M Lakshmi

24MSRDF016

**Report on Forensic Log Analyzer CLI Tool**

Day 1 – Reverse Engineering the Log Format

The project began with an exploration of the .vlog file format. Sample session logs were reviewed to understand their structure, pattern, and metadata components such as timestamps, event types, user actions, and target paths. A key observation was that each line followed a consistent format including a hexadecimal ID, timestamp, event type, user or IP, and a target.

Day 2 – Data Extraction Parser in Python

Python script was developed using regular expressions (re module) to parse structured ‘.vlog’ entries. The parser successfully extracted fields such as timestamp, event type, action, actor (user/IP), and target path. It also differentiated valid entries from malformed ones, saving invalid lines to a separate logs.csv for auditability. This script ensured that all clean logs could be transformed into structured data for further analysis. And we get malformed logs and parsed logs csv files as output.

Day 3 – Log Categorization & Timeline Generation

The structured log entries were classified into categories: process, file, and user events based on the XR- tag in the event type. A timeline was generated by sorting all events chronologically and saving them to timeline.csv. This timeline served as a clean, organized dataset representing system activity, which would be crucial for anomaly detection and visualization. And as output we generated timline and invalid logs csv files.

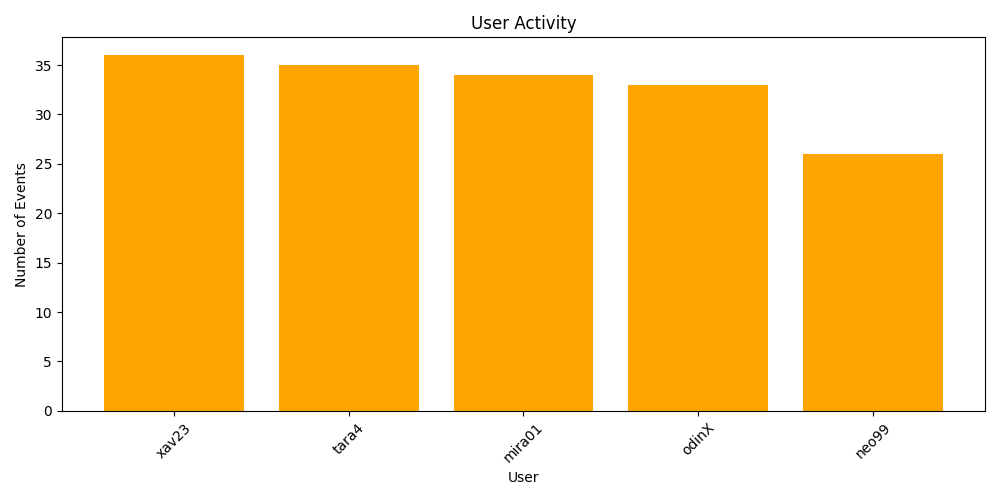
Day 4 – Identify Suspicious Activity

A rule-based approach was implemented to identify suspicious activity. Specifically, the tool flagged scenarios where a user executed a KILL command on a shadow process (XR-SHDW) followed by a DELETE operation (XR-DEL) within a 10-second window. These sequences were considered potentially malicious and were saved in a separate file, flaggedanomalies.csv. This enabled quick identification of harmful behavior hidden in routine logs.

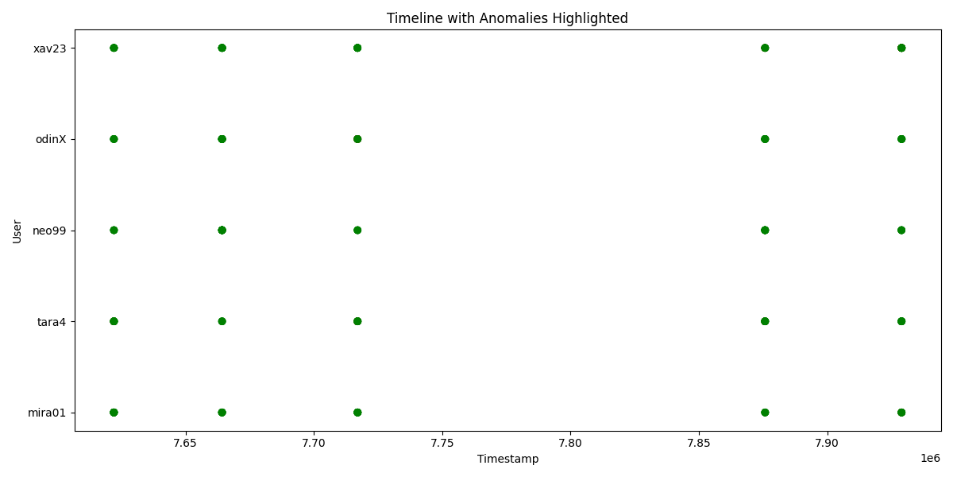
Day 5 – Create Visualization

Visual representations of the log data were created using Matplotlib and Plotly. Graphs such as event\_frequency.png showed the volume of log entries over time, user\_activity.png highlighted activity per user, and anomalies\_highlighted.png visually marked suspicious sequences. Interactive HTML versions were also generated using Plotly for deeper insights, enabling dynamic exploration of user patterns and event trends.The output was html and png files.

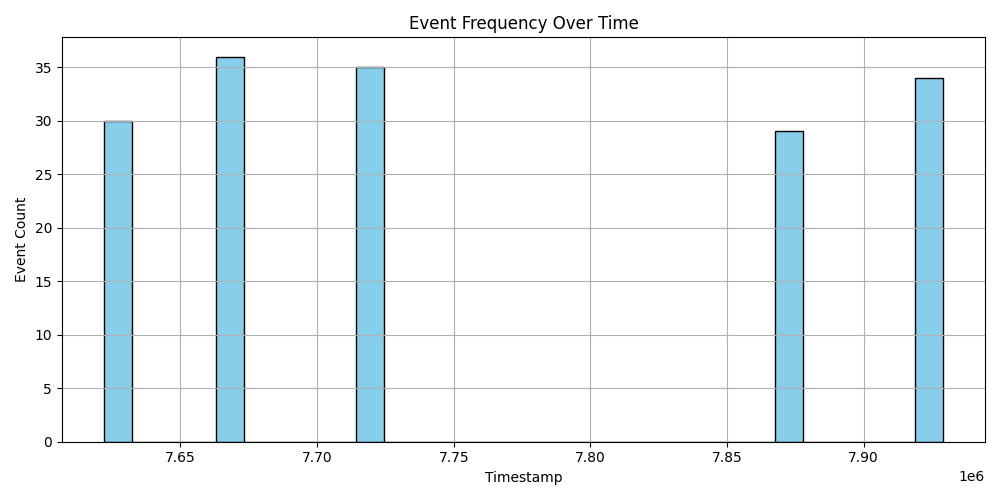
User activity-



Anomalies highlighted-



Event frequency-



Day 6 – Integrate into CLI Tool

All scripts were integrated into a single command-line tool, cli\_tool.py, using Python’s argparse library. The CLI accepted input directory paths and optional flags like --timeline, --alerts, and --summary. This made the tool flexible and easy to use in forensic labs. The --timeline flag generated structured and invalid logs, --alerts ran anomaly detection and saved plots, while --summary printed key insights to the terminal.

Conclusion:

The Forensic Log Analyzer CLI Tool project we had managed to convert raw ‘.vlog’ data into actionable intelligence for forensic investigation. By painstakingly reverse-engineering the format of the logs, building a resilient parser, and implementing fairly advanced anomaly detection rules, the tool automates a few of the more mundane stages of log analysis. With highly efficient data visualization and local command-line interface, which is great for the investigators so that they can outrightly identify questionable behavior, understand user behavior patterns, and use both the evidence structures and visualization evidence to speedily conclude the investigation, this tool is very much a step forward toward automating and enhancing forensic readiness.