

# DEVELOP A CYBERSECURITY MONITORING TOOL

***BSC(Hons)Cybersecurity***

*Pullwan Devashish: 2412298*

*Peethamree Parvish: 2412277*

*Jhorreea Kaushik: 2414372*

*Rungiah Tirouvalen: 2410805*

# TABLE OF CONTENT

1. Abstract & Introduction.....	3
2. Investigation.....	4
3. Analysis.....	5-6
4. Design.....	7-10
5. Implementation.....	11-15
6. Testing/Evaluation.....	16-21
7. References.....	22

# INTRODUCTION AND ABSTRACT

## INTRODUCTION

For our 2nd Computer Programming (ICT 1214Y) assignment, we have been assigned a task that deals with the development of a cybersecurity monitoring tool: a python application that reads a type of log file and provides useful perception after having processed the data. From the various choices of log file types, our team has been keen on working on authentication log files. Along with our knowledge and apprehension on authentication log files, object-oriented principles and various data structures, we have also made use of sources from the internet in the realization of this assignment.

In addition to this report is our source code, as per the project requirements.

## ABSTRACT

Our report provides an in-depth evaluation of authentication log files through investigation, analysis, design, implementation, and an assessment as per the assignment's criteria. The investigation focuses on what authentication logs are and how they can be used to identify common security threats. The analysis emphasises on techniques for deriving valuable insights from an authentication log file that we have created ourselves so that security monitoring or identifying abnormalities are made possible. As for the design segment, we have provided a systematic approach for our autonomous log analysis solution in accordance with our findings in the investigation and analysis.

For illustration, we have used draw.io to make our own class diagram. The implementation section outlines the development of our python-based solution that effectively processes and evaluates information in authentication log files. To determine the validity and effectiveness of our code, we have used our authentication log file to test the code and to evaluate the output, accordingly, ensuring that it adheres to the aims we had set in the early stages.

# INVESTIGATION

## An introduction on Authentication Log Files

Authentication log files (auth logs) are critical records used to keep track of user approbation events on a system. Information about system access, unsuccessful authentication attempts, and login attempts are all included in these logs (Sematext, 2025). Examining and evaluating authentication log files generated by operating systems and authentication mechanisms, is crucial in understanding patterns and identifying potential breaches and thus enhance the overall security posture (Isaiah, 2025).

They are typically stored at different locations based on the respective OS, for instance, `/var/log/auth.log` on Ubuntu and `'eventvwr.msc'` (event viewer) on Windows.

## Common security threats identified through Authentication Logs

Authentication logs reveal several types of security threats, such as:

- Brute force attacks: Repeated failed login attempts from the same IP indicate the aim of guessing passwords through automated attempts.
- Credential stuffing: Multiple login attempts with different usernames from a common IP address indicate that attackers could be using stolen credentials obtained from past data breaches.
- Log manipulation and deletion: Irregular gaps in the records or missing logs could indicate that attackers are tampering authentication logs to hide their activities in the system.
- Insider threats: Suspicious authentication behaviour from legitimate users, i.e., login attempts outside of working hours.
- Session hijacking: Logs showing a sudden change in session details or IP addresses could indicate that an attacker has taken over an authenticated session.

(Massey & Brown, 2020)

Along with other industry standards, the GDPR (General Data Protection Regulation), HIPAA (Health Insurance Portability and Accountability Act) command the logging and monitoring of authentication events in various fields.

Since not all attacks are revealed immediately, it is essential that authentication logs are kept in a safe location and for the proper amount of time.

## ANALYSIS

In this section, we shall study and evaluate the different ways of interpreting an authentication log file to detect any security threats.

Authentication logs record various types of information related to system access, such as:

- Timestamps: The date and time of each authentication attempt.
- Usernames: The identity of the user attempting to log in.
- Source IP Address: The IP address from which the login request originated.
- Success or Failure Status: Whether the authentication attempt was a success or not.
- Reason for Failure: Unsuccessful authentication (e.g., incorrect password, inaccurate biometrics).
- Session Duration and Activity: Information about start and termination of sessions, and thus the duration.
- (Knupfer, 2024)

An example of a log entry from an authentication log file is as follows:

*Jan 13 12:34:56 server1 sshd[5678]: Failed password for invalid user poule from 192.168.1.50 port 42412 ssh2*

From the entry, we are able to gather the following information:

1. The date and time of the event: **Jan 13 12:34:56**
2. The name of the server from where the event took place: **server1**
3. The protocol that recorded the event and its ID: **sshd, ID=5678**
4. The authentication attempt was unsuccessful: **Failed password**
5. The username used in login attempt is invalid: **for invalid user poule**
6. The IP address from which the attempt was made: **192.168.1.50**
7. The source port used from which the connection was initiated: **port 42412**
8. The version of the protocol (SSH) used: **ssh2 (SSH version 2)**

These deductions can then be used to detect and mitigate threats accordingly.

## ANALYSIS

Below is a segment of log entries that we shall consider:

*Jan 13 12:34:56 server1 sshd[5678]: Failed password for invalid user poule  
from 192.168.1.50 port 42412 ssh2*

*Jan 13 12:34:57 server1 sshd[5678]: Failed password for invalid user kowalski  
from 192.168.1.50 port 42413 ssh2*

*Jan 13 12:34:58 server1 sshd[5678]: Failed password for invalid user rico  
from 192.168.1.50 port 42414 ssh2*

After analysing, we can conclude that the multiple failed login attempts with different usernames in a short period of time from the same IP address indicate a brute force attack. Therefore, the attacker's IP address (192.168.1.50 in this case) can be blocked by manually setting firewall rules.

In larger authentication files, we can use commands in a source code to group/classify failed login attempts, the concerned IPs and other relevant information to identify threats easier, rather than relying solely on the chronological order of the logs.

Another instance that suggests a potential security breach that can be detected through authentication logs after grouping records is abnormal logins (e.g., a user successfully logging in from geographically distant places having different IP addresses in a short time interval indicates compromised credentials)

# DESIGN

## Purpose of authentication log file analysis

To monitor events happening on a system, authentication log files are regularly created and updated to contain information about the system as well as its users. To ensure this process is carried out effectively, an algorithm must be configured to help analyse and display desired output from an authentication log file.

The program should be able to:

- extract data from an authentication log file.
- analyse user authentication attempts.
- detect and track password change events.
- manage user account creation and deletion events.
- monitor user session activities and duration.

## Authentication log file structure

The log file stores records in plain text format, where each record holds information on users and events that occur during communication on the system. The algorithm should easily access data in the file to provide desired outputs while monitoring the system to identify any discrepancies.

The main elements of the log file are:

- Timestamps
- Usernames
- IP addresses
- Authorization Status
- Session information

# DESIGN

## Analysis Approach

The log file analysis is performed by parsing through each stored record using regular expressions to extract relevant data. The following sections provide an overview of what each part of the algorithm will carry out as well as its methodologies used.

### Authentication log analysis

- This section monitors any user authentication attempt, it extracts details about:
- Successful login attempts to monitor access.
- Each failed login attempts by its respective user and IP to track for any anomalies.
- Monitor any granted Sudo commands to restrict privileges provided.
- Output authorization attempts detail in concise and effective format.

### Password Change Analysis

This section emphasizes detecting and tracking any password change event that may pose as potential security issues by:

- Extracting usernames upon detecting a password change event
- Recording number of password change attempts by each user using a dictionary

### Account Management Analysis

- This section tracks creation and deletion events among the users in the system. This helps in ensuring proper control over account management by:
- Retrieving usernames and timestamps of users who have either created or deleted their accounts.
- Storing data about each account created and deleted in dictionaries to be able to provide a clear summary.
- Providing timestamps to help with any anomaly tracking.



# DESIGN

## Session and Logout Analysis

This section analyses user session activities which include login and logout durations. It can help in identifying users with suspicious sessions and detect possible issues like unauthorized access via:

- Extracting timestamps attached to their corresponding login and logout.
- Tracking session duration for each user to get a logout pattern.
- Summarizing information on each session in appropriate format.

## Tools and techniques used in designing the algorithm

- Regular expressions(regex): to link any specific log patterns effectively.
- Python's built-in data structures such as: Dictionaries, to store data of each user and events that occur during their session.
- Datetime module: to handle timestamps throughout the different events
- File Handling: files are analysed line by line to maximise efficiency while handling large data files without the risk of overloading memory.

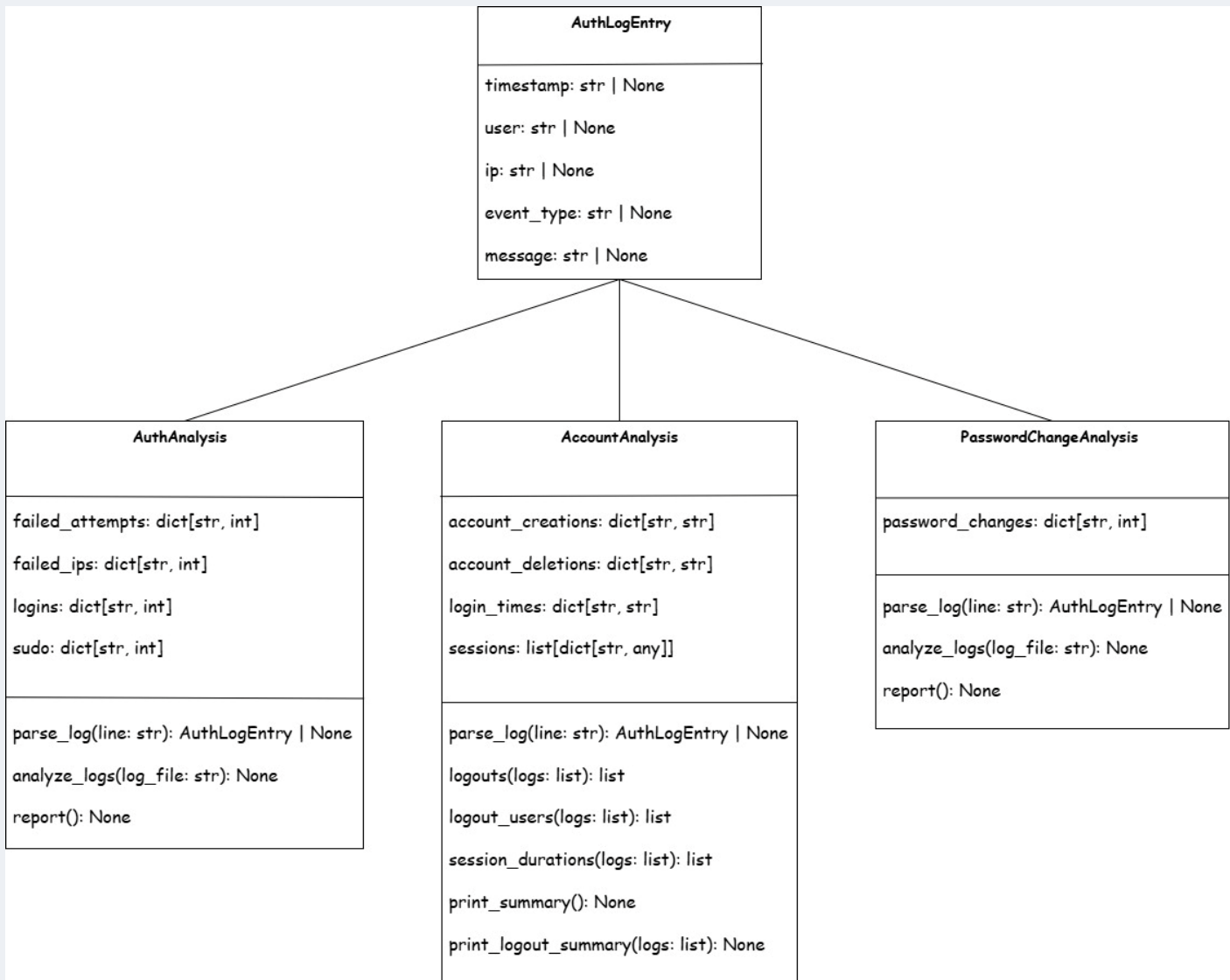
## **Class Diagram**

### Overview of the Diagram

The following class diagram represents our proposed log analysis system that will analyse authentication logs.

It contains one main log entry class which is connected to three analysis classes. Each analysis class (AuthAnalysis, AccountAnalysis and PasswordChangeAnalysis) is linked to main log entry class (AuthLogEntry) because they are dependant on authentication log records to be run.

# DESIGN



## Function of each class

- **AuthLogEntry:** stores log details such as timestamp, user IP, event type and message.
- **AuthAnalysis:** stores unsuccessful login attempts, failed IPs, sudo commands and authorised logins.
- **PasswordChangeAnalysis:** analyses password change events.
- **AccountAnalysis:** monitors account creation, deletion, login times and session durations

# IMPLEMENTATION

## Implementation

The purpose of this implementation is to analyze authentication logs, extracting key information such as failed login attempts, successful logins, and sudo command usage. The following sections detail the structure and logic used to parse and categorize log entries.

### Programming Language & Libraries

The implementation was carried out in Python, utilizing the following libraries:

- `re`: Regular expressions were used to identify and extract relevant data from authentication log entries.
- `datetime`: This library is utilized for handling timestamps when necessary.
- `dataclasses`: A Python feature that allows us to define simple data structures (in this case, `AuthLogEntry`) with minimal boilerplate code.

## Key Components

### AuthLogEntry Class

The `AuthLogEntry` class defines a data structure for storing relevant details from each log entry. This includes:

- `timestamp`: The time the log entry was recorded.
- `user`: The username associated with the log event.
- `ip`: The IP address from which the attempt was made.
- `event_type`: The type of event (e.g., failed login, successful login, sudo command).
- `message`: Any additional information associated with the event.

## IMPLEMENTATION

```
@dataclass
class AuthLogEntry:
    timestamp: str = None
    user: str = None
    ip: str = None
    event_type: str = None
    message: str = None
```

### AuthAnalysis Class

The AuthAnalysis class encapsulates the logic for analyzing authentication logs. It maintains dictionaries to categorize the log entries:

- `failed_attempts`: Stores failed login attempts keyed by username.
- `failed_ips`: Stores failed login attempts keyed by IP address.
- `logins`: Stores successful login events.
- `sudo`: Stores sudo command executions.

The key method in this class is `parse_log`, which processes a single line of the log. This method uses regular expressions to identify various patterns in the log entries:

- **Failed login attempts**: The `failed_login_pattern` regex captures failed login attempts, extracting the username and IP address.
- **Successful login attempts**: The `successful_login_pattern` regex captures successful logins with the same details.
- **Sudo command usage**: The `sudo_pattern` regex identifies sudo command entries and captures the user and command executed.

## IMPLEMENTATION

### Code:

```
class AuthAnalysis:
    def __init__(self):
        self.failed_attempts = {}
        self.failed_ips = {}
        self.logins = {}
        self.sudo = {}

    def parse_log(self, line: str):
        failed_login_pattern = r"Failed password for (invalid user )?(?P<user>[\w-]+) from (?P<ip>\d+\.\d+\.\d+\.\d+)"
```

This line defines a regular expression (regex) pattern that will be used to identify failed login attempts. Let's break down the regex:

- Failed password for: Matches the literal text "Failed password for" in the log entry.
- (invalid user )?: Optionally matches the phrase "invalid user", which can appear before the username if the user does not exist.
- (?P<user>[\w-]+): This is a named group called user, which captures the username attempting to log in. It matches any sequence of alphanumeric characters or hyphens ([\w-]), followed by one or more occurrences (+).
- from (?P<ip>\d+\.\d+\.\d+\.\d+): This captures the IP address from which the login attempt originated. The pattern \d+\.\d+\.\d+\.\d+ matches an IP address format (e.g., "192.168.0.1").

## IMPLEMENTATION

```
successful_login_pattern = r"Accepted password for (?P<user>[\w\-.]+) from  
(?P<ip>\d+\.\d+\.\d+\.\d+)"
```

This line defines a similar regex pattern for identifying successful login attempts:

- Accepted password for: Matches the literal text "Accepted password for" in the log entry.
- (?P<user>[\w\-.]+): Captures the username that successfully logged in.
- from (?P<ip>\d+\.\d+\.\d+\.\d+): Captures the IP address from which the successful login attempt was made.

```
sudo_pattern = r"sudo: (?P<user>[\w\-.]+) :.*COMMAND=(?P<command>.+)"
```

This line defines the regex pattern for detecting sudo command usage:

- sudo:: Matches the literal text "sudo:" in the log entry.
- (?P<user>[\w\-.]+): Captures the username who executed the sudo command.
- :.\*: Matches any characters after the colon until the next part.
- COMMAND=(?P<command>.+): Captures the command that was executed with sudo. The .+ matches any characters after "COMMAND=" until the end of the line.

# IMPLEMENTATION

## Challenges & Solutions

Throughout the development of this implementation, the following challenges were encountered and resolved:

- Handling diverse log formats: The main challenge was ensuring that various log entry formats were correctly parsed. To handle this, regular expressions were carefully crafted to match both successful and failed login patterns, as well as sudo command usage. This approach allows the program to process different formats consistently.
- Efficient categorization: Storing parsed log entries in dictionaries (failed\_attempts, failed\_ips, logins, sudo) enables efficient categorization and retrieval of data. This structure ensures that each log entry is processed only once, and the categorization can be easily accessed for further analysis.

By following this approach, the implementation ensures accurate and efficient analysis of authentication logs, providing insights into failed login attempts, successful logins, and sudo command usage.

# TESTING/ EVALUATION

## Overview

The goal of the Testing/Evaluating part of this report is to verify if the program can correctly process an authentication log file and provide us with useful insights required. The following will be evaluated:

- Accurately identifies and categorizes event types. (successful/ failed logins, sudo command usage, password change, account creation/deletion and log out events)
- Accurately calculates session duration for both login and logout type events.
- Output clearly the analysis of each module

## Testing Approach

The following are the types of testing that were adopted:

1. **Module Testing:** Each separate module of the application (Authentication, Password Change, and Account Analysis) was tested on its own. We created our own test input to ensure that the pattern recognition of various event types and counting was correct.
2. **Integration Testing:** The final test log file was assessed to see if all modules parsed the logs successfully and if the final report was coherent.
3. **Regression Testing:** The new log with a different event type was created again to see if the old code still worked, and it did, meaning it was successful even after some changes.
4. **Output Verification:** The output that the application produced was compared to what was originally expected output for each category to ensure that the number of counts, timestamps, usernames, and IPs were all present.



# TESTING/ EVALUATION

## Testing Environment

- **Programming Language:** Python
- **Setup:** The tests were executed in Visual Studio Code. Libraries such as re, datetime and dataclasses were used.
- **Test Data:** The file “finalauthlogt.txt” was used for the final testing of the program as it contains various types of log entries covering all aspect of the program, mainly:
  1. Failed Login Attempts.
  2. Successful Login Attempts.
  3. Sudo Command Usage.
  4. Password Change.
  5. Account Creation and Deletion Events.
  6. Logout Events and Session Durations.

## Specific Test Scenarios

The following test cases were designed to ensure that all the modules of the program are working coherently and delivering accurate outcomes:

### **1. Failed Login Attempts Test:**

- Log Entry: Jan 13 12:34:56 server1 sshd[5678]: Failed password for invalid user poule from 192.168.1.50 port 42412 ssh2
- Expected Outcome: The user(poule) and their corresponding IP (192.168.1.50) should be recorded with an incremented count of Failed Login attempt and Failed IP.

### **2. Successful Login Attempts:**

- Log Entry: Dec 22 10:15:34 server1 sshd[1234]: Accepted password for romain from 192.168.1.1 port 22 ssh2
- Expected Outcome: The user(romain) should be recorded with an incremented count for Successful Login Attempts.

### **3. Sudo Command Usage:**

- Log Entry: Apr 7 12:37:24 server1 sudo: poseidon : TTY=pts/0 ; PWD=/home/poseidon ; USER=root ; COMMAND=/sbin/reboot
- Expected Outcome: The user(poseidon) should be recorded with an incremented count for Sudo Command Usage.

## TESTING/ EVALUATION

### 4. Password Change:

- Log Entry: Jul 20 12:56:00 server passwd: password changed for ashmit
- Expected Outcome: The user(ashmit) should be recorded with an incremented count for Password Change.

### 5. Account Creation and Deletion Events:

- Log Entry: Mar 22 20:36:00 User soso created
- Expected Outcome: The user(soso) should be recorded with an incremented count of Account Created.

### 6. Logout Events and Duration sessions:

- Log Entry: Sep 25 15:18:55 server1 sshd[9101]: session closed for user pouxi
- Expected Outcome: The user(pouxi) should be recorded, and the session duration should be accurately calculated

## Evaluation

The following are the 3 main pillars that we will use to evaluate our program.

### 1. Accuracy:

- Accurate detection and parsing of all logs.
- Accurate number of occurrences and correct event types identified.

### 2. Efficiency:

- Ability to process a complete log file in a timely manner.
- Resource usage was appropriate for the log file tested.

### 3. Output:

- Quality of output reports.
- Relevance of results (how long sessions were open and comprehensive reports of what each person did on their respective accounts.

# TESTING/ EVALUATION

## Report Analysis

These results are based on the “finalauthlog.txt” file.

### 1. Authentication Analysis:

- **Failed Login Attempts-** The program correctly identified and categorized the event type as failed login and recorded both the username(poule) and corresponding IP (192.168.1.50).
- **Successful Login Attempts-** The program correctly identified and categorized the event type as successful login and recorded both the username(romain).
- **Sudo Command Usage-** The sudo command by user poseidon was correctly recorded and counted.

#### Authentication Log Insights:

##### Failed User Login Attempts:

Jan 13 12:34:56-poule: 1

##### Failed IP Login Attempts:

Jan 13 12:34:56-192.168.1.50: 1

##### Successful Logins:

Feb 3 11:00:41-soso: 1

Sep 25 15:05:29-pouxi: 1

Dec 22 10:15:34-romain: 1

##### Sudo Command Usage:

Apr 7 12:37:24-poseidon: 1

# TESTING/ EVALUATION

## 2. Password Change:

- Three password change events were correctly identified and recorded.

```
Password Changes:  
Jul 20 12:56:00-ashmit: 1  
Oct 8 22:41:39-pikachu: 1  
Dec 28 12:41:34-megatron: 1
```

## 3. Account Analysis:

- Account Creation - The creation of accounts of users soso and michelangelo were correctly identified and recorded alongside with their timestamps.
- Account Deletion- The deletion event was captured, and user baron was recorded.

```
Account Creation and Deletion Report:
```

```
Created Accounts:
```

```
Jan 22 20:36:06 - soso
```

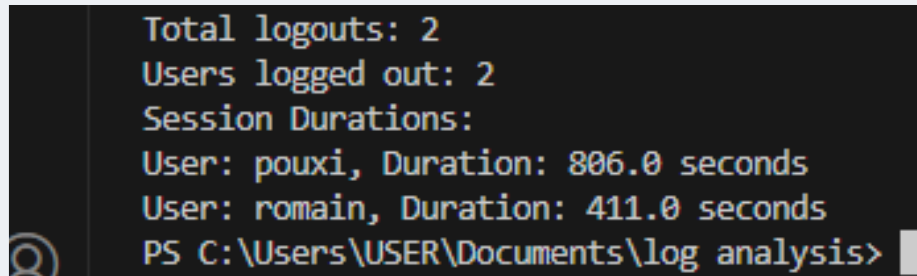
```
Apr 25 10:46:56 - michelangelo
```

```
Deleted Accounts:
```

```
Jul 29 17:37:01 - baron
```

## TESTING/ EVALUATION

- **Logout Events and Session Durations-** The Logout events were correctly identified, and the users' session durations were accurately calculated based on the timestamps from "Accepted password" and "session closed".

A terminal window with a dark background and light-colored text. On the left side, there is a small circular icon containing a stylized '@' symbol. The text in the terminal is as follows:

```
Total logouts: 2
Users logged out: 2
Session Durations:
User: pouxi, Duration: 806.0 seconds
User: romain, Duration: 411.0 seconds
PS C:\Users\USER\Documents\log analysis>
```

### Conclusion

The Authentication Log Analysis Program was fairly tested and evaluated and the assessment reveals that the program was able to process the "finalauthlog.txt" file by extracting insightful information about authentication activities and user account management events. As revealed above by the Report Analysis the program was able to correctly identify and categorize the different event types and session durations of the multiple users were also derived accurately.

Ultimately, the application assessed is a worthwhile tool for authentication log analysis and system security auditing. It provides professionals with a structured presentation of results required for proper investigation and mitigation of incidents. Where this application could be further improved, would be adding real-time notification features or more robust session tracking and reported errors; however, the application operates effectively within its intended realm from a performance and dependability perspective in accordance with precision and performance and also consistency.

## REFERENCES

1. Isaiah, A., 2025. *Monitoring Linux Authentication Logs: A Practical Guide*. [Online] Available at: <https://betterstack.com/community/guides/logging/monitoring-linux-auth-logs/> [Accessed 14 March 2025].
2. Knupfer, F., 2024. *Authentication 801 monitoring* [Online] Available at: <https://newrelic.com/blog/best-practices/authentication-log-monitoring> [Accessed 25 February 2025].
3. Massey, J. & Brown, M., 2020. *Best practices for monitoring authentication logs*. [Online] Available at: <https://www.datadoghq.com/blog/how-to-monitor-authentication-logs/> [Accessed 24 March 2025].
4. Sematext, 2025. *17 Linux Log Files You Must Be Monitoring*. [Online] Available at: <https://sematext.com/blog/17-linux-log-files-you-must-be-monitoring/> [Accessed 27 March 2025].