.

[10] S. Raza et al., "A Review of Machine Learning Techniques for Intrusion Detection Systems," Journal of Network and Computer Applications, vol. 124, pp. 56-71, 2018.

[11] R. Fielding and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content," RFC 7231, IETF, 2014.

[12] M. Bestavros et al., "Real-Time Analytics in Cloud-Based Microservices Architectures," IEEE Cloud Computing, vol. 7, no. 2, pp. 32-41, 2020.

[13] S. Axelsson, "Intrusion Detection Systems: A Survey and Taxonomy," Technical Report 99-15, Dept. of Computer Engineering, Chalmers University, 2000.

[14] J. Lee and C. Kim, "Adaptive Threshold-Based DDoS Detection in API-Driven Environments," Computers & Security, vol. 96, p. 101891, 2020.

[15] Docker Inc., "Docker Documentation: Containerization for Microservices," 2024. https://docs.docker.com/