APILogGuard FOR SECURITY EVENTS

Major Project (Phase -1) submitted in partial fulfillment of the requirement for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

Under the esteemed guidance of

Dr S. Radha
Associate Professor

By

ABBURI BHAVYA SRI (21R11A05A5)
KOSURU BHARATH KUMAR (21R11A05C8)
BUSHOLLA SRINATH (21R11A05B5)

Department of Computer Science and Engineering
Accredited by NBA

Geethanjali College of Engineering and Technology
(UGC Autonomous)
(Affiliated to J.N.T.U.H, Approved by AICTE, New Delhi)
Cheeryal (V), Keesara (M), Medchal.Dist.-501 301.

December-2024

Geethanjali College of Engineering & Technology

(UGC Autonomous)

(Affiliated to JNTUH, Approved by AICTE, New Delhi)
Cheeryal (V), Keesara(M), Medchal Dist.-501 301.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Accredited by NBA

CERTIFICATE

This is to certify that the B.Tech Major Project report entitled **"APILogGuard FOR SECURITY EVENTS"** is a bonafide work done by **Abburi Bhavya Sri (21R11A05A5), Kosuru Bharath Kumar (21R11A05C8), Busholla Srinath (21R11A05B5)**, in partial fulfillment of the requirement of the award for the degree of Bachelor of Technology in "**Computer Science and Engineering**" from Jawaharlal Nehru Technological University, Hyderabad during the year 2023-2024.

Internal Guide	**HOD - CSE**
Dr S. Radha	**Dr A SreeLakshmi**
Associate Professor	Professor

External Examiner 1

External Examiner 2

Geethanjali College of Engineering & Technology

(UGC Autonomous)

(Affiliated to JNTUH Approved by AICTE, New Delhi)
Cheeryal (V), Keesara(M), Medchal Dist.-501 301.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Accredited by NBA

DECLARATION BY THE CANDIDATE

We, **Abburi Bhavya Sri, Kosuru Bharath Kumar, Boshulla Srinath,** bearing Roll Nos. **21R11A05A5, 21R11A05C8, 21R11A05B5** hereby declare that the project report entitled **"APILogGuard for security events"** is done under the guidance of **Dr. S. Radha**, **Associate Professor**, Department of Computer Science and Engineering, Geethanjali College of Engineering and Technology, is submitted in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology in Computer Science and Engineering**.

This is a record of bonafide work carried out by me/us in **Geethanjali College of Engineering and Technology** and the results embodied in this project have not been reproduced or copied from any source. The results embodied in this project report have not been submitted to any other University or Institute for the award of any other degree or diploma.

Abburi Bhavya Sri(21R11A05A5)

Kosuru Bharath Kumar(21R11A05C8)

Boshulla Srinath(21R11A05B5)

Department of CSE,

Geethanjali College of Engineering and Technology,

Cheeryal.

ACKNOWLEDGEMENT

We would like to express our sincere thanks to **Dr. A. Sree Lakshmi, Professor, Head of Department of Computer Science,** Geethanjali College of Engineering and Technology, Cheeryal, whose motivation in the field of software development has made us to overcome all hardships during study and successful completion of project.

We would like to express our profound sense of gratitude to all for having helped us in completing this dissertation. We would like to express our deep-felt gratitude and sincere thanks to our guide **Dr.S. Radha, Associate Professor,** Department of Computer Science, Geethanjali College of Engineering and Technology, Cheeryal, for her skillful guidance, timely suggestions, and encouragement in completing this project successfully.

We would like to express our sincere gratitude to our **Principal Prof. Dr. S. Udaya Kumar** for providing the necessary infrastructure to complete our project. We are also thankful to our Chairman **Mr. G. R. Ravinder Reddy** for providing an interdisciplinary and progressive environment.

Finally, we would like to express our heartfelt thanks to our parents who were very supportive both mentally and financially for their encouragement to achieve our set goals.

Abburi Bhavya Sri(21R11A05A5)

Kosuru Bharath Kumar(21R11A05C8)

Boshulla Srinath(21R11A05B5)

ABSTRACT

In today’s fast-evolving digital world, Application Programming Interfaces (APIs) are essential for seamless communication between systems, applications, and services. They enable quick data sharing and functionality, but their widespread use also makes them vulnerable to security threats like brute force attacks, unauthorized IP access, and other malicious activities. To tackle these risks, this project introduces a well-rounded API logging and monitoring system. At its core is a middleware solution designed to efficiently handle requests, track API metrics, and log key data for deeper insights.

This middleware will function as a security watchdog, integrating effortlessly into the API environment to identify and respond to potential threats in real time. It will monitor for brute force attacks by keeping an eye on repeated failed login attempts, flag suspicious IP addresses, and assess behavioral patterns to detect irregular activities. Moreover, the system’s advanced logging features will create detailed records of all API interactions, helping with compliance requirements and forensic investigations.

To stay ahead of threats, the system incorporates automated alert mechanisms that instantly notify administrators of suspicious activities through SMS, email, or collaboration tools like Slack. Complementing this is a suite of visualization tools, including Prometheus and Grafana, which deliver clear, real-time dashboards to monitor API performance and security health.

Designed to align with cutting-edge Security Information and Event Management (SIEM) practices, this project emphasizes proactive threat detection and prevention. By establishing a secure and scalable monitoring framework, API developers and administrators will gain the tools they need to protect their systems effectively, ensuring the reliability and confidentiality of their APIs.

LIST OF ABBREVIATIONS

Acronym	Abbreviations
API	Application Programming Interface
CIDR	Classless Inter-Domain Routing
DDoS	Distributed Denial of Service
DNS	Domain Name System
IP	Internet Protocol
JWT	JSON Web Token
SIEM	Security Information and Event Management
SMTP	Simple Mail Transfer Protocol
SNS	Simple Notification Service
TLS	Transport Layer Security
UI	User Interface

TABLE OF CONTENTS

1. INTRODUCTION

1.1 About the project

In today's interconnected digital world, APIs (Application Programming Interfaces) have become the cornerstone of modern software systems, facilitating seamless communication between applications, services, and systems. From enabling real-time data sharing in cloud-based platforms to powering the functionality of mobile applications, APIs are indispensable to businesses and developers alike. However, with this critical role comes increased exposure to security vulnerabilities, making APIs a prime target for malicious actors.

This project focuses on developing a comprehensive **APILogGuard system** aimed at addressing the growing challenges of API security. The proposed system leverages a middleware program that acts as an intermediary between API endpoints and their consumers. This middleware is responsible for tracking incoming requests, maintaining detailed logs of interactions, and recording the number of API calls per client. These logs are not only essential for operational monitoring but also serve as a foundation for detecting and analyzing security incidents.

A key feature of the project is its ability to detect security threats in real time. The system is designed to identify patterns that indicate brute force attacks, such as repeated failed login attempts, as well as unusual activities like access from suspicious or unauthorized IP addresses. Furthermore, it employs advanced mechanisms to monitor anomalous behavior, such as sudden spikes in traffic or irregular API usage patterns, which could signify an ongoing attack or misuse.

In addition to logging and detection, the project integrates real-time alerting mechanisms to notify administrators of potential threats. Notifications are sent via modern communication channels, including AWS SNS, email, or messaging platforms, ensuring swift responses to security incidents. To provide actionable insights and enhance decision-making, the system is integrated with visualization tools like Prometheus and Grafana, offering clear, dynamic dashboards that display API usage metrics, system health, and security alerts.

By combining robust logging, proactive monitoring, and intelligent threat detection, this project provides a holistic solution for API security. It ensures not only the integrity and availability of API services but also supports compliance requirements by maintaining historical records of API activity for audits and forensic analysis. Ultimately, the project empowers API developers and administrators to build, monitor, and maintain secure and resilient systems, keeping pace with the evolving landscape of cybersecurity threats.

1.2 Goal

The primary goal of this project is to develop a middleware-based logging and monitoring system to enhance API security. APIs are crucial for modern systems but are also vulnerable to various security threats. This project aims to create a framework that tracks and logs incoming requests, monitors API interactions in real time, and detects potential security risks such as brute force attacks, unauthorized IP access, and unusual usage patterns.

The system will maintain detailed logs of API calls, including request counts and IP addresses, providing a valuable resource for security audits and incident investigations. It will also detect suspicious activities, such as repeated failed login attempts and abnormal traffic patterns, and send automated alerts to administrators for quick response.

By integrating real-time monitoring and automated alerts, the project seeks to ensure the resilience of APIs against security threats, providing a secure, reliable environment for both users and administrators while enabling proactive threat detection and rapid response.

1.3 Objective

1. **Request Handling and Logging**: Develop middleware to track API requests, record call counts, and store detailed logs for future use and analysis.
2. **Threat Detection**: Implement mechanisms to identify and flag security threats, including brute force attempts, access from unauthorized IPs, and unusual API usage patterns.
3. **Real-Time Monitoring**: Integrate with monitoring tools like Prometheus and Grafana to provide visual insights into API activity and system health.
4. **Automated Alerts**: Set up notification systems using services like AWS SNS to send real-time alerts to administrators when security incidents are detected.
5. **Historical Data Analysis:** Maintain comprehensive logs to enable forensic analysis and support compliance requirements.
6. **Scalability and Efficiency**: Design the system to handle high API traffic while maintaining performance and ensuring seamless integration with existing applications.

2. SYSTEM ANALYSIS

2.1 Existing System

Currently, many organizations rely on basic logging systems to monitor API interactions and detect security incidents. However, these existing systems often lack real-time threat detection, comprehensive logging, and automated alerting. Most conventional logging systems primarily focus on recording errors, requests, and response statuses without much emphasis on security-specific events, such as brute force attacks, unusual IP access, or abnormal usage patterns. Moreover, these systems tend to be reactive, only identifying issues after they have caused significant damage. Furthermore, they typically lack integration with modern monitoring tools like Prometheus or Grafana, making it difficult to analyze large volumes of data or track long-term trends effectively.

Drawbacks of the Existing System:

- **Lack of Real-Time Threat Detection:** Existing systems often fail to detect and respond to threats in real time. For example, brute force attacks, repeated failed login attempts, and unusual API usage patterns are not flagged immediately, which allows attackers to exploit vulnerabilities without prompt intervention.
- **Limited Logging Capabilities:** Current logging mechanisms are often too basic, capturing only the fundamental data like request methods and error messages. This leaves out detailed security-related information such as request frequency, IP location, and suspicious patterns, which are crucial for detecting complex attacks.
- **Manual Incident Response**: Many systems still rely on manual investigation and intervention when security incidents occur. There are no automated alerts to notify administrators of potential threats, which delays the response time and increases the potential for damage.

- **Lack of Integration with Advanced Monitoring Tools**: Traditional systems usually do not integrate with modern monitoring tools like **Prometheus** and **Grafana**, which limits the ability to gain real-time insights into system performance and security. As a result, administrators have difficulty identifying trends or spikes in traffic that might indicate a security breach.
- **Inefficient Handling of High Traffic Volumes**: Existing systems often struggle to handle large amounts of traffic efficiently while maintaining security logging and monitoring. The lack of scalable solutions means that as API usage grows, the system's performance and ability to track security events may degrade.

2.2 Proposed System

The proposed system is designed to address the limitations of existing systems by providing a middleware-based API logging and monitoring solution. This system will not only track and log API requests but also analyze incoming traffic in real-time for potential security threats. It will include features like brute force attack detection, unusual IP address monitoring, and abnormal usage pattern identification. In addition, the system will integrate with advanced monitoring tools like Prometheus and Grafana to offer detailed, real-time insights into API performance and security. Automated alerts will notify administrators of potential threats via communication channels like AWS SNS, ensuring quick and efficient responses to incidents. The system will provide a comprehensive, proactive security framework that enhances API resilience and supports compliance by maintaining detailed historical logs for audit purposes.

Benefits of the Proposed System

- **Real-Time Threat Detection and Response:** The proposed system will actively monitor API traffic in real time, enabling the immediate identification of security threats like brute force attacks, unauthorized IP access, and unusual usage patterns. This proactive approach allows for faster response times and mitigation of attacks before they can escalate.
- **Comprehensive Logging and Monitoring:** The system will provide detailed logs of all API interactions, capturing not only basic request data but also security-related metrics like IP addresses, request frequencies, and error rates. This enables in-depth analysis and forensic investigation, enhancing security visibility and auditability.
- **Automated Alerts for Suspicious Activities:** By integrating automated alerting mechanisms (e.g., via AWS SNS), the system will notify administrators immediately when suspicious activities or potential security incidents are detected. This reduces the time taken to identify and respond to attacks, minimizing risks and enhancing overall security posture.
- **Scalable and Efficient Performance:** The system will be designed to handle high traffic volumes while maintaining effective logging and monitoring. By integrating with tools like Prometheus and Grafana, it provides scalable, real-time insights into API performance and security, ensuring it can grow with the needs of the application.
- **Compliance and Audit Readiness:** The system will help organizations meet security and data privacy regulations (e.g., GDPR, CCPA) by maintaining secure, accessible logs of all API interactions. This ensures that organizations can demonstrate compliance and respond to audit requests efficiently, while also enhancing overall data protection and security practices.

2.3 Feasibility Study

2.3.1 Technical Feasibility

The technical feasibility of the proposed system is highly viable given the availability of existing technologies and frameworks. The middleware will be developed using well-established programming languages like Node.js, Python, or Go, all of which are capable of handling high-throughput traffic and ensuring seamless integration with other components like Prometheus and Grafana for real-time monitoring. Additionally, the system will utilize popular logging frameworks such as Winston or Log4js for efficient request tracking. The integration of AWS SNS for automated alerts and the use of secure protocols (like TLS and OAuth) ensures that the system is both secure and scalable. Given the maturity of these technologies, the system can be implemented with ease, and existing infrastructure can be adapted to support it, ensuring smooth integration.

2.3.2 Impact on Environment

The proposed system is expected to have minimal direct environmental impact, as it primarily involves software solutions rather than hardware-intensive operations. The focus on cloud-based technologies such as AWS and Grafana allows for energy-efficient scaling, as cloud providers often optimize their data centers for power usage. By avoiding the need for large-scale on-premises hardware, the project can reduce the carbon footprint associated with physical server infrastructure. Furthermore, the use of highly efficient logging and monitoring systems will reduce unnecessary data storage, optimizing resource use and indirectly supporting sustainability efforts.

2.3.3 Safety

In terms of safety, the system will be designed to prioritize the security and confidentiality of user data. Logging sensitive information, such as personally identifiable information (PII), will be avoided unless explicitly required for security purposes. The system will adhere to best practices for data protection, including encryption of data in transit and at rest. In addition, access controls and authentication mechanisms will ensure that only authorized personnel can access system logs and security alerts. By detecting and mitigating attacks like brute force or DDoS in real time, the system also contributes to the overall safety of the API environment, reducing vulnerabilities that could be exploited by malicious actors.

2.3.4 Ethics

The ethical feasibility of the system is central to ensuring that it operates in compliance with data protection laws and respects user privacy. The system will avoid the collection of personally identifiable information (PII) unless absolutely necessary and will anonymize or hash any sensitive data stored in logs. The project will adhere to ethical guidelines for transparency, ensuring that users are informed about what data is being collected and how it will be used. In terms of security, the system will also follow ethical practices in detecting and responding to attacks, ensuring that no harm is caused to legitimate users or businesses while preventing malicious activity.

2.3.5 Cost

The cost feasibility of the project is manageable, especially with the widespread availability of open-source tools and cloud services. The primary costs will involve software development, including developer salaries or contractor fees for specialists in middleware and API security. Cloud services, such as AWS for hosting and Prometheus and Grafana for monitoring and alerting, will also incur costs, though many of these services offer affordable pricing tiers, particularly for small to medium-sized deployments. Additionally, maintenance costs for server infrastructure and cloud services will be required to support the ongoing logging, monitoring, and alerting system. However, these costs are likely to be offset by the reduction in security incidents, which can be costly in terms of both reputation and financial losses. The system’s ability to quickly detect and respond to security threats will minimize the impact of potential breaches, making the overall system cost-effective in the long run.

2.3.6 Type

The proposed system is a middleware-based security solution that operates as a proactive monitoring and logging tool. It will function as a scalable, customizable component that can be integrated into existing API infrastructures, regardless of size. The system’s ability to track and analyze API requests in real time, detect security threats, and send automated alerts makes it a vital tool for modern API security. It is designed to be highly adaptable, capable of scaling with growing traffic loads, and can be used across various industries and applications, from small businesses to large enterprises. The system’s modular architecture allows for easy updates and feature enhancements, ensuring its relevance and usability in a rapidly evolving security landscape.

This feasibility study indicates that the project is both technically feasible and sustainable from an environmental, ethical, and cost perspective. It will provide significant value by enhancing API security while remaining aligned with industry best practices and regulations.

2.4 SCOPE OF THE PROJECT

This project will focus on providing a secure and scalable solution for logging and monitoring APIs. The key areas of development include:

- **Middleware for request handling**: Tracking, logging, and analyzing API requests.
- **Threat detection and alerts**: Identifying and notifying administrators of brute force attempts, unusual IP addresses, and abnormal usage patterns.
- **Real-time monitoring and visualization**: Integration with Prometheus and Grafana for dynamic API performance and security dashboards.
- **Compliance and auditing**: Maintaining detailed logs for compliance and forensic purposes. The system will be applicable to any API-driven application, ranging from small web apps to large-scale enterprise systems.

2.5 SYSTEM CONFIGURATION

1. SOFTWARE REQUIREMENTS

- ❖ **Operating System:** Linux-based (Ubuntu/Debian) or Windows Server for development and production environments.
- ❖ **Programming Language:** JavaScript (Node.js), Python, or Go (depending on the middleware framework chosen).
- ❖ **Database:** MySQL or MongoDB for storing logs and other related data.
- ❖ **Logging Framework:** Winston or Log4js (for logging API interactions).

- ❖ **Monitoring Tools:** Prometheus and Grafana for real-time monitoring and visualization.
- ❖ **Security Protocols:** OAuth for authentication, TLS for secure communication.
- ❖ **Notification System:** AWS SNS for automated alerts.

2. HARDWARE REQUIREMENTS

- ❖ **CPU:** Multi-core processor (minimum 4 cores).
- ❖ **RAM:** At least 8 GB of RAM.
- ❖ **Storage:** Minimum 100 GB of disk space (preferably SSD) for log storage.
- ❖ **Network:** High-speed internet connection (minimum 100 Mbps).

2.6 System Requirement Specification

The **System Requirements Specification (SRS)** for the project outlines the detailed functional and non-functional requirements for the system:

- **Functional Requirements**:
  - The system must be able to log API requests in real time.
  - The middleware should be able to detect brute force attacks by tracking failed login attempts.
  - It should monitor and flag unusual IP access and abnormal API usage patterns.
  - The system must integrate with Prometheus and Grafana for performance monitoring and alert visualization.

- Automated alerts should be sent to administrators via AWS SNS when suspicious activities are detected.

- **Non-Functional Requirements**:
  - The system must handle high volumes of API requests with minimal performance impact.
  - It must ensure scalability to support growing traffic loads.
  - The system should be highly available, with minimal downtime.
  - It should ensure data privacy and comply with relevant security standards (GDPR, CCPA).
  - The system should be easy to deploy and configure with minimal manual intervention.

2.7 Action Plan

Sprint No	Task	Objective	Timeline
Sprint 1	Learning Core Technologies	Familiarize with Node.js, React.js, MongoDB, and tools like Prometheus, Grafana, and Elasticsearch.	2 weeks
Sprint 2	Implement API Gateway	Set up API Gateway for routing, authentication, rate-limiting, and logging.	1 weeks
Sprint 3	Logging Service Development	Develop Logging Service to capture API logs and metadata.	1 weeks
Sprint 4	Implement Monitoring Service	Set up Monitoring Service to track key API metrics.	1 weeks
Sprint 5	Set Up Alerting System	Implement Alerting System for threshold-based notifications.	1 weeks
Sprint 6	Visualization and Analytics Dashboard	Create dashboard to visualize API metrics and logs	1 weeks
Sprint 7	Storage and Database Integration	Integrate storage solutions for logs (Elasticsearch/MongoDB) and metrics (Prometheus).	1 weeks

Sprint 8	Security Enhancements	Encrypt sensitive data, apply rate-limiting, and prevent DDoS attacks.	1 weeks
Sprint 9	Integration Testing	Perform integration testing for all system components.	2 weeks
Sprint 10	Load Testing and Optimization	Conduct load testing and optimize system performance.	1 weeks
Sprint 11	Final Bug Fixes and UAT	Fix bugs from testing and perform User Acceptance Testing (UAT).	1 weeks
Sprint 12	Deployment and Documentation	Deploy system and create detailed documentation.	1 weeks

3. LITERATURE SURVEY

1. **Event-Driven API Gateways: Enabling Real-time Communication in Modern Microservices Architecture**

   Year: 2024

   About the Paper: This paper explores the critical role of event-driven API gateways in modern microservices architectures, focusing on enabling real-time communication and integration. It highlights the benefits of asynchronous communication, scalability, and elasticity that event-driven architectures bring to distributed systems. The study discusses the challenges and best practices for implementing these architectures, emphasizing the importance of components like message brokers and stream processing systems. By presenting case studies from companies like XYZ Corp and ABC Inc., the paper illustrates how event-driven API gateways can significantly improve system performance, reduce downtime, and enhance scalability. The insights provided underscore the importance of adopting event-driven principles to build resilient, high-performing systems that can efficiently handle real-time data.

2. **Designing Robust API Monitoring Solutions**

   Year: 2023

   About the Paper: This paper focuses on addressing the challenges associated with monitoring Application Programming Interface (API) calls in modern software systems. The authors identify six key challenges—transparency, recall, coverage, handling derived flows, relevant calls, and capturing output values—encountered when designing API monitoring tools. To overcome these, they propose SNIPER, a robust API monitoring solution that employs two distinct implementation approaches: Dynamic Binary Instrumentation (DBI) and hardware-assisted virtualization. These methods ensure comprehensive monitoring with high accuracy and minimal artifacts, even in adversarial environments. The system targets Windows environments and covers a wide array of library and system calls. SNIPER demonstrates its effectiveness through extensive testing on real-world programs and complex malware, contributing significantly to fields like security research, malware analysis, and program debugging.

3. **WebAPI Evolution Patterns: A Usage-Driven Approach**

   Year: 2023

   About the Paper: This paper investigates the evolution of Web APIs (WAPIs) by emphasizing the importance of consumer behavior in driving changes. By analyzing usage logs through process mining techniques, the authors aim to identify usage patterns that suggest the need for modifications, such as creating new parameters, merging endpoints, or addressing redundancy. The study was applied to real-world WAPIs in the education and health sectors, demonstrating its effectiveness in capturing valuable insights. Feedback from both consumers and providers confirmed the potential of this approach to optimize WAPI design and ensure it evolves in response to actual user behavior. The findings suggest that this method can systematically improve API structures, enhance usability, and better align with user needs.

4. **Data visualization and monitoring with Grafana and Prometheus**

   Year: 2021

   About the Paper: This paper discusses the implementation of visual monitoring for IT environments using Prometheus and Grafana. It compares these tools with existing systems like Zabbix and LibreNMS. The study aimed to improve data collection and visualization through these new systems without replacing the older ones. It explores the setup of Prometheus for data collection and Grafana for visualization. The work demonstrates how these tools can be utilized in different environments, with future improvements focusing on encryption and alert systems.

5. **Design, monitoring, and testing of microservices systems: The practitioners' perspective**

   Year: 2021

   About the Paper: The paper investigates how microservices systems are designed, monitored, and tested in the industry, using a mixed-methods approach involving surveys and interviews with 106 practitioners. The study identifies key strategies like Domain-Driven Design (DDD) and business capability for decomposing systems, and highlights monitoring practices such as resource usage and load balancing. It finds that microservices complexity challenges design, monitoring, and testing. The paper calls for solutions addressing security, complexity, and improved testing and monitoring strategies.

6. **Log-based software monitoring: a systematic mapping study**

   Year: 2021

   About the Paper: The paper "Log-based Software Monitoring: A Systematic Mapping Study" explores the role of logs in software monitoring, focusing on their use for detecting system behaviors, diagnosing issues, and improving system reliability. The study highlights the importance of automated log analysis tools and infrastructure to enhance developer productivity. The paper reviews machine learning techniques applied to log analysis for context-driven insights and anomaly detection, offering valuable information on how to process and analyze logs efficiently. These findings could be applied to your API project, enhancing log monitoring, anomaly detection, and alert systems for improved security.

7. **Tools and Benchmarks for Automated Log Parsing**

Year: 2019

About the Paper: This comprehensive evaluation study reviews 13 automated log parsers applied across 16 datasets from various domains, including distributed systems and operating systems. The paper highlights the unstructured nature of logs and the critical role of parsing them into structured data for effective anomaly detection, performance analysis, and system diagnostics. The study evaluates parsers like Drain, SLCT, and IPLoM based on accuracy, robustness, and efficiency. The results underline the importance of automated log parsing in managing modern systems with large-scale log data. This work also provides tools and benchmarks to guide future research and industry adoption of log parsing solutions.

8. **What Public Transit API Logs Tell Us about Travel Flows**

Year: 2016

About the Paper: This paper delves into the analysis of public transit API logs to understand travel patterns and commuter behavior. By utilizing the iRail API, a route planning tool for Belgium’s railways, the authors examine query logs spanning several years to uncover insights into travel flows and commuting trends. The study highlights the importance of visualizing data through tools like Origin-Destination (OD) matrices and chord diagrams to represent travel dynamics effectively. The findings reveal that Brussels acts as a central transit hub, with regional variations in travel patterns—Flanders exhibiting polycentric travel behaviors while Wallonia shows a monocentric pattern focused on Brussels. This research emphasizes the value of API logs in smart city applications, providing actionable insights for urban planning, mobility management, and real-time travel optimization.

9. **Malware Detection Systems Based on API Log Data Mining**

Year: 2015

About the Paper: This paper focuses on developing a malware detection system that analyzes dynamic API call logs to distinguish between benign and malicious programs. By leveraging data mining techniques and classification algorithms such as Naive Bayesian, Decision Trees (J48), and Support Vector Machines, the system achieves a high detection rate of 95% while maintaining low computational complexity through feature selection. The study demonstrates that dynamic analysis, particularly monitoring concealed behaviors using API calls, is more effective than traditional static analysis. The findings underscore the potential of using behavioral data mining for robust malware identification, particularly in systems where traditional signature-based methods fail.

S No	Author (Publisher)	Paper Title	Year	Parameters and Objectives of Paper
1	ResearchGate	Event-Driven API Gateways	2024	Explores event-driven API gateways for real-time communication, scalability, and elasticity in microservices.
2	IEEE	Designing Robust API Monitoring Solutions	2023	Emphasizes transparency and security in API monitoring for malware analysis and debugging.
3	Rediana Koçi	WebAPI Evolution Patterns: A Usage-Driven Approach	2023	Analyzes API usage logs for structural relationships, behavioral patterns, and metrics, supporting API evolution.
4	Leppänen	Data Visualization and Monitoring with Grafana & Prometheus	2021	Focuses on integrating Prometheus for data collection and Grafana for visualization to monitor networks.

5	Muhammad Waseem	Design, Monitoring, and Testing of Microservices Systems	2021	Explores design, monitoring, and testing in microservices, focusing on practitioner challenges.
6	Marieke Huisman	Log-based Software Monitoring: A Systematic Mapping Study	2021	Reviews logging practices, infrastructure, and analysis for reliability, addressing developer tooling challenges.
7	IEEE	Tools and Benchmarks for Automated Log Parsing	2019	Evaluates log parsers for accuracy, robustness, and efficiency in handling unstructured data.
8	Peter Colpaert	What Public Transit API Logs Tell Us About Travel Flows	2016	Uses API logs to analyze travel patterns, aiding urban planning and mobility decisions.
9	IEEE	Malware Detection Systems Based on API Log Data Mining	2015	Achieves high malware detection accuracy by analyzing dynamic API call behaviors.

S No	Paper Title	Algorithm Used	Observations and Impact on your Project
1	Event-Driven API Gateways	Event-driven architecture using message brokers and stream processing.	Enhances scalability, real-time communication, and system performance in your API monitoring framework.
2	Designing Robust API Monitoring Solutions	Dynamic Binary Instrumentation (DBI) and hardware-assisted virtualization.	Offers a framework for accurate API call monitoring, enhancing debugging and security measures in your project.
3	WebAPI Evolution Patterns: A Usage-Driven Approach	Process mining for log analysis and process modeling.	Aids in optimizing API structure, reducing redundancy, and enhancing usability based on consumer behavior.
4	Data Visualization and Monitoring with Grafana & Prometheus	Prometheus for data collection; Grafana for visualization.	Enhances data insight capabilities and informs the use of tools for real-time monitoring and alerting in your API project.
5	Design, Monitoring, and Testing of	None specifically; emphasizes	Guides design and testing strategies for the microservices

	Microservices Systems	architectural patterns and practices.	architecture in your project.
6	Log-based Software Monitoring: A Systematic Mapping Study	Machine learning for contextual log analysis.	Provides insights for automating log processing and identifying suspicious activities, enhancing your API security logging and monitoring project.
7	Tools and Benchmarks for Automated Log Parsing	Clustering (Drain, LogSig) and frequent pattern mining (SLCT, IPLoM).	Reinforces efficient log parsing for anomaly detection, informing improved log management strategies.
8	What Public Transit API Logs Tell Us About Travel Flows	Data aggregation, spatial grouping, OD matrices, and chord diagrams.	Highlights the value of analyzing patterns in API logs, applicable to detecting usage trends in your API monitoring solution.
9	Malware Detection Systems Based on API Log Data Mining	Naive Bayes, Decision Tree (J48), and Support Vector Machine (SVM).	Suggests incorporating behavioral analysis for detecting anomalies in API call patterns, boosting security monitoring.

4. CONCLUSION

In conclusion, our work on the **APILogGuard Security Events** project has laid a strong foundation by addressing critical challenges in API security and operational efficiency. By thoroughly analyzing the vulnerabilities in API infrastructures, such as brute force attacks, unusual IP access, and suspicious behavior patterns, we established a clear problem statement. An in-depth literature survey provided insights into state-of-the-art practices like event-driven architectures, advanced log parsing methods, and data visualization techniques using tools like Prometheus and Grafana. These insights have guided our project's design to integrate real-time monitoring, automated alerts, and robust analytics for enhanced security.

Moving forward, the project's emphasis on scalability, user-friendly middleware, and integration with modern SIEM practices will ensure it meets the dynamic needs of API-driven systems. This comprehensive approach not only addresses current security concerns but also anticipates future challenges by embedding learning-based mechanisms for continuous improvement. With this groundwork in place, we are poised to implement a solution that will strengthen API defenses, enhance operational transparency, and establish a benchmark in secure API management systems.

5. REFERENCES

1. https://www.researchgate.net/profile/Anusha-Kondam-2/publication/382621307_Event-Driven_API_Gateways_Enabling_Real-time_Communication_in_Modern_Microservices_Architecture/links/66a531a6c6e41359a843e4f7/Event-Driven-API-Gateways-Enabling-Real-time-Communication-in-Modern-Microservices-Architecture.pdf

2. https://ieeexplore.ieee.org/document/9645216

3. https://www.sciencedirect.com/science/article/pii/S0164121223000043

4. https://www.theseus.fi/handle/10024/512860

5. https://www.sciencedirect.com/science/article/abs/pii/S0164121221001588

6. https://peerj.com/articles/cs-489/

7. https://ieeexplore.ieee.org/document/8804456

8. https://dl.acm.org/doi/abs/10.1145/2872518.2891069

9. https://ieeexplore.ieee.org/document/7273364