# Personal Report: Mini SIEM (Security Information & Event Management)

## Project Title:

**Mini SIEM: Real-Time Log Monitor & Brute-Force Detection**

---

## Overview

As part of my hands-on cybersecurity development, I built a Python-based Mini SIEM system. The project focuses on real-time log monitoring and brute-force attack detection, with a live dashboard interface built using Flask and styled with Tailwind CSS. This project simulates a real-world lightweight Security Information and Event Management tool.

The main components include:

- A **log monitor** that watches system logs for suspicious activity
- A **brute-force detection engine** using pattern tracking and time windows
- A **Flask web dashboard** to display live alerts
- A **brute-force attack simulator** to test the detection logic

---

## Features

- Real-time file monitoring using multithreading
- Detection of brute-force login patterns based on failed attempts within a short time window
- Threshold alerting and log tracking
- Live dashboard with auto-refresh capability
- Alert history stored in memory and displayed in a styled UI

- Manual simulation of failed logins to test the system

---

## Development Process & Errors

The initial setup involved writing a real-time log monitor that scanned `auth.log` for repeated failed login attempts from the same IP address. I used a dictionary to store timestamps per IP and compared them against a time window to flag potential brute-force activity.

When I began testing with the brute-force simulator, I noticed that no logs were appearing in the `auth.log` file. Despite no errors in the script, the file was simply not being updated.

This led me to debug the simulator script. I added a `try-except` block and introduced print statements to verify if log entries were being written. I also ensured that the `logs/` directory existed before attempting to write. After adding the following check:

python
CopyEdit
`os.makedirs(os.path.dirname(LOG_FILE), exist_ok=True)`

The issue was resolved. The log simulator was now reliably writing failed login attempts, and the monitor was detecting them in real time.

I also made sure the log lines were properly formatted — specifically, that the IP address was the last item in each line — since the monitor extracts it using `line.split()[-1]`.

---

## Final Touches

To improve usability:

- I added confirmation prints like `Written log: ...` and `Simulation complete` after each brute-force simulation.

- I set up the Flask dashboard to auto-refresh every few seconds, so new alerts could be seen without manual reloads.

- I modularized the components so each part of the system (monitoring, simulation, and UI) could be improved independently in the future.

---

## Reflections

This project helped reinforce my understanding of real-time security monitoring, basic SIEM logic, and the use of threading in Python. It also forced me to carefully debug subtle problems, like directory issues or silent write failures — the kind of detail that's easy to miss in security tooling.

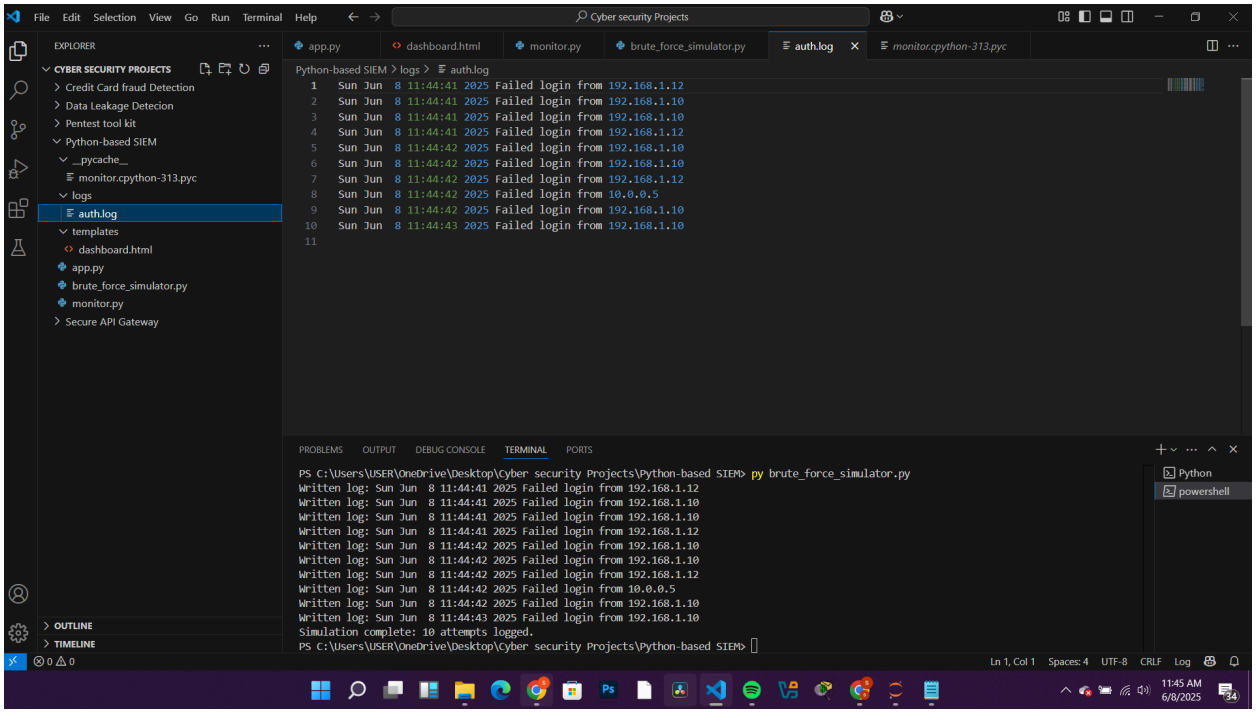
Next steps:

- Extend alert storage using SQLite
- Add user authentication to the dashboard
- Incorporate regex-based detection for other types of log anomalies
- Package the system in Docker for easier deployment

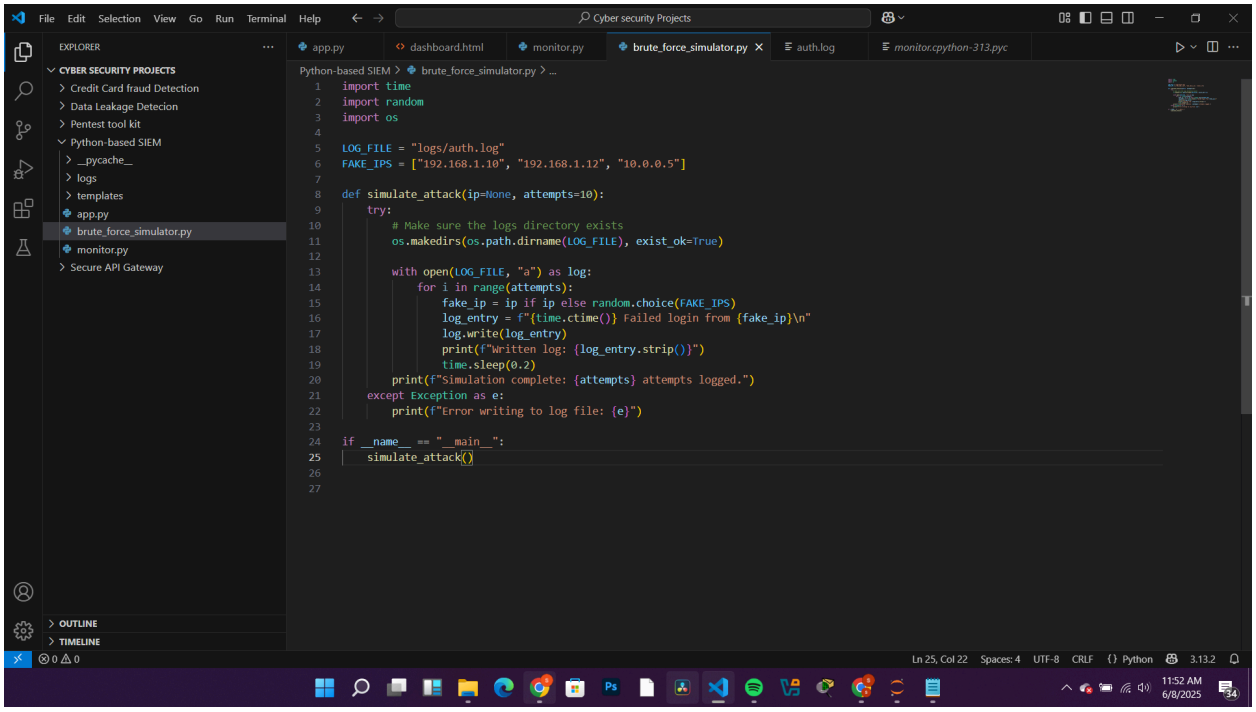
---

## Photographic Evidence

Auth log file



## Brute Force Code




## Flask Website