

# 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

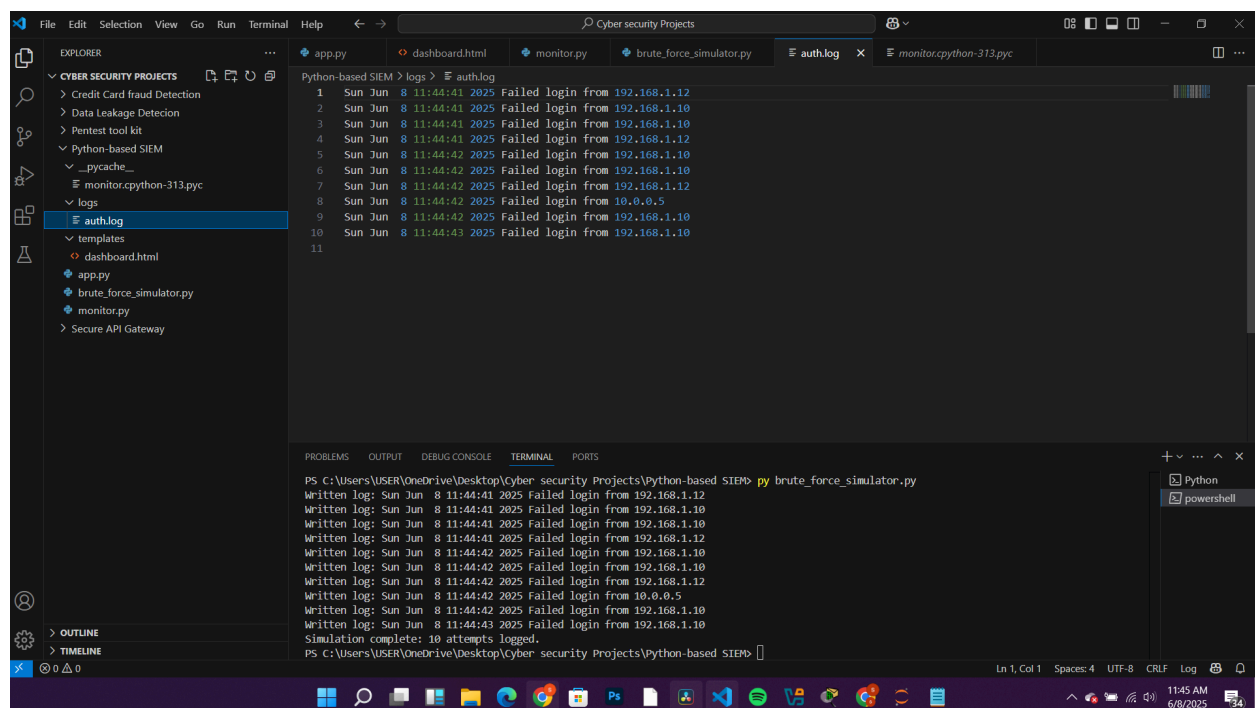
This project helped reinforce my understanding of real-time security monitoring, basic SIEM logic, and the use of threading in Python. It also forced me to carefully debug subtle problems, like directory issues or silent write failures — the kind of detail that's easy to miss in security tooling.

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

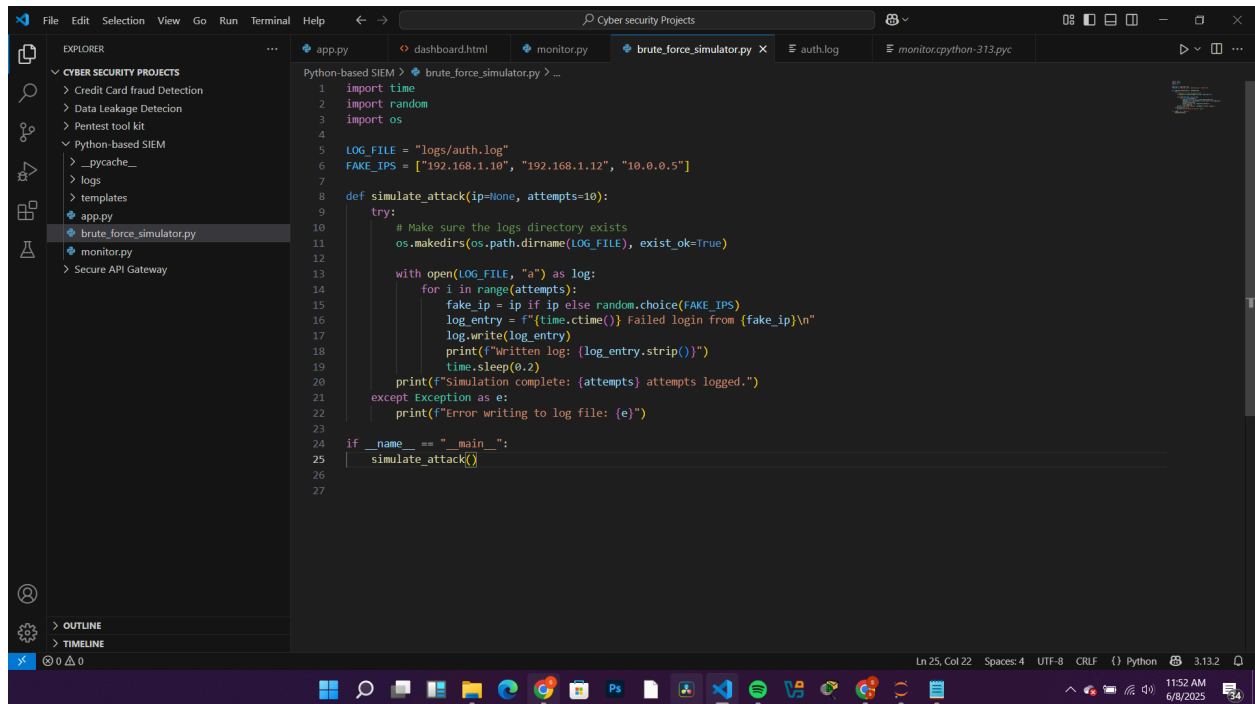
### Auth log file



```
Python-based SIEM > logs > auth.log
1 Sun Jun 8 11:44:41 2025 Failed login from 192.168.1.12
2 Sun Jun 8 11:44:41 2025 Failed login from 192.168.1.10
3 Sun Jun 8 11:44:41 2025 Failed login from 192.168.1.10
4 Sun Jun 8 11:44:41 2025 Failed login from 192.168.1.12
5 Sun Jun 8 11:44:42 2025 Failed login from 192.168.1.10
6 Sun Jun 8 11:44:42 2025 Failed login from 192.168.1.10
7 Sun Jun 8 11:44:42 2025 Failed login from 192.168.1.12
8 Sun Jun 8 11:44:42 2025 Failed login from 10.0.0.5
9 Sun Jun 8 11:44:42 2025 Failed login from 192.168.1.10
10 Sun Jun 8 11:44:43 2025 Failed login from 192.168.1.10
11
```

```
PS C:\Users\USER\OneDrive\Desktop\Cyber security Projects\Python-based SIEM> py brute_force_simulator.py
Written log: Sun Jun 8 11:44:41 2025 Failed login from 192.168.1.12
Written log: Sun Jun 8 11:44:41 2025 Failed login from 192.168.1.10
Written log: Sun Jun 8 11:44:41 2025 Failed login from 192.168.1.10
Written log: Sun Jun 8 11:44:41 2025 Failed login from 192.168.1.12
Written log: Sun Jun 8 11:44:42 2025 Failed login from 192.168.1.10
Written log: Sun Jun 8 11:44:42 2025 Failed login from 192.168.1.10
Written log: Sun Jun 8 11:44:42 2025 Failed login from 192.168.1.12
Written log: Sun Jun 8 11:44:42 2025 Failed login from 10.0.0.5
Written log: Sun Jun 8 11:44:42 2025 Failed login from 192.168.1.10
Written log: Sun Jun 8 11:44:43 2025 Failed login from 192.168.1.10
Simulation complete: 10 attempts logged.
PS C:\Users\USER\OneDrive\Desktop\Cyber security Projects\Python-based SIEM>
```

## Brute Force Code



```
Python-based SIEM > brute_force_simulator.py > ...
1 import time
2 import random
3 import os
4
5 LOG_FILE = "logs/auth.log"
6 FAKE_IPS = ["192.168.1.10", "192.168.1.12", "10.0.0.5"]
7
8 def simulate_attack(ip=None, attempts=10):
9     try:
10         # Make sure the logs directory exists
11         os.makedirs(os.path.dirname(LOG_FILE), exist_ok=True)
12
13         with open(LOG_FILE, "a") as log:
14             for i in range(attempts):
15                 fake_ip = ip if ip else random.choice(FAKE_IPS)
16                 log_entry = f"{time.ctime()} Failed login from {fake_ip}\n"
17                 log.write(log_entry)
18                 print(f"Written log: {log_entry.strip()}")
19                 time.sleep(0.2)
20             print(f"Simulation complete: {attempts} attempts logged.")
21     except Exception as e:
22         print(f"Error writing to log file: {e}")
23
24 if __name__ == "__main__":
25     simulate_attack()
26
27
```

## Flask Website

