Refined Project Description

Log File Analyzer for Intrusion Detection

Objective:
To develop a Python-based tool that detects suspicious patterns in log files (such as Apache and SSH logs). The tool identifies brute-force attacks, scanning attempts, and Denial of Service (DoS) patterns.

Tools & Libraries Used:

- **Python**: Primary programming language
- **regex**: Regular expressions for log parsing
- **pandas**: For data manipulation and analysis
- **matplotlib**: For visualizing data

Project Outline

1. **Parse Logs**:
    - Use Python to read Apache and SSH log files.
    - Utilize **regex** to extract relevant data from each log line (IP, timestamps, actions).
2. **Detect Suspicious Patterns**:
    - Identify suspicious activities like brute-force attacks (multiple failed login attempts), scanning, and DoS patterns.
3. **Visualize Data**:
    - Generate simple visualizations of access patterns (by IP and time) using **matplotlib** and **pandas**.
4. **Cross-reference with IP Blacklist**:
    - Compare the extracted IPs with a public IP blacklist to flag known malicious addresses.
5. **Export Incident Reports**:
    - Export suspicious activity reports (e.g., failed login attempts, blacklisted IPs) to a CSV file for further analysis.

Refined Python Implementation (Beginner-Friendly)

Here’s the refined version of my code, with detailed comments and structure improvements.

```
import re
import pandas as pd
import matplotlib.pyplot as plt
```

```
# Step 1: Parse Apache Logs (Basic example)
def parse_apache_log_line(line):
    """
    Parses a single Apache log line to extract relevant details using regex.
    Returns: IP, timestamp, request, status code, size.
    """
    log_pattern = r'(\S+) - - \[(.*?)\] "(.*?)" (\d{3}) (\d+)'
    match = re.match(log_pattern, line)
    if match:
        ip = match.group(1)
        timestamp = match.group(2)
        request = match.group(3)
        status = match.group(4)
        size = match.group(5)
        return ip, timestamp, request, status, size
    return None

# Step 2: Parse SSH Logs (Detect Accepted/Failed Logins)
def parse_ssh_log_line(line):
    """
    Parses a single SSH log line to detect login attempts.
    Returns: timestamp, host, action (Accepted/Failed), user, and IP address.
    """
    ssh_pattern = r'(\w{3} \d+ \d+:\d+:\d+) (\S+) sshd\[.*\]: (Accepted|Failed) password for (\S+)
from (\S+) port \d+ ssh2'
    match = re.match(ssh_pattern, line)
    if match:
        timestamp = match.group(1)
        host = match.group(2)
        action = match.group(3)
        user = match.group(4)
        ip = match.group(5)
        return timestamp, host, action, user, ip
    return None

# Step 3: Detect Brute Force Attempts (Multiple Failed Logins)
def detect_brute_force_attempts(ssh_log_lines):
    """
    Detects IP addresses with multiple failed login attempts (brute-force detection).
    Flags IPs with more than 5 failed attempts.
    """
    failed_attempts = {}
    for line in ssh_log_lines:
        parsed = parse_ssh_log_line(line)
        if parsed and parsed[2] == 'Failed':
            ip = parsed[4]
            failed_attempts[ip] = failed_attempts.get(ip, 0) + 1
    # Flag IPs with more than 5 failed attempts as suspicious
    suspicious_ips = {ip: count for ip, count in failed_attempts.items() if count > 5}
    return suspicious_ips
```

```
# Step 4: Visualize Access Patterns (by IP)
def plot_access_patterns(ip_list):
    """
    Creates a bar chart to visualize the frequency of access attempts by IP.
    """
    ip_counts = pd.Series(ip_list).value_counts()
    ip_counts.plot(kind='bar', figsize=(12,6))
    plt.title('Access Counts by IP Address')
    plt.xlabel('IP Address')
    plt.ylabel('Access Count')
    plt.show()

# Step 5: Cross-reference with IP Blacklist
def check_ip_blacklist(ip, blacklist):
    """
    Checks if an IP is present in a public blacklist.
    Returns: True if the IP is blacklisted, False otherwise.
    """
    return ip in blacklist

# Step 6: Export Incidents to CSV
def export_incidents(incident_dict, filename='incidents.csv'):
    """
    Exports the detected suspicious incidents (IPs and counts) to a CSV file.
    """
    df = pd.DataFrame(list(incident_dict.items()), columns=['IP', 'Failed Attempts'])
    df.to_csv(filename, index=False)
    print(f"Incidents exported to {filename}")

# Sample Usage
if __name__ == '__main__':
    # Sample SSH log lines (in practice, replace with actual file reading)
    ssh_log_sample = [
        "Jan 12 17:13:09 node1 sshd[18347]: Failed password for invalid user admin from 192.0.2.10
port 42304 ssh2",
        "Jan 12 17:14:09 node1 sshd[18348]: Failed password for invalid user admin from 192.0.2.10
port 42305 ssh2",
        "Jan 12 17:15:09 node1 sshd[18349]: Failed password for invalid user admin from 192.0.2.10
port 42306 ssh2",
        "Jan 12 17:16:09 node1 sshd[18350]: Failed password for invalid user admin from 192.0.2.10
port 42307 ssh2",
        "Jan 12 17:17:09 node1 sshd[18351]: Failed password for invalid user guest from 203.0.113.9
port 33322 ssh2",
        "Jan 12 17:18:09 node1 sshd[18352]: Accepted password for user from 198.51.100.5 port
42308 ssh2"
    ]

    # Detect brute force attempts
    suspicious_ips = detect_brute_force_attempts(ssh_log_sample)
    print("Suspicious IPs with multiple failed login attempts:", suspicious_ips)

    # Visualize access patterns of suspicious IPs
```

```
plot_access_patterns([ip for ip in suspicious_ips])

# Export the incidents (failed attempts) to CSV
export_incidents(suspicious_ips)

# Public IP blacklist (for example)
public_blacklist = {"192.0.2.10", "203.0.113.9"}

# Check if suspicious IPs are in the blacklist
for ip in suspicious_ips:
    if check_ip_blacklist(ip, public_blacklist):
        print(f"IP {ip} found in blacklist!")
```

Key Improvements and Clarifications:

1. **Code Structure**:
    - Modularized functions for each task (log parsing, brute-force detection, visualization,).
    - Simplified the regex patterns and ensured they are beginner-friendly.
2. **Comments**:
    - Each function includes docstrings explaining its purpose and what it returns.
    - Inline comments help guide the reader through the code's logic.
3. **Brute Force Detection**:
    - The `detect_brute_force_attempts` function now explicitly counts failed login attempts by IP and flags IPs with more than 5 failures as suspicious.
4. **Visualization**:
    - The `plot_access_patterns` function uses **matplotlib** to generate a bar chart, showing access counts by IP.
5. **Exporting Reports**:
    - The `export_incidents` function exports the suspicious IPs and their failed attempt counts to a CSV file.
6. **Blacklist Check**:
    - The `check_ip_blacklist` function simply checks if the suspicious IPs exist in a predefined blacklist.

Project Deliverables:

1. **Log Parsing**: The code will read and parse Apache and SSH logs.
2. **Suspicious Activity Detection**: The script detects brute-force attempts and flags suspicious IPs.

3. **Visualization**: Access patterns are visualized with matplotlib.
4. **IP Blacklist Checking**: The script cross-references suspicious IPs with a public IP blacklist.
5. **Incident Reporting**: Suspicious activities are logged and exported to a CSV file.