Problem: The Digital Watchtower Detecting Intruders

Blockchain Web3
Cyber Security Club

August 7, 2025

Objective

Learn to perform basic log analysis to detect security threats. You'll implement a **sliding window algorithm** to identify potential **brute-force attacks** by monitoring user login patterns in real-time.

The Intelligence Briefing

Threat: The Brute-Force Barrage

Imagine a thief trying to open a door. Instead of picking the lock, they have a massive ring with millions of keys and are trying them one by one, as fast as they can. This is a **brute-force attack** in the digital world. An attacker uses automated scripts to hammer a login page with thousands of password guesses, hoping one will eventually be correct. A tell-tale sign of this attack is a high number of failed login attempts from a single IP address in a very short time.

Countermeasure: Algorithmic Log Analysis

As a security analyst, your most powerful tool is the server's log file. It records every event, including every failed login. Your job is to be the watchtower, scanning these logs for suspicious patterns. You need to focus on a narrow, recent slice of time. This is where a **sliding window algorithm** comes in. You'll only count failures that have occurred within a specific time frame (e.g., the last 10 minutes). As time moves forward, this window "slides" along the log data.

Your Toolkit: Core Data Structures

This mission doesn't require special libraries, just your programming wits and Python's built-in tools.

- **Dictionaries (`dict`):** The perfect tool to keep track of activity for each individual IP address. The IP address will be the key, and the value can be a list of recent failure timestamps.
- **Lists (`list`):** To store the timestamps of failed logins for each IP. You will use this list to implement your sliding window.

Your Mission

Your mission is to write a single Python function, `find_suspicious_ips`, that analyzes a list of log entries and identifies any IP addresses that show signs of a brute-force attack.

Rules of Engagement

Your algorithm must follow these specific rules:

1. **Log Format:** Each log entry is a string: `"TIMESTAMP,IP_ADDRESS,STATUS"`.
2. **Attack Threshold:** An IP address is considered **suspicious** if it has **more than 5** `FAIL` statuses within any **10-minute (600-second) window**.
3. **The "Oops, I Forgot" Reset:** If an IP address has a `SUCCESS` login, all their prior failed attempts should be immediately forgotten and no longer count towards the attack threshold.

Implementation Guide

Function Skeleton

```python
def find_suspicious_ips(logs: list[str]) -> list[str]:
    """
    Parses logs to find IPs with >5 failed logins in a 10-minute window.
    """
    # Your logic here.
    # Hint: A dictionary to store lists of failure timestamps per IP is a great start.
    # ip_failures = {'1.2.3.4': [timestamp1, timestamp2, ...]}
    pass
```

Example

```
logs = [
    "1672531200,192.168.1.101,FAIL",
    "1672531205,192.168.1.101,FAIL",
    "1672531210,203.0.113.45,FAIL",      # Attacker's 1st fail
    "1672531230,192.168.1.101,FAIL",
```

```
    "1672531260,192.168.1.101,SUCCESS",   # This IP is now clear
    "1672531300,203.0.113.45,FAIL",      # Attacker's 2nd fail
    "1672531310,203.0.113.45,FAIL",      # Attacker's 3rd fail
    "1672531320,203.0.113.45,FAIL",      # Attacker's 4th fail
    "1672531330,203.0.113.45,FAIL",      # Attacker's 5th fail
    "1672531340,203.0.113.45,FAIL",      # Attacker's 6th fail ->
   BREACH!
    "1672531900,203.0.113.45,FAIL"       # This is > 600s after the 1st
    fail
]
```

Expected Output: `["203.0.113.45"]`

Why? The IP `203.0.113.45` accumulates 6 failed logins between timestamp `1672531210` and `1672531340`. The time difference is 130 seconds, which is well within the 600-second window. The IP `192.168.1.101` is not flagged because its failures were cleared by a successful login.