LOG ANALYSER

Author: Yvette K

CONTENTS

Introduction

The purpose of this report is to understand log data, using python to carefully investigate log files, especially the ones in /var/log/auth.log, and to automate extracting relevant information to examine and determine any unusual activities. There is a vast amount of information in log files that clarifies how the system works and its security.

By parsing and analysing data from critical log files, raw log data can be transformed into actionable intelligence, providing a clear and comprehensive understanding of system behaviours, security threats and operational anomalies.

Methodologies

To generate sufficient data to conduct a meaningful study, we utilised ubuntu to connect to our kali machine remotely through ssh to test a series of commands to be captured on kali's auth.log.

A python script was then created with geany to automate extraction of data from /var/log/auth.log and language was set (*#!/user/bin/python3*).

with statement and *open()* function were used to open the file for us to work on the data.

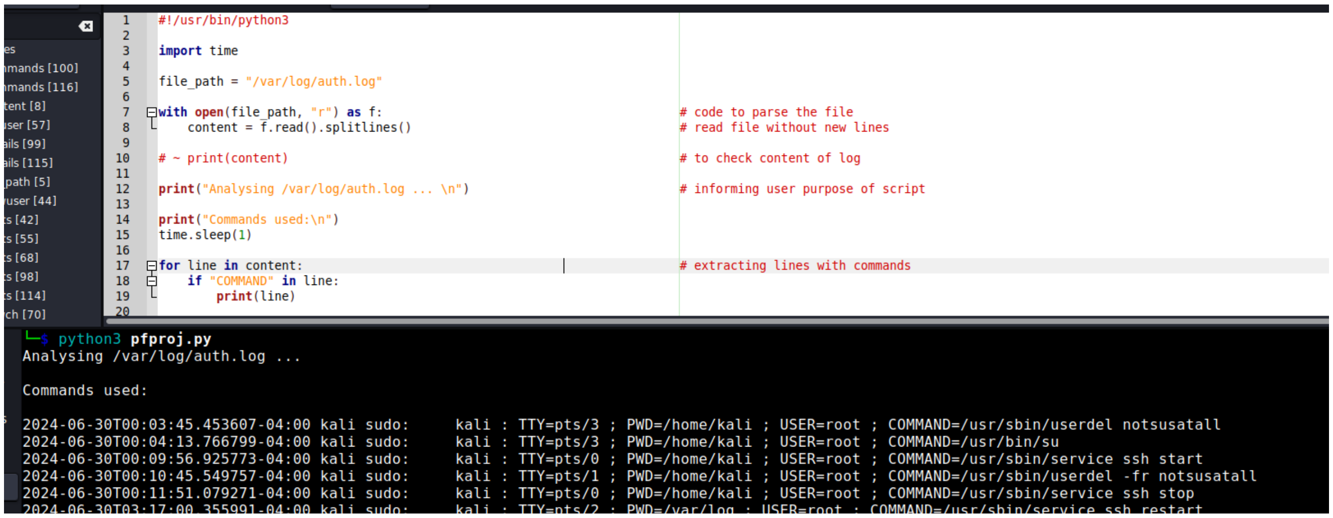
For Loops were used to print lines with keywords like "Command", "adduser" to extract information that we want to analyse.

Lines were split into sections so that only necessary information was presented to the user.

time.sleep() commands were used to suspend execution of the calling thread for a more comfortable speed for user to read and absorb the material.

Discussion

with statement and *open()* function were used to open the file for us to work on the data in /var/log/auth.log. *read* and *splitlines* commands were used to process the data such that it would be easy to obtain exact information required.



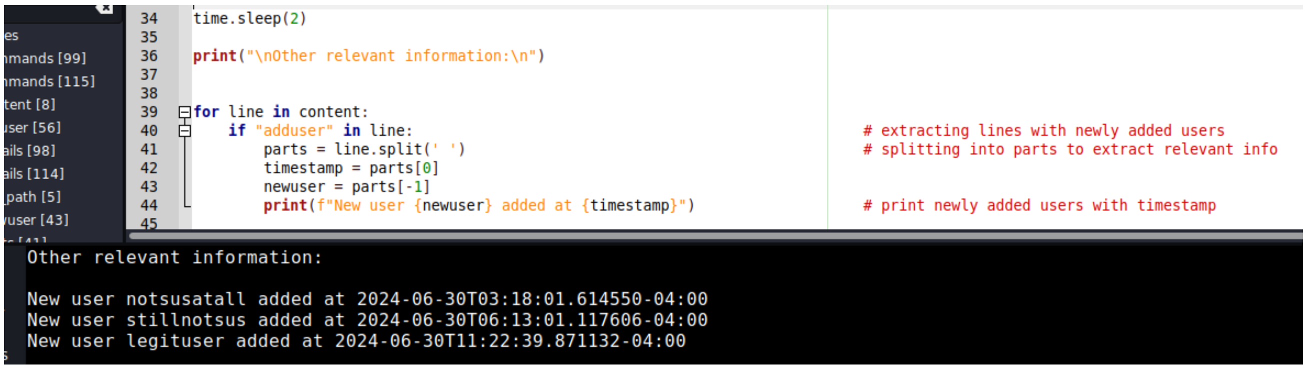
```python
#!/usr/bin/python3

import time

file_path = "/var/log/auth.log"

with open(file_path, "r") as f:                                      # code to parse the file
    content = f.read().splitlines()                                  # read file without new lines

# ~ print(content)                                                   # to check content of log

print("Analysing /var/log/auth.log ... \n")                          # informing user purpose of script

print("Commands used:\n")
time.sleep(1)

for line in content:                                                 # extracting lines with commands
    if "COMMAND" in line:
        print(line)
```

```
└─$ python3 pfproj.py
Analysing /var/log/auth.log ...

Commands used:

2024-06-30T00:03:45.453607-04:00 kali sudo:     kali : TTY=pts/3 ; PWD=/home/kali ; USER=root ; COMMAND=/usr/sbin/userdel notsusatall
2024-06-30T00:04:13.766799-04:00 kali sudo:     kali : TTY=pts/3 ; PWD=/home/kali ; USER=root ; COMMAND=/usr/bin/su
2024-06-30T00:09:56.925773-04:00 kali sudo:     kali : TTY=pts/0 ; PWD=/home/kali ; USER=root ; COMMAND=/usr/sbin/service ssh start
2024-06-30T00:10:45.549757-04:00 kali sudo:     kali : TTY=pts/1 ; PWD=/home/kali ; USER=root ; COMMAND=/usr/sbin/userdel -fr notsusatall
2024-06-30T00:11:51.079271-04:00 kali sudo:     kali : TTY=pts/0 ; PWD=/home/kali ; USER=root ; COMMAND=/usr/sbin/service ssh stop
2024-06-30T03:17:00.355991-04:00 kali sudo:     kali : TTY=pts/2 ; PWD=/var/log ; USER=root ; COMMAND=/usr/sbin/service ssh restart
```

The first set of information we wanted to convey to the user was the command usage captured in the log file. We extracted the lines containing "COMMAND" using for loop and printed the results.

To increase user readability, time module was imported to incorporate *time.sleep()* commands at appropriate junctures for information to be printed in meaningful intervals to allow users time to read and process.

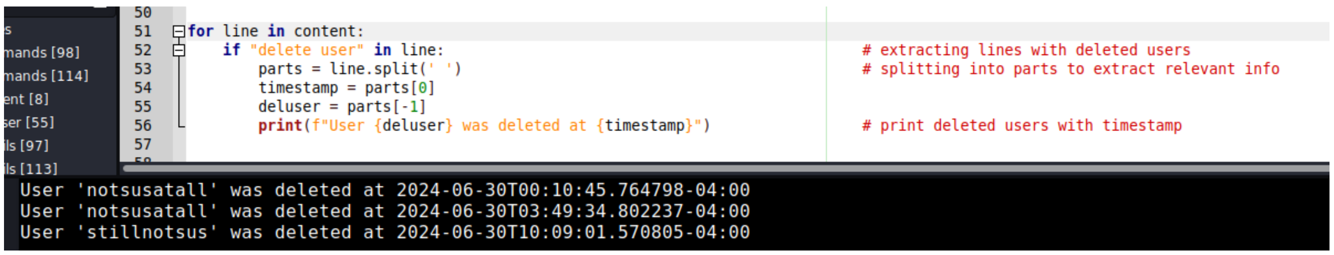
Additional headers also explained the purpose of the script and information relayed to users.



```python
time.sleep(2)

print("\nOther relevant information:\n")


for line in content:
    if "adduser" in line:                                            # extracting lines with newly added users
        parts = line.split(' ')                                      # splitting into parts to extract relevant info
        timestamp = parts[0]
        newuser = parts[-1]
        print(f"New user {newuser} added at {timestamp}")            # print newly added users with timestamp
```

```
Other relevant information:

New user notsusatall added at 2024-06-30T03:18:01.614550-04:00
New user stillnotsus added at 2024-06-30T06:13:01.117606-04:00
New user legituser added at 2024-06-30T11:22:39.871132-04:00
```

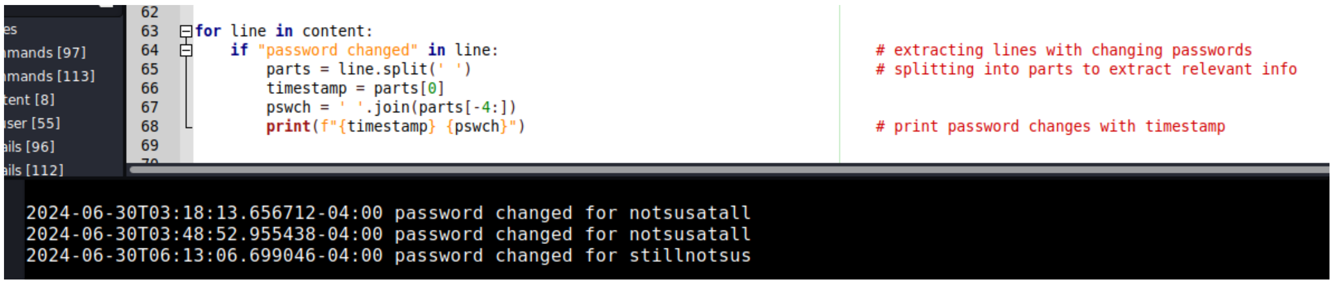
To monitor user authentication changes, we looked at a few different information. We first looked at newly added users by filtering "adduser" to obtain username created and its timestamp.



```
for line in content:
    if "delete user" in line:                                        # extracting lines with deleted users
        parts = line.split(' ')                                      # splitting into parts to extract relevant info
        timestamp = parts[0]
        deluser = parts[-1]
        print(f"User {deluser} was deleted at {timestamp}")          # print deleted users with timestamp
```

```
User 'notsusatall' was deleted at 2024-06-30T00:10:45.764798-04:00
User 'notsusatall' was deleted at 2024-06-30T03:49:34.802237-04:00
User 'stillnotsus' was deleted at 2024-06-30T10:09:01.570805-04:00
```

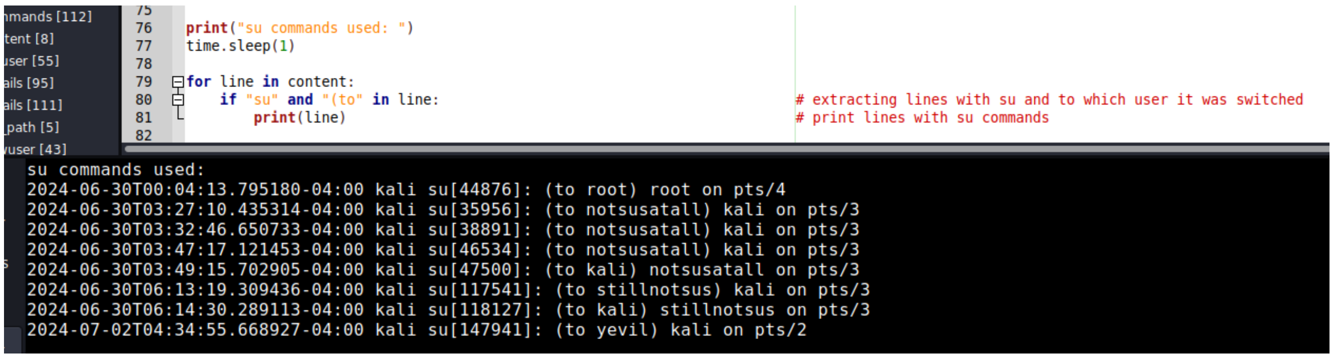
We also looked at which users were deleted using for loop and could judge from the timestamps how much time lapsed between creation and deletion of the user.



```
for line in content:
    if "password changed" in line:                                   # extracting lines with changing passwords
        parts = line.split(' ')                                      # splitting into parts to extract relevant info
        timestamp = parts[0]
        pswch = ' '.join(parts[-4:])
        print(f"{timestamp} {pswch}")                                # print password changes with timestamp
```

```
2024-06-30T03:18:13.656712-04:00 password changed for notsusatall
2024-06-30T03:48:52.955438-04:00 password changed for notsusatall
2024-06-30T06:13:06.699046-04:00 password changed for stillnotsus
```

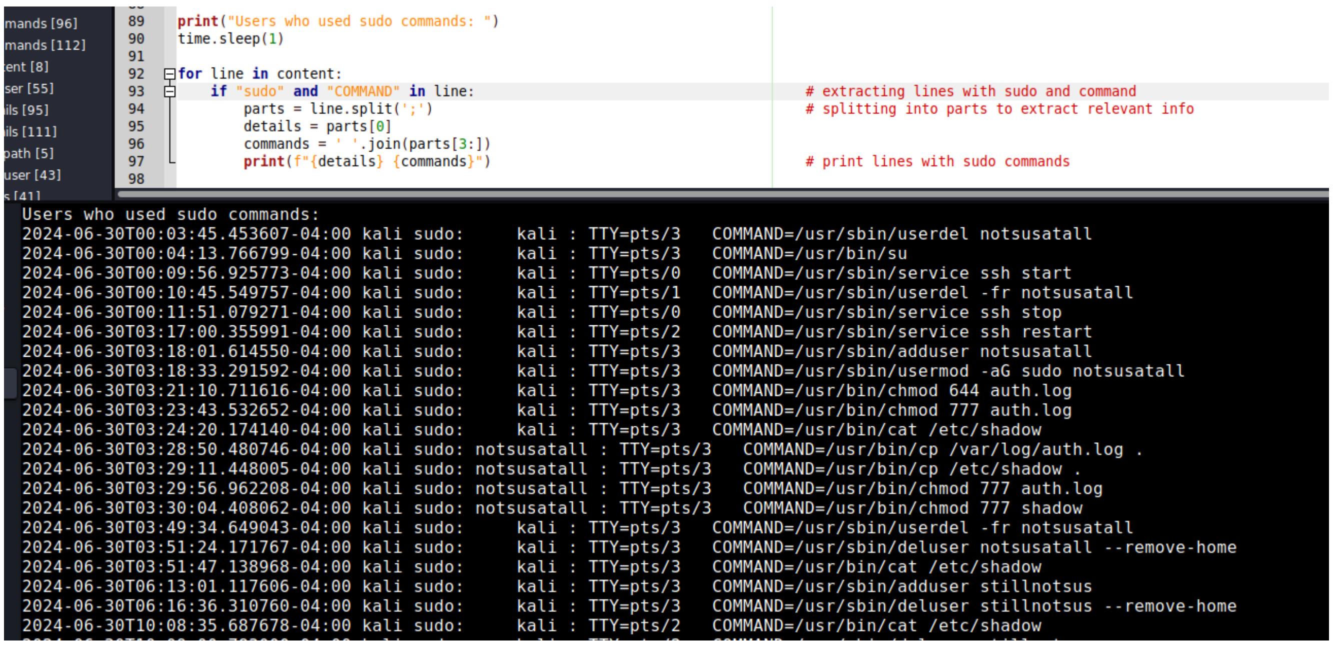
Filtering for “password changed”, we extracted information on which user’s passwords were changed and at what time.



```
print("su commands used: ")
time.sleep(1)

for line in content:
    if "su" and "(to" in line:                                       # extracting lines with su and to which user it was switched
        print(line)                                                  # print lines with su commands
```

```
su commands used:
2024-06-30T00:04:13.795180-04:00 kali su[44876]: (to root) root on pts/4
2024-06-30T03:27:10.435314-04:00 kali su[35956]: (to notsusatall) kali on pts/3
2024-06-30T03:32:46.650733-04:00 kali su[38891]: (to notsusatall) kali on pts/3
2024-06-30T03:47:17.121453-04:00 kali su[46534]: (to notsusatall) kali on pts/3
2024-06-30T03:49:15.702905-04:00 kali su[47500]: (to kali) notsusatall on pts/3
2024-06-30T06:13:19.309436-04:00 kali su[117541]: (to stillnotsus) kali on pts/3
2024-06-30T06:14:30.289113-04:00 kali su[118127]: (to kali) stillnotsus on pts/3
2024-07-02T04:34:55.668927-04:00 kali su[147941]: (to yevil) kali on pts/2
```

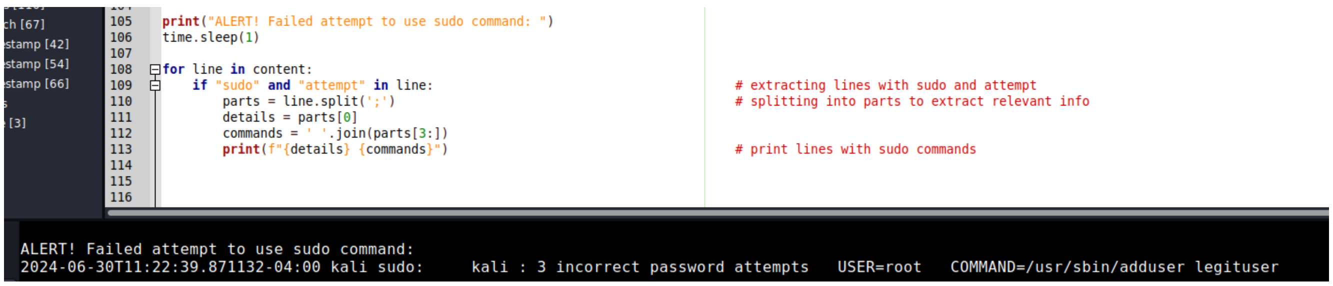
Details for when users used the su command were filtered and showed what users were switched to.



```python
print("Users who used sudo commands: ")
time.sleep(1)

for line in content:
    if "sudo" and "COMMAND" in line:                    # extracting lines with sudo and command
        parts = line.split(';')                         # splitting into parts to extract relevant info
        details = parts[0]
        commands = ' '.join(parts[3:])
        print(f"{details} {commands}")                  # print lines with sudo commands
```

```
Users who used sudo commands:
2024-06-30T00:03:45.453607-04:00 kali sudo:     kali : TTY=pts/3   COMMAND=/usr/sbin/userdel notsusatall
2024-06-30T00:04:13.766799-04:00 kali sudo:     kali : TTY=pts/3   COMMAND=/usr/bin/su
2024-06-30T00:09:56.925773-04:00 kali sudo:     kali : TTY=pts/0   COMMAND=/usr/sbin/service ssh start
2024-06-30T00:10:45.549757-04:00 kali sudo:     kali : TTY=pts/1   COMMAND=/usr/sbin/userdel -fr notsusatall
2024-06-30T00:11:51.079271-04:00 kali sudo:     kali : TTY=pts/0   COMMAND=/usr/sbin/service ssh stop
2024-06-30T03:17:00.355991-04:00 kali sudo:     kali : TTY=pts/2   COMMAND=/usr/sbin/service ssh restart
2024-06-30T03:18:01.614550-04:00 kali sudo:     kali : TTY=pts/3   COMMAND=/usr/sbin/adduser notsusatall
2024-06-30T03:18:33.291592-04:00 kali sudo:     kali : TTY=pts/3   COMMAND=/usr/sbin/usermod -aG sudo notsusatall
2024-06-30T03:21:10.711616-04:00 kali sudo:     kali : TTY=pts/3   COMMAND=/usr/bin/chmod 644 auth.log
2024-06-30T03:23:43.532652-04:00 kali sudo:     kali : TTY=pts/3   COMMAND=/usr/bin/chmod 777 auth.log
2024-06-30T03:24:20.174140-04:00 kali sudo:     kali : TTY=pts/3   COMMAND=/usr/bin/cat /etc/shadow
2024-06-30T03:28:50.480746-04:00 kali sudo: notsusatall : TTY=pts/3   COMMAND=/usr/bin/cp /var/log/auth.log .
2024-06-30T03:29:11.448005-04:00 kali sudo: notsusatall : TTY=pts/3   COMMAND=/usr/bin/cp /etc/shadow .
2024-06-30T03:29:56.962208-04:00 kali sudo: notsusatall : TTY=pts/3   COMMAND=/usr/bin/chmod 777 auth.log
2024-06-30T03:30:04.408062-04:00 kali sudo: notsusatall : TTY=pts/3   COMMAND=/usr/bin/chmod 777 shadow
2024-06-30T03:49:34.649043-04:00 kali sudo:     kali : TTY=pts/3   COMMAND=/usr/sbin/userdel -fr notsusatall
2024-06-30T03:51:24.171767-04:00 kali sudo:     kali : TTY=pts/3   COMMAND=/usr/sbin/deluser notsusatall --remove-home
2024-06-30T03:51:47.138968-04:00 kali sudo:     kali : TTY=pts/3   COMMAND=/usr/bin/cat /etc/shadow
2024-06-30T06:13:01.117606-04:00 kali sudo:     kali : TTY=pts/3   COMMAND=/usr/sbin/adduser stillnotsus
2024-06-30T06:16:36.310760-04:00 kali sudo:     kali : TTY=pts/3   COMMAND=/usr/sbin/deluser stillnotsus --remove-home
2024-06-30T10:08:35.687678-04:00 kali sudo:     kali : TTY=pts/2   COMMAND=/usr/bin/cat /etc/shadow
```

Filtering for sudo commands showed which used superuser privileges to run commands.



```python
print("ALERT! Failed attempt to use sudo command: ")
time.sleep(1)

for line in content:
    if "sudo" and "attempt" in line:                    # extracting lines with sudo and attempt
        parts = line.split(';')                         # splitting into parts to extract relevant info
        details = parts[0]
        commands = ' '.join(parts[3:])
        print(f"{details} {commands}")                  # print lines with sudo commands
```

```
ALERT! Failed attempt to use sudo command:
2024-06-30T11:22:39.871132-04:00 kali sudo:     kali : 3 incorrect password attempts   USER=root   COMMAND=/usr/sbin/adduser legituser
```

Alerts were set to highlight users who failed attempts to use sudo commands and monitor what they attempted to do.

For the purpose of being concise, some information was filtered out but there is more information and details that can be extracted such as which user created the new user, which groups they added the new user into, which IP address the remote user accessed our machine from, etc. We can also filter for “Failed password” to see who attempted and failed to access remotely.

Conclusion

Server log files are an invaluable source of information for cybersecurity practitioners. They contain records of various events and activities that occur on a server, such as requests, errors, warnings and more. Parsing these log files can provide insights into system performance, security issues and user behaviour. Effective cybersecurity measures should optimise monitoring of logs, especially security, access and server logs.

Security logs document security related events, such as failed login attempts, authentication failures and intrusion detection system alerts. Depending on the setup, they can also capture information related to access control changes and authentication success.

Access logs record information about attempts to access organisational resources, including individual files. This information can include list of accessed files, user authentication details, time of access, login/logout timestamps, permissions and network connections. Alerts can be set up for odd hours of access and/or unusual high activity such as in the instance of bruteforce attack.

Server logs created automatically by servers that document all the activities they perform, including client IP addresses and number and types of page requests, will be useful in cybersecurity forensics.

While log files are helpful in many ways, the number of log files generated can be very large. It might not be possible for IT teams to manually review every file, limiting their practical use and constraining the potential benefits.

In addition to high volumes, differences in file formats among the various log files can also hinder admins from making full use of them. Without a standardised format, users need to review and analyse each log file separately, which can be cumbersome and time-consuming. Slow log processing also prevents organisations from taking fast action, before the information becomes outdated and less useful.

Since log files are unstructured, parsing and cleaning them can be difficult. Parsing log files involves extracting relevant information from them, such as timestamps, error messages and more. Cleaning log files involves removing irrelevant information, filtering out specific entries or transforming the data into more structured, human readable format. Using automated scripts to extract relevant information makes the process more efficient, reduces human errors and can also automate analysis.

Data parsing is important because it enables cybersecurity professionals to extract insights from large sets of data quickly and efficiently, and to automate repetitive tasks that involve processing data. When parsing data, it is important to pay attention to format of the data and choose the appropriate method for parsing it. It is also important to validate and clean the data before using it in applications that automate data parsing. This includes checking for missing or invalid values, removing duplicates and converting data types as needed.

Log management is a fundamental aspect of cybersecurity and IT operations. It encompasses a series of crucial processes aimed at collecting, storing, analysing and monitoring log data generated by various systems, applications and devices within an organisation’s IT infrastructure.

After collection of logs, log data is typically stored in a centralised repository, which not only facilitates easier access, but also helps maintain data integrity and security. It ensures that logs are protected from unauthorised alterations or deletions, which is essential for forensic investigations and compliance audits.

The log analysis step involves systematically reviewing logs to extract meaningful insights. Analysts sift through vast volumes of log data, searching for patterns, anomalies and suspicious activities that may indicate a security incident or a violation of compliance standards. In essence, log analysis is the process of turning raw data into actionable intelligence. It allows organisations to identify potential security threats, understand system performance issues and uncover compliance adherence gaps. For instance, it can reveal failed login attempts, unauthorised access or unusual network traffic, providing early warning signs of security breaches.

Real-time vigilance is essential in our digital age. Log monitoring and alerting mechanisms should continuously track incoming logs, enabling organisations to detect and respond to security incidents as they occur. Automated alerting systems play a significant role in this process, instantly notifying cybersecurity practitioners of potential threats and to respond to them immediately.

By setting up predefined rules and thresholds, once they are triggered – such as a sudden spike in login failures or an unexpected change in system configuration – alerts are generated. These alerts serve as early warnings, prompting rapid responses to mitigate threats before they escalate into significant breaches.

Log management is the invisible protector of our digital infrastructure, silently watching, recording and warning of potential dangers. Properly executed, it empowers organisations to proactively protect their data, ensure system performance and adhere to compliance requirements.

In the unfortunate event of a security breach, log data becomes invaluable for conducting forensic investigations. Detailed logs can help security teams trace the origins of an attack, identify the affected systems and understand the attacker’s methods. This information is vital for law enforcement and can lead to the prosecution of cybercriminals.

Not all cybersecurity threats come from external actors. Insider threats, whether intentional or accidental, can be equally damaging. Log management allows organisations to monitor employee activities and detect any unusual behaviour that may indicate insider threat.

Recommendations

In the pursuit of effective log management, organisations should meticulously implement the following practices.

Define Clear Logging Policies
Establish logging policies that outline what events should be logged, where they should be stored and for how long. Ensure that these policies align with the organisation’s security and compliance requirements.

Centralise Log Collection
Centralising log collection simplifies the process and allows for more efficient analysis. Use dedicated log collection tools or services to gather logs from various sources into a centralised repository.

Use Secure Log Storage
Protect log data by using secure storage solutions with access controls and encryption. Unauthorised access to log data can pose significant security risks.

Regularly Review and Analyse Logs
Allocate time and resources for regular log analysis. Automated tools can assist in flagging potential issues, but human expertise is crucial for in-depth analysis.

Implement Real-Time Monitoring
Set up real-time monitoring and alerting to respond swiftly to security incidents. Automated alerts ensure that potential threats are addressed promptly.

Retention and Backup Strategy
Determine how long log data needs to be retained based on regulatory requirements and business needs. Implement a backup strategy to prevent data loss in case of hardware failures or other issues.

References

"Parse and Clean Log Files in Python" *GeeksforGeeks*, 20 March 2024, https://www.geeksforgeeks.org/parse-and-clean-log-files-in-python/

"How To Use Python To Parse Server Log Files" *GeeksforGeeks*, 26 March 2024, https://www.geeksforgeeks.org/how-to-use-python-to-parse-server-log-files/

Driscoll, Mike. "Python sleep(): How to Add Time Delays to Your Code" *Real Python*, https://realpython.com/python-sleep/

"Parsing Data in Python: A Tutorial for Beginners" *Pierian Training*, 19 May 2023, https://pieriantraining.com/parsing-data-in-python-a-tutorial-for-beginners/#:~:text=In%20Python%20programming%2C%20data%20parsing,convert%20them%20into%20usable%20formats.

Awati, Rahul and Wigmore, Ivy. "What is a log file?" *TechTarget*, June 2024, https://www.techtarget.com/whatis/definition/log-log-file

"Securing the Digital World | The Importance of Log Management in Cybersecurity: A Comprehensive Guide" *NetWitness*, 16 November 2023, https://www.netwitness.com/blog/the-importance-of-log-management-in-cybersecurity-a-comprehensive-guide/