**VRV Security’s Python Intern Assignment**

import re

import csv

import os

from collections import defaultdict

from typing import Dict, List, Tuple

class LogAnalyzer:

    def \_\_init\_\_(self, log\_file\_path: str, failed\_login\_threshold: int = 10):

        """

        Initialize the LogAnalyzer with configuration parameters.

        :param log\_file\_path: Path to the log file

        :param failed\_login\_threshold: Number of failed login attempts to flag as suspicious

        """

        # Ensure the log file exists, create if it doesn't

        self.log\_file\_path = self.ensure\_log\_file(log\_file\_path)

        self.failed\_login\_threshold = failed\_login\_threshold

        # Data storage

        self.ip\_request\_counts = defaultdict(int)

        self.endpoint\_access\_counts = defaultdict(int)

        self.failed\_login\_attempts = defaultdict(int)

        self.most\_accessed\_endpoint = None

    def ensure\_log\_file(self, log\_file\_path: str) -> str:

        """

        Ensure the log file exists, create a sample log if it doesn't.

        :param log\_file\_path: Proposed log file path

        :return: Verified log file path

        """

        # Sample log content if file doesn't exist

        sample\_log\_content = """192.168.1.1 - - [03/Dec/2024:10:12:34 +0000] "GET /home HTTP/1.1" 200 512

203.0.113.5 - - [03/Dec/2024:10:12:35 +0000] "POST /login HTTP/1.1" 401 128 "Invalid credentials"

10.0.0.2 - - [03/Dec/2024:10:12:36 +0000] "GET /about HTTP/1.1" 200 256

192.168.1.1 - - [03/Dec/2024:10:12:37 +0000] "GET /contact HTTP/1.1" 200 312

198.51.100.23 - - [03/Dec/2024:10:12:38 +0000] "POST /register HTTP/1.1" 200 128

203.0.113.5 - - [03/Dec/2024:10:12:39 +0000] "POST /login HTTP/1.1" 401 128 "Invalid credentials"

192.168.1.100 - - [03/Dec/2024:10:12:40 +0000] "POST /login HTTP/1.1" 401 128 "Invalid credentials"

10.0.0.2 - - [03/Dec/2024:10:12:41 +0000] "GET /dashboard HTTP/1.1" 200 1024

198.51.100.23 - - [03/Dec/2024:10:12:42 +0000] "GET /about HTTP/1.1" 200 256

192.168.1.1 - - [03/Dec/2024:10:12:43 +0000] "GET /dashboard HTTP/1.1" 200 1024

203.0.113.5 - - [03/Dec/2024:10:12:44 +0000] "POST /login HTTP/1.1" 401 128 "Invalid credentials"

203.0.113.5 - - [03/Dec/2024:10:12:45 +0000] "POST /login HTTP/1.1" 401 128 "Invalid credentials"

192.168.1.100 - - [03/Dec/2024:10:12:46 +0000] "POST /login HTTP/1.1" 401 128 "Invalid credentials"

10.0.0.2 - - [03/Dec/2024:10:12:47 +0000] "GET /profile HTTP/1.1" 200 768

192.168.1.1 - - [03/Dec/2024:10:12:48 +0000] "GET /home HTTP/1.1" 200 512

198.51.100.23 - - [03/Dec/2024:10:12:49 +0000] "POST /feedback HTTP/1.1" 200 128

203.0.113.5 - - [03/Dec/2024:10:12:50 +0000] "POST /login HTTP/1.1" 401 128 "Invalid credentials"

192.168.1.1 - - [03/Dec/2024:10:12:51 +0000] "GET /home HTTP/1.1" 200 512

198.51.100.23 - - [03/Dec/2024:10:12:52 +0000] "GET /about HTTP/1.1" 200 256

203.0.113.5 - - [03/Dec/2024:10:12:53 +0000] "POST /login HTTP/1.1" 401 128 "Invalid credentials"

192.168.1.100 - - [03/Dec/2024:10:12:54 +0000] "POST /login HTTP/1.1" 401 128 "Invalid credentials"

10.0.0.2 - - [03/Dec/2024:10:12:55 +0000] "GET /contact HTTP/1.1" 200 512

198.51.100.23 - - [03/Dec/2024:10:12:56 +0000] "GET /home HTTP/1.1" 200 512

192.168.1.100 - - [03/Dec/2024:10:12:57 +0000] "POST /login HTTP/1.1" 401 128 "Invalid credentials"

203.0.113.5 - - [03/Dec/2024:10:12:58 +0000] "POST /login HTTP/1.1" 401 128 "Invalid credentials"

10.0.0.2 - - [03/Dec/2024:10:12:59 +0000] "GET /dashboard HTTP/1.1" 200 1024

192.168.1.1 - - [03/Dec/2024:10:13:00 +0000] "GET /about HTTP/1.1" 200 256

198.51.100.23 - - [03/Dec/2024:10:13:01 +0000] "POST /register HTTP/1.1" 200 128

203.0.113.5 - - [03/Dec/2024:10:13:02 +0000] "POST /login HTTP/1.1" 401 128 "Invalid credentials"

192.168.1.100 - - [03/Dec/2024:10:13:03 +0000] "POST /login HTTP/1.1" 401 128 "Invalid credentials"

10.0.0.2 - - [03/Dec/2024:10:13:04 +0000] "GET /profile HTTP/1.1" 200 768

198.51.100.23 - - [03/Dec/2024:10:13:05 +0000] "GET /about HTTP/1.1" 200 256

192.168.1.1 - - [03/Dec/2024:10:13:06 +0000] "GET /home HTTP/1.1" 200 512

198.51.100.23 - - [03/Dec/2024:10:13:07 +0000] "POST /feedback HTTP/1.1" 200 128"""

        # If file doesn't exist, create it with sample content

        if not os.path.exists(log\_file\_path):

            print(f"Log file {log\_file\_path} not found. Creating a sample log file.")

            try:

                with open(log\_file\_path, 'w') as file:

                    file.write(sample\_log\_content)

                print(f"Sample log file created at {log\_file\_path}")

            except Exception as e:

                print(f"Error creating log file: {e}")

                # Fallback to using current directory

                log\_file\_path = 'sample.log'

                with open(log\_file\_path, 'w') as file:

                    file.write(sample\_log\_content)

        return log\_file\_path

    def parse\_log\_file(self):

        """

        Parse the log file and extract relevant information.

        """

        try:

            with open(self.log\_file\_path, 'r') as file:

                for line in file:

                    # IP Address extraction

                    ip\_match = re.match(r'^(\d+\.\d+\.\d+\.\d+)', line)

                    if ip\_match:

                        ip\_address = ip\_match.group(1)

                        self.ip\_request\_counts[ip\_address] += 1

                    # Endpoint extraction

                    endpoint\_match = re.search(r'"[A-Z]+ (/[^\s]+)', line)

                    if endpoint\_match:

                        endpoint = endpoint\_match.group(1)

                        self.endpoint\_access\_counts[endpoint] += 1

                    # Failed login detection

                    if "401" in line and "Invalid credentials" in line:

                        self.failed\_login\_attempts[ip\_address] += 1

            # Find most accessed endpoint

            self.most\_accessed\_endpoint = max(

                self.endpoint\_access\_counts.items(),

                key=lambda x: x[1]

            )

        except Exception as e:

            print(f"Error parsing log file: {e}")

            return False

        return True

    def get\_top\_ip\_requests(self) -> List[Tuple[str, int]]:

        """

        Get IP addresses sorted by request count in descending order.

        :return: Sorted list of (IP, request\_count) tuples

        """

        return sorted(

            self.ip\_request\_counts.items(),

            key=lambda x: x[1],

            reverse=True

        )

    def get\_suspicious\_ips(self) -> List[Tuple[str, int]]:

        """

        Identify IPs with failed login attempts exceeding the threshold.

        :return: List of (IP, failed\_login\_count) tuples

        """

        return [

            (ip, count)

            for ip, count in self.failed\_login\_attempts.items()

            if count > self.failed\_login\_threshold

        ]

    def save\_results\_to\_csv(self):

        """

        Save analysis results to a CSV file.

        """

        try:

            with open('log\_analysis\_results.csv', 'w', newline='') as csvfile:

                csv\_writer = csv.writer(csvfile)

                # IP Requests Section

                csv\_writer.writerow(['Requests per IP'])

                csv\_writer.writerow(['IP Address', 'Request Count'])

                for ip, count in self.get\_top\_ip\_requests():

                    csv\_writer.writerow([ip, count])

                csv\_writer.writerow([])  # Empty row for readability

                # Most Accessed Endpoint Section

                csv\_writer.writerow(['Most Accessed Endpoint'])

                csv\_writer.writerow(['Endpoint', 'Access Count'])

                csv\_writer.writerow([

                    self.most\_accessed\_endpoint[0],

                    self.most\_accessed\_endpoint[1]

                ])

                csv\_writer.writerow([])  # Empty row for readability

                # Suspicious Activity Section

                csv\_writer.writerow(['Suspicious Activity'])

                csv\_writer.writerow(['IP Address', 'Failed Login Count'])

                for ip, count in self.get\_suspicious\_ips():

                    csv\_writer.writerow([ip, count])

            print("Results saved to log\_analysis\_results.csv")

        except Exception as e:

            print(f"Error saving results to CSV: {e}")

    def display\_results(self):

        """

        Display analysis results in the terminal.

        """

        print("\n--- Log Analysis Results ---\n")

        # IP Requests

        print("Requests per IP Address:")

        print("{:<20} {:<15}".format("IP Address", "Request Count"))

        print("-" \* 35)

        for ip, count in self.get\_top\_ip\_requests():

            print("{:<20} {:<15}".format(ip, count))

        # Most Accessed Endpoint

        print("\nMost Frequently Accessed Endpoint:")

        print(f"{self.most\_accessed\_endpoint[0]} (Accessed {self.most\_accessed\_endpoint[1]} times)")

        # Suspicious Activity

        suspicious\_ips = self.get\_suspicious\_ips()

        if suspicious\_ips:

            print("\nSuspicious Activity Detected:")

            print("{:<20} {:<15}".format("IP Address", "Failed Login Attempts"))

            print("-" \* 35)

            for ip, count in suspicious\_ips:

                print("{:<20} {:<15}".format(ip, count))

        else:

            print("\nNo suspicious activity detected.")

    def run\_analysis(self):

        """

        Execute the complete log file analysis.

        """

        # Parse log file

        if not self.parse\_log\_file():

            print("Analysis failed. Check the log file and try again.")

            return

        # Display results

        self.display\_results()

        # Save to CSV

        self.save\_results\_to\_csv()

def main():

    # Use current directory or specify a full path

    log\_file\_path = 'sample.log'

    # Print current working directory for debugging

    print("Current Working Directory:", os.getcwd())

    analyzer = LogAnalyzer(log\_file\_path)

    analyzer.run\_analysis()

if \_\_name\_\_ == "\_\_main\_\_":

    main()