**Task 02 : Debugging the Python Script**

**Bug 1: Incorrect Regular Expression Pattern**

***Problem Identified:***

In the function **parse\_log\_line()** , the original regex pattern was too specific and incorrectly structured. It tried to extract "(\w+\.\w+\.\w+)" as the domain and "(\w+ /.+ HTTP/\d\.\d)" as the request.

On inspecting the regex pattern and comparing it with standard log formats, it was clear the pattern wouldn't match common cases like "GET /index.html HTTP/1.1"

The pattern expected \w+ /.+, which would fail if the method or path had unexpected characters.

***Fix Applied:***

Updated the regex to:  
pattern = r'(\d+\.\d+\.\d+\.\d+) - (\w+) \[(\d{2}/\w{3}/\d{4}:\d{2}:\d{2}:\d{2} \+\d{4})\] "\S+" "\S+ .+ HTTP/\d\.\d" (\d{3}) (\d+) (\d+)'

**Bug 2: Status Code Not Treated as Integer**

***Problem Identified:***

In the function is\_error\_status() , the status code was being compared using numeric comparison (status >= 400), but it was returned as a string from match.groups().

When running the program received the following error: Exception has occurred: TypeError '>=' not supported between instances of 'str' and 'int'. Assuming status is an integer (like HTTP status codes 200, 404, etc.), but in reality, it's a string (e.g., '404' or '500').

***Fix Applied:***

Converting to int ensures code comparing numbers with numbers. The try … except block ensures code doesn't crash if status is not convertible to an integer.

*def is\_error\_status(status):*

*try:*

*status = int(status)*

*return 400 <= status <= 599*

*except ValueError:*

*return False*

**Bug 3: Final Window Not Checked**

***Problem Identified:***

If the final time window in the log file doesn’t cross the 5-minute threshold, it is never evaluated for error rate.  
  
When going through the code logic, after looping through all lines, the script doesn't re-check the final window’s error rate unless a new timestamp forces it.

***Fix Applied:***

Added a final check after the loop to evaluate the last batch of requests:  
  
*if window\_requests > 0:*

*error\_rate = window\_errors / window\_requests*

*if error\_rate > error\_threshold:*

*print(f"Alert! Error rate {error\_rate:.2%} exceeds threshold at {current\_window\_start}")*

**Bug 4: Error Rate Display Not in readable format**

***Problem Identified:***

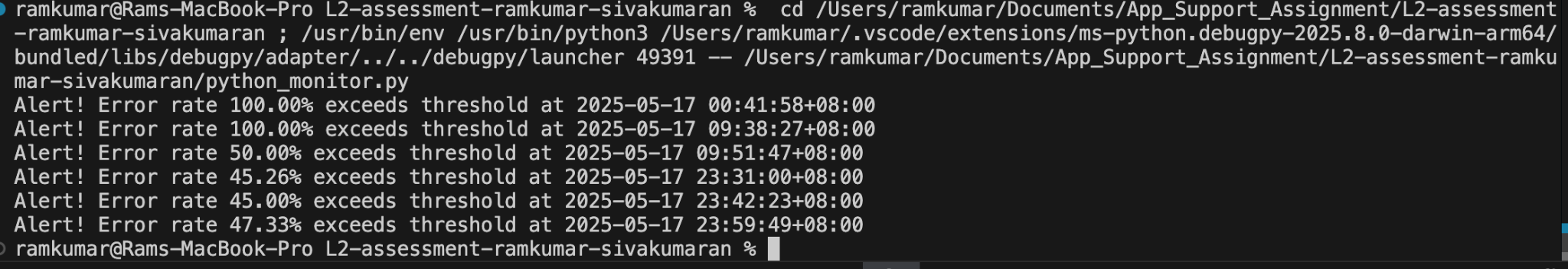
The error rate was printed as a float , which isn't easily interpreted by humans as a percentage.  
  
*print(f"Alert! Error rate {error\_rate}% exceeds threshold at {current\_window\_start}")*

% after a float doesn't format it as a percentage—just appends the % symbol.

***Fix Applied:***

Used Python’s percentage formatting with two decimal places for clarity:  
  
*print(f"Alert! Error rate {error\_rate:.2%} exceeds threshold at {current\_window\_start}")*

**Final output**

**

**Task 03 : Attack Pattern Detection Report**

**detect\_attack\_patterns.py** Python script that parses the log file and detects potential attack patterns, including:

* High Request rate IPs.
* High Error Rate IPs.
* Potential Brute-force Login IPs.

### **Script mainly flows the following steps:**

* Parses logs using regex
* Counts requests per IP
* Flags:
  + IPs with > threshold requests  
    IPs with > threshold 4xx/5xx errors
  + IPs frequently hitting /login

Here for this assessment threshold values are assigned in a assumption as follows :

# Configuration thresholds

maxRequestsPerIp = 5

maxErrorsPerIp = 3

loginRequestThreshold = 3

**Task 05 : CDN Performance Optimization**

To enhance CDN and application layer performance, implementing **aggressive, behavior-aware traffic control and caching optimization at the CDN edge.**

**1. Behavior-Based IP Rate Limiting**

* **High Request Rate IPs**: Block IPs that exceed a predefined request threshold in a short time window.
* **High Error Rate IPs**: Identify IPs generating excessive 4xx/5xx errors and subject them to stricter rate limits or bans.
* **Brute-Force Login Attempts**: For IPs that repeatedly access /login endpoints, enforce multi-factor authentication after a certain number of attempts to mitigate brute-force login attacks.

**2. Caching Optimization**

* **Leverage CDN Caching**: Serve static assets (JavaScript, CSS, images) with extended TTL via Cache-Control headers to reduce origin traffic.
* **Conditional Requests**: Implement ETag and Last-Modified headers to support efficient client-side caching and reduce unnecessary data transfers.

**3. Monitoring and Dynamic Adaptation**

* Integrate real-time monitoring tools (Grafana, Prometheus, ELK stack) to:
  + Visualize traffic anomalies
  + Track effectiveness of rate-limiting and caching policies

—---------------

*Assessment done by: Ramkumar Sivakumaran*

*Github :* [*https://github.com/Ramkumar96/L2-assessment-ramkumar-sivakumaran*](https://github.com/Ramkumar96/L2-assessment-ramkumar-sivakumaran)