# ITERATIVE LOG ANALYSIS TOOL FOR VISUAL REPRESENTATION IN DETECTING MALICIOUS ACTIVITIES

**S.Saravana Kumar**, Department of CSE (Cyber Security), Dr.Mahalihngam College of Engineering and Technology, Coimbatore, India
saravanacs84@gmail.com
**P.Harishini**, Department of CSE (Cyber Security), Dr.Mahalingam College of Engineering and Technology, Coimbatore, India
harishiniraj2020@gmail.com
**S. Sathya Shalini**, Department of CSE (Cyber Security), Dr.Mahalingam College of Engineering and Technology, Coimbatore, India
vijayakumart@drmcet.ac.in
**Dr. P. Vivekanandan**, Department of CSE (Cyber Security), Dr.Mahalingam College of Engineering and Technology, Coimbatore, India
drpvivekanandan@gmail.com

**Abstract**

Log analysis is the process of converting raw or unstructured log files into structured data and making intelligent decisions on that structured data. In every field like software testing, the analysis of log files is designed to monitor and check application performance. Logs are unstructured text lines containing systematic information regarding application work and actions such as IP Address, date, time, viewed sites, potential domains, status code, components, levels, nodes, query information, loading-time, user-agent, and port-number. Logs include several types such as INFO, WARNING, FATAL, SEVERE, and ERROR. The Log Analyzer Tool is a Python-based application designed to analyze log files for suspicious activities including malware, unauthorized access, phishing attempts, file tampering, security breaches, and more. The tool works across macOS, Windows, and Linux, offering a user-friendly graphical interface for log file selection and scan initiation. This project represents an industry-level, production-grade solution that strengthens cybersecurity infrastructure, reduces manual monitoring efforts, improves incident response time, enhances system reliability, and protects sensitive digital assets.

**Keywords** System Monitoring, Log File Analytics, Performance Analysis, Security Event Detection, Visual Representation

## I. Introduction

In modern computing environments, systems, servers, and applications continuously generate log files that document operational events, user interactions, transaction records, performance metrics, and error information. These logs function as a critical source of system intelligence, enabling administrators to monitor application behavior, diagnose faults, and maintain infrastructure reliability. From a cybersecurity perspective, log files provide indispensable forensic evidence and behavioral insights that help identify unauthorized access attempts, abnormal usage patterns, malicious payload execution, privilege escalation attempts, and system misconfigurations.

Effective log analysis enables organizations to proactively detect security incidents before they escalate into major breaches. Authentication logs can reveal brute-force attacks, web server logs may indicate SQL injection attempts or directory traversal activity, and system logs can highlight configuration anomalies or service failures. Therefore, structured and systematic log examination forms the backbone of incident detection, threat hunting, and compliance auditing processes.

However, the exponential growth in data generation has significantly increased the volume and complexity of log files. Large-scale systems may generate thousands or even millions of log entries per day. Manually reviewing such extensive datasets is both time-consuming and prone to oversight. Human-based analysis often struggles to identify subtle attack patterns that are distributed across multiple entries or occur within short time intervals. As a result, delays in identifying security threats can increase organizational risk exposure.

Although various commercial log management and Security Information and Event Management (SIEM) platforms are available, these solutions often require substantial computational resources, advanced configuration expertise, and high licensing costs. Small-scale organizations, educational institutions, and individual developers may find such systems financially or operationally impractical. Additionally, enterprise-grade platforms may introduce complexity that is unnecessary for localized or offline analysis scenarios.

To address these challenges, this project proposes the development of an automated and iterative Log Analyzer Tool implemented using Python. The proposed solution focuses on offline processing of structured log files, enabling efficient data extraction, transformation, statistical computation, and visualization without dependency on heavy infrastructure or continuous network connectivity. The system leverages Python's robust ecosystem, including libraries for regular expression-based text parsing, structured data manipulation, and graphical representation. decision-making processes. Security analysts and system administrators can quickly identify abnormal trends, high-frequency request sources, or error surges without manually scanning thousands of log entries.

Furthermore, the proposed solution is designed to be lightweight, modular, and adaptable. It provides a cost-effective alternative for developers, system administrators, academic researchers, and cybersecurity students who require structured log analysis capabilities without enterprise-level deployment complexity. The modular architecture also allows future enhancements such as real-time log streaming, machine learning-based anomaly detection, threshold-driven alerting mechanisms, and integration with centralized monitoring platforms.

## II. Related Work

Log analysis has been widely studied in the fields of system administration and cybersecurity over the past decade, with significant research and tool development reported between 2015 and 2023. Traditional approaches rely on command-line utilities such as grep, awk, and sed, which are effective for simple searches but lack scalability, automation, and visualization capabilities when handling large log datasets. Advanced platforms such as Splunk, ELK Stack, and Graylog provide real-time analytics and interactive dashboards; however, they involve complex deployment procedures, high resource consumption, and substantial operational overhead.

Recent research, particularly from 2020 onwards, has emphasized the need for lightweight, customizable log analysis solutions suitable for educational, forensic, and small-scale environments. Python-based tools have gained popularity due to their flexibility, ease of development, and availability of powerful data-processing and visualization libraries. Unlike existing large-scale platforms, the proposed system introduces a lightweight Python-based log analyzer that supports both command-line output and graphical user interface (GUI) visualization, enabling efficient offline analysis with minimal infrastructure requirements. The novelty of this project lies in its combined focus on automated parsing, visual interpretation, and resource-efficient execution, making it well suited for academic use, offline forensic analysis, and small organizations that require effective log analysis without complex setup or high cost.

## III. Proposed Methodology

The proposed Automated Iterative Log Analysis Tool follows a structured and modular workflow consisting of log ingestion, parsing, iterative analysis, and visualization.

### A. Log Collection and Parsing

Log files are provided to the system as input in standard formats such as Apache web server logs or custom application logs. The system utilizes regular expression-based pattern matching to extract relevant attributes from raw textual entries.

Key extracted fields include:

• IP addresses

• Timestamps

• Request methods (e.g., GET, POST)

• URLs accessed

• HTTP status codes

• Response sizes

The extracted information is transformed into structured data formats, enabling systematic storage and further analytical processing.

**B. Iterative Log Analysis**

Once structured, the log data undergoes iterative statistical evaluation. The extracted dataset is stored in tabular form to enable efficient filtering, grouping, and aggregation operations.

The analysis module computes metrics such as:

• Request frequency per IP address

• Distribution of status codes

• Identification of frequently occurring IP addresses

• Error occurrence trends

Through iterative evaluation, the system identifies abnormal access patterns, repeated failed attempts, and irregular traffic behaviors that may indicate malicious activity. This structured approach enhances anomaly detection accuracy compared to manual inspection methods.

**C. Visualization**

To improve interpretability, the analyzed results are presented using graphical representations such as bar charts and summarized statistical reports. Visualization simplifies the understanding of complex log patterns by converting textual data into intuitive graphical formats.

These visual outputs enable administrators and security analysts to quickly identify abnormal trends, detect potential security incidents, and make informed decisions without manually reviewing extensive log files.

## IV. System Architecture

The proposed Log Analyzer Tool follows a modular and layered system architecture designed to ensure clarity, scalability, and ease of maintenance. The architecture is composed of four primary modules, each responsible for a specific stage in the log processing pipeline. This structured separation of responsibilities enhances flexibility and simplifies future upgrades or integration with advanced analytical components.

**A. Input Module**

The Input Module serves as the entry point of the system. It is responsible for handling log file selection, format verification, and preliminary validation checks. The module ensures that the provided log files conform to supported structures such as Apache server logs or custom-defined application logs.

Validation mechanisms are implemented to detect incomplete entries, unsupported formats, or corrupted

**B. Identify the Headings**

Headings, or heads, are organizational devices that guide the reader through your paper. There are two types: component heads and text heads.

Component heads identify the different components of your paper and are not topically subordinate to each other. Examples include Acknowledgments and References and, for these, the correct style to use is "Heading 5". Use "figure caption" for your Figure captions, and "table head" for your table title. Run-in heads, such as "Abstract", will require you to apply a style (in this case, italic) in addition to the style

files before further processing. By performing early-stage validation, this module prevents downstream errors and improves overall system reliability.

**C. Parsing Module**

The Parsing Module processes raw log entries and converts unstructured textual data into a structured format suitable for analysis. Regular expression-based extraction techniques are employed to identify and isolate critical log attributes, including IP addresses, timestamps, request methods (GET, POST, etc.), accessed URLs, HTTP status codes, and response sizes.

The extracted data is organized into structured data representations such as tabular or dictionary-based formats. This structured transformation enables efficient querying, filtering, and aggregation during the analysis phase.

**D. Analysis Module**

The Analysis Module performs statistical computations and anomaly identification based on the structured dataset. It applies iterative evaluation techniques to derive meaningful insights, such as:

- Frequency of requests per IP address
- Distribution of HTTP status codes
- Identification of repeated failed login attempts
- Detection of unusually high traffic patterns

Through systematic aggregation and comparison, the module assists in recognizing suspicious behaviour, potential brute-force attempts, abnormal traffic spikes, and misconfiguration-related errors. This analytical component forms the core intelligence layer of the system.

**E. Visualization Module**

The Visualization Module transforms analytical results into graphical representations for intuitive interpretation. Statistical summaries are presented using visual formats such as bar charts and summarized reports, allowing users to quickly identify patterns and irregularities within large datasets.

By converting numerical outputs into visual insights, this module enhances usability and supports faster decision-making for administrators and security analysts.

The modular design significantly improves maintainability, scalability, and adaptability. Each module operates independently yet integrates seamlessly within the processing pipeline. This separation of concerns enables easier debugging and performance optimization, independent module enhancement without affecting the entire system, and simplified integration of future extensions.

## V. Algorithm: Iterative Log Analysis and Visualization

The proposed algorithm follows a systematic approach to log analysis:

Input: Log file L

Output: Malicious activity report and visual representation

Step 1: Initialization

- Start the log analysis process
- Initialize required data structures for storing parsed log entries and detection results
- Load necessary libraries and configure processing environment

Step 2: Log File Input

- Accept the sample log file L through either Command-Line Interface (CLI) or Graphical User Interface (GUI)
- Validate the log file format and check for structural consistency
- If the file is invalid or corrupted, terminate the process with an error message

Step 3: Detection Pattern Loading

- Load predefined regular expression patterns associated with common malicious behaviors
- Allow optional loading of user-defined detection patterns
- Detection patterns may include indicators related to malware-related signatures, unauthorized access attempts, phishing-related URLs, file tampering activities, and general security breaches

Step 4: Iterative Parsing and Analysis

- Read the log file L line by line
- For each log entry, apply parsing rules using regular expressions to extract key attributes (IP address, timestamp, request type, status code, etc.)

- Compare extracted attributes against loaded detection patterns
- If a match is found, flag the entry as suspicious
- Store flagged entries in a separate malicious activity dataset
- Aggregate statistical metrics such as frequency of suspicious IP addresses, repeated failed login attempts, and distribution of abnormal status codes
- Continue the iteration until all log entries are processed

Step 5: Visualization and Termination

- Generate graphical representations of the analysis results
- Produce comprehensive malicious activity report
- Terminate the process

## VI. User-Facing Problem

Log files are a primary source of information for monitoring system operations and identifying security incidents. However, end users face significant difficulty in analyzing log data due to its large volume, continuous generation, and unstructured nature. Manual analysis using basic command-line tools is inefficient, time-consuming, and does not provide a clear overview of system behaviour.

Existing enterprise log analysis platforms offer advanced features but introduce practical challenges such as complex setup, high resource consumption, and licensing costs. These factors limit their usability for students, researchers, and small organizations, especially in offline or academic environments.

As a result, users lack a simple and efficient log analysis solution that balances functionality and usability. There is a need for a lightweight system that enables both command-line analysis for quick inspection and graphical visualization for improved understanding, while avoiding heavy infrastructure requirements.

## VII. Experimental Results and Discussion

The proposed system was tested using both sample and real-world log datasets. The tool successfully parsed large log files and extracted relevant attributes with high accuracy. Analysis results highlighted frequent error codes, peak traffic periods, and suspicious IP addresses.

Visual representations significantly improved interpretability compared to textual summaries. The system demonstrated efficient performance with minimal resource usage, making it suitable for offline forensic analysis and academic applications.

These results confirm that automated and visual log analysis enhances security monitoring and operational efficiency. The tool provides analysis results through both command-line output and a graphical user interface (GUI). The command-line interface displays concise statistical summaries and key findings,

enabling quick inspection and scripting-based usage. In parallel, the GUI presents the analyzed data using interactive charts and visual reports, which significantly improve interpretability compared to plain textual summaries.

## VIII. Conclusion and Future Work

This paper presented an Automated Iterative Log Analysis Tool for detecting malicious activities through visual representation. By automating log parsing and analysis, the system reduces manual effort and improves threat detection efficiency. The lightweight and extensible design enables deployment in various environments without complex infrastructure requirements.

Future work includes implementing real-time log monitoring, integrating machine learning techniques for anomaly detection, and developing an enhanced graphical user interface to improve usability.

## References

[1] D. Beazley and B. K. Jones, Python Cookbook, 3rd ed. Sebastopol, CA, USA: O'Reilly Media, 2013.

[2] M. Lutz, Learning Python, 5th ed. Sebastopol, CA, USA: O'Reilly Media, 2013.

[3] K. Scarfone and P. Mell, Guide to Vulnerability Assessment, NIST Special Publication 800-115. Gaithersburg, MD, USA: National Institute of Standards and Technology (NIST), 2007.

[4] OWASP Foundation, "Logging and Monitoring." [Online]. Available: https://owasp.org. [Accessed: Feb. 26, 2026].

[5] National Institute of Standards and Technology (NIST), Guide to Computer Security Log Management, NIST Special Publication 800-92. Gaithersburg, MD, USA, 2006.

[6] A. Chuvakin, K. Schmidt, and C. Phillips, Logging and Log Management: The Authoritative Guide to Understanding the Concepts Surrounding Logging and Log Management. Waltham, MA, USA: Syngress, 2013.

[7] B. Liu, Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data, 2nd ed. Berlin, Germany: Springer, 2011.

[8] C. Tankard, "Advanced persistent threats and how to monitor and deter them," Network Security, vol.2011, no. 8, pp. 16-19, Aug. 2011.

[9] S. Axelsson, "Intrusion detection systems: A survey and taxonomy," Dept. Comput. Eng., Chalmers Univ., Sweden, Tech. Rep., 2000.

[10] W. Lee and S. J. Stolfo, "Data mining approaches for intrusion detection," in Proc. 7th USENIX Security Symp., San Antonio, TX, USA, 1998, pp. 79-93.

[11] Splunk Inc., "Splunk Enterprise Security Overview." [Online]. Available: https://www.splunk.com. [Accessed: Feb. 26, 2026].

[12] Elasticsearch B.V., "Elastic Stack (ELK Stack) Documentation." [Online]. Available: https://www.elastic.co. [Accessed: Feb. 26, 2026].

[13] Graylog, Inc., "Graylog Open Source Log Management." [Online]. Available: https://www.graylog.org. [Accessed: Feb. 26, 2026].

[14] M. Du, F. Li, G. Zheng, and V. Srikumar, "DeepLog: Anomaly detection and diagnosis from system logs through deep learning," in Proc. ACM Conf. Comput. Commun. Security (CCS), 2017, pp. 1285-1298.

[15] H. He, J. Zhu, Z. Zheng, and M. Lyu, "Drain: An online log parsing approach with fixed depth tree," in Proc. IEEE Int. Conf. Web Services (ICWS), 2017, pp. 33-40.