

Log Monitoring Workflow

Zehra Nur Ozer

May 2024

Contents

Executive Summary.....	3
Work Structure.....	3
Findings	3
Python Language.....	3
Bash Script.....	5
Expected Output	6
References.....	7

Executive Summary

This project, prepared for Turn a New Leaf, outlines the challenging task of monitoring network logs for unusual traffic. The primary programming languages used are Bash and Python. The company uses both Windows & Linux machines and has two web servers in its network. Following the work structure steps, the insides of the logs are interpreted as described below.

Work Structure

Workflow: This step starts with the monitoring schedule. In this phase, any unusual patterns or behaviours in the log data are detected, and the security logs are checked for any signs of intrusion attempts. Turn a New Leaf requests that members log in to the company system every Thursday to confirm or update their status. The logs monitored in this project are from December 4, 2023, with Windows and September 8, 2023, in Linux.

Programming: To collect data, tools like syslog-ng, Splunk or Graylog are used, and for analysis, the scripting languages Python and Bash are utilized.

Expected Output: This phase identifies the logs' patterns and potential security incidents.

Documentation: Includes findings and weekly report. Email report is provided every Friday.

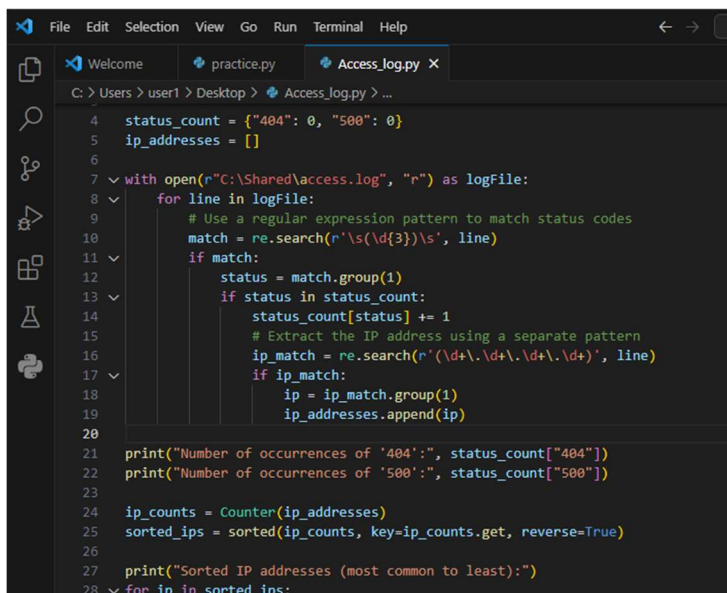
Unusual Behaviour: Any unusual login attempts, or multiple failed logins can be considered suspicious activities.

Potential Iteration: A live monitoring or data visualization tool can be used for additional information.

Findings

Python Language

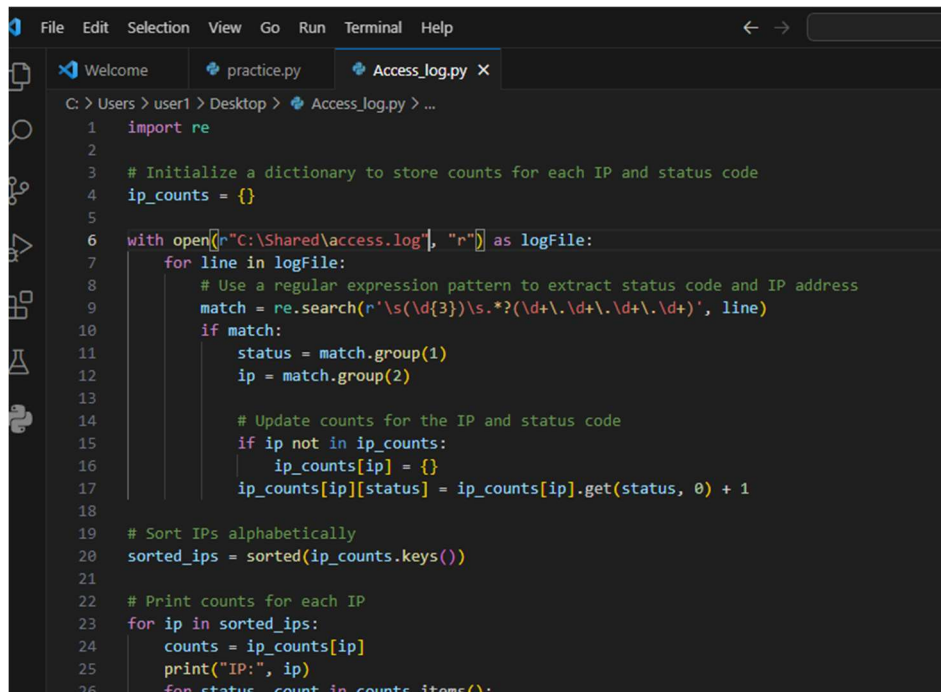
The insides are collected using python language (with VsCode) as below.



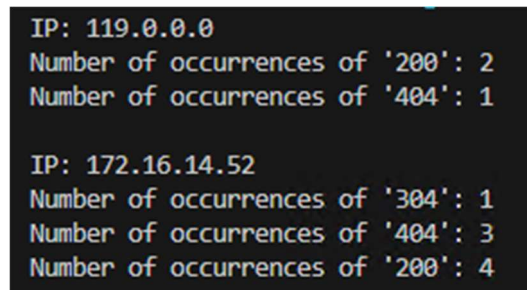
```
4 status_count = {"404": 0, "500": 0}
5 ip_addresses = []
6
7 with open(r"C:\Shared\access.log", "r") as logFile:
8     for line in logFile:
9         # Use a regular expression pattern to match status codes
10        match = re.search(r'\s(\d{3})\s', line)
11        if match:
12            status = match.group(1)
13            if status in status_count:
14                status_count[status] += 1
15            # Extract the IP address using a separate pattern
16            ip_match = re.search(r'(\d+\.\d+\.\d+\.\d+)', line)
17            if ip_match:
18                ip = ip_match.group(1)
19                ip_addresses.append(ip)
20
21 print("Number of occurrences of '404':", status_count["404"])
22 print("Number of occurrences of '500':", status_count["500"])
23
24 ip_counts = Counter(ip_addresses)
25 sorted_ips = sorted(ip_counts, key=ip_counts.get, reverse=True)
26
27 print("Sorted IP addresses (most common to least):")
28 for ip in sorted_ips:
```

```
Number of occurrences of '404': 7
Number of occurrences of '500': 0
```

This indicates 7 instances where the server responded with a "404 Not Found" error. This typically means the requested resource could not be found on the server. The reason can be human error or a malicious attack. This could be due to broken links or attempts to access non-existent resources.

A screenshot of a code editor window showing a Python script named 'Access_log.py'. The script is designed to parse an access log file. It imports the 're' module, initializes a dictionary 'ip_counts', and opens a file 'C:\Shared\access.log'. It uses a regular expression to extract status codes and IP addresses from each line. The script then updates the counts for each IP and status code, sorts the IPs alphabetically, and prints the counts for each IP.

```
1 import re
2
3 # Initialize a dictionary to store counts for each IP and status code
4 ip_counts = {}
5
6 with open(r"C:\Shared\access.log", "r") as logFile:
7     for line in logFile:
8         # Use a regular expression pattern to extract status code and IP address
9         match = re.search(r'\s(\d{3})\s.*?(\d+\.\d+\.\d+\.\d+)', line)
10        if match:
11            status = match.group(1)
12            ip = match.group(2)
13
14            # Update counts for the IP and status code
15            if ip not in ip_counts:
16                ip_counts[ip] = {}
17            ip_counts[ip][status] = ip_counts[ip].get(status, 0) + 1
18
19 # Sort IPs alphabetically
20 sorted_ips = sorted(ip_counts.keys())
21
22 # Print counts for each IP
23 for ip in sorted_ips:
24     counts = ip_counts[ip]
25     print("IP:", ip)
26     for status, count in counts.items():
```

A screenshot showing the output of the Python script. It displays the IP address and the count of occurrences for specific status codes. For IP 119.0.0.0, there are 2 occurrences of '200' and 1 occurrence of '404'. For IP 172.16.14.52, there is 1 occurrence of '304', 3 occurrences of '404', and 4 occurrences of '200'.

```
IP: 119.0.0.0
Number of occurrences of '200': 2
Number of occurrences of '404': 1

IP: 172.16.14.52
Number of occurrences of '304': 1
Number of occurrences of '404': 3
Number of occurrences of '200': 4
```

When sorting IP addresses:

172.16.14.52: This IP address appears in the log entries, with 4 occurrences of the '200' status code ('OK'), 3 occurrences of the '404' status code, and 1 occurrence of the '304' status code ('Not Modified').

119.0.0.0: This IP address appears once in the log entries, with 2 occurrences of the '200' status code and 1 occurrence of the '404' status code. Frequent 404 is an unusual spike and it might indicate misconfigurations as well.

Further analysis and investigation may be needed to understand the causes behind the occurrence of specific status codes, such as '404' and '500', and to address any potential issues or security concerns.

Bash Script

The insides are collected using the bash script as below.

The first indication is all the logs are from the same IP address. And we see the attempts are back-to-back in seconds.

```
user@user-pc: ~
File Edit View Search Terminal Help
user@user-pc:~$ ./practice.sh
172.16.14.50 - - [08/Sep/2023:12:26:56 -0400] "GET / HTTP/1.1" 200 3477 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/116.0.0.0 Safari/537.36"
172.16.14.50 - - [08/Sep/2023:12:26:58 -0400] "GET /icons/ubuntu-logo.png HTTP/1.1" 200 3623 "http://172.16.14.52/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/116.0.0.0 Safari/537.36"
172.16.14.50 - - [08/Sep/2023:12:27:04 -0400] "GET /favicon.ico HTTP/1.1" 404 491 "http://172.16.14.52/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/116.0.0.0 Safari/537.36"
172.16.14.50 - - [08/Sep/2023:12:27:33 -0400] "GET /test HTTP/1.1" 404 491 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/116.0.0.0 Safari/537.36"
172.16.14.50 - - [08/Sep/2023:12:28:26 -0400] "-" 408 0 "-" "-"
user@user-pc:~$ S
```

(wikipedia)

```
user@user-pc:/var/log/apache2$ awk '{print $1}' access.log.1
172.16.14.50
172.16.14.50
172.16.14.50
172.16.14.50
172.16.14.50
```

"404 Not Found" status code in a log file. These two attempts are from the same IP address and there might need a further investigation to determine whether it was a malicious attempt or not.

```
user@user-pc:/var/log/apache2$ grep -n '404' access.log.1
3:172.16.14.50 - - [08/Sep/2023:12:27:04 -0400] "GET /favicon.ico HTTP/1.1" 404 491 "http://172.16.14.52/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/116.0.0.0 Safari/537.36"
4:172.16.14.50 - - [08/Sep/2023:12:27:33 -0400] "GET /test HTTP/1.1" 404 491 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/116.0.0.0 Safari/537.36"
```

Bash script that checks for IP addresses with two occurrences of a "200 OK" status code in a log file. This script will parse the log file, count the "200" status codes per IP address occurrences, and then output the IPs that meet the criteria. (chatgpt)

```
user@user-pc:/var/log/apache2$ grep -n '200' access.log.1
1:172.16.14.50 - - [08/Sep/2023:12:26:56 -0400] "GET / HTTP/1.1" 200 3477 "-" "Mozilla/5.0 (Windows NT 10.0; Win
64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/116.0.0.0 Safari/537.36"
2:172.16.14.50 - - [08/Sep/2023:12:26:58 -0400] "GET /icons/ubuntu-logo.png HTTP/1.1" 200 3623 "http://172.16.14
.52/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/116.0.0.0 Safari/
537.36"
```

Expected Output

Indicators of Compromise (IoCs) must be included for interpretation. These are the artifacts that are observed on the network. Some potential IoC's for the monitoring are.

Multiple Failed Login Attempts: A high number of consecutive failed login attempts from the same IP address could indicate a brute-force attack.

Unusual IPs: Access from IP addresses that do not normally interact with your network

Unusual Resource Access: Repeated 404 errors or attempts to access non-existent resources, which might indicate reconnaissance activity.

To determine whether there is an anomaly, it is also suggested that the system be improved. Setting up real-time alerting can help detect unusual logs, as can implementing the system with a tool like Prometheus with Alertmanager, Grafana, or Watcher.

Frequent documentation can allow for catching the incidents early. Log management tools can assist with this task by scheduling daily, weekly or monthly reports.

References

(n.d.). Retrieved from <https://chatgpt.com/?oai-dm=1>

(n.d.). Retrieved from <https://web.compass.lighthouselabs.ca/p/14/days/w04d1>

(n.d.). Retrieved from https://en.wikipedia.org/wiki/List_of_HTTP_status_codes

(n.d.). Retrieved from <https://sematext.com/blog/log-monitoring-tools/#:%7E:text=Best%20Log%20Monitoring%20Tools%201%201.%20Sematext%20Logs,...%206%206.%20Splunk%20...%207%207.%20Datadog>