



LogHawk Technical Report

Prepared by: John Okenyi

Date: June 25, 2025



Executive Summary/Introduction	3
Network Devices Information	3
Information Collection Methodology	4
Recommended Topology	6
Automation with Cron (Linux)	6
References	8



Executive Summary/Introduction

Cyber threats are growing rapidly, and early detection is essential to preventing major damage. This report presents **LogHawk**, a simple and open-source tool designed to help monitor computer systems by scanning log files and catching warning signs before they turn into full-blown issues. **LogHawk** reads system logs and looks for things like too many failed logins, strange traffic patterns, critical system errors, or script activity. These could be signs of someone trying to break in or that something has gone wrong. By spotting these early, security teams can react fast and stay ahead of threats (MITRE ATT&CK, n.d.). Designed with flexibility and simplicity in mind, LogHawk is lightweight, easy to use, and customizable. It can be deployed across systems and integrated with automation tools like cron for continuous protection. This approach supports Canada's cybersecurity objectives for small and medium organizations by promoting proactive, automated, and transparent practices (Canadian Centre for Cyber Security, 2022).

Network Devices Information

LogHawk processes logs generated by different parts of a computer system. Here's a breakdown of the key log files it monitors:

Log File	What It Tracks
access.log	Tracks who accessed what (like website visits or file access)
app.log	What happened inside the app (like errors, warnings, or suspicious actions)
auth.log	Who tried to log in to the system and whether they succeeded or failed
system.log	What happened inside the operating system — like alarms, blocked connections, crashes or high CPU usage

Figure 1: Logs used for LogHawk testing

The screenshot shows a Windows File Explorer window with the address bar displaying the path: This PC > John_shared (\\VBoxSvr) (Z:) > loghawk. The window contains a table of files with columns for Name, Date modified, Type, and Size.

Name	Date modified	Type	Size
access	6/25/2025 4:26 PM	Text Document	2 KB
app	6/25/2025 4:26 PM	Text Document	2 KB
auth	6/25/2025 4:26 PM	Text Document	3 KB
system	6/25/2025 4:27 PM	Text Document	2 KB



Information Collection Methodology

LogHawk uses regular expressions (re module in Python) to find suspicious patterns in logs. These patterns are based on common tactics seen in real-world cyberattacks (MITRE ATT&CK, n.d.). Here's what the tool is designed to catch:

Threat Type	What to Look For	Log File(s)
Too Many Failed Logins	"401" errors, "Failed password" messages	access.log, auth.log
Unusual Traffic Spikes	Same IP making too many requests	access.log
Critical System Errors	Logs with keywords like ERROR, CRITICAL	app.log, system.log
Suspicious Script Activity	Filenames like malware.py, *.sh, cron, etc.	system.log
Suspicious IP Behavior	Requests from blacklisted IPs (e.g., 203.x.x.x, 45.x.x.x)	All logs
Unauthorized Access Attempts	"Unauthorized", "blocked", or "restricted endpoint" keywords	app.log, system.log

The table outlines key security threats LogHawk should scan for across various log files, such as failed login attempts, traffic spikes, critical errors, unauthorized access, suspicious IPs, and script activity. Each threat is matched to specific log files and patterns to watch for, helping LogHawk identify potential attacks or unusual behavior.

Figure 2 Code Execution for Automated Security Monitoring

```
loghawk.py
1  import re
2
3  # === Load each log file ===
4  with open("access.log") as f:
5      access_log = f.readlines()
6  with open("app.log") as f:
7      app_log = f.readlines()
8  with open("auth.log") as f:
9      auth_log = f.readlines()
10 with open("system.log") as f:
11     system_log = f.readlines()
12
13 # === Define patterns and Labels ===
14 patterns = {
15     "access.log": {
16         "Too Many 401 Errors (Access Denied)": r'401',
17         "Repeated Access to /admin": r'/admin',
18         "Blocked wp-login Attempts": r'/wp-login\.php',
19     },
20     "app.log": {
21         "CRITICAL Errors in App": r'CRITICAL:',
22         "ERROR Messages in App": r'ERROR:',
23         "WARNINGS in App": r'WARNING:',
24         "Repeated Access to /admin": r'/admin',
25     },
26     "auth.log": {
27         "Failed SSH Logins": r'Failed password',
28         "Invalid User Logins": r'invalid user',
29     },
30 }
```



```
loghawk.py
29 },
30 "system.log": {
31     "WARNINGS in App": r'WARNING:',
32     "Suspicious Script Execution (CRON)": r'(malware\.py|malicious\.py|security_check\.py)',
33     "Firewall Blocking": r'Firewall: blocked connection',
34     "Unusual Traffic or CPU": r'(Suspicious activity|High CPU usage|Unusual number of connections)',
35 }
36 }
37
38 # === Function to extract IP from a Log Line ===
39 def extract_ip(line):
40     match = re.search(r'(\d{1,3}(\.?\d{1,3}){3})', line)
41     return match.group(1) if match else "No IP"
42
43 # === Function to scan log content ===
44 def scan_log(file_name, log_lines, rule_set):
45     print(f"\n==== Scanning {file_name} =====")
46     for label, pattern in rule_set.items():
47         regex = re.compile(pattern)
48         for line in log_lines:
49             if regex.search(line):
50                 ip = extract_ip(line)
51                 print(f"[{file_name}] {label} ⚠ Line: {line.strip()} | IP: {ip}")
52
53 # === Run the scans ===
54 scan_log("access.log", access_log, patterns["access.log"])
55 scan_log("app.log", app_log, patterns["app.log"])
56 scan_log("auth.log", auth_log, patterns["auth.log"])
57 scan_log("system.log", system_log, patterns["system.log"])
```

```
loghawk.py
38 "system.log": {

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python + - ... ^ x
[auth.log] Invalid User Logins ⚠Line: Feb 17 10:50:33 server1 sshd[2143]: Failed password for invalid user xyz from 45.67.89.102 port 30011
0 ssh2 | IP: 45.67.89.102
[auth.log] Invalid User Logins ⚠Line: Feb 17 11:15:30 server1 sshd[2143]: Failed password for invalid user admin from 198.51.100.33 port 55
0321 ssh2 | IP: 198.51.100.33

==== Scanning system.log ====
[system.log] WARNINGS in App ⚠Line: Feb 17 10:14:45 server1 kernel: [12348.123456] WARNING: Suspicious activity detected on eth0 (IP: 203..
0.113.99) | IP: 203.0.113.99
[system.log] WARNINGS in App ⚠Line: Feb 17 10:27:40 server1 kernel: [12351.123456] WARNING: Potential DDoS detected on interface eth1 | IPP
: No IP
[system.log] Suspicious Script Execution (CRON) ⚠Line: Feb 17 10:17:30 server1 CRON[5678]: (root) CMD (/usr/bin/python3 /opt/scripts/malwa
re.py) | IP: No IP
[system.log] Suspicious Script Execution (CRON) ⚠Line: Feb 17 10:25:12 server1 CRON[6789]: (user3) CMD (/usr/bin/python3 /opt/scripts/mali
cious.py) | IP: No IP
[system.log] Suspicious Script Execution (CRON) ⚠Line: Feb 17 10:59:05 server1 CRON[8901]: (user1) CMD (/usr/bin/python3 /opt/scripts/secu
rity_check.py) | IP: No IP
[system.log] Firewall Blocking ⚠Line: Feb 17 10:10:00 server1 kernel: [12346.789012] Firewall: blocked connection from 10.0.0.99 to 192.166
8.1.1 on port 22 | IP: 10.0.0.99
[system.log] Firewall Blocking ⚠Line: Feb 17 10:23:35 server1 kernel: [12350.789123] Firewall: blocked connection from 10.0.0.105 to 192.11
68.1.1 on port 23 | IP: 10.0.0.105
[system.log] Firewall Blocking ⚠Line: Feb 17 10:50:35 server1 kernel: [12355.789012] Firewall: blocked connection from 192.168.1.12 to 1922
.168.1.1 on port 21 | IP: 192.168.1.12
[system.log] Unusual Traffic or CPU ⚠Line: Feb 17 10:14:45 server1 kernel: [12348.123456] WARNING: Suspicious activity detected on eth0 (II
P: 203.0.113.99) | IP: 203.0.113.99
[system.log] Unusual Traffic or CPU ⚠Line: Feb 17 10:35:50 server1 kernel: [12352.567890] Unusual number of connections from 203.0.113.1000
| IP: 203.0.113.100
[system.log] Unusual Traffic or CPU ⚠Line: Feb 17 10:55:12 server1 kernel: [12356.123456] SYSTEM ALERT: High CPU usage detected | IP: No II
P
PS C:\Users\Admin\Documents\Lighthouse\John_shared\loghawk>
```

The above screenshots shows the Python script developed which acts as a **log file analyzer** that automatically scans system logs for security threats and errors. It works by first opening and reading four types of log files (access.log, app.log, auth.log, and system.log). The script then checks each file against a predefined list of suspicious patterns - like failed login attempts (Failed password), unauthorized access to admin areas (/admin), malware scripts (malware.py), firewall blocks, and system warnings. When it finds matches, it extracts relevant details (particularly IP addresses) and prints clear warnings. The script essentially automates what would otherwise be manual log inspection, helping system administrators quickly identify potential security breaches or technical issues. The code uses basic programming concepts like file reading, pattern matching regular expressions, and functions to organize the scanning process efficiently



Simple Explanation of the Code in *figure 2 Code Execution for Automated Security Monitoring*

Part	Code explanation
import re	This allows us to search for patterns using words, numbers, or symbols.
with open(...)	We're opening the log files so the script can read them.
patterns = {...}	We create a dictionary of things we want to catch (like errors, failed logins, and weird scripts).
scan_log(...)	This goes through each log and looks for the patterns.
print(...)	Shows what was found so you can see it right in your terminal.

Recommended Topology

Tool Name: LogHawk

Programming Language: Python 3

Target Logs: Web server, authentication, system, and app logs

Execution: Command-line script

Output: Human-readable console output with timestamps

Automation: Compatible with cron

Deployment: Standalone servers, security operations centers (SOCs), or remote environments

Automation with Cron (Linux)

To make LogHawk run every 10 minutes automatically, the following cron job setup was used:

```
*/10 * * * * /usr/bin/python3 /media/sf_John_shared/loghawk/loghawk.py >>
```

```
/media/sf_John_shared/loghawk/loghawk.log 2>&1
```

Code	Explanation
*/10 * * * *	Runs the script every 10 minutes
/usr/bin/python3	Path to Python executable on the system
/media/sf_John_shared/loghawk/loghawk.py	Full path to the LogHawk Python script
/media/sf_John_shared/loghawk/loghawk.log 2>&1	Saves both normal and error output to loghawk.log file



Figure 3 Cron job entry in crontab -e editor

```
(student@kali)-[/media/sf_John_shared/loghawk]
$ crontab -e

no crontab for student - using an empty one
Select an editor. To change later, run select-editor again.
 1. /bin/nano          ← easiest
 2. /usr/bin/vim.basic
 3. /usr/bin/vim.tiny

Choose 1-3 [1]: 1
No modification made

(student@kali)-[/media/sf_John_shared/loghawk]
$ crontab -l









# 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
*/10 * * * * /usr/bin/python3 /media/sf_John_shared/loghawk/loghawk.py >> /media/sf_John_shared/loghawk/loghawk.log 2>&1
```

Automating LogHawk with cron to check logs continuously

These screenshots demonstrate how we've automated LogHawk, our security monitoring tool, to run continuously like a digital watchdog. In the first image, we set up the scheduling system (called *crontab*), choosing a simple text editor to configure it. The second image shows the automated schedule we created: LogHawk will run every hour (at the 10-minute mark) to scan system logs for threats—such as hacking attempts or system errors—and save its findings to a log file. This "set it and forget it" approach ensures 24/7 protection without manual oversight, aligning with our goal of proactive cybersecurity. The automation acts like a regular security patrol, quietly working in the background to detect and report risks before they escalate.



File Structure (Project Folders)

 loghawk.py	26/06/2025 23:34	Python Source File	3 KB
 .loghawk.py.swp	27/06/2025 13:41	SWP File	1 KB
 access.log	25/06/2025 23:39	Text Document	2 KB
 app.log	25/06/2025 16:26	Text Document	2 KB
 auth.log	25/06/2025 16:26	Text Document	3 KB
 loghawk.log	28/06/2025 00:00	Text Document	14 KB
 LogHawk_README.txt	27/06/2025 17:25	Text Document	2 KB
 system.log	25/06/2025 16:27	Text Document	2 KB

Attached is the link to the project files:

<https://drive.google.com/drive/folders/1qIFQgqjROHrORUbMCFxGVsold05S7ehn?usp=sharing>

Project Folder Summary (LogHawk)

Filename	Type	Purpose
loghawk.py	Python Source File	The main LogHawk script that scans the logs for threats.
.loghawk.py.swp	Swap File (Hidden)	Temporary file from the text editor — can be deleted.
access.log	Text Document	Web server logs (monitored for 401 errors, /admin abuse, etc.).
app.log	Text Document	App logs (monitored for CRITICAL/ERROR/WARNING messages).
auth.log	Text Document	SSH login attempts (checked for failed logins, invalid users, etc.).
loghawk.log	Text Document (Output)	Auto-generated report by loghawk.py containing threats found.
LogHawk_README.txt	Text Document	Project's instructions — how to use and run LogHawk.
system.log	Text Document	System events (checked for cron scripts, CPU spikes, etc.).

References

- Canadian Centre for Cyber Security. (2022). *Baseline cybersecurity controls for small and medium organizations*. <https://www.cyber.gc.ca>
- MITRE ATT&CK Framework. <https://attack.mitre.org/>
- Python D9ocs. (2024). *re — Regular expression operations*. <https://docs.python.org/3/library/re.html>
- Stack Overflow Community Contributions
- Ubuntu Manual Pages – cron, rsyslog, auth.log

