Assignment 1 – COEN 346
Deadline: October 11 by 11:59pm
Weight: 8%

Submission instructions:

- Copy all Java files and your report file in a folder and put your student id(s)_(assignment number) as the folder name.
- Zip the folder and Submit it via Moodle.
- If you are working in pair put the student id of both students as the name of the Zip file.

Multithreaded Zero-Day Attack

1- Introduction

Vulnerabilities in software can be found by hackers, security experts, or users. In this case, we have to patch software as soon as possible to protect users from any particular cybercrimes (e.g., stoles sensitive information, shutting the system down, etc.). Another scenario also occurs; the crackers will keep it hidden to sell on black auctions on the dark web. These people buy these vulnerabilities to design cyber-attacks on specific dates and times with perfect plans. Thus, the security experts do not have enough time to protect the system. This term, known as ”**Zero-day attack**.” If the only method to avoid zero-day assaults was to wait for the software maker to patch their program, you'd be waiting a long time. While there is no one **magic bullet** solution to protect your network from all zero-day vulnerabilities, there are certain things you and/or your organization can do to be secure in the future: strong antispam and antivirus security, which provide more costs.

2- Proposed Solution

Speeding up the search process for these system vulnerabilities can reduce the risk. In this case, we need to study the system behaviors. One of the most efficient methods that help to automatically analyze system behavior is logging. Logging is a common programming method that involves adding statements to source code to save critical runtime information. The numerous uses of logs in software system management activities, such as anomaly detection, fault debugging, performance diagnostics, workload modeling, and system behavior understanding, demonstrate the necessity of logging. It can be clearly seen that it is impossible to read and analyze logs to discover this complicated information manually. To this end, from your experience in course COEN 346, your task is to design and implement a parallel solution to read logs with multi-threading technique. You are supposed implement Java code with a multi-level of threads. This assignment has two levels/types of threads (Master and Workers), as explained in the following sections. The main goal is to search for specific vulnerability patterns on the given dataset. The dataset is one text files contains artificial logs. Each line inside the file represents a single log statement with the following attributes:

- Date and Time of task.
- Service Id
- Log statement with a length of 265 bytes

Here is an example of one line of the dataset file:

VM1 20210907_ 080513 AM Services_3757 Log:**ZY8BXAW5P2QZU9Q5P01PXNEI3PZ……..**

You need to use the supported Java file (LevenshteinDistance.Java) that contains the Levenshtein Distance algorithm. This algorithm helps with searching the logs for the possible **Vulnerability Pattern**. This Java class contains three functions and an attribute of type Boolean called ***acceptable_change***:

- **Calculate()** function: This function takes two string parameters with the same length and returns their differences as an integer.
- **Measure_Change_Ratio() :** This function measures the change ratio between the compared strings. If the similarity ratio between two strings is equal or bigger than 0.05 it sets the ***acceptable_change*** attribute to true.
- **isAcceptable_change():** This function returns the value of ***acceptable_change*** attribute

2-1 Master Thread

The master thread is your first launched threads (by main thread) that reads the textual file and loads its content into an array of strings. Each item of this array contains one line from the file. This task should be done on the class constructor handling IO exception if the file does not exist.

The Master Thread class has four attributes as follows:

- **Vulnerability Pattern**: This attribute stores the vulnerability pattern that we need to search for.
- **worker_number**: This is the number of running threads under the master thread. The initial value for this attribute is 2.
- **Count**: This is the number of detected vulnerabilities that is used for dynamically increasing the number of worker threads.
- **approximate_avg**: This is the average number of detected vulnerabilities (**Count**/number of lines in the file) and is used for increasing the number of launched worker threads; In each iteration, if the difference between the previous value of **approximate_average** and the updated one is more than 20% (0.2), the master thread increases the number of launched worker threads **(worker_number**) for the next iteration by 2 .

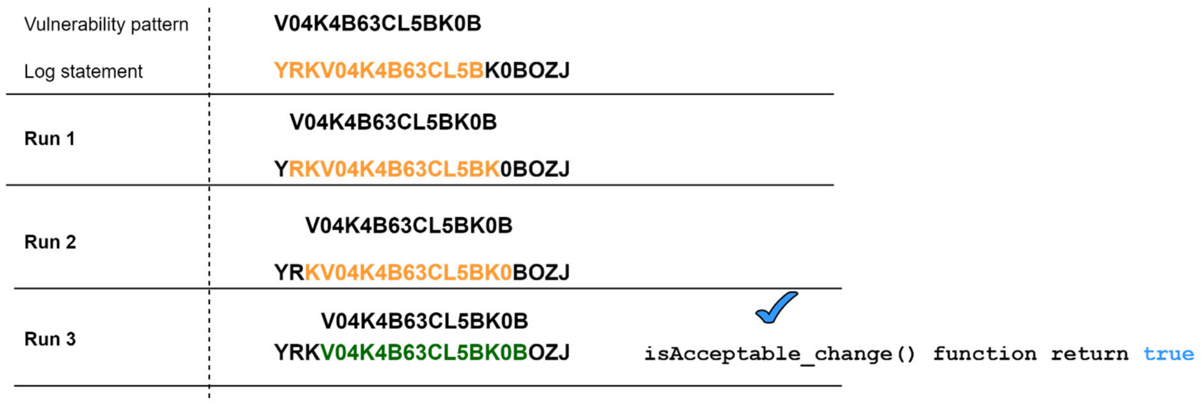
The master thread works in an iterative way. It starts by launching two worker threads under its control and starting from the beginning of the array, assigns one line (one item of the array) to each thread. Each of these worker threads searches its assigned log for the **Vulnerability Pattern** and if the pattern is found (if **acceptable_change** is true), it increases the value of **Count** by 1. More detail about the worker threads is available in the next section.

After these worker threads finished their tasks, the master thread calculates the updated value of **approximate_avg** based on the new value of **Count**, and if it is at least 20% more than the previous value of **approximate_ave,** it first sleeps for 2000 millisecond and then increases the number of workers by 2 for the next round (4 here) and so on. Otherwise, it launches the previous number of workers in the next iteration. ***Please note that this sleep time helps you to track the transition points where the number of launched worker threads is increase by 2***.

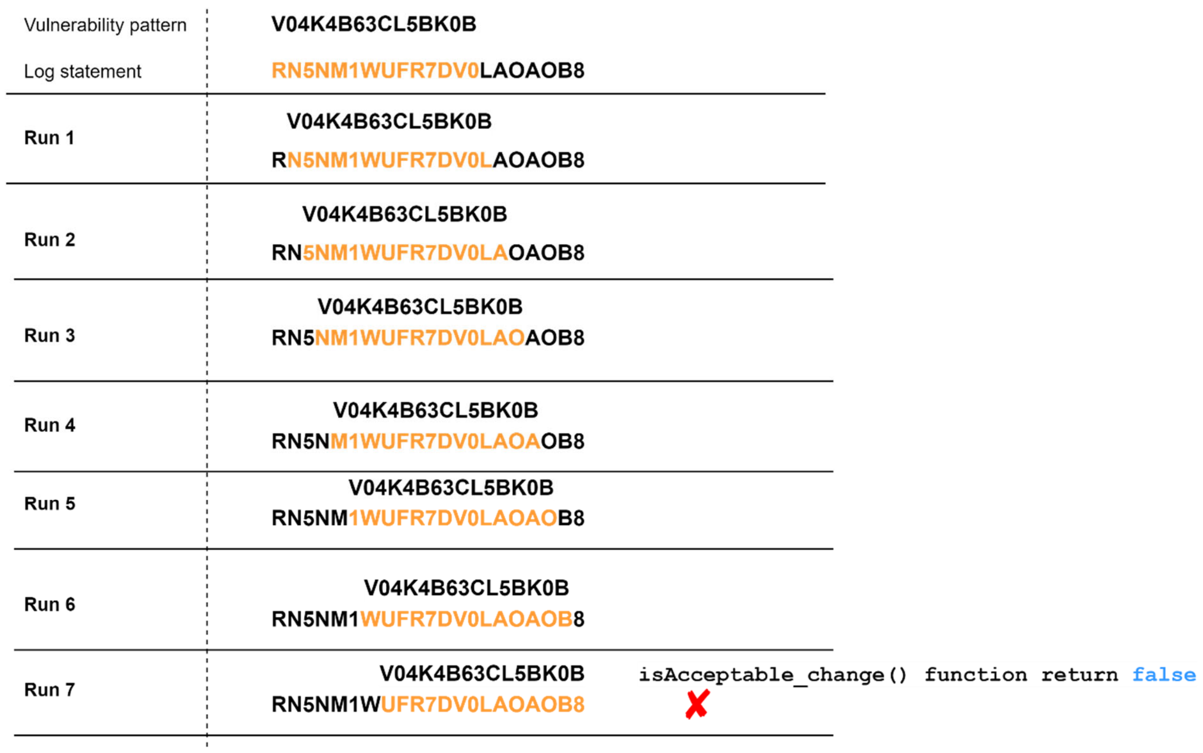
2-2 Worker Thread

The worker threads task is to search its assigned log statement for the possible similarity with the **Vulnerability Pattern** inside one log statement. To be able to that, the worker thread needs to start from the beginning of its assigned log and then recursively passes the substrings equal to the length of the Vulnerability Pattern to the levenshtein java functions to search the log statement for the Vulnerability Pattern. If at the end of the search process the value of **acceptable_change** is true, the worker thread must increase the number of detected vulnerabilities (**Count** attribute) of the master thread. (Note: multiple threads will access the shared attribute)

You can consider **Vulnerability Pattern** = **V04K4B63CL5BK0B** in your implementation. Here are two examples of successful and unsuccessful search processes for the worker thread:



Vulnerability pattern	V04K4B63CL5BK0B
Log statement	YRKV04K4B63CL5BK0BOZJ
Run 1	V04K4B63CL5BK0B YRKV04K4B63CL5BK0BOZJ
Run 2	V04K4B63CL5BK0B YRKV04K4B63CL5BK0BOZJ
Run 3	V04K4B63CL5BK0B ✓ YRKV04K4B63CL5BK0BOZJ isAcceptable_change() function return true



Vulnerability pattern	V04K4B63CL5BK0B
Log statement	RN5NM1WUFR7DV0LAOAOB8
Run 1	V04K4B63CL5BK0B RN5NM1WUFR7DV0LAOAOB8
Run 2	V04K4B63CL5BK0B RN5NM1WUFR7DV0LAOAOB8
Run 3	V04K4B63CL5BK0B RN5NM1WUFR7DV0LAOAOB8
Run 4	V04K4B63CL5BK0B RN5NM1WUFR7DV0LAOAOB8
Run 5	V04K4B63CL5BK0B RN5NM1WUFR7DV0LAOAOB8
Run 6	V04K4B63CL5BK0B RN5NM1WUFR7DV0LAOAOB8
Run 7	V04K4B63CL5BK0B isAcceptable_change() function return false RN5NM1WUFR7DV0LAOAOB8 ✗

3- Supported Files

You will have two package folders provided to you as follows:

1- **Support:** This folder contains **LevenshteinDistance.java** means the implementation of text similarity algorithm for two strings of the same length is already provided to you.
2- **DataSet:** This folder contains **vm 1.txt** which is the dataset of VM logs.

4- Submitted Documents

The assignment can be done in a group of two students. The deliverable consists of a **well-commented code** and at **least on page report specifying the high-level description of the code** (description of the methods/functions/threads and the flow of the program).You also need to explain the synchronization method that you used and also discuss the rationale behind it. This assignment will take 8% of your total mark for programming assignments. Also for this assignment 80% of the mark is dedicated to your code and 20% to your report.