Turn a New Leaf

Log Monitoring Workflow

Prepared by: Prinson D'silva
Date: January 7th, 2025

Table of Contents

Summary

This report is created for Turn a New Leaf which is a non-profit organization supporting employment for youth belonging to rural communities. Their usual protocol for members based on government regulatory requirements is to log into the system weekly (Thursdays) and confirm or update their employment status along with information to their active job listings and ones they are currently pursuing.

The purpose of this report is to provide an update to management of Turn a New Leaf, based on results obtained from suspicious activities like an unusual number of incorrect logins or access to an unknown website which would be sent to management for review.

Workflow:

The workflow for this project was divided into the following steps:

- Log Collection
- Analysis of Collected logs
- Report of analysis

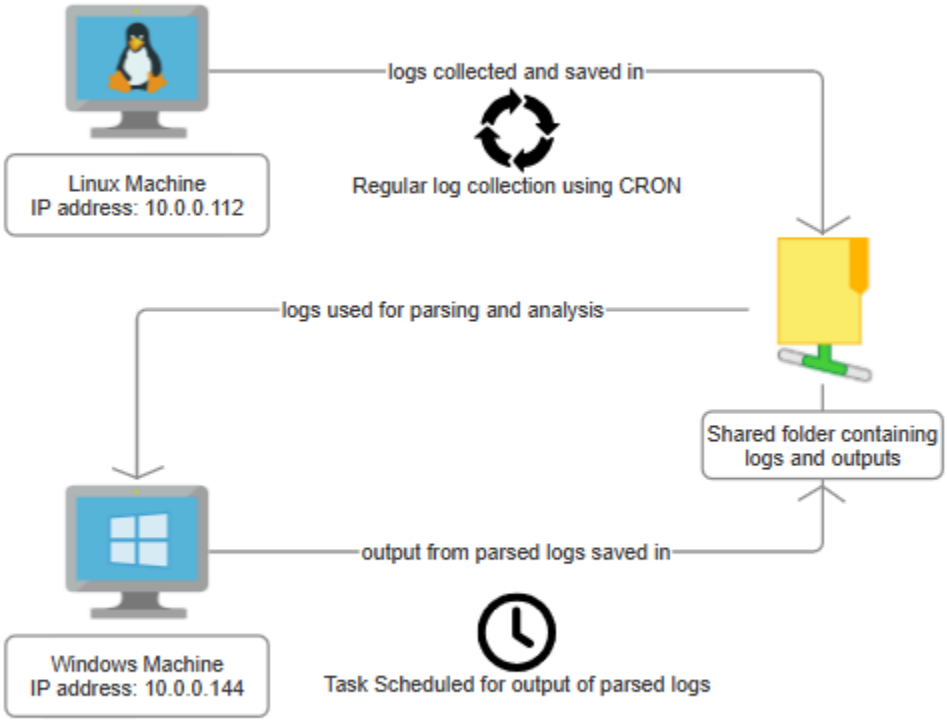


Fig 1: Log Monitoring Workflow (Turn a New Leaf)

Solutions

As mentioned, the log workflow monitoring for Turn a New Leaf is divided into the following sections:

1. Log Collection

The logs would be collected from the following files that would flag as potential Indicators of Compromise (IOCs):

- Apache2 Web-Server logs - The log would scan for "HTTP 404 - File not found" error which tells an attacker that the requested file doesn't exist rather than that they don't have access to the file. This can help the attacker to decide their next step. [OWASP, 2025]
- Authentication logs - A large number of failed login attempts could indicate a potential Brute Force Attack either by Password guessing or Spraying. [Mitre, 2025]

These files would be copied over from the listed directories to the shared output folder between both systems using a bash script called **logs_collected.sh**

```
#Define source and destination path
Source1="/var/log/apache2/access.log"
Source2="/var/log/auth.log"
Destination="/media/sf_Shared/Logs/Output"

#Move the log file from Source1 into the Destination folder
if [ -f "$Source1" ]; then
        cp "$Source1" "$Destination"
        echo "Copied $Source1 to $Destination on $(date)" >> /media/sf_Shared/Logs/Output/moved_files.txt

#Check for any existing logs, if not then display the following message
else
        echo "File $Source1 not found on $(date)" >> /media/sf_Shared/Logs/Output/moved_files.txt
fi

#Now Move the log file from Source2 into the Destination folder
if [ -f "$Source2" ]; then
        cp "$Source2" "$Destination"
        echo "Copied $Source2 to $Destination on $(date)" >> /media/sf_Shared/Logs/Output/moved_files.txt

#Check for any existing logs, if not then display the following message
else
        echo "File $Source2 not found on $(date)" >> /media/sf_Shared/Logs/Output/moved_files.txt
fi
```

Fig 2: Bash script used for moving logs to Output folder (logs_collected.sh)

The output of this script is a generated text file which shows the records of the copied files:

Open

moved_files.txt
sf_Shared /media/sf_Shared/Logs/Output

```
Copied /var/log/apache2/access.log to /media/sf_Shared/Logs/Output on Wed Jan  8 08:55:54 PM EST 2025
Copied /var/log/auth.log to /media/sf_Shared/Logs/Output on Wed Jan  8 08:55:54 PM EST 2025
Copied /var/log/apache2/access.log to /media/sf_Shared/Logs/Output on Wed Jan  8 08:57:02 PM EST 2025
Copied /var/log/auth.log to /media/sf_Shared/Logs/Output on Wed Jan  8 08:57:02 PM EST 2025
Copied /var/log/apache2/access.log to /media/sf_Shared/Logs/Output on Wed Jan  8 10:57:05 PM EST 2025
Copied /var/log/auth.log to /media/sf_Shared/Logs/Output on Wed Jan  8 10:57:05 PM EST 2025
Copied /var/log/apache2/access.log to /media/sf_Shared/Logs/Output on Wed Jan  8 10:57:50 PM EST 2025
Copied /var/log/auth.log to /media/sf_Shared/Logs/Output on Wed Jan  8 10:57:51 PM EST 2025
Copied /var/log/apache2/access.log to /media/sf_Shared/Logs/Output on Wed Jan  8 11:06:53 PM EST 2025
Copied /var/log/auth.log to /media/sf_Shared/Logs/Output on Wed Jan  8 11:06:53 PM EST 2025
Copied /var/log/apache2/access.log to /media/sf_Shared/Logs/Output on Thu Jan  9 12:05:10 AM EST 2025
Copied /var/log/auth.log to /media/sf_Shared/Logs/Output on Thu Jan  9 12:05:10 AM EST 2025
Copied /var/log/apache2/access.log to /media/sf_Shared/Logs/Output on Thu Jan  9 12:08:12 AM EST 2025
Copied /var/log/auth.log to /media/sf_Shared/Logs/Output on Thu Jan  9 12:08:12 AM EST 2025
Copied /var/log/apache2/access.log to /media/sf_Shared/Logs/Output on Thu Jan  9 12:26:13 AM EST 2025
Copied /var/log/auth.log to /media/sf_Shared/Logs/Output on Thu Jan  9 12:26:13 AM EST 2025
```

This is the Output of the log files after copying them over to the shared folder: -

- Access log:

Open

access.log
sf_Shared /media/sf_Shared/Logs/Output

```
127.0.0.1 - - [09/Jan/2025:00:25:25 -0500] "HEAD /nonexistentpage HTTP/1.1" 404 140 "-" "curl/7.81.0"
127.0.0.1 - - [09/Jan/2025:00:25:29 -0500] "HEAD /nonexistentpage HTTP/1.1" 404 140 "-" "curl/7.81.0"
127.0.0.1 - - [09/Jan/2025:00:25:31 -0500] "HEAD /nonexistentpage HTTP/1.1" 404 140 "-" "curl/7.81.0"
127.0.0.1 - - [09/Jan/2025:00:25:33 -0500] "HEAD /nonexistentpage HTTP/1.1" 404 140 "-" "curl/7.81.0"
127.0.0.1 - - [09/Jan/2025:00:25:34 -0500] "HEAD /nonexistentpage HTTP/1.1" 404 140 "-" "curl/7.81.0"
127.0.0.1 - - [09/Jan/2025:00:25:37 -0500] "HEAD /nonexistentpage HTTP/1.1" 404 140 "-" "curl/7.81.0"
127.0.0.1 - - [09/Jan/2025:00:25:51 -0500] "HEAD /tighthouselabs HTTP/1.1" 404 140 "-" "curl/7.81.0"
```

- Authentication log:

Open

auth.log
sf_Shared /media/sf_Shared/Logs/Output

```
Jan  5 16:30:01 linux-server CRON[3681]: pam_unix(cron:session): session closed for user root
Jan  5 17:03:47 linux-server gdm-password]: pam_unix(gdm-password:auth): authentication failure; logname= uid=0 euid=0 tty=/dev/tty1 ruser= rhost=  user=student
Jan  5 17:04:08 linux-server gdm-password]: gkr-pam: unlocked login keyring
Jan  5 17:11:14 linux-server gdm-password]: pam_unix(gdm-password:auth): authentication failure; logname= uid=0 euid=0 tty=/dev/tty1 ruser= rhost=  user=student
Jan  5 17:11:20 linux-server gdm-password]: gkr-pam: unlocked login keyring
Jan  5 17:17:01 linux-server CRON[4264]: pam_unix(cron:session): session opened for user root(uid=0) by (uid=0)
Jan  5 17:17:01 linux-server CRON[4264]: pam_unix(cron:session): session closed for user root
Jan  5 17:23:09 linux-server gdm-password]: gkr-pam: unlocked login keyring
Jan  5 17:27:31 linux-server polkitd(authority=local): Operator of unix-session:2 successfully authenticated as unix-user:student to gain TEMPORARY authorization
helper for unix-process:4436:557102 [/bin/sh -c pkexec /usr/libexec/gvfsd-admin "$@" --address $DBUS_SESSION_BUS_ADDRESS --dir $XDG_RUNTIME_DIR gvfsd-admin --spa
(owned by unix-user:student)
Jan  5 17:27:31 linux-server pkexec: pam_unix(polkit-1:session): session opened for user root(uid=0) by (uid=1000)
Jan  5 17:27:31 linux-server pkexec[4437]: student: Executing command [USER=root] [TTY=unknown] [CWD=/home/student] [COMMAND=/usr/libexec/gvfsd-admin --spawner :
unix:path=/run/user/1000/bus --dir /run/user/1000]
Jan  5 17:27:35 linux-server polkitd(authority=local): Operator of unix-session:2 successfully authenticated as unix-user:student to gain TEMPORARY authorization
for unix-process:3870:462832 [/usr/bin/nautilus --gapplication-service] (owned by unix-user:student)
Jan  5 17:27:52 linux-server polkitd(authority=local): Operator of unix-session:2 successfully authenticated as unix-user:student to gain TEMPORARY authorization
for unix-process:4463:559068 [/usr/bin/gedit --gapplication-service] (owned by unix-user:student)
Jan  5 17:28:01 linux-server polkitd(authority=local): Operator of unix-session:2 successfully authenticated as unix-user:student to gain TEMPORARY authorization
for unix-process:4497:560090 [/usr/bin/gedit --gapplication-service] (owned by unix-user:student)
Jan  5 17:30:01 linux-server CRON[4548]: pam_unix(cron:session): session opened for user root(uid=0) by (uid=0)
Jan  5 17:30:01 linux-server CRON[4548]: pam_unix(cron:session): session closed for user root
Jan  5 19:23:00 linux-server CRON[4574]: pam_unix(cron:session): session opened for user root(uid=0) by (uid=0)
Jan  5 19:23:00 linux-server CRON[4574]: pam_unix(cron:session): session closed for user root
```

For automation of this process, a CRON job was created which runs the script from logs_collected.sh weekly, i.e. every Friday at midnight.

```
  GNU nano 6.2
* 0 * * 5 /media/sf_Shared/Logs/Scripts/logs_collected.sh
```

Fig 3: CRON job for running logs_collected.sh

2. Analysis of Collected Logs:

The logs collected from the Linux system would then be used by the Windows system to run a complex script using python.

The output of this script would summarize our concerns, i.e. the number of failed login attempts as well as 404 errors thrown by the web server, given our criteria is met, which is:

There are >5 errors or failed logins in < 5 minutes.

The script run in the Windows system using python is called **parsed_logs.sh**

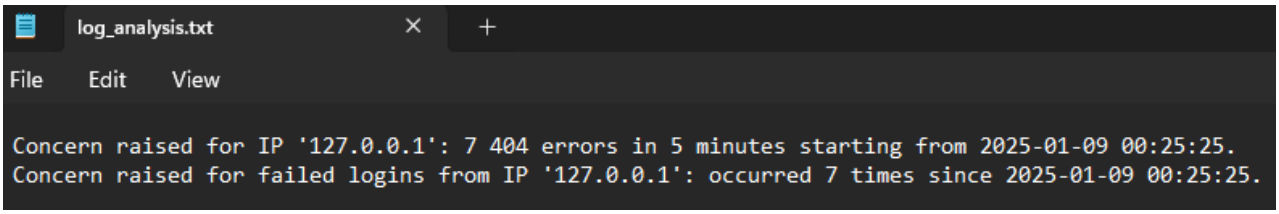
Z: > Logs > Scripts > parsed_logs.py > ...

```python
import re
from collections import defaultdict
from datetime import datetime, timedelta

# Define the regex patterns for "failed Login" and "404" errors
failed_login_pattern = re.compile(r"failed password")
error_404_pattern = re.compile(r"\b404")

# Path to the linux log files
auth_log_file_path = r"Z:\Logs\Output\auth.log"
web_server_file_path = r"Z:\Logs\Output\access.log"

# Group logs by IP
logs_by_ip = defaultdict(list)

# Parse the logs and add relevant entries
def parse_logs(file_path, pattern, is_404=False):
    with open(file_path, 'r') as file:
        for line in file:
            if pattern.search(line):
                parts = line.split(" ")  # Split into all parts without limiting the number of splits
                ip = parts[0]  # IP is assumed to be the first part
                timestamp = parts[3][1:]  # Remove the leading '[' in timestamp

                try:
                    if is_404:
                        # For 404 logs, parse the request and status
                        web_file_request = parts[5]  # 5th index is the requested file path
                        action = parts[8]  # 8th index is the HTTP status code
                        logs_by_ip[ip].append((timestamp, web_file_request, action))
                    else:
                        logs_by_ip[ip].append((timestamp, line))
                except IndexError:
                    print(f"Error parsing log line: {line}")  # Handle unexpected log formats

# Check for concerns: 5 or more 404 errors in 5 minutes and failed logins
def check_log_concerns(logs_by_ip, is_404=False):
    concerns = []
    for ip, logs in logs_by_ip.items():
        # Convert timestamps into datetime objects for sorting and comparison
        logs.sort(key=lambda x: datetime.strptime(x[0], "%d/%b/%Y:%H:%M:%S"))
        for i in range(len(logs)):
            base_time = datetime.strptime(logs[i][0], "%d/%b/%Y:%H:%M:%S")
            # Count logs within the 5-minute window from base_time
            count = sum(1 for log in logs if base_time <= datetime.strptime(log[0], "%d/%b/%Y:%H:%M:%S") < base_time + timedelta(minutes=5))

            if count >= 5:  # If there are 5 or more events within 5 minutes
                concern_type = "404 errors" if is_404 else "failed logins"
                concerns.append((ip, count, base_time, concern_type))
                break  # No need to check further logs for the same IP
    return concerns

# Check for 404 errors and failed logins concerns
access_log_404_concerns = check_log_concerns(logs_by_ip, is_404=True)
login_failed_concerns = check_log_concerns(logs_by_ip, is_404=False)

# Output results
output_file_path = r'Z:\Logs\Output\log_analysis.txt'
with open(output_file_path, 'w') as outputfile:
    # Write 404 concerns
    for ip, count, start_time, concern_type in access_log_404_concerns:
        outputfile.write(f"Concern raised for IP '{ip}': {count} {concern_type} in 5 minutes starting from {start_time}.\n")

    # Write login failed concerns
    for ip, count, start_time, concern_type in login_failed_concerns:
        outputfile.write(f"Concern raised for {concern_type} from IP '{ip}': occurred {count} times since {start_time}.\n")
```

Fig 4: Python script used to parse logs from Output folder back to the Output folder (parsed_logs.py)

3. Report of Analysis:

The output from the above script is sent back to the Shared folder in the file called **log_analysis.txt** and the results are as follows:



Similar to the CRON job set up for collecting weekly logs on our Linux system, this report would be obtained on a weekly basis as well by using Windows Task Scheduler. The scheduler would run the **parsed_logs**.py script at 12:00 AM, every Friday of every week.

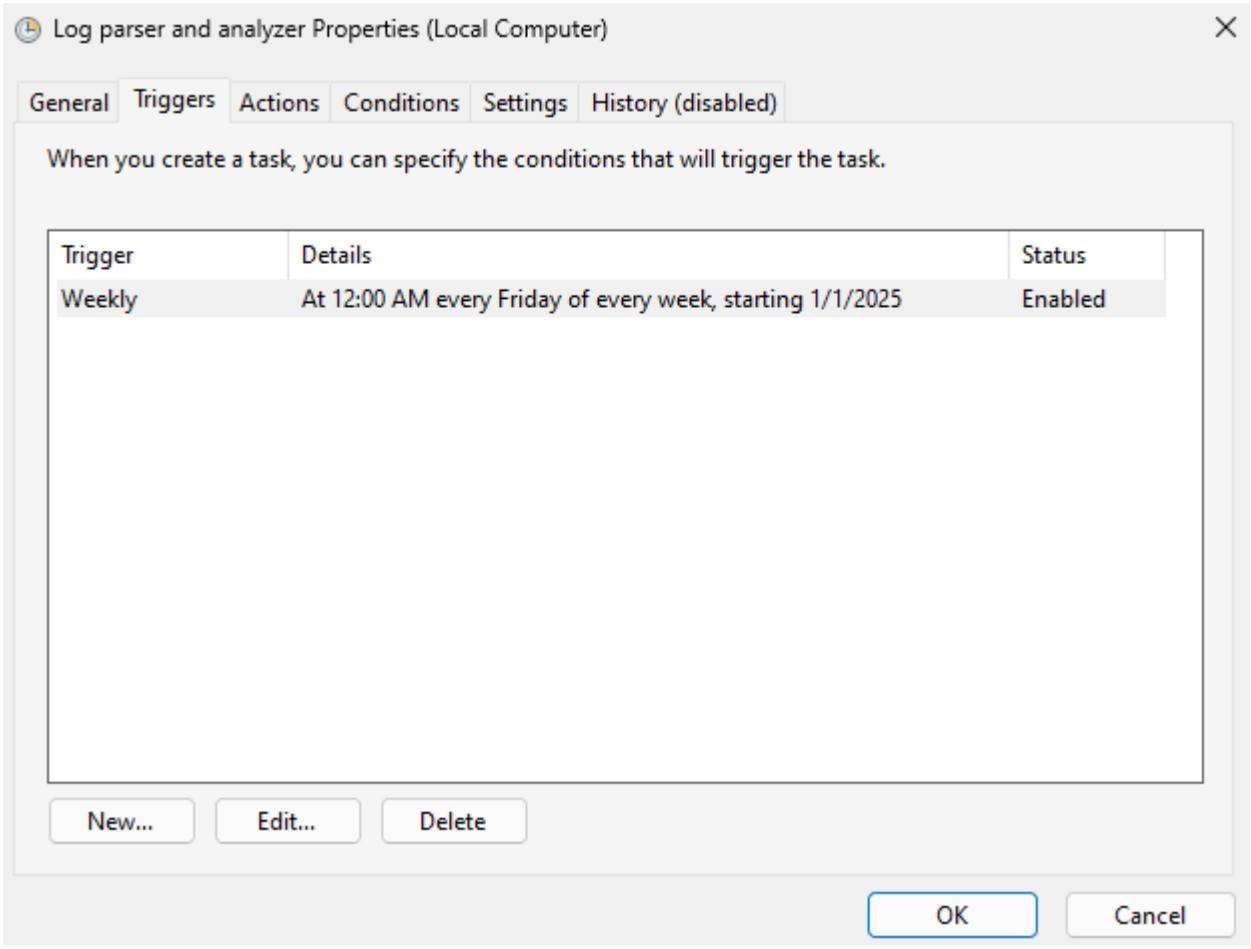


Fig 5: Windows Task scheduler set to run the mentioned script on a weekly basis (parsed_logs.py)

Potential Iterations

Although this workflow gives us our desired output, the solutions used might not be the best from a security point of view. For that, we have a couple of Iterations that could be used to improve our current Workflow considering more resources would available to implement these:

- Frequent log collection - Instead of weekly log collection, it would be more beneficial to collect logs on a daily basis. This would help monitor the network thoroughly and keep an eye out for potential attacks outside of working hours.
- Scalability & Storage Enhancement - If frequent log collection is implemented, increasing storage and capacity of logs in the shared folder would be ideal in order to prevent the scheduled scripts from running without downtime.
- Send email notification - It would be assumed that management would not be accessing the shared folder frequently or outside of working hours. In event of an emergency or a potential concern, it would be beneficial to have an email notification sent out. This can be executed by simply sending a plain-text email through python using a tool like smtplib which can be done by starting a secure SMTP connection that is encrypted [Real Python, 2025].

Conclusion

Based on our Workflow, our Solutions implemented help us identify vulnerabilities pertaining to IOCs such as Reconnaissance and Brute Force Attacks.

These can be slightly enhanced considering the potential iterations proposed which would improve detection of threats by a developed log monitoring workflow.

As cybersecurity is a rapidly evolving field, we should keep in mind there is always room for improvements and further enhancements for securing more advanced systems.

References

- Easy Cron (Retrieved January, 2025) - Cron Tab Generator
  Retrieved from https://crontab-generator.org/
- Real Python (Retrieved January, 2025) - Sending Emails With Python
  Retrieved from https://realpython.com/python-send-email/
- OWASP (Retrieved January, 2025) - Missing Error Handling
  Retreived from https://owasp.org/www-community/vulnerabilities/Missing_Error_Handling
- Cyble (Updated December, 2024) - What is IOC in Cybersecurity?
  Retrieved from https://cyble.com/knowledge-hub/what-is-ioc-in-cybersecurity/
- MITRE (Retrieved January, 2025) - Techniques > Enterprise > Brute Force
  Retrieved from https://attack.mitre.org/techniques/T1110/