Python Project: Automated Security Log Parser

Project by: Reut Abergel

Table of Contents

1. Introduction

This document serves as a technical guide for the Automated Security Log Parser. The primary function of this script is to perform rapid security auditing on Linux authentication logs (/var/log/auth.log).

The script transforms dense, unstructured system logs into a clear, actionable security report by:

- **Parsing:** Extracting critical data points like Timestamps, Usernames, and Executed Commands.
- **Alerting:** Automatically flagging suspicious keywords (e.g., nmap, /etc/shadow) and failed authentication attempts.
- **Tracking:** Monitoring the lifecycle of user accounts (Creation/Deletion) and privilege escalation attempts (su, sudo).

2. Target Audience

This guide and tool are designed for:

- **Security Analysts:** Professionals needing to quickly audit user activity and identify potential breaches without manually reading thousands of lines.
- **System Administrators:** Users requiring a "Red Flag" alert system for unauthorized administrative changes.
- **Cybersecurity Students:** Individuals learning how to process system logs and implement security logic using Python.

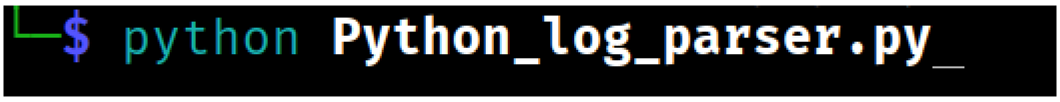
3. Requirements

To successfully execute this script, the following requirements must be met:

1. **Operating System:** Linux (Debian/Kali) with rsyslog installed.
2. **Environment:** Python interpreter.
3. **Privileges:** Root/Sudo privileges are mandatory to read the protected /var/log/auth.log file.
4. **Dependencies:** Standard Python libraries (os, sys).

4. Operational Guide

Step 1 Execution & Initialization: The script must be run with elevated privileges to access the system logs.
Command:



```
└─$ python Python_log_parser.py_
```

Step 2 Automated File Validation: Upon start, the script utilizes the os library to perform a safety check. It verifies that the target log file (/var/log/auth.log) exists. If the file is missing (e.g., rsyslog is not installed), it halts execution and provides the specific remediation command (sudo apt-get install rsyslog).

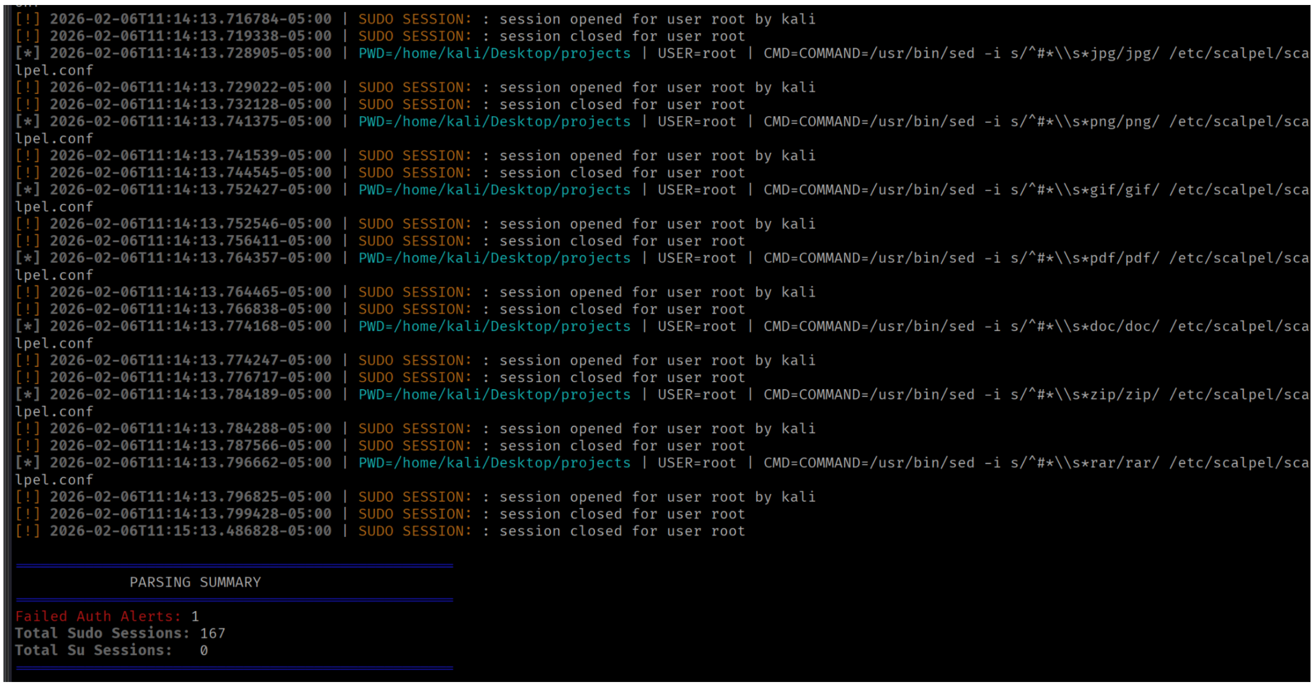
Step 3 Real-Time Parsing Loop: The script initiates a read loop, iterating through the log file line-by-line. It automatically filters out empty lines or incomplete entries using try/except blocks to ensure stability during processing.

Step 4 Red Flag Detection: Every line is scanned against a predefined "Blacklist" of suspicious keywords. High-risk commands like nc (netcat), nmap, or attempts to read sensitive files like /etc/shadow trigger an immediate **"RED FLAG DETECTED"** alert in the output.

Step 5 User & Privilege Auditing: The tool categorizes events into distinct security contexts:

- **New Users:** Detects and reports the creation of new accounts.
- **Privilege Escalation:** Tracks every usage of sudo and su, calculating successful sessions versus failed authentication attempts.

Step 6 Summary & Statistics: At the conclusion of the analysis, the script generates a statistical summary, providing a quick-glance view of the system's security posture (Total Failed Logins, Total Sudo Sessions, etc.).



```
[!] 2026-02-06T11:14:13.716784-05:00 | SUDO SESSION: : session opened for user root by kali
[!] 2026-02-06T11:14:13.719338-05:00 | SUDO SESSION: : session closed for user root
[*] 2026-02-06T11:14:13.728905-05:00 | PWD=/home/kali/Desktop/projects | USER=root | CMD=COMMAND=/usr/bin/sed -i s/^#*\\s*jpg/jpg/ /etc/scalpel/sca
lpel.conf
[!] 2026-02-06T11:14:13.729022-05:00 | SUDO SESSION: : session opened for user root by kali
[!] 2026-02-06T11:14:13.732128-05:00 | SUDO SESSION: : session closed for user root
[*] 2026-02-06T11:14:13.741375-05:00 | PWD=/home/kali/Desktop/projects | USER=root | CMD=COMMAND=/usr/bin/sed -i s/^#*\\s*png/png/ /etc/scalpel/sca
lpel.conf
[!] 2026-02-06T11:14:13.741539-05:00 | SUDO SESSION: : session opened for user root by kali
[!] 2026-02-06T11:14:13.744545-05:00 | SUDO SESSION: : session closed for user root
[*] 2026-02-06T11:14:13.752427-05:00 | PWD=/home/kali/Desktop/projects | USER=root | CMD=COMMAND=/usr/bin/sed -i s/^#*\\s*gif/gif/ /etc/scalpel/sca
lpel.conf
[!] 2026-02-06T11:14:13.752546-05:00 | SUDO SESSION: : session opened for user root by kali
[!] 2026-02-06T11:14:13.756411-05:00 | SUDO SESSION: : session closed for user root
[*] 2026-02-06T11:14:13.764357-05:00 | PWD=/home/kali/Desktop/projects | USER=root | CMD=COMMAND=/usr/bin/sed -i s/^#*\\s*pdf/pdf/ /etc/scalpel/sca
lpel.conf
[!] 2026-02-06T11:14:13.764465-05:00 | SUDO SESSION: : session opened for user root by kali
[!] 2026-02-06T11:14:13.766838-05:00 | SUDO SESSION: : session closed for user root
[*] 2026-02-06T11:14:13.774168-05:00 | PWD=/home/kali/Desktop/projects | USER=root | CMD=COMMAND=/usr/bin/sed -i s/^#*\\s*doc/doc/ /etc/scalpel/sca
lpel.conf
[!] 2026-02-06T11:14:13.774247-05:00 | SUDO SESSION: : session opened for user root by kali
[!] 2026-02-06T11:14:13.776717-05:00 | SUDO SESSION: : session closed for user root
[*] 2026-02-06T11:14:13.784189-05:00 | PWD=/home/kali/Desktop/projects | USER=root | CMD=COMMAND=/usr/bin/sed -i s/^#*\\s*zip/zip/ /etc/scalpel/sca
lpel.conf
[!] 2026-02-06T11:14:13.784288-05:00 | SUDO SESSION: : session opened for user root by kali
[!] 2026-02-06T11:14:13.787566-05:00 | SUDO SESSION: : session closed for user root
[*] 2026-02-06T11:14:13.796662-05:00 | PWD=/home/kali/Desktop/projects | USER=root | CMD=COMMAND=/usr/bin/sed -i s/^#*\\s*rar/rar/ /etc/scalpel/sca
lpel.conf
[!] 2026-02-06T11:14:13.796825-05:00 | SUDO SESSION: : session opened for user root by kali
[!] 2026-02-06T11:14:13.799428-05:00 | SUDO SESSION: : session closed for user root
[!] 2026-02-06T11:15:13.486828-05:00 | SUDO SESSION: : session closed for user root


              PARSING SUMMARY

Failed Auth Alerts: 1
Total Sudo Sessions: 167
Total Su Sessions:   0
```

5. Purpose & Automation

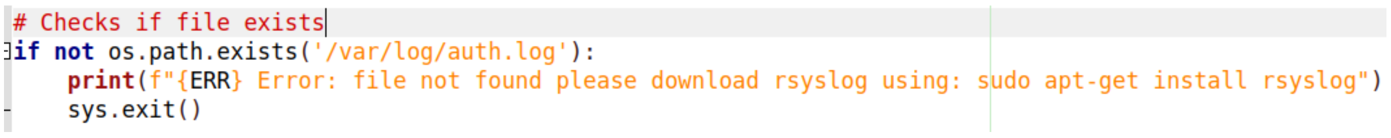
The goal is to transform a messy, dense log file into a readable security report.

1. **Noise Reduction:** Raw logs are often cluttered with system IDs (uid=0) and repetitive data. The script uses string manipulation to sanitize the output, presenting only the essential information (Who, What, When).
2. **Proactive Monitoring:** Instead of waiting for a manual review, the script proactively hunts for indicators of compromise (IOCs) using keyword matching.
3. **Resilience:** By implementing robust error handling (try...except IndexError), the script is designed to skip over malformed log lines rather than crashing, ensuring the audit completes even on corrupted files.

6. Technical Analysis

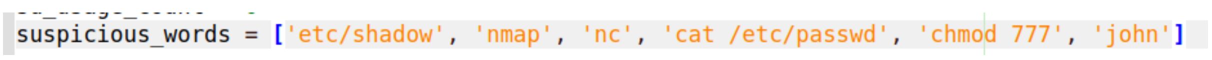
The script relies on four core logic blocks to manage the workflow:

1. Function Log Ingestion & Safety: Ensures the script only runs in a valid environment. Uses os.path.exists to check for the file.

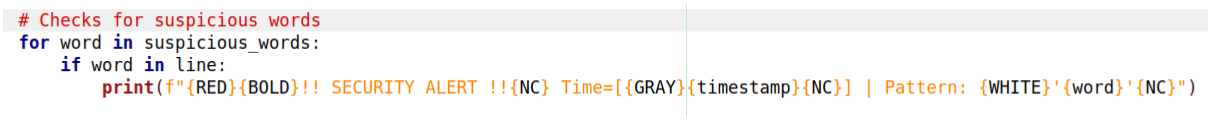


```
# Checks if file exists
if not os.path.exists('/var/log/auth.log'):
    print(f"{ERR} Error: file not found please download rsyslog using: sudo apt-get install rsyslog")
    sys.exit()
```

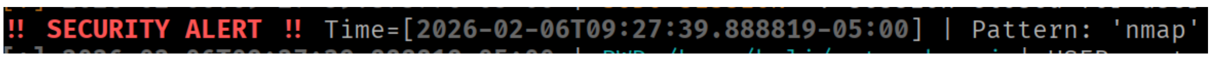
2. Function Keyword Detection (Blacklist): Identifies high-priority threats immediately. **Logic:** Iterates through a list of dangerous strings (suspicious_words) and checks if they appear in the current line.



```
suspicious_words = ['etc/shadow', 'nmap', 'nc', 'cat /etc/passwd', 'chmod 777', 'john']
```

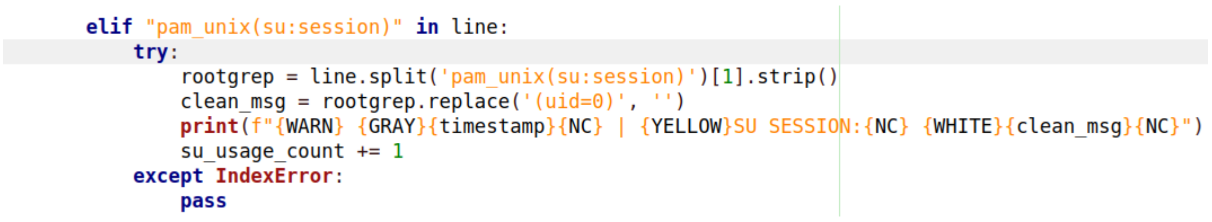


```
# Checks for suspicious words
for word in suspicious_words:
    if word in line:
        print(f"{RED}{BOLD}!! SECURITY ALERT !!{NC} Time=[{GRAY}{timestamp}{NC}] | Pattern: {WHITE}'{word}'{NC}")
```



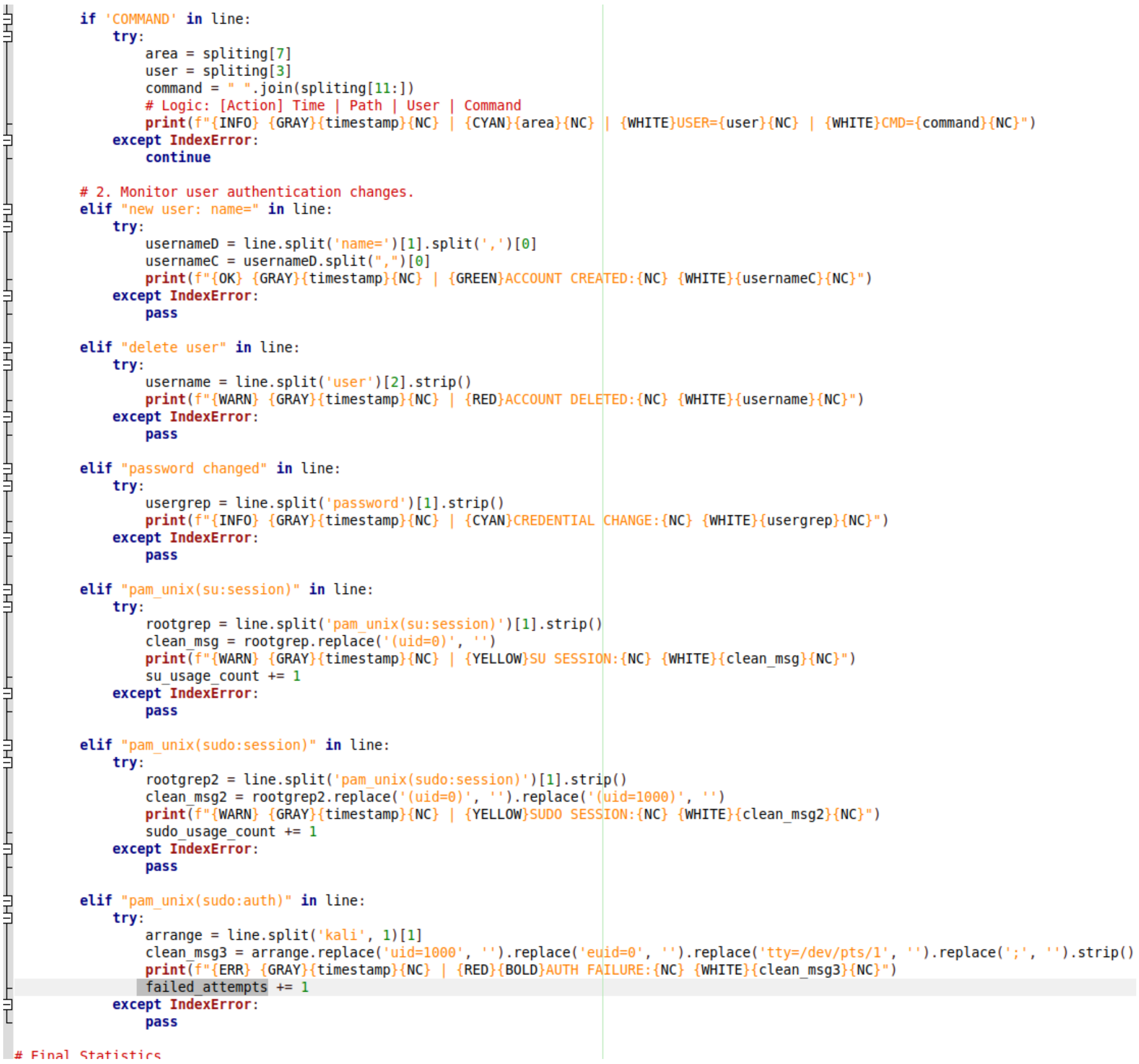
```
!! SECURITY ALERT !! Time=[2026-02-06T09:27:39.888819-05:00] | Pattern: 'nmap'
```

3. Function Data Sanitization: Cleans up the log output for human readability. **Logic:** Uses chained .replace() methods to remove technical noise like (uid=0) and .strip() to remove whitespace.



```
elif "pam_unix(su:session)" in line:
    try:
        rootgrep = line.split('pam_unix(su:session)')[1].strip()
        clean_msg = rootgrep.replace('(uid=0)', '')
        print(f"{WARN} {GRAY}{timestamp}{NC} | {YELLOW}SU SESSION:{NC} {WHITE}{clean_msg}{NC}")
        su_usage_count += 1
    except IndexError:
        pass
```

4. Function Event Categorization: distinguishing between normal administrative tasks and potential threats.
Logic: Uses if/elif statements to route different log types (New User vs. Sudo vs. Failed Login) to specific processing blocks.



```python
        if 'COMMAND' in line:
            try:
                area = spliting[7]
                user = spliting[3]
                command = " ".join(spliting[11:])
                # Logic: [Action] Time | Path | User | Command
                print(f"{INFO} {GRAY}{timestamp}{NC} | {CYAN}{area}{NC} | {WHITE}USER={user}{NC} | {WHITE}CMD={command}{NC}")
            except IndexError:
                continue

        # 2. Monitor user authentication changes.
        elif "new user: name=" in line:
            try:
                usernameD = line.split('name=')[1].split(',')[0]
                usernameC = usernameD.split(",")[0]
                print(f"{OK} {GRAY}{timestamp}{NC} | {GREEN}ACCOUNT CREATED:{NC} {WHITE}{usernameC}{NC}")
            except IndexError:
                pass

        elif "delete user" in line:
            try:
                username = line.split('user')[2].strip()
                print(f"{WARN} {GRAY}{timestamp}{NC} | {RED}ACCOUNT DELETED:{NC} {WHITE}{username}{NC}")
            except IndexError:
                pass

        elif "password changed" in line:
            try:
                usergrep = line.split('password')[1].strip()
                print(f"{INFO} {GRAY}{timestamp}{NC} | {CYAN}CREDENTIAL CHANGE:{NC} {WHITE}{usergrep}{NC}")
            except IndexError:
                pass

        elif "pam_unix(su:session)" in line:
            try:
                rootgrep = line.split('pam_unix(su:session)')[1].strip()
                clean_msg = rootgrep.replace('(uid=0)', '')
                print(f"{WARN} {GRAY}{timestamp}{NC} | {YELLOW}SU SESSION:{NC} {WHITE}{clean_msg}{NC}")
                su_usage_count += 1
            except IndexError:
                pass

        elif "pam_unix(sudo:session)" in line:
            try:
                rootgrep2 = line.split('pam_unix(sudo:session)')[1].strip()
                clean_msg2 = rootgrep2.replace('(uid=0)', '').replace('(uid=1000)', '')
                print(f"{WARN} {GRAY}{timestamp}{NC} | {YELLOW}SUDO SESSION:{NC} {WHITE}{clean_msg2}{NC}")
                sudo_usage_count += 1
            except IndexError:
                pass

        elif "pam_unix(sudo:auth)" in line:
            try:
                arrange = line.split('kali', 1)[1]
                clean_msg3 = arrange.replace('uid=1000', '').replace('euid=0', '').replace('tty=/dev/pts/1', '').replace(';', '').strip()
                print(f"{ERR} {GRAY}{timestamp}{NC} | {RED}{BOLD}AUTH FAILURE:{NC} {WHITE}{clean_msg3}{NC}")
                failed_attempts += 1
            except IndexError:
                pass

# Final Statistics
```

7. Technical Skills Gained

Through the development of this project, I developed the following technical skills:

- **Python String Manipulation:** learned how to use split(), join(), and replace() to parse complex, unstructured text data.
- **Security Logic Implementation:** Learned to translate security concepts (Blacklisting, Privilege Tracking) into functional code.
- **Robust Error Handling:** Implemented try...except blocks to prevent script crashes when encountering unexpected data formats..

8. Summary

This project resulted in a lightweight yet powerful Security Auditing tool. By combining string manipulation with real-time log parsing logic, it allows an analyst to instantly identify Red Flags and track user activity without manually sifting through thousands of lines of raw data. It demonstrates a strong ability to automate security tasks using Python.