Log Monitoring Workflow

Prepared by: Anastasiya Gruneva

Date: January 7th, 2025

Table of Contents

Executive summary/Introduction 3
Solution - Log Monitoring Workflow 4
Potential iterations 12
Conclusion 13
References 14

Executive summary/Introduction

Proactive monitoring of web servers is essential for ensuring both system stability and security. One of the critical HTTP status codes to monitor is the HTTP 404 error, which occurs when a requested resource is not found on the server. While 404 errors may appear insignificant, frequent occurrences can point to broken links, misconfigured URLs, or even malicious activities like bot attacks scanning for vulnerabilities.

This document outlines a log monitoring workflow designed to track 404 errors, enabling the identification of unusual traffic patterns. The solution involves four key steps:

1. **Log File Collection from Linux Machine**: A Bash script extracts lines containing 404 errors from Apache's access log and saves them to a shared directory for further analysis.
2. **Automation with Cron**: A cron job automates the log collection process, ensuring it runs regularly without manual intervention.
3. **Log Analysis with Python Script**: A Python script analyzes the collected logs, counting 404 errors and identifying the IP addresses associated with them, while applying a threshold mechanism to flag unusual activity.
4. **Task Scheduling on Windows**: The analysis is automated using Task Scheduler on Windows, ensuring that the Python script runs on a weekly basis after the logs are transferred.

By implementing this workflow, organizations can detect and address potential issues early, improving server reliability, security, and performance. This approach ensures that 404 errors are consistently monitored, enabling timely interventions when anomalies occur.

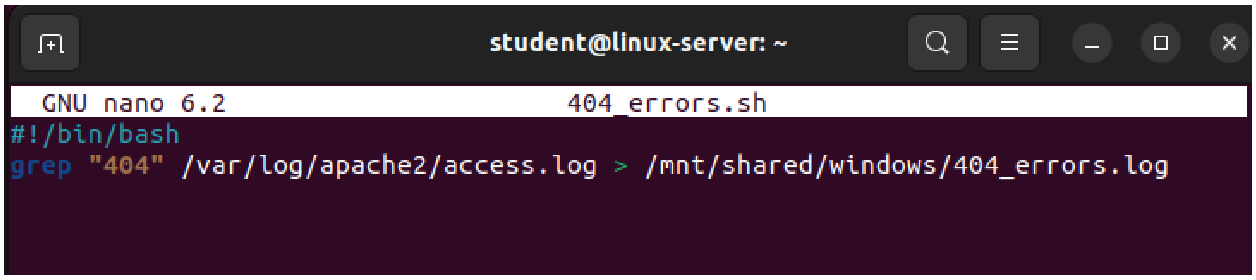
Solution - Log Monitoring Workflow

To monitor unusual network traffic, we will focus on tracking 404 errors. These errors occur when a requested resource is not found on the server, which can indicate issues such as broken links, incorrect URLs, or malicious activities like automated bots scanning for vulnerabilities. By monitoring 404 errors, we can detect patterns or anomalies and take timely actions to maintain system stability and security. (Zanini, 2024)

Workflow for Monitoring 404 Errors

Step 1: Collect Log Files from Linux Machine

- Use a Bash script to extract lines containing HTTP 404 errors from the Apache access log and save them to a shared directory accessible from the Windows system. (Musukuma, retrieved 2025-01-06)
- Bash Script:



```
#!/bin/bash
grep "404" /var/log/apache2/access.log > /mnt/shared/windows/404_errors.log
```

Figure 1. Bash script is designed to find all occurrences of HTTP 404 errors in an Apache web server's access log and move them to a shared file. (LinuxCommand.org, retrieved 2025-01-06)

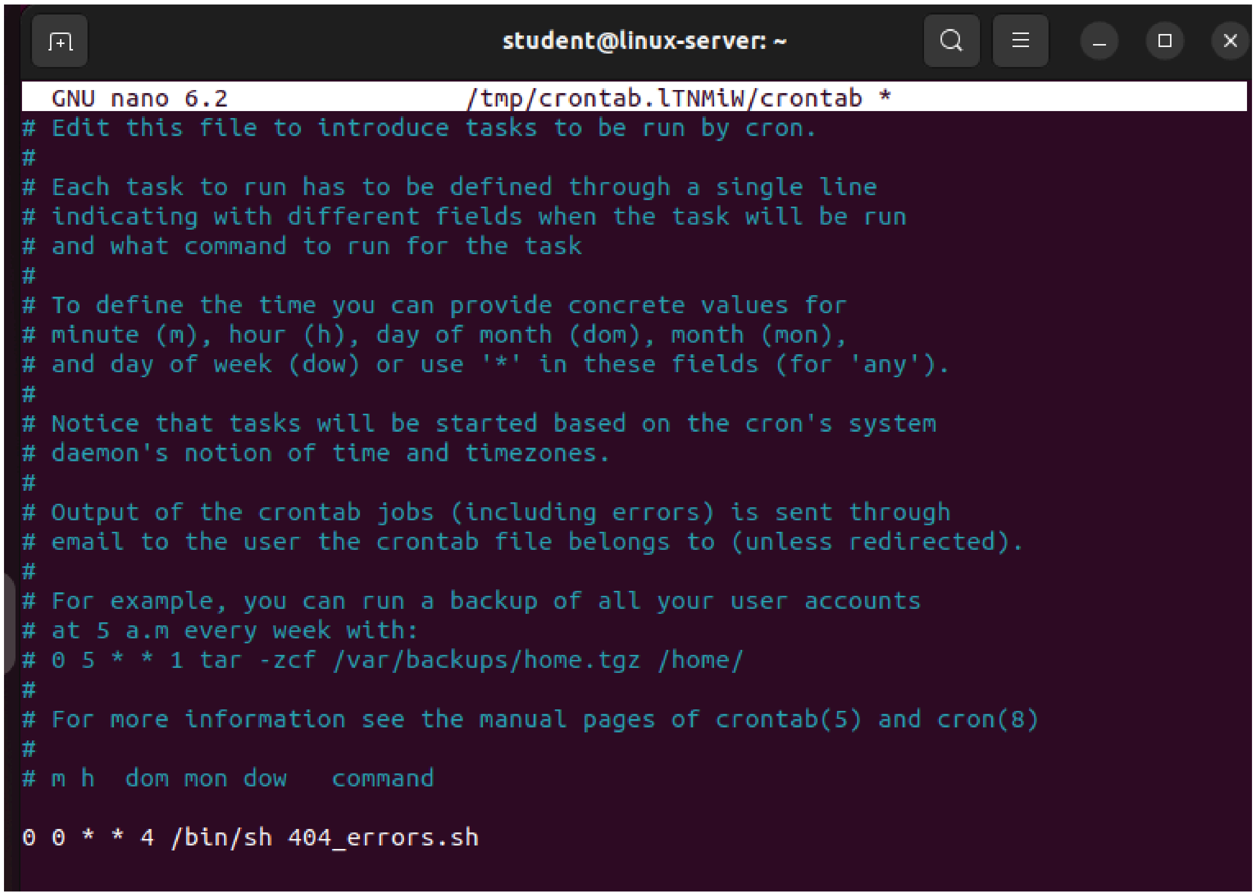
Here's what each part of the code does:

1. **#!/bin/bash**
   This is the shebang line that specifies the script should be run using the Bash shell.
2. **grep "404"**
   The grep command searches for the string "404" in the input provided. In this case, it looks for lines containing 404, which typically represents a "Not Found" error in HTTP logs.
3. **/var/log/apache2/access.log**
   This is the path to the Apache access log file.
4. **>**
   This is the output redirection operator. It directs the output of the grep command to a specified file.
5. **/mnt/shared/windows/404_errors.log** This is the target file where the script saves the results. It's located in the /mnt/shared/windows/ directory. This is a

location that is accessible from a Windows system.

Step 2: Automate Collection with Cron

- Configure a cron job to execute the Bash script weekly to ensure regular log collection.

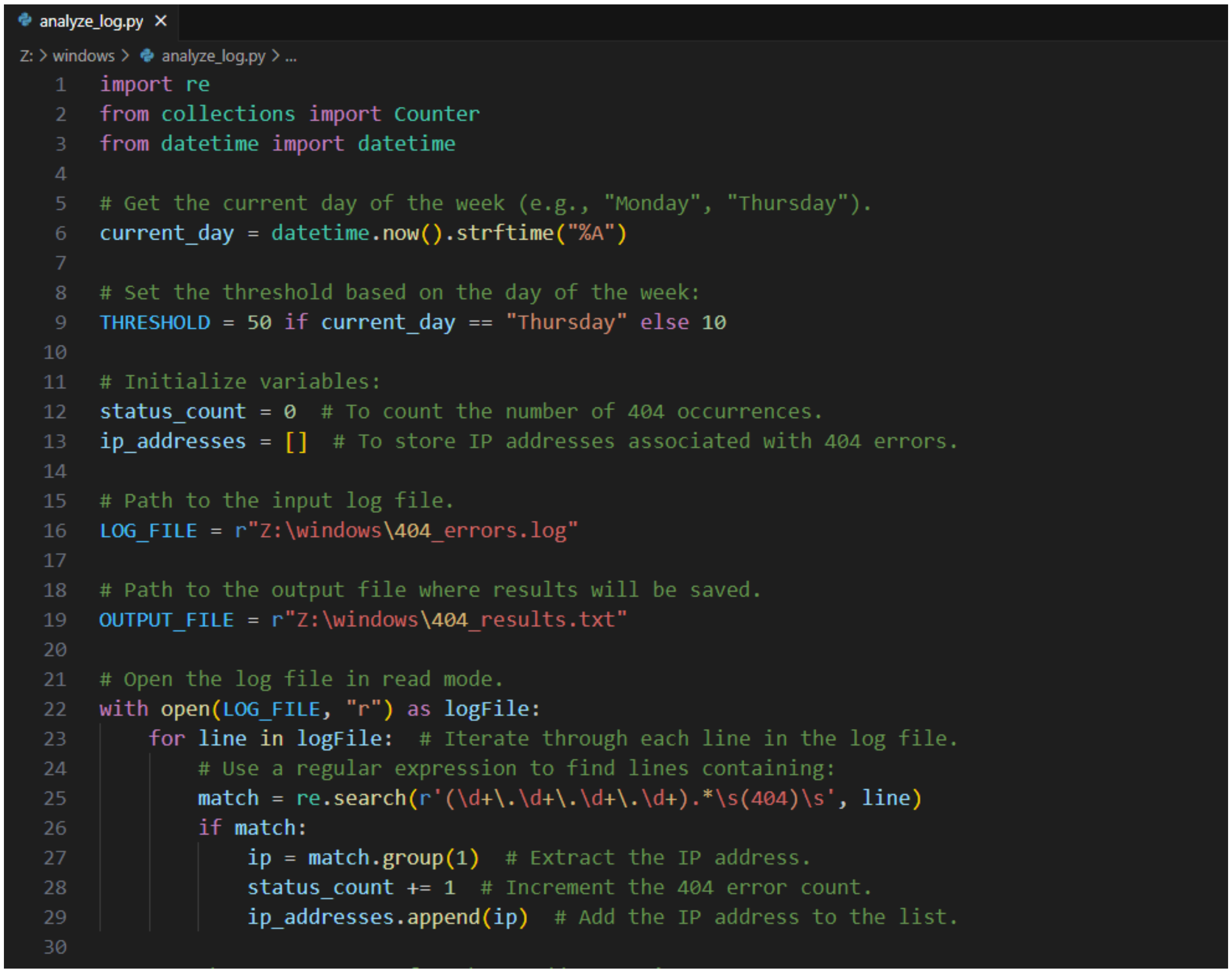


```
 GNU nano 6.2                 /tmp/crontab.lTNMiW/crontab *
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').
#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command

0 0 * * 4 /bin/sh 404_errors.sh
```

Figure 2. This Cron Job schedules the script to run every Thursday at midnight. (Ubuntu, retrieved 2025-01-06)

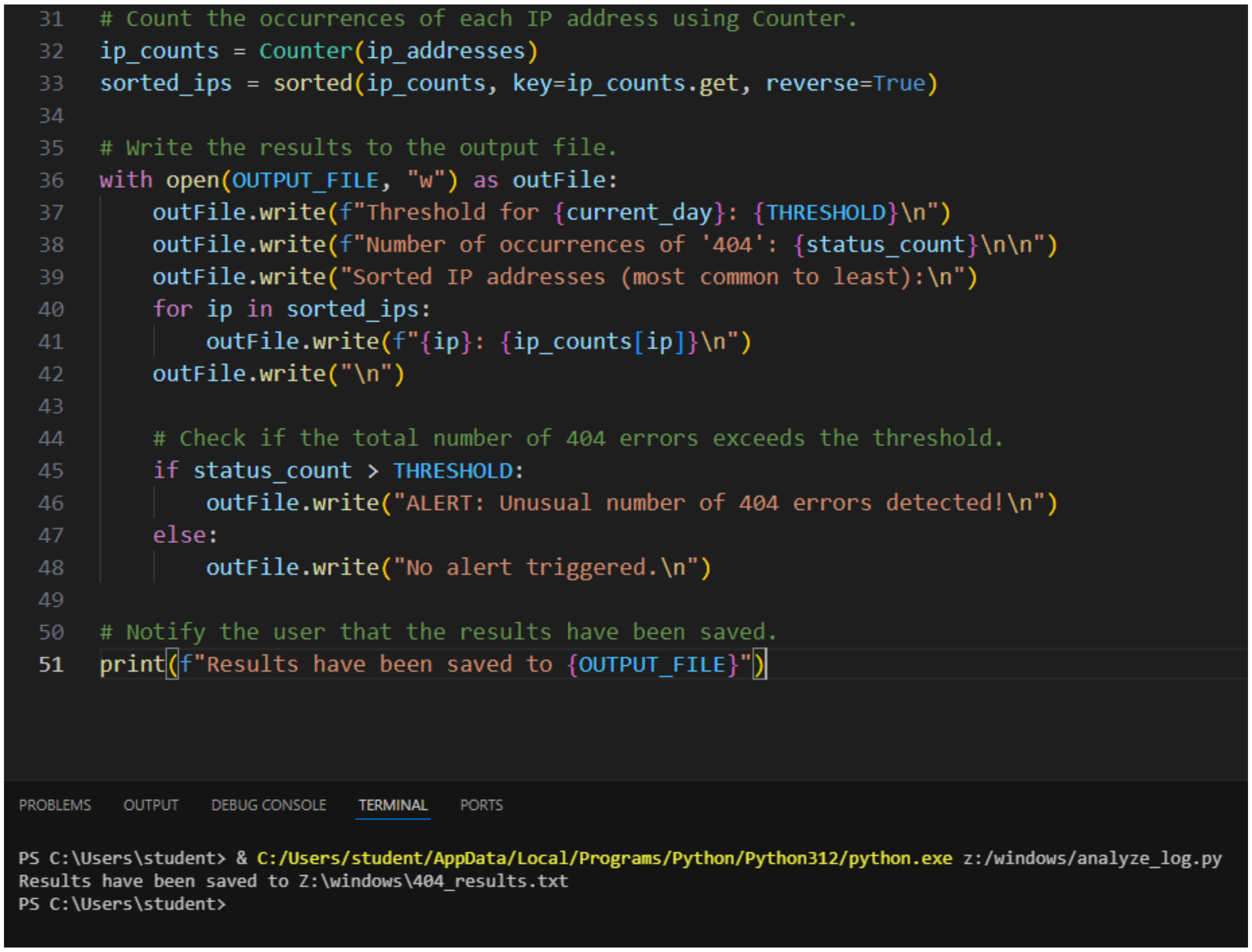
Step 3: Analyze Logs with Python Script

- On the Windows machine, use a Python script to analyze the log file.
- The script automates the process of analyzing web server logs for 404 errors.
- It provides insights into the frequency and sources (IP addresses) of these errors.
- The threshold mechanism helps flag unusual activity, such as an unexpectedly high number of 404 errors, which might indicate issues like broken links, server misconfiguration, or malicious activity.



```python
import re
from collections import Counter
from datetime import datetime

# Get the current day of the week (e.g., "Monday", "Thursday").
current_day = datetime.now().strftime("%A")

# Set the threshold based on the day of the week:
THRESHOLD = 50 if current_day == "Thursday" else 10

# Initialize variables:
status_count = 0  # To count the number of 404 occurrences.
ip_addresses = []  # To store IP addresses associated with 404 errors.

# Path to the input log file.
LOG_FILE = r"Z:\windows\404_errors.log"

# Path to the output file where results will be saved.
OUTPUT_FILE = r"Z:\windows\404_results.txt"

# Open the log file in read mode.
with open(LOG_FILE, "r") as logFile:
    for line in logFile:  # Iterate through each line in the log file.
        # Use a regular expression to find lines containing:
        match = re.search(r'(\d+\.\d+\.\d+\.\d+).*\s(404)\s', line)
        if match:
            ip = match.group(1)  # Extract the IP address.
            status_count += 1  # Increment the 404 error count.
            ip_addresses.append(ip)  # Add the IP address to the list.

```

Figure 3. The first part of Python script.



```python
# Count the occurrences of each IP address using Counter.
ip_counts = Counter(ip_addresses)
sorted_ips = sorted(ip_counts, key=ip_counts.get, reverse=True)

# Write the results to the output file.
with open(OUTPUT_FILE, "w") as outFile:
    outFile.write(f"Threshold for {current_day}: {THRESHOLD}\n")
    outFile.write(f"Number of occurrences of '404': {status_count}\n\n")
    outFile.write("Sorted IP addresses (most common to least):\n")
    for ip in sorted_ips:
        outFile.write(f"{ip}: {ip_counts[ip]}\n")
    outFile.write("\n")

    # Check if the total number of 404 errors exceeds the threshold.
    if status_count > THRESHOLD:
        outFile.write("ALERT: Unusual number of 404 errors detected!\n")
    else:
        outFile.write("No alert triggered.\n")

# Notify the user that the results have been saved.
print(f"Results have been saved to {OUTPUT_FILE}")
```

```
PS C:\Users\student> & C:/Users/student/AppData/Local/Programs/Python/Python312/python.exe z:/windows/analyze_log.py
Results have been saved to Z:\windows\404_results.txt
PS C:\Users\student>
```

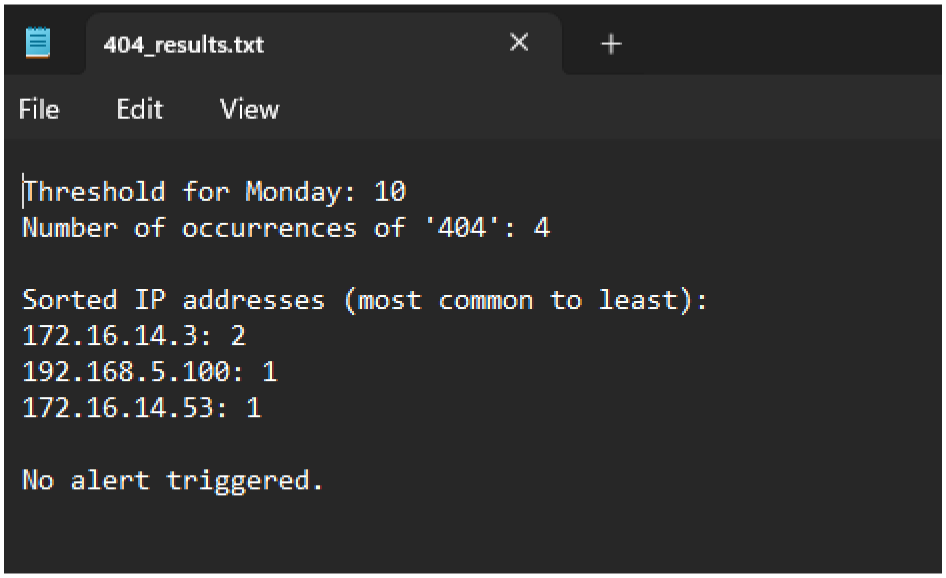
Figure 4. The second part of Python script.

What the script does (W3Schools, retrieved 2025-01-06):

1. **Determine the Current Day of the Week:**
   a. The script retrieves the current day (e.g., "Monday" or "Thursday") using the datetime module.
2. **The threshold is adjusted based on traffic expectations:**
   a. **Thursday (Threshold: 50):** High traffic due to mandatory member logins, so a higher threshold accounts for expected 404 errors from user mistakes or broken links.
   b. **Other Days (Threshold: 10):** Lower traffic, so even a small number of 404 errors could indicate unusual activity or issues.
3. **Read and Analyze the Log File:**
   a. The script opens a specified log file (Z:\windows\404_errors.log) in read mode.
   b. It processes each line in the log file to:
      - Identify lines containing a **404 error** (using a regular expression).
      - Extract the **IP address** from each matching line.
      - Count the total occurrences of 404 errors and collect the associated

IP addresses.

4. **Count and Sort IP Addresses:**
    a. The script uses the Counter class from the collections module to count how many times each IP address appears in the log.
    b. The IP addresses are sorted by frequency in descending order (most frequent first).
5. **Save the Results to an Output File:**
    a. The script writes the following information to an output file (Z:\windows\404_results.txt):
        - The threshold value for the current day.
        - The total number of 404 errors found.
        - A sorted list of IP addresses with their respective counts.
        - An alert message if the number of 404 errors exceeds the threshold, or a note that no alert was triggered.
6. **Notify the User:**
    a. Once the results are saved, the script prints a message to the console, indicating the location of the output file.



```
Threshold for Monday: 10
Number of occurrences of '404': 4

Sorted IP addresses (most common to least):
172.16.14.3: 2
192.168.5.100: 1
172.16.14.53: 1

No alert triggered.
```

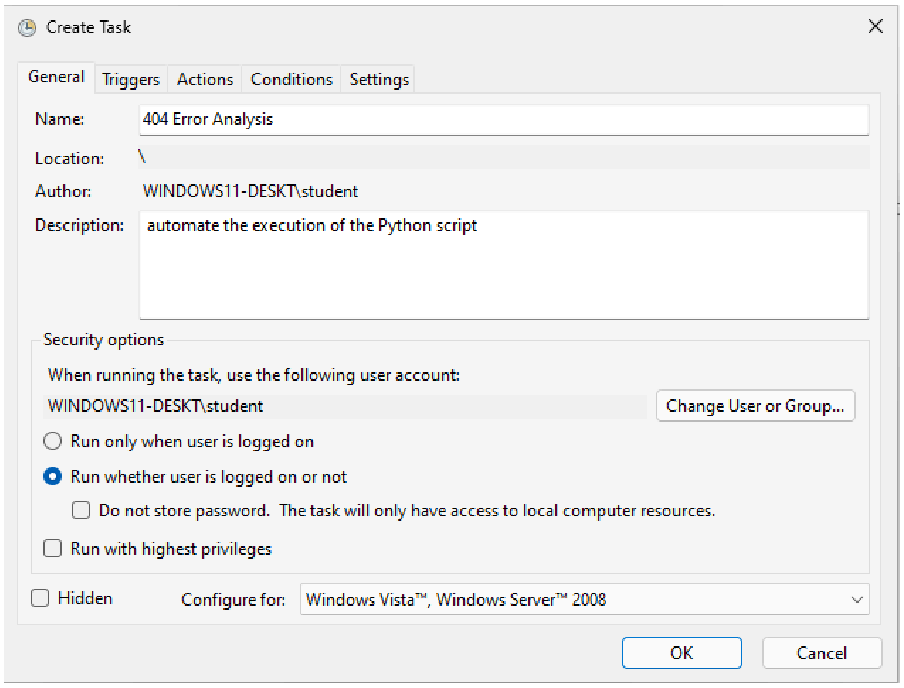
Figure 5. Output file.

Step 4: Schedule the Analysis Task

- Use the Task Scheduler on Windows to automate the execution of the Python script. Configure the task to run weekly after the log file has been transferred. (Microsoft, 2024)

To Schedule the Task on Windows:

- Open Task Scheduler.
- Create a new task with a descriptive name (e.g., "404 Error Analysis").



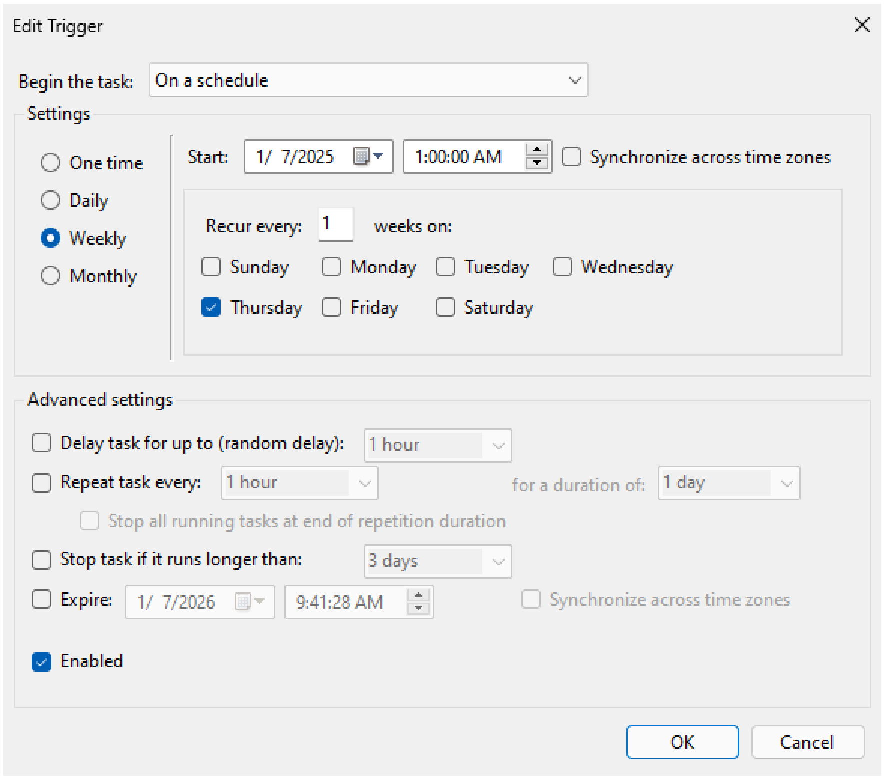
Name: 404 Error Analysis
Location: \
Author: WINDOWS11-DESKT\student
Description: automate the execution of the Python script

Security options
When running the task, use the following user account:
WINDOWS11-DESKT\student
- [ ] Run only when user is logged on
- [✓] Run whether user is logged on or not
  - [ ] Do not store password. The task will only have access to local computer resources.
- [ ] Run with highest privileges

[ ] Hidden   Configure for: Windows Vista™, Windows Server™ 2008

Figure 6. The Task Scheduler Create Task window

- Set the trigger to a specific day and time (e.g., Thursday at 1 AM).


Edit Trigger

Begin the task: On a schedule

Settings

( ) One time
( ) Daily
(●) Weekly
( ) Monthly

Start: 1/ 7/2025 1:00:00 AM [ ] Synchronize across time zones

Recur every: 1 weeks on:

[ ] Sunday [ ] Monday [ ] Tuesday [ ] Wednesday
[✓] Thursday [ ] Friday [ ] Saturday

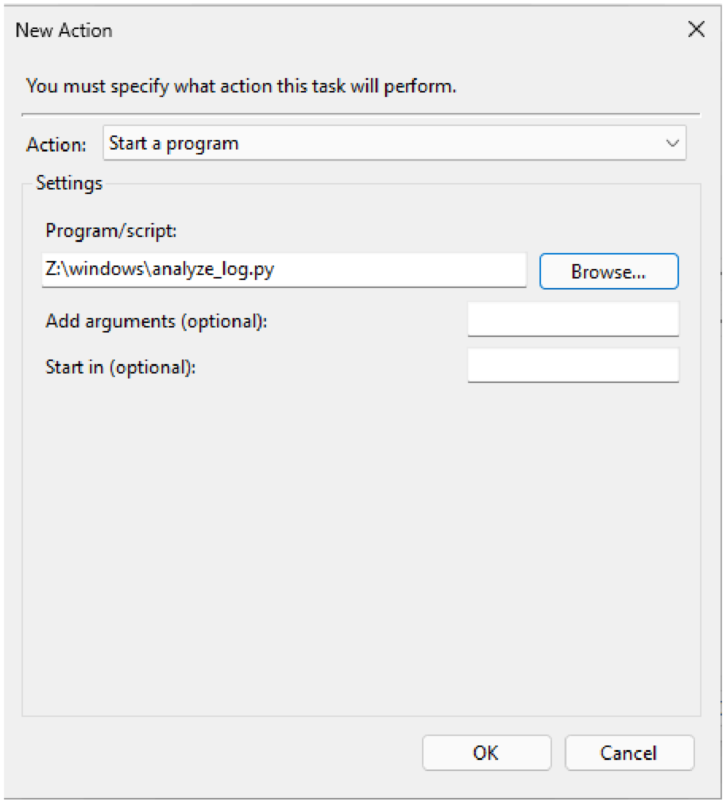
Advanced settings

[ ] Delay task for up to (random delay): 1 hour
[ ] Repeat task every: 1 hour for a duration of: 1 day
[ ] Stop all running tasks at end of repetition duration
[ ] Stop task if it runs longer than: 3 days
[ ] Expire: 1/ 7/2026 9:41:28 AM [ ] Synchronize across time zones

[✓] Enabled

Figure 7. Edit Trigger window.

- Set the action to execute the Python script (Z:\windows\analyze_logs.py).



Z:\windows\analyze_log.py

Figure 8. New Action window.

By implementing this workflow, you can efficiently monitor 404 errors and document unusual traffic patterns. This process ensures your manager receives timely updates with actionable insights, helping maintain system reliability and security.

Potential iterations

While the initial workflow is designed to monitor 404 errors, future iterations could enhance the process by monitoring additional HTTP status codes, such as 500 (server errors) or 403 (forbidden access). Integrating more complex data analysis tools, such as machine learning algorithms for anomaly detection, could help identify more subtle patterns that suggest potential attacks or performance issues. Furthermore, expanding the script's capabilities to send automatic alerts or generate real-time dashboards could significantly improve response times and allow quicker troubleshooting. These enhancements could help develop a more comprehensive and proactive monitoring system.

Conclusion

The log monitoring workflow described in this document provides a robust method for detecting and analyzing HTTP 404 errors in web server logs. By automating the collection, analysis, and reporting of these errors, the organization can ensure that abnormal traffic patterns or server misconfigurations are identified promptly. This process supports ongoing system optimization, security monitoring, and helps avoid potential issues that could impact user experience or system reliability. As web infrastructure continues to grow and evolve, adapting the workflow to address additional error codes and leveraging advanced monitoring tools will be essential for maintaining high performance and security standards.

References

Writing shell scripts. LinuxCommand.org. (retrieved 2025-01-06).
https://linuxcommand.org/lc3_writing_shell_scripts.php

Task Scheduler: Automating tasks on Windows. Microsoft. (2024 -07-15).
https://learn.microsoft.com/en-us/windows/win32/taskschd/task-scheduler-start-page

Musukuma, M. (retrieved 2025-01-06). *Best tips for monitoring and filtering your web server logs*. Papertrail.
https://www.papertrail.com/solution/tips/best-tips-for-monitoring-and-filtering-your-web-server-logs/

Zanini, A. (2024-11-07). *HTTP status codes: A comprehensive guide*.
https://semaphoreci.com/blog/http-status-codes

CronHowto. Ubuntu. (retrieved 2025-01-06).
https://help.ubuntu.com/community/CronHowto

Python regular expressions. W3Schools. (retrieved 2025-01-06).
https://www.w3schools.com/python/python_regex.asp