Assignment 3: System Security

Hjorvar Logi Ingvarsson & Marit Rudlang

Overview

As specified in the assignment description the program consists of four parts: Initial input, activity engine, analysis engine and an alert engine. The initial input, activity engine, and analysis engine all relate to the Honeypot event modeller. The Alert Engine uses the events modelled in the honeypot to detect and alert if there is a plausible intrusion. An flag is set if the data is out of bounds of a given threshold. The following sections describe the separate parts in more detail.

Initial Input

For holding the data in the input files events.txt and stats.txt we have two simple classes: Event and Stat that have fields for every type of data defined in the document. We take the text files apart with split and store the values in two ArrayLists, one of them for Event objects and one for Stat objects. We store all numbers as doubles in the program (except weight), but operate on them. The event objects has boolean fields for turning off the bounds (min and max) when they're not specified. The Stat object is just name, standard deviation and mean (String, double, double).

For checking the consistency we are doing a run through of the lists to check that the size (eg the number of events and the number of stats) is the same, and that they have the same event names in both the files. If an inconsistency is found it is printed, and the program is aborted.

Inconsistencies such as the min and max values specified by Stats.txt being outside of the scope of the normal distribution are handled by stopping the generation of data after a certain amount of tries (it tries a million times to get a value in the specified zone before giving up).

Activity Engine and the Logs

The events are generated by creating a random object and asking for Gaussian values (random number following a normal distribution with a mean 0 and standard deviation of 1) and scaling them to have whatever mean and deviation is specified in Stats.txt for that event.

The actual logs are generated on a per day basis. We generate event entries sequentially but give them a random timestamp. The events that are of type E or C get a random magnitude until the sum of the magnitudes is equal to the previously generated total. These entries are stored in an ArrayList that then gets sorted with respect to time, thereby randomizing the order of events, and printed to a file called Day#.txt (# is replaced with the number of the day).

This ensures that the values are random but if we take an infinite sample it will follow the distribution specified in Stats.txt. There is a weakness in the implementation relating to how we split the total into separate events. Since it's difficult to make sure the values will be believable, don't follow a pattern and that the total gets split into enough parts for the log to look real. There may also be some rounding errors making the sum of the parts less than the total.

The data is written to Days.txt as it is generated, the format of Days.txt is simple:

- The first line is the names of the events separated by " : "
- The following lines start with "Day#: " and then have the frequencies of the events in their respective order separated by " : ".

Note: the hashtag is replaced with the number of the day in question (starting with 1).

In order to get the data back we split with ": " then strip whitespace and convert to numbers. The whitespace is to help with human readability.

Analysis Engine

The analysis engine uses the total values that were created in the activity engine and stored in the Days.txt. Since the document only contains the total values, and the log values are stored in a separate log file it is virtually impossible to pick up log reading anomalies at this stage, and it is improbable that they are apparent as the log values are generated from the total values, and not the other way around.

It reads through the days.txt file and calculates the mean and standard deviation on each event. This information is then stored as a Stat-object (since the data we're asked to calculate corresponds to a Stat object) and saved in an ArrayList. The data can easily be written to a file if assessment by a human is needed but the program itself handles it with an ArrayList.

We haven't thought of any possible statistical anomalies to check for(?).

Alert Engine

The program is split into two parts, the preprocessing part and the interactive part. The alert engine is pretty much analogous to the interactive part. The preprocessing part takes the inputs from the user creates the base data then works out the mean and standard deviation for each event in that data and returns it to the interactive part.

The interactive part (or the 'alert engine') asks if the user wants to stop the program or input a new statistics file. If the user enters an invalid filename the program exits with an exception, if the input has too few parameters it keeps asking for new input until the input has two arguments. Then it starts generating events based on that file, at the end of each day it will run a check to see if the sum of “distances” is greater than the threshold (double the sum of weights).

The distance for each event is defined as:

Weight * |Observed_Frequency - Mean| / Standard_Deviation

If the check is true then that day is flagged by the alert engine; if not, nothing happens. Either way the program keeps running until it finishes its workload and the user doesn’t want to continue.

Output

Upon completion the program has generated one file for each day with logging information from the honeypot and a Days.txt file containing the statistical information generated from the logs.

As the program finishes after the user has given input, it prints a flag for any data generated that seems suspicious, as well as the number of the day that data was generated.