Name: reut abergel

SID:s8

Unit: TMagen7736.38

Program code: XE105

<u>python project</u>

<u>Table of Contents:</u>

1. INTRODUCTION:

Hi! This is a friendly guide for my Python script. The script's role is to act as an Automated Security Log Parser for Linux authentication logs.

It specifically targets the `/var/log/auth.log` file to identify potential security breaches, unauthorized access attempts, and administrative changes.

1.2 Who is this guide for and how does it helps

1. Security Analysts looking to audit user activity quickly.
2. Students who want to know how to parse system logs using Python.
3. System Administrators who need a Red Flag alert system for suspicious terminal commands.

How does it help?

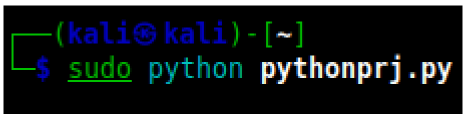
1. Explains the logic used to identify Red Flags in the log and suspicious activity.
2. Breaks down the script's parsing logic (Timestamps, Users, and Commands).
3. Provides a summary of the statistics tracked during the analysis.

1.3 WHAT YOU WILL NEED:

1. Linux OS (Kali): The script is designed to read `/var/log/auth.log`, a standard file in Debian-based systems.
2. Python: Ensure Python is installed.
3. Rsyslog: The script specifically checks for this. If missing, you can install it via
4. Permissions: You generally need `sudo` or root privileges to read the system auth logs.

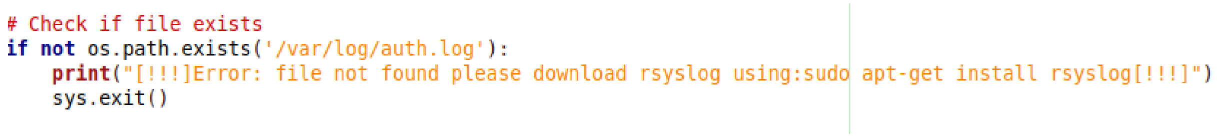
1.4 HOW TO RUN:

1. Execution: Run the script using the Python interpreter with root privileges:



```
┌──(kali㉿kali)-[~]
└─$ sudo python pythonprj.py
```

2. Automatic Safety Check: The script immediately checks if the log file exists using pic



```
# Check if file exists
if not os.path.exists('/var/log/auth.log'):
    print("[!!!]Error: file not found please download rsyslog using:sudo apt-get install rsyslog[!!!]")
    sys.exit()
```

3. If it's missing, it provides the exact command needed to fix the environment and exits safely.

2. PURPOSE OF THE TOOL:

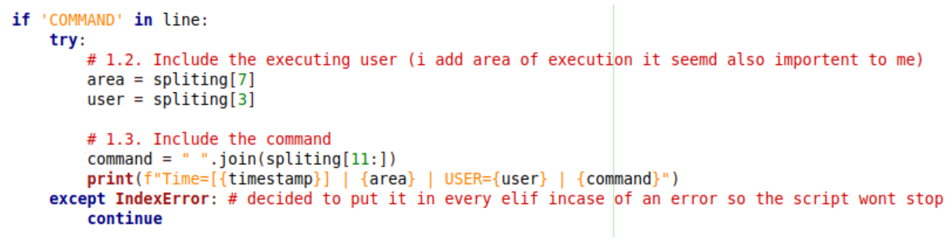
The goal is to transform a messy, dense log file into a readable security report:

1. Suspicious Keyword: The script monitors for high-risk commands like `nmap`, `nc`, and attempts to access `/etc/shadow`.
2. User Lifecycle Tracking: It automatically flags when users are created or deleted.
3. Privilege Escalation Monitoring: It tracks every time a user attempts to use `su` or `sudo`, counting both successful sessions and failed authentication attempts.

3. WHAT I LEARNED FROM BUILDING THE SCRIPT:

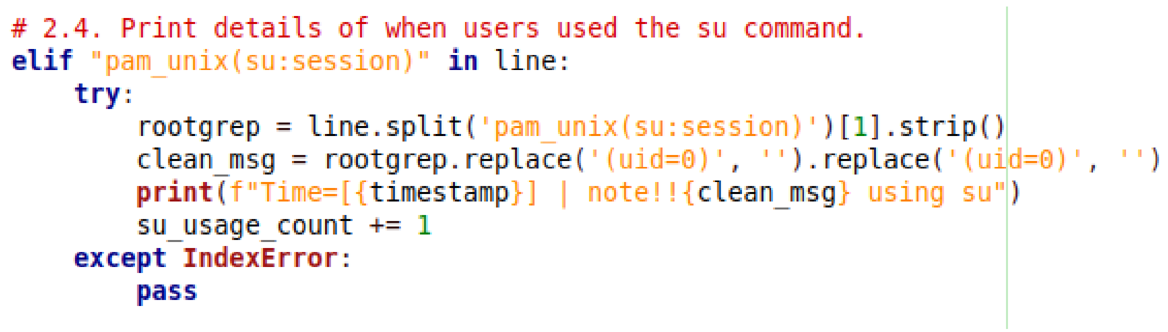
Building this script taught me several advanced Python concepts for data processing and security auditing:

1) Error Handling with `try` and `except`: I learned how to use `try` to attempt data extraction and `except IndexError` to skip lines that don't fit the expected format. This prevents the script from crashing when it hits a messy or incomplete log entry.



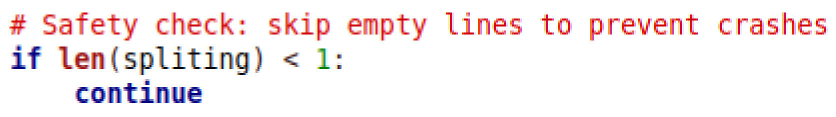
```
if 'COMMAND' in line:
    try:
        # 1.2. Include the executing user (i add area of execution it seemd also importent to me)
        area = spliting[7]
        user = spliting[3]

        # 1.3. Include the command
        command = " ".join(spliting[11:])
        print(f"Time=[{timestamp}] | {area} | USER={user} | {command}")
    except IndexError: # decided to put it in every elif incase of an error so the script wont stop
        continue
```

2) <u>Data Cleaning and Sanitization: </u>I learned how to use `.replace()` and `.strip()` to remove the information i dont need, like `(uid=0),` from the log strings, ensuring the output only shows the essential information.



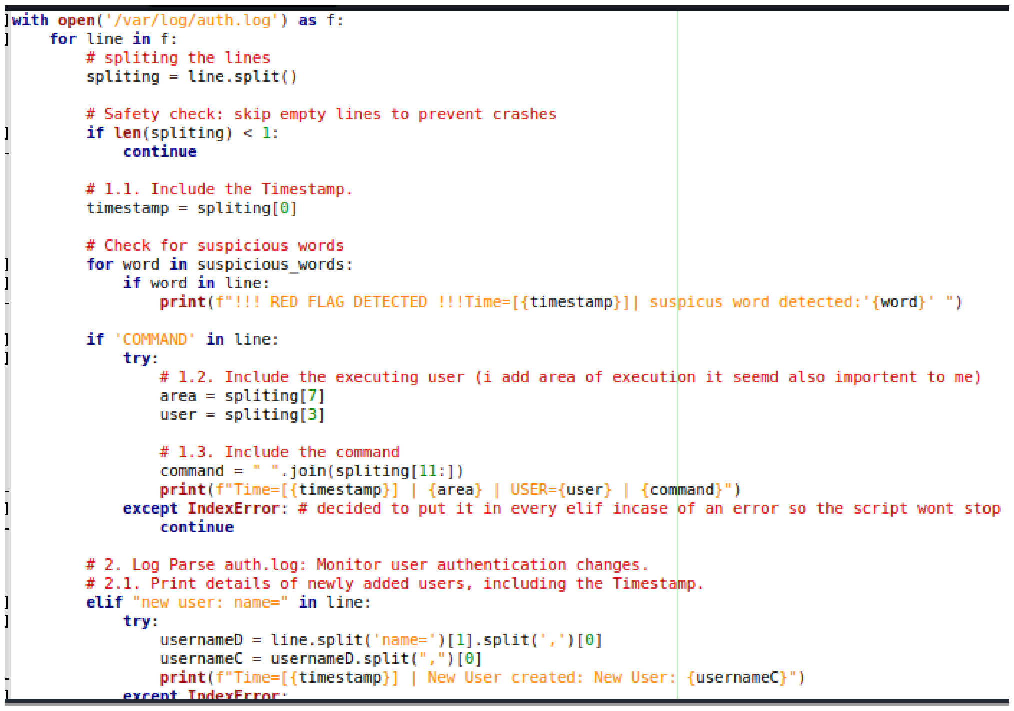
```
# 2.4. Print details of when users used the su command.
elif "pam_unix(su:session)" in line:
    try:
        rootgrep = line.split('pam_unix(su:session)')[1].strip()
        clean_msg = rootgrep.replace('(uid=0)', '').replace('(uid=0)', '')
        print(f"Time=[{timestamp}] | note!!{clean_msg} using su")
        su_usage_count += 1
    except IndexError:
        pass
```

3) <u>Safety Checks for Empty Lines: </u>I discovered how to use `if len(splitting) < 1: continue` to skip empty lines, which is a critical safety check to prevent errors during processing.



```
# Safety check: skip empty lines to prevent crashes
if len(spliting) < 1:
    continue
```

4)<u>Python Indentation and Logic</u>: I learned that in Python, indentation isn't just for looks, it's essential for writing a clean, functional script.



```python
with open('/var/log/auth.log') as f:
    for line in f:
        # spliting the lines
        spliting = line.split()

        # Safety check: skip empty lines to prevent crashes
        if len(spliting) < 1:
            continue

        # 1.1. Include the Timestamp.
        timestamp = spliting[0]

        # Check for suspicious words
        for word in suspicious_words:
            if word in line:
                print(f"!!! RED FLAG DETECTED !!!Time=[{timestamp}]| suspicus word detected:'{word}' ")

        if 'COMMAND' in line:
            try:
                # 1.2. Include the executing user (i add area of execution it seemd also importent to me)
                area = spliting[7]
                user = spliting[3]

                # 1.3. Include the command
                command = " ".join(spliting[11:])
                print(f"Time=[{timestamp}] | {area} | USER={user} | {command}")
            except IndexError: # decided to put it in every elif incase of an error so the script wont stop
                continue

        # 2. Log Parse auth.log: Monitor user authentication changes.
        # 2.1. Print details of newly added users, including the Timestamp.
        elif "new user: name=" in line:
            try:
                usernameD = line.split('name=')[1].split(',')[0]
                usernameC = usernameD.split(",")[0]
                print(f"Time=[{timestamp}] | New User created: New User: {usernameC}")
            except IndexError:
```

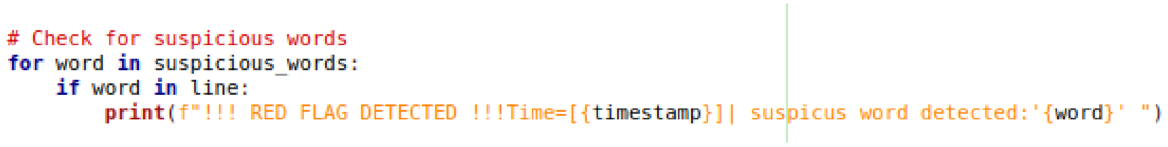
4. ANALYSIS OF FUNCTIONS AND LOGIC:

Script Logic: The Parsing Loop

The script opens the log and iterates through every line, using `split()` to isolate data points.

Red Flag Detection:

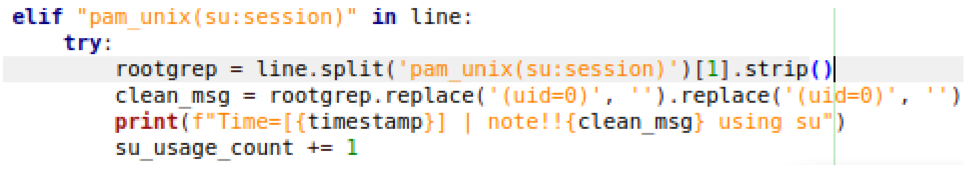
For word in suspicious_words: This loop ensures that every single line is scanned against a blacklist I created of dangerous strings. It provides immediate visual feedback to the analyst when a critical security event occurs.



```python
# Check for suspicious words
for word in suspicious_words:
    if word in line:
        print(f"!!! RED FLAG DETECTED !!!Time=[{timestamp}]| suspicus word detected:'{word}' ")
```

String Cleaning:

Raw logs are often cluttered with system IDs (UIDs). The script uses `.replace()` and `.strip()` to make the output cleaner, making the report human readable by removing the unwanted parts.

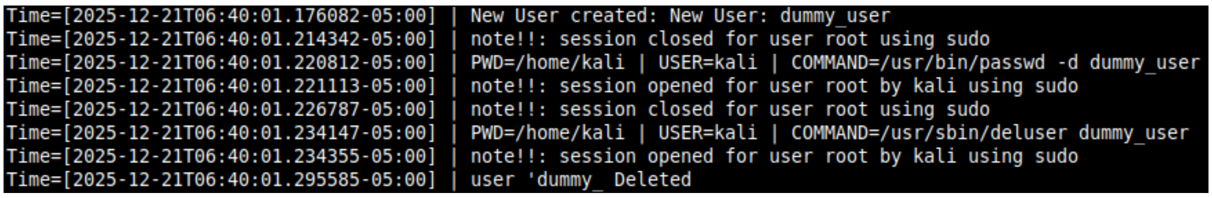


```
elif "pam_unix(su:session)" in line:
    try:
        rootgrep = line.split('pam_unix(su:session)')[1].strip()
        clean_msg = rootgrep.replace('(uid=0)', '').replace('(uid=0)', '')
        print(f"Time=[{timestamp}] | note!!{clean_msg} using su")
        su_usage_count += 1
```

5. SUMMARY OF OUTPUTS:

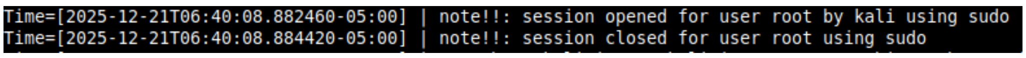
The script categorizes findings into several buckets:

Detects **new user**, **delete user**, and **password change** events



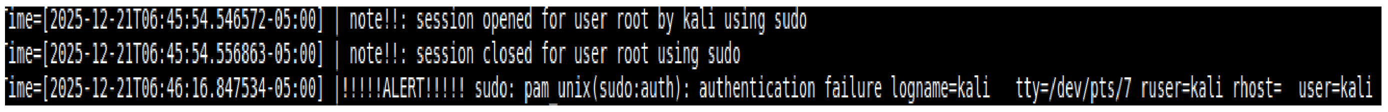
```
Time=[2025-12-21T06:40:01.176082-05:00] | New User created: New User: dummy_user
Time=[2025-12-21T06:40:01.214342-05:00] | note!!: session closed for user root using sudo
Time=[2025-12-21T06:40:01.220812-05:00] | PWD=/home/kali | USER=kali | COMMAND=/usr/bin/passwd -d dummy_user
Time=[2025-12-21T06:40:01.221113-05:00] | note!!: session opened for user root by kali using sudo
Time=[2025-12-21T06:40:01.226787-05:00] | note!!: session closed for user root using sudo
Time=[2025-12-21T06:40:01.234147-05:00] | PWD=/home/kali | USER=kali | COMMAND=/usr/sbin/deluser dummy_user
Time=[2025-12-21T06:40:01.234355-05:00] | note!!: session opened for user root by kali using sudo
Time=[2025-12-21T06:40:01.295585-05:00] | user 'dummy_ Deleted
```

Monitors **pam_unix** sessions for both **su** and **sudo**.



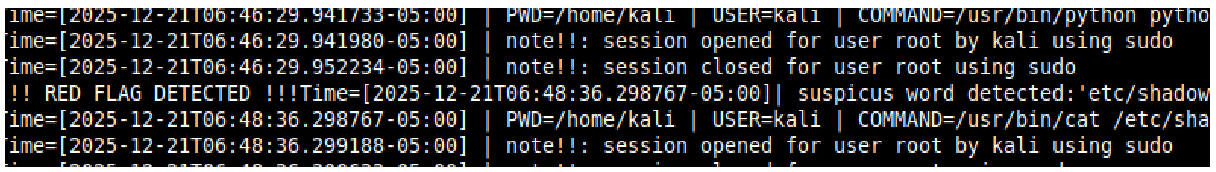
```
Time=[2025-12-21T06:40:08.882460-05:00] | note!!: session opened for user root by kali using sudo
Time=[2025-12-21T06:40:08.884420-05:00] | note!!: session closed for user root using sudo
```

Triggers a **!!!ALERT!!!** For failed sudo authentication attempts



```
ime=[2025-12-21T06:45:54.546572-05:00] | note!!: session opened for user root by kali using sudo
ime=[2025-12-21T06:45:54.556863-05:00] | note!!: session closed for user root using sudo
ime=[2025-12-21T06:46:16.847534-05:00] |!!!!!ALERT!!!!! sudo: pam_unix(sudo:auth): authentication failure logname=kali   tty=/dev/pts/7 ruser=kali rhost=  user=kali
```

Detects suspicious keywords like **nc, nmap,/etc/shadow**, and prints them in a different way from other events, so it's easy to see

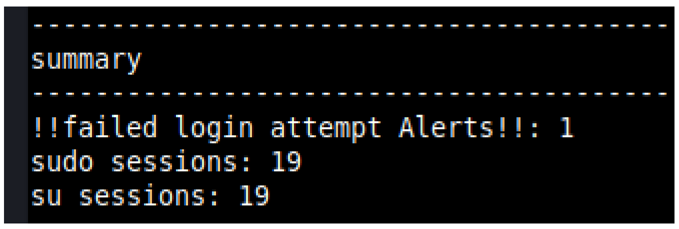


```
ime=[2025-12-21T06:46:29.941733-05:00] | PWD=/home/kali | USER=kali | COMMAND=/usr/bin/python pytho
ime=[2025-12-21T06:46:29.941980-05:00] | note!!: session opened for user root by kali using sudo
ime=[2025-12-21T06:46:29.952234-05:00] | note!!: session closed for user root using sudo
!! RED FLAG DETECTED !!!Time=[2025-12-21T06:48:36.298767-05:00]| suspicus word detected:'etc/shadow
ime=[2025-12-21T06:48:36.298767-05:00] | PWD=/home/kali | USER=kali | COMMAND=/usr/bin/cat /etc/sha
ime=[2025-12-21T06:48:36.299188-05:00] | note!!: session opened for user root by kali using sudo
```

Final Statistics:

At the end of the execution, the script provides a quick-glance summary:

- **Total Failed Login Alerts** (Calculated via `failed_attempts` counter).
- **Total Sudo Sessions**
- **Total Su Sessions**



```
----------------------------------------
summary
----------------------------------------
!!failed login attempt Alerts!!: 1
sudo sessions: 19
su sessions: 19
```

SUMMARY

This tool is a lightweight yet powerful way to perform Security Auditing. Combining string manipulation with real-time log parsing, it helps an analyst to find only the needed information without manually reading thousands of lines of log data.

Thank you for reading and using my script!