Guardian SIEM - The Complete Project Guide

Version: 2.0 (Async + SOAR + SNMP)

Chapter 1: Project Motive

1.1 What is this Project?

Guardian SIEM is a modern, lightweight Security Information and Event Management (SIEM) system built with Python, Flask, Google Gemini, and Scapy. It functions as the central brain of a SOC (Security Operations Center), designed to collect logs from multiple sources (like Windows PCs, routers, network traffic), analyze them, correlate events, and execute automated responses (SOAR).

1.2 Why is this different from other SIEM tools?

While professional SIEM tools like Splunk or QRadar are powerful, they are often incredibly expensive and complex to set up. The core philosophy of Guardian SIEM is **Simplicity** and **Intelligence**.

Key Differences:

1. **Direct AI Integration:** In traditional SIEMs, AI is often a costly add-on. In Guardian SIEM, Google Gemini is a core component. It doesn't just search logs; it understands natural language commands (in English or Bengali), correlates events, and analyzes alert severity.
2. **True SOAR Capability:** This project is not just a SIEM (Information) but a SOAR (Action) tool. When you give the AI a command like block 1.2.3.4, the server automatically executes a netsh advfirewall command to block that IP on the Windows Firewall. This "Human-in-the-Loop" automation is a highly advanced feature.
3. **Transparency (Not a "Black Box"):** Every line of code in soc_dashboard.py is under your control. You can easily add new correlation rules, new alert types, or new agents to monitor any device you want.
4. **Lightweight:** The entire system runs on Python, Flask, and SQLite, requiring no heavy server infrastructure.

Chapter 2: System Architecture

Guardian SIEM consists of three main components and one library integration:

1. **soc_dashboard.py (The Brain/Server):**
   - This is the main web server built with Flask.
   - It receives logs from agents via the /api/logs endpoint.
   - It manages the SQLite (logs.db) database.
   - It runs the **Correlation Engine** (e.g., Brute Force Alert).
   - It hosts the **Gemini AI Analyst** (for AI queries) and the **SOAR Engine** (for IP blocking).
   - It runs the **Scapy Sniffer** to monitor all network traffic.
   - It manages **Async Worker Threads** (background workers) to enrich logs with GeoIP, DNS, and AbuseIPDB (Threat Intel) data.
   - It runs an **SNMP Trap Receiver** on UDP Port 162 to catch critical alerts from network devices.
2. **agent.py (The Windows Agent):**
   - A lightweight Python script that runs on client PCs (like your desktop).
   - It uses the pywin32 library to read Windows Event Logs (Security, Application, System).
   - It collects new logs every 10 seconds and sends them to the soc_dashboard.py server.
3. **snmp_agent.py (The Router Agent):**
   - An independent Python script that also acts as an agent.
   - It uses the pysnmp library to "poll" (query) your network devices (like the MikroTik router) every 20 seconds.
   - It gathers health data (like Uptime, Traffic In/Out) and sends it to the soc_dashboard.py server as Event ID 9001.

Chapter 3: Software Requirements

To run this project, you must have a virtual environment (venv) set up in your F:\SOC_Project folder.

1. **Python 3.10+**
2. **VMware Workstation Player** (To run the MikroTik CHR virtual router).
3. **Npcap:** Required by Scapy to capture network traffic (this should have been installed with GNS3 or Wireshark, but can be installed separately).
4. **Python Libraries:** Run the following command **inside your activated venv** to install all required libraries at once:
   ```
   pip install flask requests scapy google-generativeai pysnmp pywin32
   ```

Chapter 4: Full Setup & Launch Guide

To run the complete system, you will need **three (3) separate Administrator terminals**.

Step 1: Launch the Main Server (Terminal 1)

1. Open a new Command Prompt or PowerShell **"Run as administrator"**.
2. Navigate to your project folder and activate the venv:
   ```
   cd F:\SOC_Project
   .\venv\Scripts\activate
   ```
3. Delete the old database logs.db (if it exists) for a clean start.
4. Run the main server (soc_dashboard.py):
   ```
   python soc_dashboard.py
   ```
5. You will see * Running on http://127.0.0.1:5000. Leave this terminal running.

Step 2: Launch the Windows Agent (Terminal 2)

1. Open a **second** new Command Prompt or PowerShell **"Run as administrator"**.
2. Activate the venv:
   ```
   cd F:\SOC_Project
   .\venv\Scripts\activate
   ```
3. Run the Windows agent (agent.py):
   ```
   python agent.py
   ```
4. You will see Starting monitor for 'Security' log... and Sending in batches... messages. Leave this terminal running.

Step 3: Configure the MikroTik Router (SNMP Agent)

(You only need to do this once. Ensure the VM is running in VMware).

1. Log in to your MikroTik CHR VM console (Login: admin, Password: [your_new_password]).
2. Run the following three commands one-by-one in the [admin@MikroTik] > prompt:
   - (This assumes your SIEM server's IP is 192.168.1.207)

```
/snmp community set public address=192.168.1.207
/snmp set enabled=yes
/ip firewall filter add chain=input protocol=udp dst-port=161 src-address=192.168.1.207
action=accept place-before=0
```

Step 4: Launch the SNMP Polling Agent (Terminal 3)

1. Open a **third** new Command Prompt or PowerShell **"Run as administrator"**.
2. Activate the venv:
   cd F:\SOC_Project
   .\venv\Scripts\activate

3. Run the SNMP agent (snmp_agent.py):
   python snmp_agent.py

4. You will see Monitoring Interface: ether2... and Successfully sent 1 log(s)... messages. Leave this running.

Done! Your dashboard at http://127.0.0.1:5000 is now receiving real-time logs from all three sources (Windows, Network Sniffer, and your MikroTik Router).

Chapter 5: How to Test Every Feature

Use these steps to verify that every component of your SIEM is working correctly.

Test 1: "Failed Logins" Counter (Testing agent.py)

- **Goal:** Increment the "Failed Logins" counter.
- **Method:**
  1. Go to your computer's **Lock Screen** (Press Windows Key + L).
  2. Type a **deliberately wrong password** and press Enter.
  3. Log back in with your correct password.
  4. Observe the Guardian SIEM dashboard. You will see the "Failed Logins" counter increase by one.

Test 2: "App Errors" Counter (Testing agent.py)

- **Goal:** Increment the "App Errors" counter.
- **Method:**
  1. Open an **Administrator** PowerShell or CMD terminal.
  2. Use the eventcreate command to generate a fake "App Crash" log (Event ID 1000):
     eventcreate /T ERROR /ID 1000 /L Application /SO "TestApp" /D "Test Crash"

  3. Observe the dashboard. You will see the "App Errors" counter increase by one.

Test 3: "SNMP Polling" (Testing snmp_agent.py)

- **Goal:** Verify that the ether2_Traffic_IN bytes increase from 0.
- **Method:**
  1. Open your **MikroTik VM** console window (the [admin@MikroTik] > prompt).
  2. Start a continuous ping to the internet and leave it running:
     /ping 8.8.8.8

  3. Now, look at the "Live Log Stream" on your SIEM dashboard.
  4. Observe the next "MikroTik Health Poll" log that arrives. You will see the ether2_Traffic_IN (Bytes) and ether2_Traffic_OUT (Bytes) values are now greater than zero and increasing with each new poll.

Test 4: "Threat Intel (AbuseIPDB)" (Testing Background Workers)

- **Goal:** Detect a malicious IP and automatically flag the log as "CRITICAL".
- **Method:**
  1. Open any CMD or PowerShell terminal on your computer.
  2. ping a known malicious IP (e.g., 118.25.6.39):
     ping 118.25.6.39

  3. The ping will show "Request timed out" (this is normal).
  4. Instantly, a new **INFO** (blue) log will appear on your dashboard showing the ping (Packet from 192.168.1.207 to 118.25.6.39).
  5. **Wait 1-2 minutes.** In the background, the threat_intel_enricher_thread is checking this IP.
  6. **Refresh your browser (http://127.0.0.1:5000)**.
  7. You will now see that the same log has turned **CRITICAL** (red) and its details now contain the line: Threat Intel (Dest): Malicious (100%)...

Test 5: "AI SOAR" (Testing AI Action)

- **Goal:** Use the AI to block a real IP address.
- **Method:**
  1. In the "Gemini AI Analyst" box on the dashboard, type show logs for 8.8.8.8 and click Ask.
     - The AI will respond: Found X log(s) for your query...
  2. Now, type a **new command**: block 8.8.8.8 and click Ask.
     - The AI will respond: Successfully created firewall rule to block IP: 8.8.8.8
  3. **Verify:** Open a CMD terminal and type ping 8.8.8.8. It will now fail with "Request timed out" or "General failure," because your own firewall is blocking it.

4. **Unblock:** Go back to the dashboard and type unblock 8.8.8.8. The AI will respond Successfully removed firewall rule.... The ping 8.8.8.8 command will now work again.

Chapter 6: Future Roadmap

The foundation of this project is extremely strong. To evolve it into an even more advanced platform, here are the next logical steps:

1. AI-Powered "Attack Storytelling"

- **Idea:** When a CRITICAL correlation (like "Brute Force Success") occurs, automatically trigger the AI. Have it query all related logs from that IP and User from the last 15 minutes.
- **Result:** The AI would write a simple, human-readable "Attack Story" (e.g., "1. The attacker first pinged the machine. 2. They failed to log in as 'Admin'. 3. They successfully logged in as 'Eshan'. 4. Immediately after, they created a new user named 'temp_admin'.") and attach it to the alert.

2. Predictive Threat Scoring

- **Idea:** Use the AI to score IPs *before* they break a rule. Feed it low-priority signals (e.g., "1 Failed Logon + 1 Ping + 10% AbuseIPDB score").
- **Result:** The AI would provide a "Predictive Score" (e.g., "This IP has a 65% probability of launching a brute force attack in the next hour"), allowing analysts to be proactive, not just reactive.

3. AI-Driven Auto-Remediation

- **Idea:** Instead of just blocking an IP when *you* tell it to, have the AI make the decision. When an alert triggers, the server asks Gemini: "Gemini, I have a 'Brute Force Success' on a 'Critical Workstation' from an IP with a 90% threat score. What is your recommendation: (A) Immediately Block, (B) Alert Human Only, or (C) Ignore?"
- **Result:** If the AI responds A, the server automatically calls the run_command function to block the IP, creating a truly autonomous SOAR system.