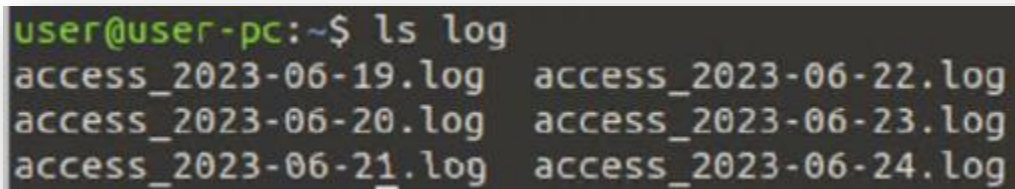


Log Monitoring Workflow Project

The objective is to develop an efficient and automated workflow for monitoring and documenting logs in a network, specifically focusing on detecting unusual network traffic and failed login attempts. This includes analyzing access logs, generating alerts for unusual behavior, documenting the monitoring process, and providing weekly updates to the manager.

My Workflow:

1. Log Retrieval: The first step is to retrieve the access log file from the web server. This can be achieved by running the `get_log.sh` script. The script creates a directory to store log files, copies the access log file to the specified directory with a timestamped name, and then executes the Python script to analyze the log. Examples of log files created:

A terminal window showing the command `ls log` and its output. The output lists eight log files: `access_2023-06-19.log`, `access_2023-06-20.log`, `access_2023-06-21.log`, `access_2023-06-22.log`, `access_2023-06-23.log`, and `access_2023-06-24.log`.

```
user@user-pc:~$ ls log
access_2023-06-19.log  access_2023-06-22.log
access_2023-06-20.log  access_2023-06-23.log
access_2023-06-21.log  access_2023-06-24.log
```

2. Log Analysis: The `detect_unusual_login.py` script analyzes the log file to detect any unusual login attempts. It counts the occurrences of specific error codes (401, 403, 400, 404, 409) and tracks the number of failed login attempts per IP address. If the number of failed login attempts exceeds a defined threshold (5 here), it considers it an unusual login attempt and includes the IP address in the summary. Additionally, it detects if there is an unusual amount of 400 errors.
3. Output Generation: The script prints the detected unusual login attempts and the corresponding number of failed login attempts for each IP address. It also notifies if there is an unusual amount of 400 errors. If no unusual login attempts or errors are found, it indicates that as well. The script provides a message indicating the completion of the detection process.

4. Documentation: To capture and document the monitoring process, we can implement a method to log the script's output to a file. For example, we can modify the Python script to append the results to a log file along with the timestamp. This log file will serve as documentation and can be shared with your manager.
5. Alerting: This comes under potential iterations: To notify my manager about any unusual activity, we can integrate an email notification system into the workflow. After the script execution and output generation, we can add code to send an email with the generated log file attached. This can be achieved using a Python library like smtplib (SMTPLIB) to send emails programmatically. We can schedule this email to be sent on a weekly basis, along with the updated log file.

Programming:

To successfully complete the task, I used 2 major programming languages Bash and Python.

1. Bash script:
 - a. **get_log.sh**: This script creates a directory to store log files, copies the access log file to the specified directory with a timestamped name, and executes the Python script to analyze the log.

```
1. #!/bin/bash
2.
3. # Destination directory to store log files
4. mkdir -p ./log
5. log_directory="./log"
6.
7. # Retrieve the access log file
8. current_date=$(date +%F)
9. access_log="/var/log/apache2/access.log"
10. new_log="${log_directory}/access_${current_date}.log"
11. cp "$access_log" "$new_log"
12.
13. # Run the Python script
14. python3 detect_unusual_login.py "$new_log"
```

2. Python script:

- a. detect_unusual_login.py: This script analyzes the log file to detect unusual login attempts and unusual 400 errors. It uses regular expressions to extract relevant information from the log entries, tracks failed login attempts per IP address, and compares the counts against defined thresholds.

```
1. import re
2. import sys
3.
4. log_file = sys.argv[1]
5. threshold = 5
6. error_codes = ['401', '403', '400', '404', '409']
7.
8. def detect_unusual_login_attempts():
9.     login_attempts = {}
10.    error_400_count = 0
11.    summary = []
12.
13.    with open(log_file, "r") as file:
14.        for line in file:
15.            match = re.search(r'(\d+\.\d+\.\d+\.\d+).*?\s(\d+)\s', line)
16.            if match:
17.                ip_address = match.group(1)
18.                status_code = match.group(2)
19.                if status_code in error_codes:
20.                    login_attempts[ip_address] = login_attempts.get(ip_address, 0) + 1
21.                    if login_attempts[ip_address] >= threshold and ip_address not in
summary:
22.                        summary.append(ip_address)
23.                    if status_code == '400':
24.                        error_400_count += 1
25.
26.    if summary:
27.        print("Unusual login attempts detected:")
28.        for ip_address in summary:
29.            print(f"IP: {ip_address} - Failed login attempts:
{login_attempts[ip_address]}")
30.
```

```

31. if error_400_count >= threshold:
32.     print(f"Unusual amount of 400 errors detected: {error_400_count}")
33.
34. if not summary and error_400_count < threshold:
35.     print("No unusual login attempts or errors found.")
36.     print("Detection complete.")
37.
38.detect_unusual_login_attempts()

```

3. Cron Job: I set up a cron job (Hira, 2023) to run the get_log.sh script automatically every 24 hours at midnight (00:00). The output of the command is appended to the file /home/user/log/cronjob_logs.

```

1. 0 0 * * * /bin/bash /home/user/get_logs.sh >>
   /home/user/log/cronjob_logs

```

Unusual Behavior: In the monitoring process, the following elements would constitute a "flag" to alert the manager:

1. Unusual Login Attempts: If the number of failed login attempts from a specific IP address exceeds the defined threshold (5), it indicates potential brute-force or unauthorized access attempts.
2. Unusual 400 Errors: If the number of 400 errors (Bad Request) exceeds the defined threshold, it suggests a high frequency of malformed or malicious requests to the server.

Any detection of unusual login attempts or an unusual amount of 400 errors would trigger an alert to the manager. The manager can then investigate and take appropriate action to mitigate any potential security risks.

Expected Outputs:

1. No unusual login attempts: If no unusual login attempts or errors are found in the log file, the script will print a message stating that no unusual login attempts or errors were detected. For example:

```
user@user-pc:~$ ./get_logs.sh
Unusual login attempts detected:
IP: 172.16.14.50 - Failed login attempts: 5
Detection complete.
```

2. Summary of Unusual Login Attempts: The script will print the IP addresses with the corresponding number of failed login attempts that exceed the threshold. For example:

```
user@user-pc:~$ ./get_logs.sh
No unusual login attempts or errors found.
Detection complete.
```

3. Detection of Unusual 400 Errors: If the number of 400 errors (Bad Request) exceeds the threshold, the script will print a message indicating the detection of an unusual amount of 400 errors. For example:

```
1. Unusual amount of 400 errors detected: 10
```

Expected results are important as they provide insights into potential security threats and issues. Unusual login attempts indicate unauthorized access attempts, while unusual 400 errors suggest malicious or malformed requests. The summary helps identify IPs requiring investigation or blocking. Detecting these issues promptly strengthens security, and absence of unusual activity assures normal network operation.

Potential improvements for the workflow include:

1. Real-Time Monitoring: Instead of running the script on a scheduled basis, explore options for real-time monitoring. Implement a solution that can continuously monitor logs and trigger alerts as soon as unusual behavior is detected.

2. More automation: Further automate the whole procedure, including sending email to the manager through scripting as mentioned earlier.

References:

SMTPLIB - SMTP protocol client. Python documentation. (n.d.).

<https://docs.python.org/3/library/smtplib.html>

Hira, Z. (2023, April 1). *How to automate tasks with Cron jobs in linux*.

freeCodeCamp.org. <https://www.freecodecamp.org/news/cron-jobs-in-linux/>