Report on System Security Assignment 3: Event Modeler and Intrusion Detection System

1. Introduction

The purpose of this assignment is to design and implement an event modeler and intrusion detection system (IDS) that monitors, analyzes, and detects anomalies in email system events. This IDS utilizes statistical methods to calculate thresholds for anomaly detection and generates alerts based on deviations from expected patterns. The system is coded in Python, using JSON files for log management.

2. Initial input

Internal Storage of Events and Statistics

The system reads event specifications from Events.txt and statistical parameters from Stats.txt, storing them in distinct data structures for efficient retrieval:

- **Events Storage:** Each event entry includes the fields name, type, min, max, and weight, distinguishing discrete (D) and continuous (C) events. These entries are stored in a list of dictionaries:

```
events = [
    {'name': 'LoginAttempt', 'type': 'D', 'min': 1, 'max': 5,
'weight': 3},
    {'name': 'DataAccess', 'type': 'C', 'min': 0.5, 'max': 10.0,
'weight': 5},
    # Additional events
]
```

- **Statistics Storage:** The mean and standard deviation for each event, as specified in `Stats.txt`, are stored in a dictionary, with each event's name serving as the key. This allows quick access to statistical parameters during event generation:

```
stats = {
    'LoginAttempt': {'mean': 3.0, 'std_dev': 1.0},
    'DataAccess': {'mean': 7.0, 'std_dev': 1.5},
    # Additional statistics
}
```

Potential Inconsistencies Between Events.txt and Stats.txt

Inconsistencies between Events.txt and Stats.txt may include:

- Missing statistical entries in Stats.txt for events listed in Events.txt.
- Extra statistical entries in Stats.txt that have no corresponding event in Events.txt.

Implementation of check_consistency function identifies missing statistics by comparing the list of event names against the stats dictionary keys. However, it does not yet detect extra entries in Stats.txt without an event in Events.txt. This may be an area to address in future iterations of the system.

Data Parsing Modules

- parse_events(file_path): Reads Events.txt to extract the event definitions, including event type (Discrete or Continuous), min and max values, and weights.
- parse_stats(file_path): Reads Stats.txt to gather statistical parameters (mean and standard deviation) for each event.
- check_consistency(events, stats): Ensures that each event in Events.txt has a corresponding entry in Stats.txt, flagging any inconsistencies.

3. Activity Engine

Event Generation Process

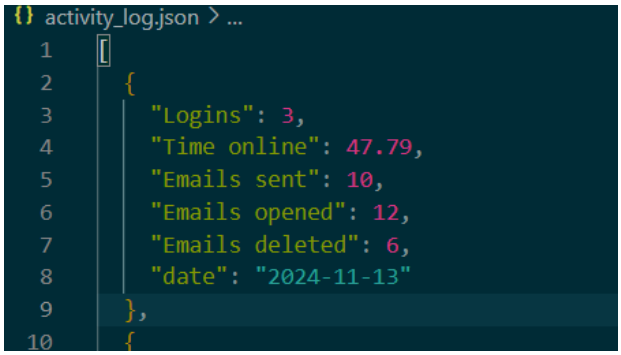
The activity engine simulates daily event occurrences with a process that distinguishes between discrete and continuous events:

1. **Discrete Events (D):** For events marked as discrete, values are generated around the specified mean using a Gaussian distribution, and the output is rounded to whole numbers to reflect countable occurrences.
2. **Continuous Events (C):** Continuous events are generated with Gaussian distribution values rounded to two decimal places, representing measurements like data access volume or session duration.

The system generates random event values by applying the random.gauss(mean, std_dev) function for each event, closely matching the specified statistics to model realistic variations in event frequencies and values.

Log File Structure

- **File Name and Format:** The generated event log is stored in activity_log.json. JSON is chosen for its structured, human-readable format, which also facilitates efficient parsing in the analysis phase.
- **Log Structure:** Each log entry includes a date and records of all generated events and their values for that day:



This format allows human-readable inspection and supports structured data access for analysis.

Activity Modules:

- generate_event_value(event, stats): Uses a Gaussian distribution to create random values for each event, differentiating between discrete and continuous event types.
- generate_daily_log(events, stats): Generates daily entries by iterating through all events and generating values.
- run_activity_engine(events, stats, num_days): Creates a log file (activity_log.json) with event data over a specified number of days, generating new entries each day.

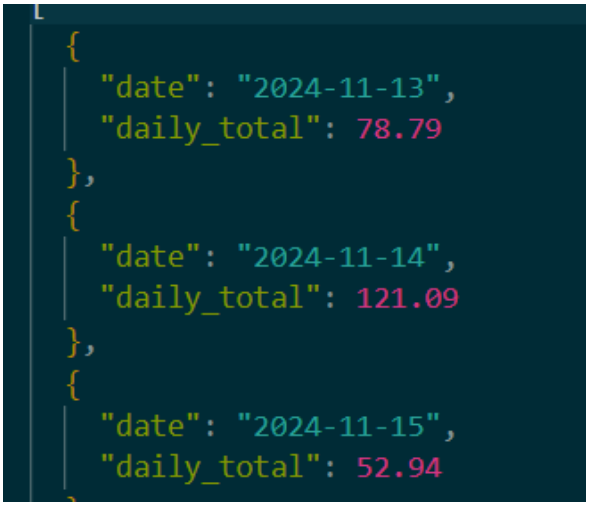
4. Analysis Engine

Daily Totals and Statistical Data Storage

During analysis, the system generates additional data files to support anomaly detection:

1. **Baseline Statistics File (baseline_stats.json):** This file contains the mean and standard deviation for each event based on daily log data. These baseline statistics provide reference points for anomaly detection:

2. **Daily Totals File (DailyTotals.json):** This file records the sum of all event values for each day, enabling tracking of daily activity levels:



These file formats provide a clear structure for statistical and daily total data, supporting efficient anomaly detection based on variations from established patterns.

Analysis Modules:

- calculate_baseline_stats(logs): Computes the mean and standard deviation for each event from generated logs.
- calculate_and_save_daily_totals(logs, output_file): Calculates the daily sum of event values to monitor fluctuations over time, saving results to DailyTotals.json.
- save_baseline_stats(baseline_stats, filename): Exports baseline statistics to baseline_stats.json for anomaly detection.

5. Alert Engine

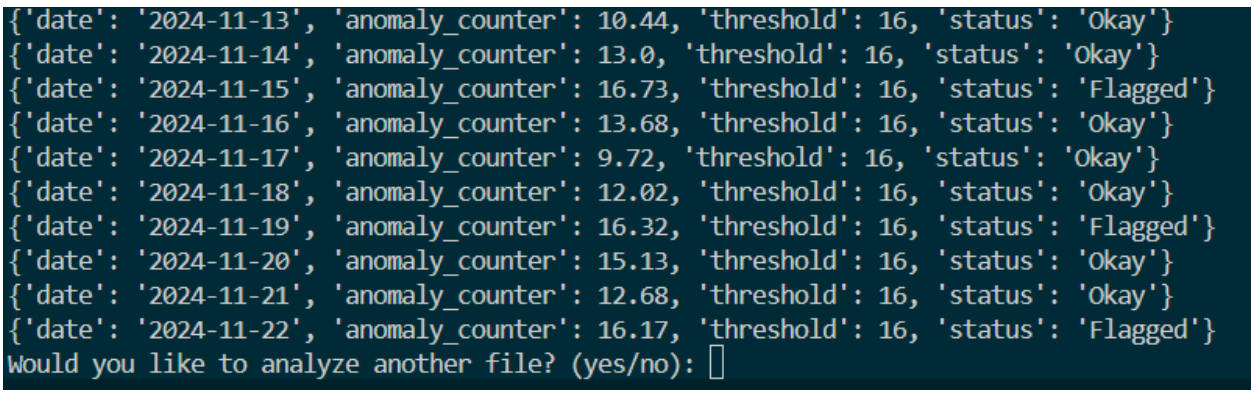
System Workflow and Execution

The main function integrates each module, creating an end-to-end system for event monitoring:

- **Step 1**: Collects user input for statistics file and the desired number of days for activity generation.
- **Step 2**: Loads events and statistics, checks for consistency, and starts the activity engine to generate logs.
- **Step 3**: Baseline statistics are calculated, and daily totals are saved.
- **Step 4**: The alert engine evaluates each log for anomalies, generating an alert status based on thresholds.

Anomaly Detection and Analysis

- **Statistical Method**: The IDS relies on z-scores, calculated as deviations from the mean divided by standard deviation. This allows the system to detect significant anomalies while ignoring normal variations.
- **Weighted Anomaly Counter**: Each event's deviation is weighted, enhancing the model's sensitivity to critical events. The threshold, set at twice the sum of all event weights, balances detection sensitivity and robustness.
- **Sample Output**: The system prints alerts with dates, anomaly counters, thresholds, and alert statuses. Below is an example output format:



Alert Engine Modules:

- calculate_anomaly_counter(daily_log, baseline_stats, events): Calculates deviations from baseline statistics for each event in a daily log, weighted by event importance.
- detect_anomalies(logs, baseline_stats, events): Flags days with anomaly counters that exceed the defined threshold as "Flagged" for review.

6. Conclusion

This IDS successfully models and monitors event data, utilizing statistical analysis to detect unusual activity in a simulated email environment. The system's modular design allows for flexibility in configuration and scalability to larger datasets or additional events.