Turn a New Leaf Log Monitoring

Prepared by: Mahad Mohamood

Date: July 23, 2024

Table of Contents

Introduction

This report outlines the implementation of cybersecurity measures within 'Turn a New Leaf' with a focus on establishing a robust workflow for monitoring network traffic to detect and address potential security threats promptly.

As the Access Log Analyst at Turn a New Leaf, the role involves monitoring network activity for unusual patterns, particularly focusing on failed login attempts and other suspicious activities. Weekly updates are provided to senior management to ensure organizational security and compliance.

To achieve effective monitoring, scripts will be developed and deployed to automate the detection of abnormal network traffic. These scripts will enable quick identification of potential security breaches, allowing for timely intervention and mitigation.

Solutions

Turn a New Leaf operates both Windows and Linux machines, with two web servers in their network. Our monitoring efforts will concentrate on the log files of these web servers.

According to middleware, "a log is a record of events that have occurred, typically including a timestamp and event details. Logs are commonly used to troubleshoot issues, monitor system performance, and identify security concerns. Software programs and systems generate log files containing information about the application, user device, time, IP address, and more."("What is log monitoring? A detailed guide (updated)," n.d.).

Furthermore, Log monitoring is the "observation of logs and metrics from a system, typically combined with dashboards, visualizations, and alerts. As engineers keep an eye on the present state of the application, they can identify issues or anomalies."("Logging vs monitoring: Improved visibility," 2024).

We will utilize Bash scripts to parse logs for the detection of abnormal network traffic. Bash is "a computer program that allows you to directly control a computer’s operating system (OS) with a graphical user interface (GUI) or command-line interface (CLI)."(Klein, 2024).

Additionally, we shall use Python to handle data analysis. According to amazon, "Python is a programming language that is widely used in web applications, software development, data science, and machine learning (ML)." ("What is Python? - Python programming language explained - AWS," n.d.).

Some of the expected outcomes of this report include the detection of failed login attempts and large data transfers using Bash, and the detailed analysis of log data using Python, ensuring swift action and thorough documentation.

Lets now proceed with a more in-depth analysis of the solutions and the corresponding code snippets.

Monitoring targets and key commands

The online system of Turn a New Leaf is maintained using an Apache web server. According to sumologic, "Apache HTTP Server is a free and open-source web server that delivers web content through the internet." (Hernandez, 2019).

Apache servers "provides a variety of different mechanisms for logging everything that happens on your server, from the initial request, through the URL mapping process, to the final resolution of the connection, including any errors that may have occurred in the process." ("Log files," n.d.).

For this report, we shall focus on the access log and error logs of the Apache web servers. According to sematext, "The Apache access logs are text files that include information about all the requests processed by the Apache server. You can expect to find information like the time of the request, the requested resource, the response code, time it took to respond, and the IP address used to request the data"(Sematext, 2023). On the other hand, "this file contains diagnostic information about any errors were encountered while processing requests"(Isaiah, 2023).

With the identified log files in focus, we will now specify the types of issues to detect and develop corresponding code snippets to filter these issues from the logs.

1. Unusual Network Traffic Patterns that deviate from the norm

In Access log, we can use the below code snippet to look for entries with unusually large Content-Length value, such as data transfer spikes exceeding 1GB in a short period.

```
awk '$10 > 1000000000' /var/log/apache2/access.log
# filters access.log for requests with a Content-Length greater than 1GB
```

In the Access log, we can use the below code snippet to look for IP addresses that deviate from the usual geographic locations of users.

```
awk '{print $1}' /var/log/apache2/access.log | sort | uniq -c | sort -nr
# list IP addresses making requests, sorted by frequency
```

2. High Number of Failed Login Attempts

In the Access log, we can use the below code snippet to look for repeated login attempts to the login endpoint, which could indicate a brute force attack

```
grep "/login" /var/log/apache2/access.log | grep "401" | awk '{print $1}' | sort |
uniq -c | awk '$1 > 5'
# filter failed login attempts (401 status code) for the /login endpoint
# counting occurrences per IP
# identify IPs with more than 5 failures
```

In the Error log, we can use the below code snippet to look for errors related to login failures.

```
grep "login failed" /var/log/apache2/error.log
# searches for error messages related to failed login attempts
```

3. Unauthorized Access Attempts

In the Access log, we can use the below code snippet to identify attempts to access restricted URLs or directories.

```
grep "/restricted-directory" /var/log/apache2/access.log
# searches for access attempts to a specified restricted directory or URL
```

In the Error log, we can use the below snippet to look for errors indicating unauthorized access.

```
grep "403 Forbidden" /var/log/apache2/error.log
# searches for 403 Forbidden errors in error.log, indicating unauthorized access
attempts
```

Key scripts and the log monitoring workflow process

With the identified log types and filtering commands established, the next step is to outline the Bash scripts for monitoring these logs and the subsequent Python scripts for their analysis.

First, we will develop the Bash script, verify its functionality using test logs, and subsequently perform Python analysis. Screenshots of terminal results will be provided at the conclusion of each phase.

Create bash script to monitor logs

1 . Go into the documents directory

```
cd Documents
```

2 . Create monitor_logs.sh file

```
nano monitor_logs.sh
```

3 . Add the below bash script.

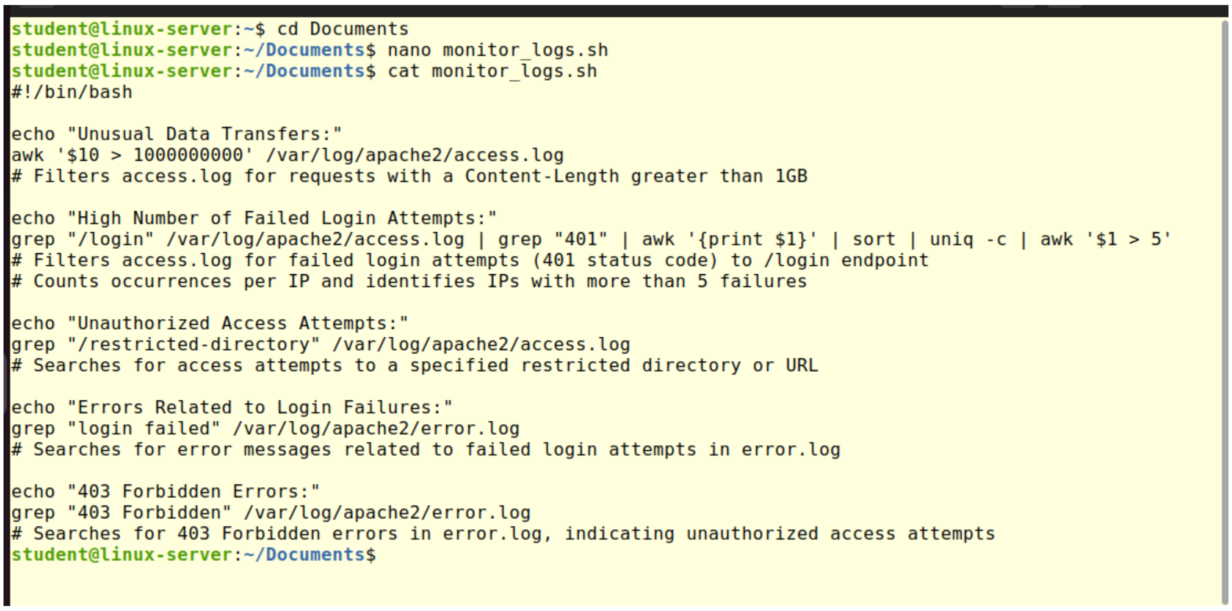
In this script we combine the code snippets from the 'what to monitor' section into the executable file called monitor_logs.sh.

```
#!/bin/bash

echo "Unusual Data Transfers:"
awk '$10 > 1000000000' /var/log/apache2/access.log
# Filters access.log for requests with a Content-Length greater than 1GB

echo "High Number of Failed Login Attempts:"
grep "/login" /var/log/apache2/access.log | grep "401" | awk '{print $1}' | sort |
uniq -c | awk '$1 > 5'
# Filters access.log for failed login attempts (401 status code) to /login
endpoint
# Counts occurrences per IP and identifies IPs with more than 5 failures

echo "Unauthorized Access Attempts:"
grep "/restricted-directory" /var/log/apache2/access.log
# Searches for access attempts to a specified restricted directory or URL

echo "Errors Related to Login Failures:"
grep "login failed" /var/log/apache2/error.log
# Searches for error messages related to failed login attempts in error.log

echo "403 Forbidden Errors:"
grep "403 Forbidden" /var/log/apache2/error.log
# Searches for 403 Forbidden errors in error.log, indicating unauthorized access
attempts
```

4 . Save and exit

```
ctrl + o
Press enter
ctrl  x
```

Below is the verification result from the Linux terminal:



```
student@linux-server:~$ cd Documents
student@linux-server:~/Documents$ nano monitor_logs.sh
student@linux-server:~/Documents$ cat monitor_logs.sh
#!/bin/bash

echo "Unusual Data Transfers:"
awk '$10 > 1000000000' /var/log/apache2/access.log
# Filters access.log for requests with a Content-Length greater than 1GB

echo "High Number of Failed Login Attempts:"
grep "/login" /var/log/apache2/access.log | grep "401" | awk '{print $1}' | sort | uniq -c | awk '$1 > 5'
# Filters access.log for failed login attempts (401 status code) to /login endpoint
# Counts occurrences per IP and identifies IPs with more than 5 failures

echo "Unauthorized Access Attempts:"
grep "/restricted-directory" /var/log/apache2/access.log
# Searches for access attempts to a specified restricted directory or URL

echo "Errors Related to Login Failures:"
grep "login failed" /var/log/apache2/error.log
# Searches for error messages related to failed login attempts in error.log

echo "403 Forbidden Errors:"
grep "403 Forbidden" /var/log/apache2/error.log
# Searches for 403 Forbidden errors in error.log, indicating unauthorized access attempts
student@linux-server:~/Documents$
```

Test the bast script with fake logs

Testing with fake logs allows us to verify that our monitoring script correctly identifies and processes relevant log entries.

This ensures the script functions as expected before deploying it in a live environment, thereby minimizing the risk of errors.

A. Create test logs :

1 . Create a Test Log File:

Save your fake log entries in a new file, e.g., test_logs.log.

```
touch ~/Documents/test_logs.log
```

2 . Open the file :

```
nano test_logs.log
```

3 . Add Fake Log Entries:

Paste fake entries into test_logs.log.

We add a mix of fake successful and errors logs to make sure that the bash script can filter for only the problematic logs

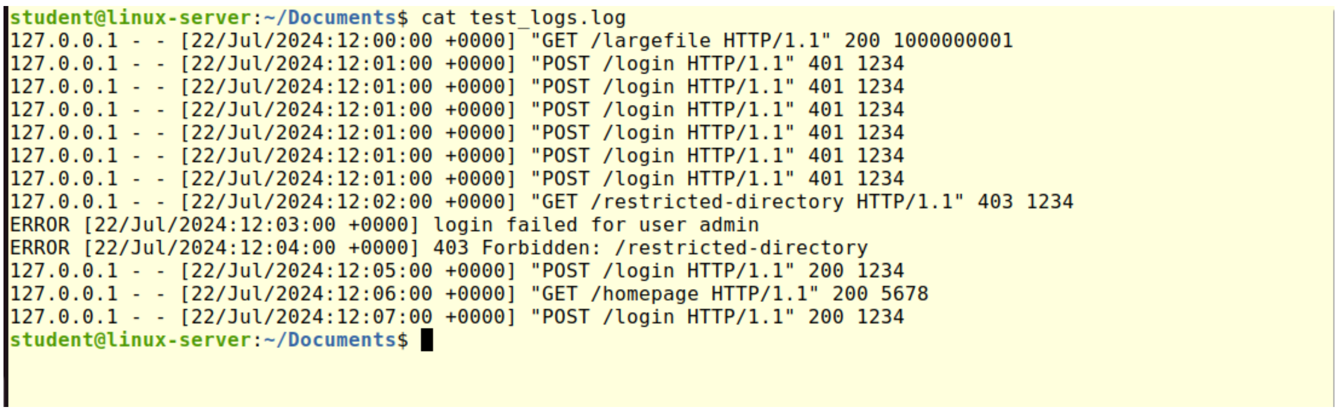
```
127.0.0.1 - - [22/Jul/2024:12:00:00 +0000] "GET /largefile HTTP/1.1" 200
1000000001
127.0.0.1 - - [22/Jul/2024:12:01:00 +0000] "POST /login HTTP/1.1" 401 1234
```

```
127.0.0.1 - - [22/Jul/2024:12:01:00 +0000] "POST /login HTTP/1.1" 401 1234
127.0.0.1 - - [22/Jul/2024:12:01:00 +0000] "POST /login HTTP/1.1" 401 1234
127.0.0.1 - - [22/Jul/2024:12:01:00 +0000] "POST /login HTTP/1.1" 401 1234
127.0.0.1 - - [22/Jul/2024:12:01:00 +0000] "POST /login HTTP/1.1" 401 1234
127.0.0.1 - - [22/Jul/2024:12:01:00 +0000] "POST /login HTTP/1.1" 401 1234
127.0.0.1 - - [22/Jul/2024:12:02:00 +0000] "GET /restricted-directory HTTP/1.1"
403 1234
ERROR [22/Jul/2024:12:03:00 +0000] login failed for user admin
ERROR [22/Jul/2024:12:04:00 +0000] 403 Forbidden: /restricted-directory
127.0.0.1 - - [22/Jul/2024:12:05:00 +0000] "POST /login HTTP/1.1" 200 1234
127.0.0.1 - - [22/Jul/2024:12:06:00 +0000] "GET /homepage HTTP/1.1" 200 5678
127.0.0.1 - - [22/Jul/2024:12:07:00 +0000] "POST /login HTTP/1.1" 200 1234
```

4 . Save and exit

```
ctrl + o
Press enter
ctrl  x
```

Below is the verification result from the Linux terminal:



```
student@linux-server:~/Documents$ cat test_logs.log
127.0.0.1 - - [22/Jul/2024:12:00:00 +0000] "GET /largefile HTTP/1.1" 200 1000000001
127.0.0.1 - - [22/Jul/2024:12:01:00 +0000] "POST /login HTTP/1.1" 401 1234
127.0.0.1 - - [22/Jul/2024:12:01:00 +0000] "POST /login HTTP/1.1" 401 1234
127.0.0.1 - - [22/Jul/2024:12:01:00 +0000] "POST /login HTTP/1.1" 401 1234
127.0.0.1 - - [22/Jul/2024:12:01:00 +0000] "POST /login HTTP/1.1" 401 1234
127.0.0.1 - - [22/Jul/2024:12:01:00 +0000] "POST /login HTTP/1.1" 401 1234
127.0.0.1 - - [22/Jul/2024:12:01:00 +0000] "POST /login HTTP/1.1" 401 1234
127.0.0.1 - - [22/Jul/2024:12:02:00 +0000] "GET /restricted-directory HTTP/1.1" 403 1234
ERROR [22/Jul/2024:12:03:00 +0000] login failed for user admin
ERROR [22/Jul/2024:12:04:00 +0000] 403 Forbidden: /restricted-directory
127.0.0.1 - - [22/Jul/2024:12:05:00 +0000] "POST /login HTTP/1.1" 200 1234
127.0.0.1 - - [22/Jul/2024:12:06:00 +0000] "GET /homepage HTTP/1.1" 200 5678
127.0.0.1 - - [22/Jul/2024:12:07:00 +0000] "POST /login HTTP/1.1" 200 1234
student@linux-server:~/Documents$
```

B. Modify monitor_logs for testing :

We modify monitor_logs.sh script to use test_logs.log instead of real Apache logs.

We do this by replacing the paths of the real Apache logs (/var/log/apache2/access.log and /var/log/apache2/error.log) with the path to the test log file ($HOME/Documents/test_logs.log).

This ensures that the script reads from test_logs.log for all its parsing and monitoring tasks, allowing for safe testing without impacting real log data

1 . Open monitor_logs.sh file

```
nano monitor_logs.sh
```

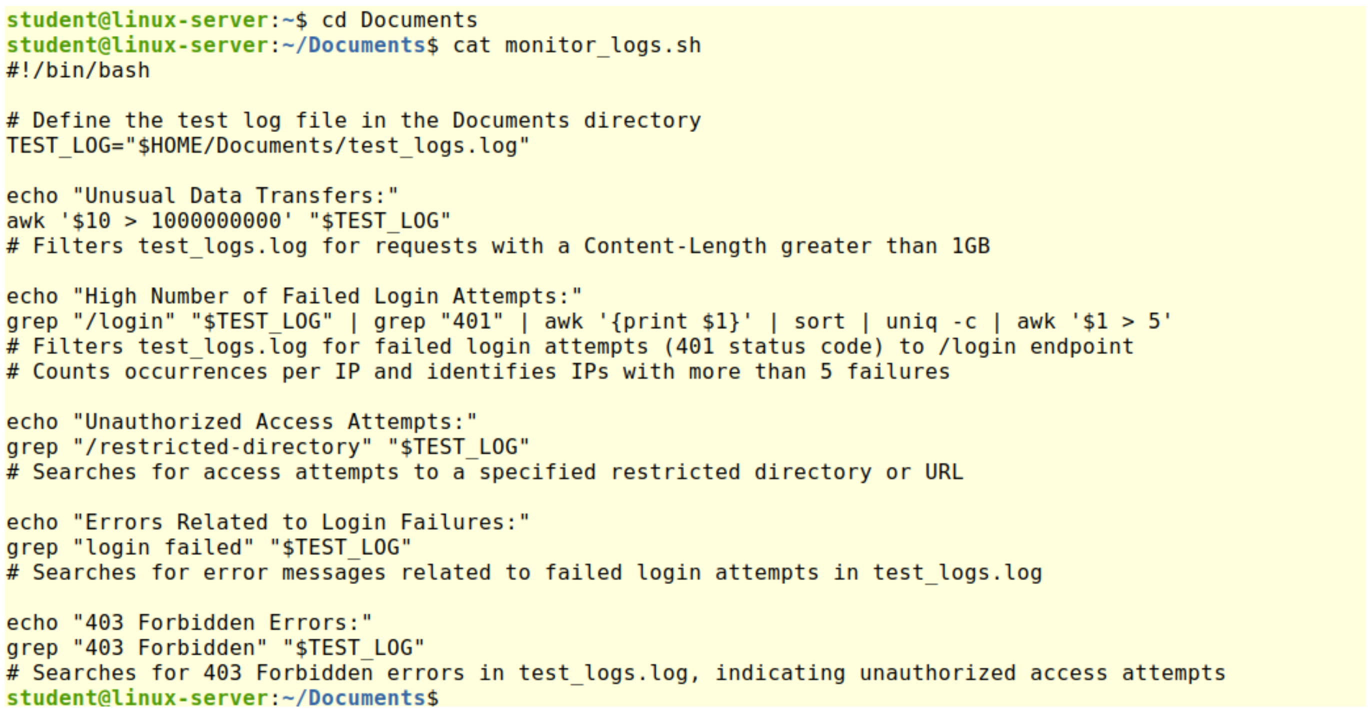
2 . Add modified bash script below

```
#!/bin/bash

# Define the test log file in the Documents directory
TEST_LOG="$HOME/Documents/test_logs.log"

echo "Unusual Data Transfers:"
awk '$10 > 1000000000' "$TEST_LOG"
# Filters test_logs.log for requests with a Content-Length greater than 1GB

echo "High Number of Failed Login Attempts:"
grep "/login" "$TEST_LOG" | grep "401" | awk '{print $1}' | sort | uniq -c | awk
'$1 > 5'
# Filters test_logs.log for failed login attempts (401 status code) to /login
endpoint
# Counts occurrences per IP and identifies IPs with more than 5 failures

echo "Unauthorized Access Attempts:"
grep "/restricted-directory" "$TEST_LOG"
# Searches for access attempts to a specified restricted directory or URL

echo "Errors Related to Login Failures:"
grep "login failed" "$TEST_LOG"
# Searches for error messages related to failed login attempts in test_logs.log

echo "403 Forbidden Errors:"
grep "403 Forbidden" "$TEST_LOG"
# Searches for 403 Forbidden errors in test_logs.log, indicating unauthorized
access attempts
```

3 . Save and exit

```
ctrl + o
Press enter
ctrl  x
```

Below is the verification result from the Linux terminal:



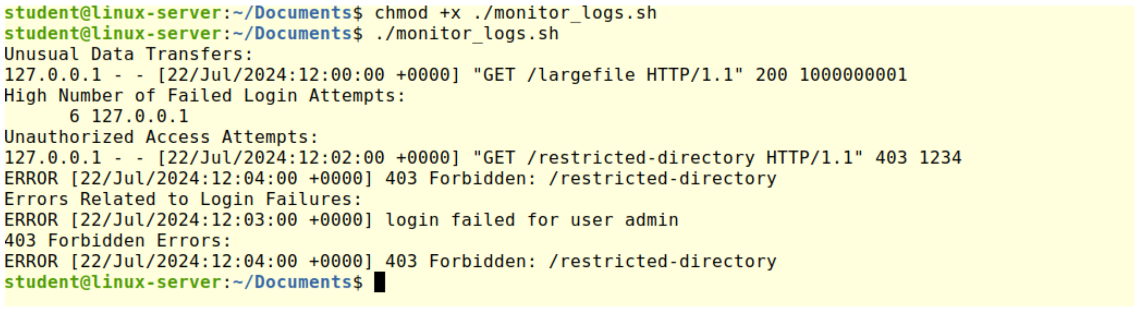
```
student@linux-server:~$ cd Documents
student@linux-server:~/Documents$ cat monitor_logs.sh
#!/bin/bash

# Define the test log file in the Documents directory
TEST_LOG="$HOME/Documents/test_logs.log"

echo "Unusual Data Transfers:"
awk '$10 > 1000000000' "$TEST_LOG"
# Filters test_logs.log for requests with a Content-Length greater than 1GB

echo "High Number of Failed Login Attempts:"
grep "/login" "$TEST_LOG" | grep "401" | awk '{print $1}' | sort | uniq -c | awk '$1 > 5'
# Filters test_logs.log for failed login attempts (401 status code) to /login endpoint
# Counts occurrences per IP and identifies IPs with more than 5 failures

echo "Unauthorized Access Attempts:"
grep "/restricted-directory" "$TEST_LOG"
# Searches for access attempts to a specified restricted directory or URL

echo "Errors Related to Login Failures:"
grep "login failed" "$TEST_LOG"
# Searches for error messages related to failed login attempts in test_logs.log

echo "403 Forbidden Errors:"
grep "403 Forbidden" "$TEST_LOG"
# Searches for 403 Forbidden errors in test_logs.log, indicating unauthorized access attempts
student@linux-server:~/Documents$
```

C. Execute monitor_logs with the test logs :

Execute your script to parse test_logs.log and verify results.

```
cd Documents

chmod +x ./monitor_logs.sh
# Ensure your monitor_logs.sh file has execute permissions

./monitor_logs.sh
# Run your script by specifying its path
```

Below is the verification result from the Linux terminal:



```
student@linux-server:~/Documents$ chmod +x ./monitor_logs.sh
student@linux-server:~/Documents$ ./monitor_logs.sh
Unusual Data Transfers:
127.0.0.1 - - [22/Jul/2024:12:00:00 +0000] "GET /largefile HTTP/1.1" 200 1000000001
High Number of Failed Login Attempts:
      6 127.0.0.1
Unauthorized Access Attempts:
127.0.0.1 - - [22/Jul/2024:12:02:00 +0000] "GET /restricted-directory HTTP/1.1" 403 1234
ERROR [22/Jul/2024:12:04:00 +0000] 403 Forbidden: /restricted-directory
Errors Related to Login Failures:
ERROR [22/Jul/2024:12:03:00 +0000] login failed for user admin
403 Forbidden Errors:
ERROR [22/Jul/2024:12:04:00 +0000] 403 Forbidden: /restricted-directory
student@linux-server:~/Documents$ 
```

Create Python Script for data analysis

Python is useful for analyzing logs because it has powerful libraries like pandas for data manipulation and analysis. It can easily handle large volumes of data and perform complex operations with simple code. Additionally, Python's readability and extensive library support make it ideal for automating and streamlining log analysis tasks.

1 . Install pandas:

Run the following command to install the pandas library:

```
pip3 install pandas
```

2 . Create python file called analyze_logs.py

```
cd Documents

nano analyze_logs.py
```

3 . Add the following script

```
import subprocess
# Import a module to run other programs from within this script

import pandas as pd
# Import a module to handle data in tables

import os
# Import a module to interact with the operating system

import re
# Import a module to work with regular expressions, which are patterns for
matching text

script_path = os.path.expanduser('~/Documents/monitor_logs.sh')
# Declare the location of the script that monitors logs

result = subprocess.run([script_path], capture_output=True, text=True)
# Run the monitor_logs.sh script and save what it outputs

output = result.stdout
# Get the output of the script

log_entries = []
# Create a empty list to store log entries from the output

log_pattern = re.compile(
    r'(?P<ip>[\d\.]+) - - \[(?P<datetime>[^\]]+)\] "(?P<method>\w+) (?P<path>
[^\s]+) HTTP/\d\.\d" (?P<status>\d{3}) (?P<size>\d+)'
)
# Create a regex pattern to find and extract log details (IP address, date/time,
method, path, status, size) from log entries

error_pattern = re.compile(r'ERROR \[(?P<datetime>[^\]]+)\] (?P<message>.+)')
# Create a pattern to find details in error messages (date/time, message)

# The below loop parses the log output and extract details
```

```
for line in output.split('\n'):
    # Go through each line in the output from the script

    line = line.strip()
    # Remove any extra spaces from the line

    if not line:
        continue
        # Skip this line if it's empty

    log_match = log_pattern.match(line)
    # Try to match the log entry pattern

    if log_match:
        # If the line matches the log entry pattern:

        log_entries.append([
            "Log",
            log_match.group('datetime'),
            log_match.group('ip'),
            log_match.group('method'),
            log_match.group('path'),
            log_match.group('status'),
            log_match.group('size'),
        ])
        # Add the log entry details to the list

        continue
        # Move to the next line

    error_match = error_pattern.match(line)
    # Try to match the error message pattern

    if error_match:
        # If the line matches the error message pattern:

        log_entries.append([
            "Error",
            error_match.group('datetime'),
            None,
            None,
            None,
            None,
            error_match.group('message'),

        ])
        # Add the error message details to the list

columns = ['Category', 'Date/Time', 'IP Address', 'Method', 'Path', 'Status',
'Size/Message']
# Set the names of the columns for the table
# Create a table from the list of log entries
df = pd.DataFrame(log_entries, columns=columns)

# Show the table
print(df)
```

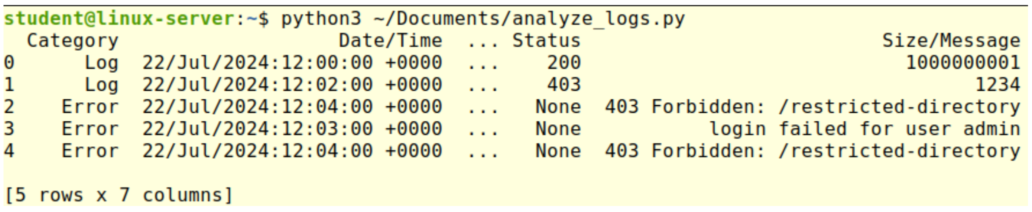
4 . Save and exit

```
ctrl + o
Press enter
ctrl  x
```

5 . Run the python script

```
cd Documents

chmod +x ./monitor_logs.sh
# Ensure your monitor_logs.sh file has execute permissions

python3 ~/Documents/analyze_logs.py
```

Below is the verification result from the Linux terminal:



```
student@linux-server:~$ python3 ~/Documents/analyze_logs.py
  Category                  Date/Time  ... Status                      Size/Message
0      Log  22/Jul/2024:12:00:00 +0000  ...    200                        1000000001
1      Log  22/Jul/2024:12:02:00 +0000  ...    403                              1234
2    Error  22/Jul/2024:12:04:00 +0000  ...   None  403 Forbidden: /restricted-directory
3    Error  22/Jul/2024:12:03:00 +0000  ...   None           login failed for user admin
4    Error  22/Jul/2024:12:04:00 +0000  ...   None  403 Forbidden: /restricted-directory

[5 rows x 7 columns]
```

Potential Iterations

To enhance workflow efficiency, focus could be placed on increasing automation of log checks and implementation of machine learning for advanced anomaly detection. Automating the log monitoring process would save time and ensure real-time alerts, while machine learning could enhance the accuracy of anomaly detection. Additionally, preparation of playbooks and escalation matrices for automated alert handling would be beneficial.

To develop the necessary skills, online courses in cybersecurity automation and machine learning could be pursued by the log analyst. Additionally, relevant certifications such as CompTIA Security+ or CISSP could be obtained to deepen knowledge and credibility. Participation in cybersecurity challenges and hackathons would provide practical experience and ensure staying updated with the latest trends and techniques.

Conclusion

The project established a comprehensive log monitoring system for Turn a New Leaf, a non-profit organization supporting rural youth employment. The workflow includes continuous monitoring of logs, with Bash scripts handling log parsing and filtering to identify unusual network traffic and potential security issues, such as failed login attempts and large data transfers, while Python is utilized for data analysis. Key improvements for the future include increasing automation and incorporating machine learning for advanced anomaly detection.

References

1. What is log monitoring? A detailed guide (updated). Middleware. (n.d.). https://middleware.io/blog/what-is-log-monitoring/
2. Logging vs monitoring: Improved visibility - crowdstrike. crowdstrike.com. (2024, July 8). https://www.crowdstrike.com/cybersecurity-101/observability/logging-vs-monitoring/
3. Klein, M. (2024, April 9). What is bash used for?. Codecademy Blog. https://www.codecademy.com/resources/blog/what-is-bash-used-for/
4. What is python? - python programming language explained - AWS. (n.d.). https://aws.amazon.com/what-is/python/
5. Hernandez, J. (2019, May 8). What is Apache? in-depth overview of Apache Web Server. Sumo Logic. https://www.sumologic.com/blog/apache-web-server-introduction/
6. Log files. Log Files - Apache HTTP Server Version 2.4. (n.d.). https://httpd.apache.org/docs/2.4/logs.html
7. Sematext, R. (2023, November 23). How to view & analyze apache access & error log files. Sematext. https://sematext.com/blog/apache-logs/
8. Isaiah, A. (2023, November 23). How to view and configure Apache Access & error logs. Better Stack Community. https://betterstack.com/community/guides/logging/how-to-view-and-configure-apache-access-and-error-logs/