System Security

Assignment 3

Oct 2024 Semester

Name: Jeslyn Ho Ka Yan, ID: 1024 1485

Name: CHEA Darayuth, ID: 10240675

Table of Content

1 Section 1: Initial Input Phase

1.1 Data Parsing and Internal Storage

The program reads data from two input files, Events.txt and Stats.txt, which define the events and their statistical properties. The program parses these files and stores the data in Python dictionaries as follows:

- **Events Dictionary (events):**
  - This dictionary stores the details of each event from Events.txt.
  - Each event is represented as a key-value pair, where the key is the event name, and the value is a dictionary with the attributes:
    - **types, min, max, weight**
- **Statistics Dictionary (stats):**
  - This dictionary stores the statistical properties of each event as defined in Stats.txt.
  - Each event name is a key, and its value is a dictionary with:
    - **Mean and standard deviation**

It would look like this

```
Parsing Events file: Events.txt
Number of events in Events.txt: 5
Events parsed successfully: {
'Logins': {'type': 'D', 'min': 0, 'max': inf, 'weight': 2},
'Time online': {'type': 'C', 'min': 0, 'max': 1440, 'weight': 3},
'Emails sent': {'type': 'D', 'min': 0, 'max': inf, 'weight': 1},
'Emails opened': {'type': 'D', 'min': 0, 'max': inf, 'weight': 1},
'Emails deleted': {'type': 'D', 'min': 0, 'max': inf, 'weight': 2}}

Parsing Stats file: Stats.txt
Number of events in Stats.txt: 5
Stats parsed successfully: {
'Logins': {'mean': 4.0, 'std_dev': 1.5},
'Time online': {'mean': 150.5, 'std_dev': 25.0},
'Emails sent': {'mean': 10.0, 'std_dev': 3.0},
'Emails opened': {'mean': 12.0, 'std_dev': 4.5},
'Emails deleted': {'mean': 7.0, 'std_dev': 2.25}}

Checking for inconsistencies...
Inconsistencies found: []
Parsed Events: {
'Logins': {'type': 'D', 'min': 0, 'max': inf, 'weight': 2},
'Time online': {'type': 'C', 'min': 0, 'max': 1440, 'weight': 3},
'Emails sent': {'type': 'D', 'min': 0, 'max': inf, 'weight': 1},
'Emails opened': {'type': 'D', 'min': 0, 'max': inf, 'weight': 1},
'Emails deleted': {'type': 'D', 'min': 0, 'max': inf, 'weight': 2}}

Parsed Stats: {
'Logins': {'mean': 4.0, 'std_dev': 1.5},
'Time online': {'mean': 150.5, 'std_dev': 25.0},
'Emails sent': {'mean': 10.0, 'std_dev': 3.0},
'Emails opened': {'mean': 12.0, 'std_dev': 4.5},
'Emails deleted': {'mean': 7.0, 'std_dev': 2.25}}
Inconsistencies found: []
```

1.2 Potential Inconsistencies

The program detects potential inconsistencies between Events.txt and Stats.txt to ensure data integrity before proceeding with event generation and analysis.

- **Missing Event Entries**:
  - If an event is present in one file but missing in the other, it is flagged as an inconsistency.
- **Type Mismatches**:
  - **Discrete Events**: Discrete events are expected to have integer values for mean in Stats.txt. If a non-integer mean is detected, it is flagged.
  - **Continuous Events**: Continuous events should allow decimal values for mean, aligning with their non-discrete nature.
- **Invalid Ranges**:
  - The program ensures that each event's min and max values are consistent. For instance, a min that exceeds max would be flagged.

1.3 Example Output for Potential Inconsistencies

1. **If the 'Logins' Line in the Stats.txt mean is a decimal value instead.**

   For example :
   ```
   Logins:4.5:1.5:
   ```

   An error output will show this:
   ```
   Inconsistencies found: ["Discrete event 'Logins' has non-integer mean in Stats.txt."]
   ```

2. **If there is a missing entry, where "Emails sent:10:3:" line is remove**

   An error output will show this:
   ```
   Inconsistencies found: ["Event 'Emails sent' in Events.txt missing in Stats.txt."]
   ```

3. **If there is a mismatch for a continuous event,**

   For example :
   ```
   Time online:150:25.00:
   ```

   An error output will show this:
   ```
   Inconsistencies found: ["Continuous event 'Time online' has integer mean in Stats.txt."]
   ```

2 Section 2: Activity Engine and Logs

2.1 Event Generation Process

Statistical Consistency:
Each event is generated to approximate its specified mean and standard deviation in Stats.txt. This ensures that the generated logs reflect realistic patterns that would be expected under normal conditions.

Handling Discrete and Continuous Events

- **Discrete Events**: For events marked as discrete (D), values are generated as integers, representing counts or occurrences. These values are based on a normal (Gaussian) distribution centered around the mean, with variation defined by the standard deviation.
- **Continuous Events**: For continuous events (C), values are generated as decimal numbers, reflecting continuous measurement values (e.g., time spent online). Continuous values are rounded to two decimal places for readability.

2.2 Log File Structure

File Name: The log file is named event_logs.json.
Format: The JSON format was chosen because it is both human-readable and easy to parse in Python, making it an effective choice for storing and retrieving structured event data.

The event_logs.json Structure

```
[
  {
    "Logins": 4,
    "Time online": 146.86,
    "Emails sent": 5,
    "Emails opened": 6,
    "Emails deleted": 4
  },
  {
    "Logins": 5,
    "Time online": 132.21,
    "Emails sent": 9,
    "Emails opened": 12,
    "Emails deleted": 6
  },
….
]
```

3 Section 3: Analysis Engine

3.1 Analysis processing:

This document contains the analysis results generated by the Analysis Engine as part of the intrusion detection system (IDS) assignment. The engine computes daily totals, mean, and standard deviation for each event based on the provided logs.

File Information:

The analysis results are stored in the file 'analysis_results.json'. This file is formatted as JSON and contains the following data for each event:

- **Daily totals:** A list of values representing the total for each day.
- **Mean:** The average value across all days.
- **Standard deviation:** The variability in daily totals.

Structure Sample:

```
// Each contianers represent difference events
{
        'Event Name': {
                'daily_totals': [list of daily totals],
                'mean': calculated mean,
                'std_dev': calculated standard deviation
        }
}
```

Sample Analysis Results:

```
{
   "Logins": {
      "daily_totals": [2, 1, 3, 2, 4],
      "mean": 2.4,
      "std_dev": 1.02
        },
   "Time online": {
      "daily_totals": [130.87, 144.42, 160.4, 107.46, 146.72],
      "mean": 137.97,
      "std_dev": 17.9
        },
   "Emails sent": {
      "daily_totals": [11, 9, 15, 10, 9],
      "mean": 10.8,
      "std_dev": 2.23
        },
    ....
 }
```

4 Section 4: Alert Engine

Introduction

The alert engine monitors and detects anomalies in simulated event data by comparing it to baseline statistics. It consists of:

1. Input Setup: Reading 'Events.txt' and 'Stats.txt'.
2. Data Generation: Simulating event logs for the specified number of days.
3. Anomaly Detection: Comparing daily event data to thresholds based on event weights and deviations.

4.1 Input Files

Events.txt:
Describes the events, their types, and constraints (min, max, weights).
Example:

```
5
Logins:D:0::2:
Time online:C:0:1440:3:
Emails sent:D:0::1:
Emails opened:D:0::1:
Emails deleted:D:0::2:
```

Stats.txt:
Provides statistical parameters (mean, standard deviation) for each event.
Example:

```
5
Logins:5:2.0
Time online:160.0:20.0
Emails sent:9:3.5
Emails opened:11:4.0
Emails deleted:6:2.5
```

4.2 Processing Details

Simulated Log Data:

– Generated data for 5 days using Gaussian distribution.
– Ensured values respected event constraints (min, max).

Anomaly Detection:

– Threshold: Threshold = 2 × Sum of Event Weights = 18.
– Daily Anomaly Counter calculated as: Deviation (std dev) × Weight.

4.3 Outputs

Simulated Data:
Saved in 'log_data.json'.
Example:

```
[
   {"Logins": 4, "Time online": 132.12, "Emails sent": 5, "Emails opened": 7,
"Emails deleted": 3},
   {"Logins": 6, "Time online": 142.24, "Emails sent": 9, "Emails opened": 10,
"Emails deleted": 5},
   ...
]
```

Anomaly Report:
Saved in 'anomaly_report.json'.
Example:

```
[
   {"day": 1, "anomaly_counter": 6.60, "threshold": 18, "status": "okay"},
   {"day": 2, "anomaly_counter": 4.43, "threshold": 18, "status": "okay"},
   ...
]
```

4.4 Results

– All 5 days were classified as 'okay' since their anomaly counters were below the threshold.

4.5 Conclusion

The alert engine functioned as expected:

1. Successfully simulated event data and detected anomalies.
2. Produced human-readable outputs for logs and anomaly reports.
3. Provided user interaction for processing additional files.