GroupWork: Samuel, Celine, Yan Ying

School of Computing & Information Technology

CSCI262 System Security SIM-2020-S4

Standard Deviation
ML
Training

Assignment 3 (14 marks, worth 14%)

Due 18 Nov 2020 20:55 (SG Time)

The groups

Students should form into groups of 2 or 3. One person from each group can email the tutor Mr. Sionggo Japit (sjapit@uow.edu.au) a list of the group members. `Please cc in the other student or students in your group`. Each group needs to submit the code and a report. Only one student from each group needs to submit. Submission is via Moodle.

Overview

Each group needs to design and implement an Email system event modeller & intrusion detection system in accordance with the system descriptions below. The implementation is to be in C, C++ or Java. While there are concrete details on the form of the initial input, and certain inputs along the way, the format of intermediate data is up to each group.

You need to provide a report in a file `Report.pdf` covering the various points through this assignment where information is required. This report should be broken into sections associated with the components as follows:

- Initial input.
- Activity engine and the logs.
- Analysis engine.
- Alert engine.

java a3 IDS Events.txt Stats.txt 10

Initial Input

You only need command line options at the setup phase, some user input is required later.

```
IDS Events.txt Stats.txt Days
```
we supply in the days we want

`Events.txt` and `Stats.txt` define the formats and the distributions of the events to be modelled. `Days` is an integer used in the next section.

Here goes an example `Events.txt` file. This file describes the events and some of their parameters.

```
5
Logins:D:0::3:
Time online:C:0:1440:2:          Events.txt
Emails sent:D:0::1:
Emails opened:D:0::1:
Emails deteled:D:0::2:
```

The first line contains the number of events being monitored. Each subsequent line is of the form

```
Event name:[CD]:minimum:maximum:weight:
```

C and D represent continuous and discrete events respectively. Discrete events must take integer values and occur one at a time, continuous events don't need to take an integer value and an occurrence of that event may be of any value. The minimum and maximum specify the allowed range for that event type across a day. Continuous events need to be recorded to two decimal places. The weights are used in the alert engine and will always be positive integers.

The file `Stats.txt` contains the distributions to be modelled for the events. Here goes an example `Stats.txt` file.

```
5  represent the number of record
Logins:4:1.5:
Time online:150.5:25.00:          Stats.txt
Emails sent:10:3:
Emails opened:12:4.5:
Emails deteled:7:2.25:
```

The first line again contains the number of events being monitored. Each subsequent line is of the form

```
Event name:mean:standard deviation:
```

Your program should appropriately report events and statistics read in, as evidence this phase works. You should include in your report a description of:

1. How you are going to store the events and statistics internally.

2. Potential inconsistencies between `Events.txt` and `Stats.txt`. You should attempt to detect those inconsistencies. If there are inconsistencies you are aware of but haven't attempted to detect them, note this in your report.

Activity Simulation Engine and the Logs

Once the intial setup has taken place, and you have read in the base files, the activity engine should start generating and logging events. Your program should give some indication as to what is happening, without being verbose.

Need print out the statement

Produce another stats.txt approxiametly consistent with the statistic specied in the given stats.txt file

Baseline_Stats is the combination of both stats and the event files

You are attempting to produce statistics approximately consistent with the statistics specified in the file `Stats.txt`. You should log for the number of `Days` specified at the initial running of `IDS`. You can, if you like, store the events in distinct files for each day, or in a single log file. This collection of events forms the baseline data for the system.

You should include in your report a description of:

What is this?

1. The process used to generate events approximately consistent with the particular distribution. This is likely to differ between discrete and continuous events.

2. The name and format of the log file, with justification for the format. You will need to be able to read the log entries for subsequent parts of the program. The log file needs to be human readable.

Analysis Engine

Store totals for each event (per day) in data file.

Your program should indicate it has completed event generation and is going to begin analyis. You can now measure that baseline data for the events and determine the statistics associated with the baseline.

Produce totals for each event for each day, store that in a data file, and determine the mean and standard deviation associated with that event across that data. Report what is happening as you consider appropriate.

Even if you are unable to produce data consistent with a given distribution you can still have the analysis engine reading and reporting on the log file.

You should include in your report the name and format of the file containing the daily totals and statistical data for the events.

Name and the format of the file

Alert Engine

32.jpg from screen shot note

The alert engine is used to check consistency between "live data" and the base line statistics. Once this phase is reached you should prompt the user for a file, containing new statistics, and a number of days. The new statistics file has the same format as `Stats.txt` from earlier but will generally have different parameters for the events. You should run your activity engine and produce data for the number of days specified. Use the analysis engine to produce daily totals, those are used in alert detection.

For each day generated you need to report on whether the there is an intrusion detected by comparing an anomaly counter with a threshold. You calcualte the anomaly counter by adding up the weighted number of standard deviations each specific tested event value is from the mean for that event, where the standard deviation and mean are those you have generated from the base data and reported, and the weight is taken from the original `Events.txt` file.

For example, if the mean number of logins per day is 4 and the standard deviation is 1.5; then if we get 1 login in a day we are 2 standard deviations from the mean. Referrring back to the weight of the login event we see it was 2 so the login event contributes 4 to our overall anomaly counter.

The threshold for detecting an intrusion is 2∗(Sums of weights) where the weights are taken from `Events.txt`. If the anomaly counter is greater or equal to the threshold you should report this as an anomaly.

You should output the threshold, and give the anomaly counter for each day as well as stating each day as okay or flagged as having an alert detected.

T_T

Once the alert engine part has finished you should return to the start of this phase, so another set of statistics and number of days can be considered. An option to quit should be provided.

Notes on submission

Things to submit:
- Report.PDF
- source code
- readme.txt

1. Submission is via Moodle. Everything will need to be uploaded in a single zip file. Please don't put subdirectories in your submission. In addition to addressing the specified points, you can include anything of significance in your report. You should report on any parts not completed.

2. One person from each group needs to make a submission. If multiple people make submissions the last one received will be marked.

3. Include the compilation instructions with your submission (i.e., provide a readme.txt file).

4. Late submissions will be marked with a 25% deduction for each day, including days over the weekend.

5. Submissions more than three days late will not be marked, unless an extension has been granted.

6. If you need an extension apply through SOLS, if possible **before** the assignment deadline.

7. Plagiarism is treated seriously. Students involved will likely receive zero.

University of Wollongong

School of Computing and Information Technology

CSIT314 Software Development Methodologies

SIM S4 2020

Group Project (40 marks)

TASKS

Your tasks are to:

1. Carefully read this document;
2. Form and structure your group, allocating roles and responsibilities to your members. **Make sure you communicate frequently and contribute significantly to your group work**. If you have little contribution to your group project, your individual mark will be significantly lower than your group's mark (or even 0 mark if you have almost no contribution).
3. Complete the development of a given software system (enclosed in this document). This should cover **all software development activities** from requirements elicitation and specification, design to implementation and testing. Your design must be based on the **b-c-e framework** discussed in the subject.
4. Choose an agile method (Scrum is highly commended) for your group to follow. Choose a tool which supports that methodology for your group to use. **Taiga https://taiga.io/ for agile method is strongly recommended** (choose the public option which allows multiple team members).
5. Demonstrate the practice of test-driven development and Continuous Integration/Continuous Delivery (CI/CD) in your project.
6. Produce a report detailing the group's work.

SUBMISSION

1. Final deliverable (35 marks): **20th November 2020**
   - Final report
   - A 15 minute recorded video of a live demo of your product
   - Source code
2. Final presentation (5 marks): **The last lab session**

All submission must be made to Moodle by one member of your group by the deadline.

GUIDLINES

1. The **final report** should cover at least the following:
   - A complete list of user stories.
   - For each user story, a list of complete tasks.
   - A complete and detailed design including UML **use case diagrams**, **use case descriptions, sequence diagrams, class diagrams, data persistence and user-interface aspects.**
   - Sufficient evidence (with screenshots and detailed text description/explanation) to demonstrate that your group has followed an agile methodology and has used a tool which supports the methodology

from the beginning to the end of the project. Note: ***make sure that you record these evidence every week (e.g. screenshots of the product backlog, sprints, the work-in-progress software system, etc.)***

- Test plans, test cases (include **unit test cases**), and test data that is sufficiently large enough to simulate the scale of the developed system. Details of unit testing procedures that have been conducted to clearly demonstrate (with sufficient evidence) that your group has followed test-driven development

- Sufficient evidence (with screenshots and detailed text description/explanation) to demonstrate the use of CI/CD in your project (i.e. the development and deployment of **at least** one feature/functionality).

- Identify and discuss ethical considerations in developing the software system in this project and how your team have addressed them.

- Identify a feature of your software application that can be developed using data-driven software development. Present a detailed plan of how this feature would be developed and integrated into your software application using the data-driven approach discussed in the subject.

- True group meeting records: agendas and meeting minutes which includes at least the following: meeting date, attendance, progress reports, review and tracking (e.g. snapshots of Gantt chart tracking or backlogs, etc.), discussion summaries, and action plans/items.

- Member contribution for the whole project (with each member's signature):
  - On the cover page of your progress/mid-project/final report, you need to provide **rating** for the contribution of each team member and a **detailed explanation of what the team member did for the project** to justify the rating (e.g. the roles that they filled, the tasks they completed, and the artefacts that they successfully delivered).
  - Everyone in the team should sign the cover page. The individual contribution of each team member is assessed by all the other members.
  - The rating scale can be a percentage number (e.g. 60%). Alternatively, the scale be in the form of "contributed", "very little", and "almost no contribution". For a team member who has "contributed", he/she will receive 100% of the group mark; for a team member who contributed "very little", he/she will receive 50% of the team mark; for students who made "almost no contribution", he/she will receive 0 marks for the entire group project. Your tutor/lecturer may make adjustment to this marking criterion based on practical situations.

A suggested plan:

- The first week after the lectures: finalize group and start working on the project. Produce the first complete version of the requirements.
- One **iteration/sprint per week** until the end of project. In each iteration:
  - Design, implement, and test a number of functionalities;
  - **Demonstrate the working system to the clients (i.e. tutors) to receive feedback during the second half of weekly labs.** ***Weekly progress will be observed and noted by the tutors and will contribute to the project management component of your project marks. Check the marking scheme in the last page for details.***
  - Continue eliciting and clarifying further requirements.

The project description is in the next page.

Project Description

Important Notes:

- *This Project Description provides only the **high-level goals** of this project. The development team **MUST elicit more detailed and specific requirements AND get feedback from "the client"** (the tutor).*
- ***The tutors will note your group's progress and interaction with the "client" since they are one of the factors contributing to the final marks.***
- ***The requirements may change during the course of the project (this is to simulate real-life projects).***
- ***At least the backend/middleware of your software product (i.e. the main code that controls/runs all application logic and hold data in memory) needs to be object oriented.***
- ***You need to form a group of 6-7 people in your same lab ASAP and register your group with your tutor (see below):***
    - ***Full time cohort: Terence Chew (tchew@uow.edu.au)***
    - ***Part time cohort: Kheng Teck Tan (ktan@uow.edu.au)***

You are asked to develop a bug-tracking system using **object-oriented design and implementation**. Using this system, users can report bugs, triage bugs, and manage a bug's lifecycle (see https://bugzilla.mozilla.org/show_bug.cgi?id=1429672 for an example).

This system has different user types: bug reporters, triagers, developers, and reviewers. Reporters here can be end-users, testers, and any other users. Developers are those assigned to fix a bug, and hence their expertise and experience (e.g. the bugs that have been fixed by them) are important. Reviewers check if a bug has been fixed with the patches provided by the developer and close it accordingly. Triager is responsible for managing all quality aspects of a bug, such as assigning it to a suitable developer, checking if it is a duplicate or invalid bug. All of these must be kept and managed by the system.

Throughout the lifecycle of a bug, users can participate in discussing how to fix it (by providing comments). User can also search for bugs through keywords, titles, assignee, etc. The system can also generate various reports such as the number of bugs reported in a month, the number of bugs resolved in a week and the best performed reporters or developers.

You must create test data that is sufficiently large enough to simulate the system (e.g. at minimum 50 users and 500 bug reports). You could write a script to randomly generate these data.

The marking scheme is in the next page for your reference.

Marking scheme

Component	*Out of*	*Marks*	*Comments*
Final Project Presentation (Recoded demo video + Q&A session)	*5*		
Final Deliverables			
Overall quality of the Final Deliverables	*2*		
User stories and tasks	*8*		
Analysis and Design (use cases, detailed design models, consistency between design models, consistency between design with code, etc.)	*9*		
Implementation (quality of code, functionalities implemented, sophistication of the solutions, consistency with design, etc.)	*6*		
Test-driven development	*2*		
CI/CD	*2*		
Ethical consideration and discussions	*2*		
Data-driven development	*2*		
Effective use of methodologies (e.g. evidence of the use of a methodology and tool support, true meeting records, **weekly progress as observed and noted in the labs**, etc.)	*2*		
Total	***40***		