# A Comprehensive Framework for Advanced API Security and Real-Time Monitoring

Terli Sai krishna

Department of Computer Science

Engineering

Amrita School of Computing

Amrita Vishwa Vidyapeetham

Chennai, India

[terlisai45@gmail.com](mailto:terlisai45@gmail.com)

Thanusree Pyatlo  
Department of Computer Science

Engineering

Amrita School of Computing

Amrita Vishwa Vidyapeetham

Chennai, India

[thanupyatlo@gmail.com](mailto:thanupyatlo@gmail.com)

Dr. S Udhaya Kumar

Department of Computer Science

Engineering

Amrita School of Computing

Amrita Vishwa Vidyapeetham

Chennai, India

[s\_udhayakumar@ch.amrita.edu](mailto:s_udhayakumar@ch.amrita.edu)

***Abstract—APIs (Application Programming Interfaces) are critical components of modern web and mobile applications, enabling seamless communication between client and server systems. However, APIs are prime targets for cyberattacks, including injection attacks, unauthorized access, API abuse, and data manipulation. Traditional logging mechanisms lack tamper resistance, making forensic investigations unreliable, while weak authentication and the absence of real-time monitoring expose systems to security breaches. Additionally, APIs are frequently targeted by malicious IPs, and without an effective reputation-based monitoring system, organizations struggle to block harmful traffic.***

***This paper introduces an Advanced API Security and Monitoring System that integrates JWT authentication, threat detection, IP reputation analysis, rate limiting, and secure headers enforcement to protect against cyber threats. The system ensures cryptographic log integrity, verifying API request hashes to prevent tampering while detecting anomalies in real time. Distributed tracing enhances API observability, and Elasticsearch-based monitoring enables scalable log management. Additionally, it maintains a comprehensive audit trail by logging all incoming API requests and responses.***

***By combining real-time monitoring, automated integrity checks, and proactive threat defenses, this system strengthens API security, ensuring tamper-proof logging, attack prevention, and resilient access control while maintaining performance and compliance.***

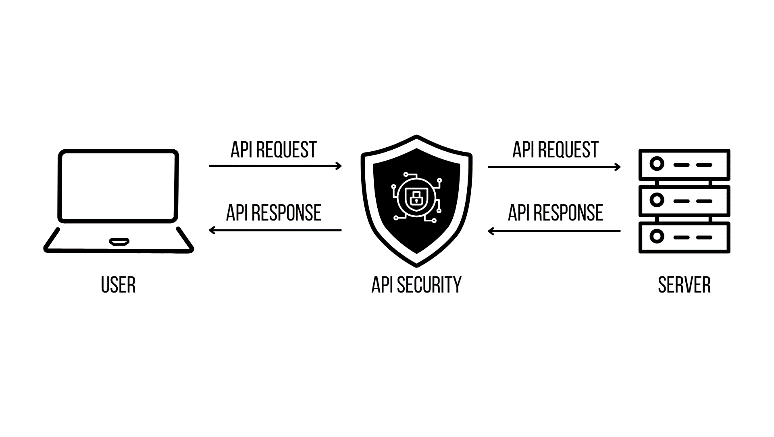
**I.INTRODUCTION**

In modern digital ecosystems, Application Programming Interfaces (APIs) play a crucial role in enabling seamless communication between web, mobile, and cloud applications. However, as organizations become increasingly reliant on APIs, they face growing cybersecurity threats such as injection attacks, unauthorized access, API abuse, data manipulation, and DDoS attacks. These vulnerabilities can lead to data breaches, financial losses, compliance violations, and reputational damage. Traditional logging mechanisms often lack tamper resistance, making forensic investigations unreliable. Ensuring the integrity, security, and observability of API interactions is essential for building a resilient API security framework.

This paper presents an Advanced API Security and Monitoring System that integrates multiple security mechanisms to protect API transactions from cyber threats. JWT authentication is implemented to enforce secure access control, ensuring that only authorized users can interact with the API. Threat detection analyzes incoming API requests for known malicious patterns, preventing exploitation through previously identified attack vectors. IP reputation analysis dynamically scores incoming requests based on historical behavior, blocking malicious actors before they can launch attacks.

To mitigate denial-of-service (DoS) threats, rate limiting is enforced, restricting excessive requests from a single source to prevent API abuse. Secure headers enforcement helps block common web exploits, including XSS, clickjacking, and MIME-type sniffing attacks. Cryptographic log integrity is maintained by hashing API requests before storage, ensuring tamper-proof audit trails. The system also performs distributed tracing, allowing security teams to monitor API transactions across different microservices for enhanced observability.

Additionally, Elasticsearch-based log management ensures scalable and efficient storage of API logs, enabling rapid search and retrieval of security incidents. The system verifies cryptographic hashes of API requests to detect unauthorized modifications, ensuring data integrity. Finally, all incoming API requests and responses are logged, providing comprehensive insights for security analysis and forensic investigations.



**Figure 1.1: A Secure API Communication Model**

Figure 1.1 shows how an API (Application Programming Interface) works with an added layer of security. When a user sends a request to a server, it first passes through an "API Security" checkpoint. This checkpoint examines the request for any threats using methods like integrity checks and threat detection. If the request is safe, it's sent to the server. Similarly, the server's response also goes through the API Security layer for a final check before reaching the user, ensuring secure communication

By combining cryptographic integrity, real-time threat detection, and scalable log management, this system provides a robust and intelligent API security framework. It enables organizations to detect, prevent, and respond to security threats effectively while maintaining continuous visibility and control over their API ecosystems.

**II. RELATED WORKS**

The increasing prevalence of Application Programming Interfaces (APIs) in modern digital ecosystems has introduced significant security challenges. Effectively securing APIs is essential to protect sensitive data and ensure the integrity of digital communications. Research has explored various approaches to enhance API security, focusing on machine learning, security testing, middleware solutions, and cloud-specific considerations. This literature review examines key contributions in these areas, highlighting the latest advancements, challenges, and proposed solutions in API security.

Paul [1] discusses the integration of machine learning (ML) in API security frameworks, emphasizing how traditional security measures often fail against sophisticated attacks like injection, credential stuffing, and Distributed Denial of Service (DDoS). By leveraging supervised and unsupervised learning models—such as clustering algorithms, decision trees, and neural networks—this approach effectively detects subtle deviations in API behavior, enabling proactive threat mitigation through dynamic security updates based on real-time monitoring. Similarly, Ali, Nasim, and Haider [5] introduce a secure middleware model for public RESTful APIs, addressing gaps in existing security frameworks. Their comprehensive solution integrates authentication, access control, threat detection, and encryption while maintaining system performance. Additionally, the middleware supports regulatory compliance (e.g., GDPR, HIPAA) and incorporates real-time logging and alerting mechanisms to ensure immediate responses to security incidents. Designed for high-performance environments, this middleware solution enhances API security while meeting the demands of large-scale applications that rely on APIs for critical operations.

Kaul and Khurana [4] extend this concept by exploring an AI-driven security model tailored for enterprise-level distributed systems. Their work addresses vulnerabilities stemming from the broad accessibility and interoperability of APIs, proposing a model that integrates robust encryption, multi-factor authentication, and advanced anomaly detection. Complementing this, Alharbi and Moulahi [2] focus on the challenges of security testing for RESTful APIs, critiquing conventional assessments for missing logic-based flaws and advocating for an automated vulnerability assessment framework that combines both static and dynamic analysis within CI/CD pipelines.

Hindka [3]provides an in-depth analysis of API security patterns and practices, identifying common pitfalls such as broken authentication, inadequate rate limiting, and insufficient logging. His recommendations include adopting OAuth 2.0 for robust authentication, centralizing security controls via API gateways, and deploying Web Application Firewalls (WAFs) to safeguard against attacks. In parallel, Kolawole [7] emphasizes the strategic importance of secure API implementations in promoting business growth and managing external integrations, advocating for stringent access controls and periodic updates to security policies to mitigate evolving threats.

Athukorale et al. [6] evaluate cybersecurity technologies in cloud environments, highlighting the unique challenges posed by multi-tenancy and dynamic workloads. Their study underscores the effectiveness of advanced techniques like data encryption, token-based authentication, and policy-driven security mechanisms. Additionally, they emphasize the necessity of integrating security strategies that adapt to the scalability of cloud-based systems, ensuring resilience against dynamic cyber threats.

Similarly, Aharon et al. [8] tackle API injection attacks by presenting a novel classification-by-retrieval framework using few-shot anomaly detection enhanced by transfer learning. Their approach leverages minimal training data to identify novel attack patterns effectively, making it a promising technique for API security in scenarios where large datasets are unavailable. Meanwhile, Baladari [9] focuses on security best practices for AI-powered chatbots in banking, stressing the need for robust authentication, encryption, and continuous threat monitoring. His study highlights the risks associated with conversational AI interfaces, particularly in financial applications where sensitive customer data is frequently exchanged.

Manoharan [10] provides a systematic analysis of API rate-limiting mechanisms—including Token Bucket, Leaky Bucket, and Sliding Window algorithms—to combat DDoS attacks, demonstrating the benefits of context-aware approaches over traditional IP-based methods. He also explores the trade-offs between different rate-limiting techniques in balancing security with API usability, emphasizing the importance of adaptive rate control to prevent legitimate user disruption. Complementing this, Adilapuram [11] explores modern authentication strategies for Java APIs, discussing the trade-offs among OAuth 2.0, JWT, API keys, and basic authentication. His study provides insights into the performance impact and security risks associated with each authentication method, making it valuable for developers selecting authentication schemes based on their API use cases.

Lewis et al. [12] introduce the Automated IP Reputation Analyzer Tool (AIPRA), which leverages public blacklist databases and machine learning to identify malicious IPs and domain names, enhancing API security through real-time risk assessments. Gadient et al. [13] examine the adoption of security-related HTTP headers in mobile app communications, revealing a general lack of support that could lead to data leaks and arbitrary code execution. Montanari et al. [14] propose a cloud-based framework to verify log trustworthiness and ensure compliance by collecting evidence data, mitigating risks associated with manipulated system logs. Similarly, Zawoad et al. [15] highlight the significance of secure logging for digital forensics by introducing FAL, a domain-specific language designed to enforce secure logging mechanisms across heterogeneous log formats, thereby improving audit reliability.

This review highlights key advancements in API security, including machine learning, automated testing, middleware solutions, and threat mitigation. Ensuring proactive security remains essential for protecting sensitive data.

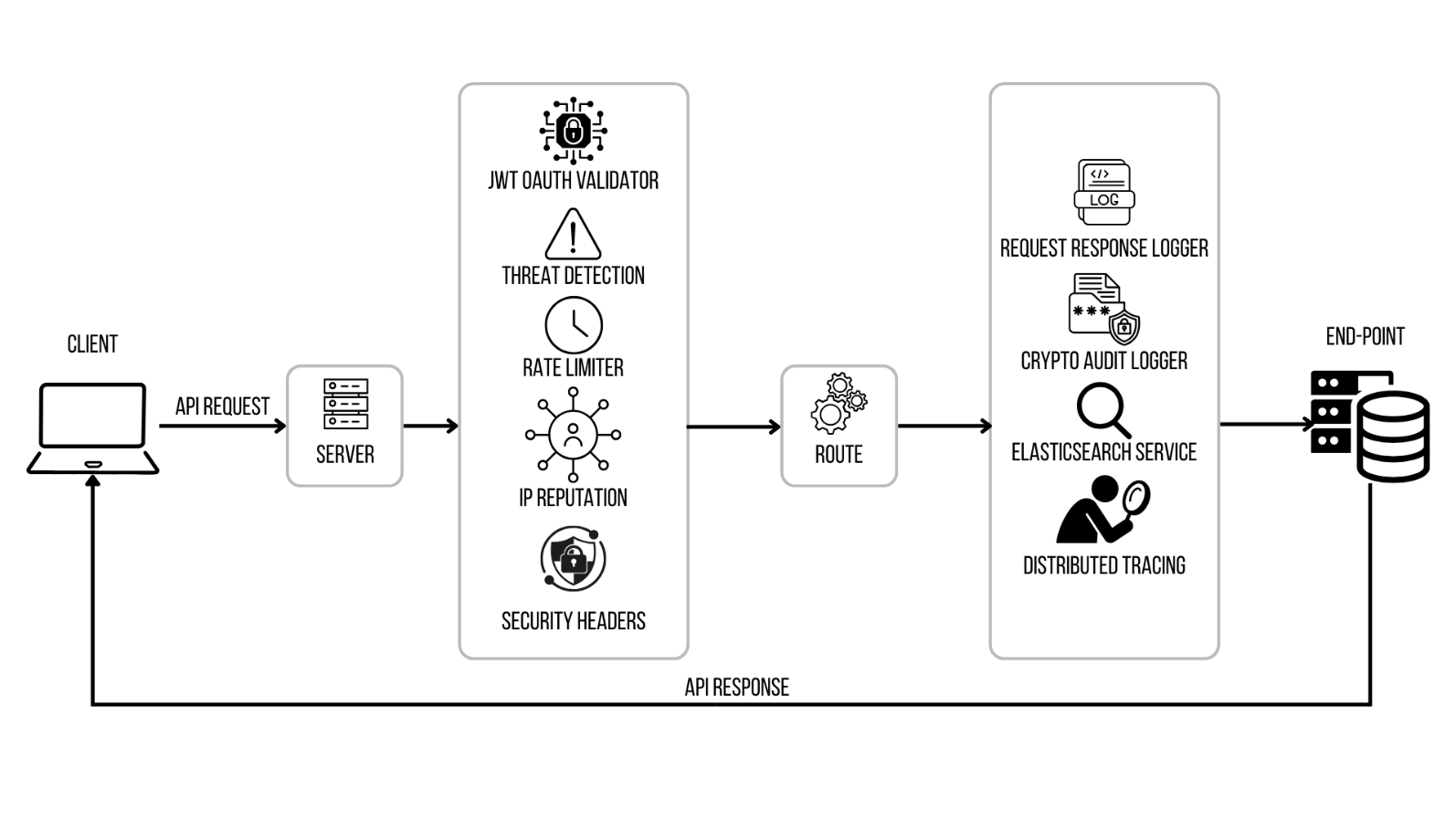
**Figure 1.2: Architecture Diagram**

Figure 1.2 illustrates the end-to-end flow of an API request within the Advanced API Security and Monitoring System. When a client sends a request, it first reaches the server and is forwarded to the security validation layer, which enforces multiple protective measures, including JWT authentication, threat detection, rate limiting, IP reputation analysis, and security headers enforcement. Once validated, the request proceeds to the forensic monitoring layer, where the Request Response Logger records transaction details, the Crypto Audit Logger ensures log integrity, Elasticsearch enables real-time threat analysis, and Distributed Tracing provides end-to-end visibility. Simultaneously, the validated request is processed by the endpoint, which generates and returns the API response to the client.

**III.METHODOLOGY**

The Advanced API Security and Monitoring System is a robust solution designed to protect APIs from cyber threats, unauthorized access, and malicious activities through a multi-layered defense approach. It integrates essential security features such as JWT authentication, IP reputation analysis, rate limiting, threat detection, secure headers enforcement, cryptographic log integrity, distributed tracing, and Elasticsearch integration for real-time monitoring. These components ensure that API environments remain secure, resilient, and protected from unauthorized intrusions. Authentication is a key security layer, and the system employs JSON Web Tokens (JWTs) for access control. When a user logs in, a JWT is generated and attached to API requests for verification. The JWT OAuth validator module verifies the token’s signature, checks expiration times, and ensures data integrity. If an invalid or expired token is detected, access is immediately denied, preventing unauthorized access attempts. This stateless authentication mechanism eliminates risks like session hijacking and credential theft, ensuring only verified users can interact with the API. Additionally, the system enforces secure session handling by rejecting compromised tokens and revoking access when anomalies are detected.

Beyond authentication, the system incorporates advanced protective measures against malicious activities. IP reputation analysis helps identify and block harmful requests by assessing incoming IPs against a database of suspicious entities. If an IP has a history of DoS attacks or brute-force attempts, it is flagged and blocked or subjected to additional security measures. Rate limiting is another crucial defense that prevents Denial-of-Service (DoS) attacks by limiting the number of requests a user or IP can make within a set timeframe. The rate limiter module dynamically enforces these restrictions, ensuring fair resource distribution and preventing service abuse. If an IP exceeds the allowed request limit, it is throttled or temporarily blocked to maintain system stability. Furthermore, adaptive rate limiting techniques are employed to detect sudden traffic spikes and adjust thresholds accordingly, reducing false positives and ensuring uninterrupted service for legitimate users.

To enhance security further, the system features a threat detection module that scans incoming requests for cyber threats such as SQL Injection (SQLi), Cross-Site Scripting (XSS), Remote Code Execution (RCE), and Server-Side Request Forgery (SSRF). Using pattern-based detection techniques, it identifies and blocks malicious inputs before they can exploit vulnerabilities. Additionally, real-time monitoring with Elasticsearch and distributed tracing helps detect anomalies like multiple failed logins from different locations or unusual API access patterns. These logs are indexed in Elasticsearch, enabling fast threat investigation and forensic analysis. If suspicious activity is detected, the system can trigger alerts or automatically block threats. Combined with cryptographic log integrity and secure headers enforcement, this security framework ensures robust API protection against modern cyber threats. Moreover, tamper-proof logging mechanisms ensure that forensic data remains unaltered, allowing for precise post-incident analysis and regulatory compliance.

**Table 1.1: Signatures for Detecting Common Web Vulnerabilities in API Requests**

|  |  |  |
| --- | --- | --- |
| S.No | Attack name | Syntax Used to Check for Attack |
| 1 | XSS (Cross-Site Scripting) | **(?i)<script.\*?> , (?i)on\w+\s\*=, (?i)javascript:, (?i)vbscript:, (?i)data:text/html, (?i)document\.cookie, (?i)document\.write, (?i)alert\(, (?i)prompt\(, (?i)eval\(, (?i)innerHTML, (?i)outerHTML** |
| 2 | SQL Injection | **(?i)union\s+select, (?i)or\s+1=1, (?i)drop\s+table, (?i)insert\s+into, (?i)update\s+.\*set, (?i)delete\s+from, (?i)benchmark\((.\*?)\), (?i)sleep\((\d+)\), (?i)load\_file\(, (?i)outfile, (?i)information\_schema** |
| 3 | Command Injection | **(?i);--, (?i)&\s\*rm\s, (?i)&\s\*del\s, (?i)&\s\*format\s, (?i)&\s\*shutdown\s, (?i)&\s\*reboot\s, (?i)system\(, (?i)exec\s+, (?i)os\.system, (?i)subprocess\.Popen** |
| 4 | Path Traversal | **(?i)\.\./, (?i)/etc/passwd, (?i)/proc/self/environ, (?i)C:\\Windows, (?i)C:\\boot.ini, (?i)\\..\\, (?i)/var/log, (?i)/root/.ssh** |
| 5 | Remote Code Execution (RCE) | **(?i)import\s+os, (?i)import\s+subprocess, (?i)import\s+pickle, (?i)import\s+eval, (?i)eval\(, (?i)exec\(, (?i)pickle\.loads, (?i)marshal\.loads** |
| 6 | Sensitive Data Exposure | **(?i)Set-Cookie:,(?i)api\_key=, (?i)password=,(?i)secret\_key=, (?i)access\_token=, (?i)private\_key=, (?i)session\_id=** |
| 7 | Server-Side Request Forgery (SSRF) | **(?i)http://169\.254\.169\.254, (?i)http://metadata\.google\.internal, (?i)http://localhost, (?i)http://127\.0\.0\.1, (?i)file://, (?i)ftp://** |
| 8 | XML External Entity (XXE) | **(?i)<!DOCTYPE, (?i)SYSTEM, (?i)ENTITY, (?i)file://, (?i)public** |
| 9 | LDAP Injection | **(?i)\\*|, (?i)&|, (?i)\(objectClass=.\*\), (?i)\(userPassword=.\*\), (?i)\(cn=.\*\), (?i)\(uid=.\*\)** |
| 10 | Server-Side Template Injection (SSTI) | **(?i){{.\*?}}, (?i){%.\*?%}, (?i)\$\{.\*?}, (?i)eval\(, (?i)exec\(** |
| 11 | Open Redirect | **(?i)\bhttps?://[^/]+@, (?i)\bhttps?://[^/]+\.example\.com@, (?i)\bhttps?:\/\/[^\/]+\/\/** |

Table 1.1 presents the detection signatures used by the API security system to identify and block common web vulnerabilities in incoming requests. It includes regex-based patterns for attacks such as XSS, SQL Injection, Command Injection, Path Traversal, and Remote Code Execution, among others. For example, SQL Injection is detected through patterns like UNION SELECT and OR 1=1, while XSS is flagged by identifying malicious JavaScript execution attempts. By leveraging these predefined signatures, the system proactively prevents exploitation of vulnerabilities, ensuring secure API interactions.

Web-based attacks, including clickjacking, cross-site scripting (XSS), and MIME-type sniffing, pose significant security risks to APIs. To mitigate these threats, the system enforces strict security headers in API responses using the security headers module. This module automatically injects essential HTTP security headers such as Strict-Transport-Security (HSTS), X-Frame-Options, Content Security Policy (CSP), and X-XSS-Protection. These headers prevent browsers from executing malicious scripts, loading untrusted content, and exposing users to security vulnerabilities. By enforcing these security best practices, the system significantly reduces its attack surface and strengthens overall protection against common web exploits. Another critical aspect of security is ensuring the integrity of logs, as tampered logs can compromise forensic investigations and compliance efforts. To address this, the crypto audit logger module cryptographically signs each log entry using SHA-256 hashing, making it resistant to unauthorized modifications. Additionally, the hash verify module recalculates and verifies these hashes, ensuring the logs remain unaltered. If any discrepancies are detected, an alert is immediately triggered, signaling possible tampering. This cryptographic approach guarantees that security logs remain reliable, accurate, and verifiable for audit purposes.

Beyond log integrity, the system incorporates distributed tracing to monitor API transactions and enhance observability. The distributed tracing module tracks API requests as they pass through different services, capturing crucial data such as response times, service interactions, and error occurrences. This allows security teams and developers to troubleshoot slow endpoints, correlate security incidents, and optimize system performance. In addition to tracing, a centralized logging system powered by Elasticsearch enables scalable API monitoring and real-time analysis. The elasticsearch service module processes logs from API transactions, security events, and threat detection mechanisms, indexing them for easy searching and visualization. Security teams can leverage Elasticsearch dashboards to identify attack patterns, detect anomalies, and respond proactively to potential threats. This robust logging solution enhances forensic investigations, accelerates threat detection, and ensures compliance with security policies, providing deeper insights into API activity.

To further reinforce data integrity, the hash verify module continuously validates the cryptographic hashes of stored logs. By recalculating and comparing SHA-256 hashes, the system detects any unauthorized modifications, ensuring logs remain intact and tamper-proof. Another essential feature is comprehensive request and response logging, handled by the request response logger module. This module records critical details such as timestamps, client IP addresses, request methods, headers, and payloads for every incoming request and outgoing response. These logs are stored securely and analyzed to detect suspicious patterns, failed authentication attempts, and abnormal API usage. By maintaining a detailed and immutable audit trail, the system strengthens security monitoring and aids forensic investigations. The Advanced API Security and Monitoring System integrates JWT authentication, IP reputation analysis, rate limiting, threat detection, secure headers enforcement, cryptographic logging, distributed tracing, and Elasticsearch-based monitoring to provide real-time threat prevention, secure authentication, and advanced forensic capabilities. This comprehensive approach ensures continuous security monitoring, rapid attack mitigation, and adherence to modern security standards while maintaining system performance and reliability.

**Table 1.2: Overview of Implemented Features**

|  |  |  |
| --- | --- | --- |
| S.No | Features | Description |
| 1 | Server | **Main Flask server handling API requests.** |
| 2 | Jwt oauth validator | **Validates JWT and OAuth-based authentication.** |
| 3 | Threat detection | **Detects and blocks various security threats.** |
| 4 | Ip reputation | **Checks IP addresses against reputation lists and determines a score.** |
| 5 | Rate limiter | **Implements rate limiting to prevent DOS attacks.** |
| 6 | Security headers | **Enforces security headers in HTTP responses.** |
| 7 | Request response logger | **Logs incoming requests and responses.** |
| 8 | Crypto audit logger | **Logs cryptographically secure (Hashed) logs.** |
| 9 | Hash verify | **Verifies cryptographic hashes for integrity.** |
| 10 | Distributed tracing | **Implements distributed tracing for monitoring.** |
| 11 | Elasticsearch service | **Manages Elasticsearch logging and queries.** |

Table 1.2 provides an overview of the key features implemented in the API security system. It includes essential modules such as the main server, which handles API requests, and the JWT OAuth validator, responsible for authentication. Security mechanisms like threat detection, IP reputation analysis, rate limiting, and security headers enforcement help mitigate various cyber threats. Logging and monitoring components, including the request-response logger, crypto audit logger, and distributed tracing, ensure comprehensive tracking of API activities. Additionally, hash verification maintains log integrity, while the Elasticsearch service enables efficient log management and analysis.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Algorithm: Advanced API Security and Monitoring System**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Require:**

1. API Gateway (G)
2. Security Policies (P = {P\_jwt, P\_ip\_reputation, P\_rate\_limit, P\_threat\_detection, P\_secure\_headers})
3. JWT Tokens, mTLS Certificates (C\_tls)
4. Incoming API Request (U\_req), Transaction Data (T\_api)
5. Logging System (L), Elasticsearch (ES), Distributed Tracing (DT)

**Ensure: Secure API transactions, real-time monitoring, and anomaly detection**

***Procedure:AUTHENTICATEANDAUTHORIZE(Ureq)***

6. T\_id ← VERIFYJWT(U\_req, T\_jwt)  
7. if T\_id = ∅ then  
8.   Reject request with status 401 Unauthorized  
9.   return  
9. if IPREPUTATIONCHECK(U\_req.ip) fails then  
10.   Reject request with status 403 Forbidden  
11.   return  
12. if RATELIMITCHECK(U\_req) exceeds threshold then  
13.   Reject request with status 429 Too Many Requests  
14.   return  
15. Forward request to API backend (M\_target)

***Procedure: PROCESSAPIREQUEST(Tapi)***

16. F\_raw ← PARSEREQUEST(T\_api)  
17. F\_clean ← THREATDETECTION(F\_raw) // Detect Threats  
18. if F\_clean = ∅ then  
19.   return Threat Detected — Request Blocked  
20. STORE F\_clean in tamper-proof log (L)  
21. TRIGGERSECURITYALERT if ANOMALYSCORE(F\_clean) > ε

***Procedure: INDEXLOGSINELEASTICSEARCH(L)***

22. if LOGINTEGRITYCHECK(L) fails then  
23.   TRIGGER AUDIT NOTIFICATION  
24. else  
25.   INDEXLOG(L, Elasticsearch)

***Procedure: DISTRIBUTEDTRACINGANDMONITORING***

26. T\_trace ← CAPTURETRACE(U\_req)  
27. L\_trace ← INDEXTRACE(T\_trace)  
28. if UNUSUALPATTERNDETECTED(L\_trace) then  
29.   TRIGGER SECURITY ALERT

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**IV. EXPERIMENTAL RESULTS**

This section presents the evaluation of the Advanced API Security and Monitoring System through a series of experiments conducted to assess the system's performance in detecting and preventing security threats, ensuring the integrity of logs, and maintaining the overall security posture of the API. The evaluation process included testing various components such as threat detection, IP reputation analysis, rate limiting, cryptographic log integrity verification, and the enforcement of security headers. The results demonstrate the system’s ability to effectively handle security threats while ensuring minimal impact on API performance.

1.Rate limiting analysis

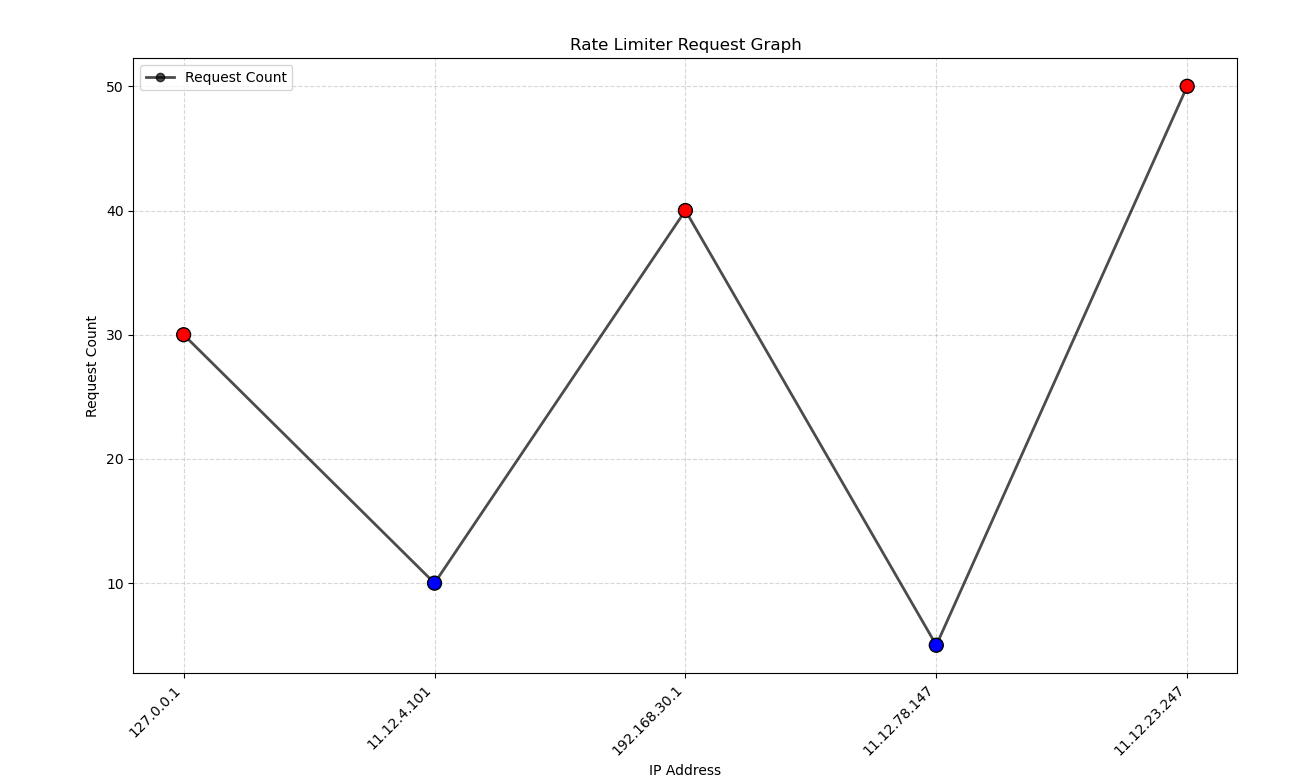
**Figure 1.3: Rate Limiter Request Distribution Across Ips**

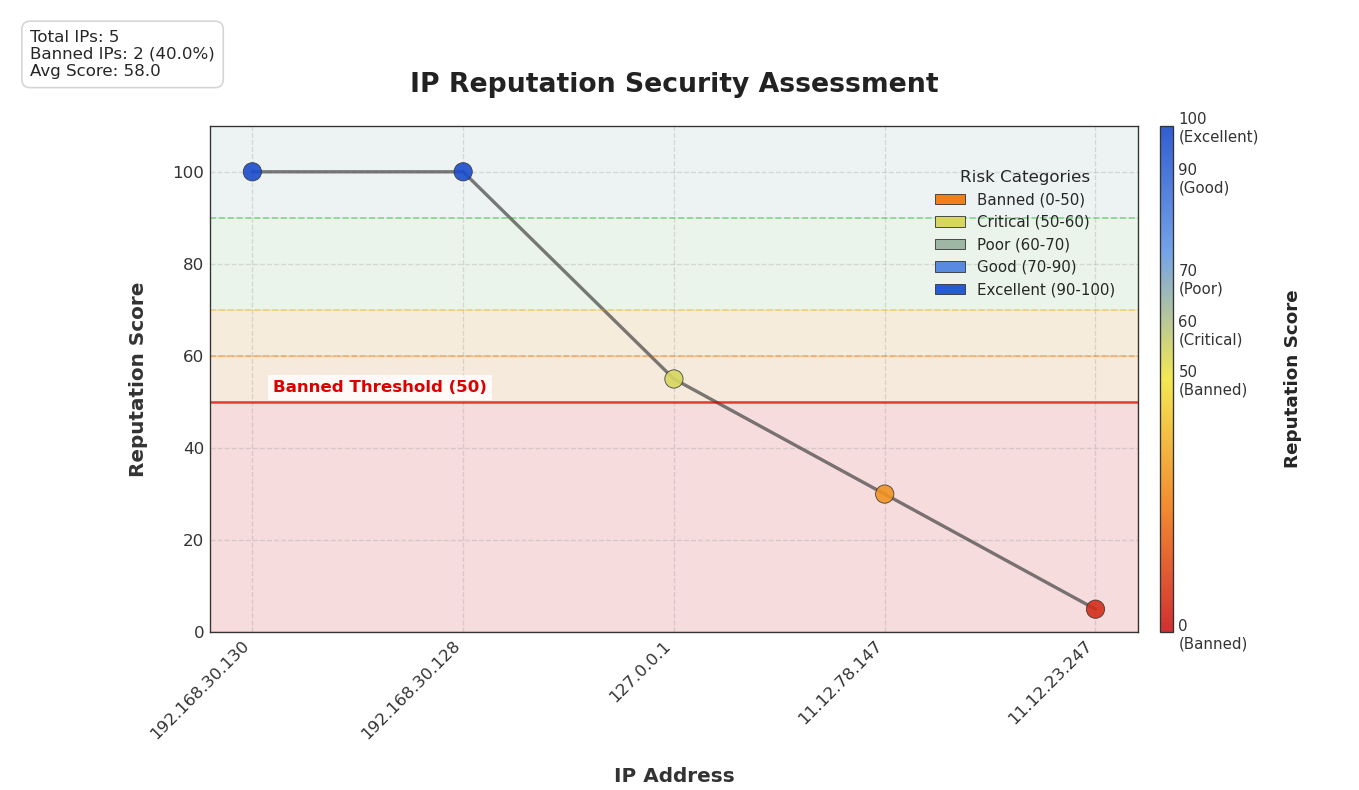
Fig 1.3 presents Rate Limiter Request Graph visualizes API request patterns from various IP addresses, highlighting potential abuse through excessive traffic. Red markers indicate IPs exceeding the allowed request threshold, triggering rate-limiting mechanisms, while blue markers represent normal traffic. This enforcement helps prevent denial-of-service attacks, brute-force attempts, and resource exhaustion by restricting high-frequency requests. By integrating rate limiting, the system ensures fair API usage, enhances security, and maintains service availability. This visualization aids in identifying traffic anomalies, optimizing request thresholds, and reinforcing API protection against malicious activities.

**Table 1.3: Rate Limiting Analysis Table**

|  |  |  |  |
| --- | --- | --- | --- |
| S.No | Ip address | Request Count | Blocked |
| 1 | 127.0.0.1 | 30 | 1 |
| 2 | 11.12.4.101 | 10 | 0 |
| 3 | 192.168.30.1 | 40 | 1 |
| 4 | 11.12.78.147 | 5 | 0 |
| 5 | 11.12.23.247 | 50 | 1 |

The table 1.3 presents an analysis of API requests based on IP addresses, tracking the number of requests made and whether the requests were blocked due to exceeding rate limits. Each row represents a unique IP address along with its request count. The "Blocked" column, where 1 indicates blocked requests and 0 indicates allowed requests, highlights which IPs triggered rate-limiting enforcement. The data reveals that IPs with high request counts, such as 127.0.0.1 (30 requests), 192.168.30.1 (40 requests), and 11.12.23.247 (50 requests), were blocked, while lower request counts were allowed, demonstrating the effectiveness of rate-limiting measures in controlling excessive traffic.

2.IP Reputation Analysis



**Fig 1.4: IP Reputation Score Distribution Across Monitored IP Addresses**

Fig 1.4 presents IP reputation system was evaluated by simulating requests from known malicious IP addresses. These blacklisted IPs, sourced from threat intelligence feeds, were used to test the system's ability to block harmful traffic. The system successfully blocked all malicious requests from these IPs while permitting legitimate traffic. Furthermore, the IP reputation list was dynamically updated with new threats, ensuring ongoing protection against emerging dangers. The graph visualizes the reputation scores of monitored IP addresses, with scores below the "Banned Threshold (50)" indicating blacklisted or high-risk IPs.

**Table 1.4: IP Reputation Assessment Results**

|  |  |  |  |
| --- | --- | --- | --- |
| S.No | Ip address | Score | Blocked |
| 1 | 192.168.30.1 | 100 | 0 |
| 2 | 11.12.4.101 | 70 | 0 |
| 3 | 127.0.0.1 | 55 | 0 |
| 4 | 11.12.23.247 | 40 | 1 |
| 5 | 11.12.78.147 | 10 | 1 |

Table 1.4 demonstrates the IP reputation system's blocking mechanism based on assigned scores. IP addresses receiving a score below 50 are classified as having a poor or banned reputation and are consequently blocked. For instance, IP address 11.12.23.247 with a score of 40 and 11.12.78.147 with a score of 10 both fall below this threshold and are marked as "Blocked = 1". Conversely, IP addresses with scores of 50 or higher are considered to have an acceptable to excellent reputation and are not blocked. Examples include 192.168.30.1 with a score of 100, 11.12.4.101 with a score of 70, and even 127.0.0.1 with a score of 55, all showing "Blocked = 0". This clearly illustrates the system's policy of automatically blocking traffic originating from IP addresses with a reputation score below 50.

2. Log Integrity Verification Using Cryptographic Hashing

**Table 1.5: Cryptographic Log Integrity Verification**

|  |  |  |  |
| --- | --- | --- | --- |
| S.No | Computed Hash (Computed from server.log) | Excepted Hash (Stored in crypt\_audit.log) | Verification |
| 1 | e100395fa770fc07dd6760c0f5891594d66d42e233ea808200ddf57aab183f0e | e100395fa770fc07dd6760c0f5891594d66d42e233ea808200ddf57aab183f0e | 0 |
| 2 | c6f922d53a77c3302990da55de1effecd942128b30284e329461e0ac8b56ce1d | 06a63db77a214c148ff58bb439534535b8261043f55dd190271da6d422e73ada | 1 |
| 3 | 6c1b42c5424ee91dd1cfa3e59eecb70ecb076f9175a122d4bc41f50e1b3f5eea | 6c1b42c5424ee91dd1cfa3e59eecb70ecb076f9175a122d4bc41f50e1b3f5eea | 0 |

Table 1.5 details the cryptographic verification of log integrity. SHA-256 hashes computed from server.log are compared against expected hashes in crypt\_audit.log. A "Verification" value of '0' confirms log integrity through matching hashes, while '1' indicates a mismatch and potential tampering. Entries 1 and 3 show successful verification, whereas entry 2 reveals a mismatch, highlighting the system's ability to detect unauthorized log modifications and ensure audit data trustworthiness.

**V. CONCLUSION**

This paper presents an advanced API security and monitoring system that integrates multiple layers of protection, including cryptographic log integrity, real-time request tracing, and Elasticsearch-based anomaly detection. It effectively mitigates security threats such as SQL injection, XSS, SSRF, and command injection through regex-based payload analysis and strict request validation. By leveraging a structured audit logging mechanism, cryptographic hashing ensures the integrity of API transactions, preventing tampering and unauthorized modifications. The traceability of requests, combined with Elasticsearch’s powerful indexing and search capabilities, enables real-time monitoring and rapid incident response. Additionally, IP reputation analysis enhances security by identifying malicious actors based on historical behavior patterns, while heatmap visualization provides an intuitive representation of threat levels to aid in proactive threat management. Overall, this approach demonstrates a robust and scalable solution for detecting, preventing, and responding to cyber threats in real time, with potential future enhancements including machine learning-based anomaly detection and automated threat response mechanisms for even greater security resilience.

**VI. REFERENCE**

[1] J. Paul, “Building a Robust API Security Framework with Machine Learning,” November 2024.

[2] S. J Alharbi and T. Moulahi, “API Security Testing: The Challenges of Security Testing for Restful APIs,” International Journal of Innovative Research in Science Engineering and Technology, vol. 8, issue 5, May 2023.

[3] M. Hindka, “Securing the Digital Backbone: An In-depth Insights into API Security Patterns and Practices,” ISA Transactions, April 2024.

[4] D. Kaul and R. Khurana, “AI to Detect and Mitigate Security Vulnerabilities in APIs: Encryption, Authentication, and Anomaly Detection in Enterprise-Level Distributed Systems,” Eigenpub Review of Science and Technology, 2021.

[5] S. Ali, F. Nasim, and K. Haider, “Secure Middleware Model for Public RESTful APIs,” AL-AASAR Journal, vol. 2, no. 1, 2025.

[6] N. K. Athukorale, C. J. Yi, L. Z. Xin, A. Y. Choo, D. Z. Yu, J. S. Jia Wei, F. Hady, L. S. Zhe, C. C. Eu, and S. R. Sindiramutty, “Evaluating Advanced Cybersecurity Technologies for Cloud Environments,” Preprints.org, January 2025.

[7] O. O. Kolawole, “Business Growth and Development Strategy in the Era of Dynamic Communication Paradigm: A Discourse of Application Programming Interface (API),” Research Journal of Mass Communication and Information Technology, vol. 11, no. 1, 2025.

[8] U. Aharon, R. Dubin, A. Dvir, and C. Hajaj, “A classification-by-retrieval framework for few-shot anomaly detection to detect API injection,” Computers & Security, vol. 150, 2025.

[9] V. Baladari, “AI-Powered Chatbots in Banking: Developer Best Practices for Enhancing Efficiency and Security,” International Journal of Advanced Research in Science Communication and Technology, January 2024.

[10] M. Manoharan, “API Rate Limiting Mechanisms in SaaS Applications: A Systematic Analysis of DDoS Protection Strategies,” International Journal of Scientific Research in Computer Science, Engineering and Information Technology, vol. 10, no. 6, 2024

[11] S. Adilapuram, “Unveiling Modern Authentication Strategies for Java APIs: Exploring OAuth 2.0, JWT, API Keys, and Basic Authentication,” Journal of Scientific and Engineering Research, vol. 9, no. 9, 2022.

[12] J. L. Lewis, G. F. Tambaliuc, H. S. Narman, and W. -S. Yoo, "IP Reputation Analysis of Public Databases and Machine Learning Techniques"

[13] P. Gadient, O. Nierstrasz, and M. Ghafari, "Security Header Fields in HTTP Clients."

[14] M. Montanari, J. H. Huh, D. Dagit, R. B. Bobba, and R. H. Campbell, "Evidence of Log Integrity in Policy-based Security Monitoring."

[15] S. Zawoad, M. Mernik, and R. Hasan, "FAL: A Forensics Aware Language for Secure Logging."