**Day 1: Reverse Engineering the Log Format**

I began by manually analyzing raw .vlog session files to understand their hidden structure. Through careful inspection, I identified repeating patterns and fields like log ID, timestamp, event type, actor type, action, and target path. Regular expressions were crafted to extract this structure for parsing.

**Tools & Libraries Used:**

* re (regular expressions) for pattern extraction
* VS Code / any text editor
* Sample .vlog files for analysis

**Day 2: Data Extraction Parser in Python**

I developed a Python script that parses each .vlog file line-by-line and structures the entries into a clean DataFrame. Malformed or corrupt lines were skipped to ensure the parser could handle real-world logs gracefully. Once parsed, the cleaned data was exported directly into an **Excel spreadsheet** for easy access and review.

**Tools & Libraries Used:**

* os and re for file handling and pattern matching
* datetime for timestamp conversion
* pandas for structured storage and manipulation
* pandas.DataFrame.to\_excel() for exporting to Excel (.xlsx)

**Day 3: Log Categorization & Timeline Generation**

Using the structured data, I categorized logs by their event types (e.g., user, file, process, IP). A full timeline was created by sorting the events chronologically based on their timestamp. The timeline was then exported to a CSV file to assist with temporal analysis.

**Tools & Libraries Used:**

* pandas for event grouping and timeline sorting
* datetime for timestamp formatting
* CSV export via pandas.DataFrame.to\_csv()

**Day 4: Identify Suspicious Activity**

I implemented rule-based logic to flag anomalous behaviors in the logs, such as:

* Shadow load followed by deletion
* Creation followed by deletion of the same file
* Execution of suspicious binaries
* Deletion of executables in sensitive paths

Anomalies were extracted and saved into both a .csv report and a **styled .html table** for easier viewing.

**Tools & Libraries Used:**

* pandas and datetime for filtering and rule checks
* Custom logic for rule-based anomaly detection
* pandas.to\_html() for generating HTML anomaly reports

**Day 5: Create Visualization**

To visualize patterns and anomalies, I created plots showing event frequency over time. Static charts were generated using matplotlib, while optional interactive graphs were made using plotly. These visuals helped spot spikes, patterns, and user activity trends.

**Tools & Libraries Used:**

* matplotlib for static bar charts (.png)
* plotly for interactive dashboards (.html)
* pandas for preparing visualization data

**Day 6: CLI Tool Integration & Report Generation**

Everything was wrapped into a CLI tool using argparse. The tool accepts a log directory and various flags to control output: --summary, --timeline, --alerts, and --pdf. A comprehensive **PDF report** was also generated using fpdf, which included summary statistics, action breakdowns, anomaly highlights, and the timeline chart.

**Tools & Libraries Used:**

* argparse for command-line argument parsing
* fpdf for creating PDF reports
* pandas and matplotlib for data and embedded visuals
* Modularized Python scripts to separate logic cleanly