Mamadou Diao Kaba (27070179)

Assignment Report 1

The vulnerability detection system is designed to identify potential security vulnerabilities within log statements using a multithreaded approach. The system consists of two main components: the MasterThread and WorkerThread classes. This report provides a high-level overview of the code structure, functionality of methods/functions/threads, and the flow of the program. Additionally, it discusses the synchronization method employed and the rationale behind it.

MasterThread Class:

Attributes:

Vulnerability Pattern: Stores the pattern to be searched for.

worker_number: Number of worker threads.

Count: Number of detected vulnerabilities.

Avg: Initial value of the approximate average.

approximate_avg: Average number of detected vulnerabilities.

Methods:

1. **run()**: This method is the main entry point for the master thread. It controls the flow of the program by calling other methods such as **startWorkers()**, **calculateApproximateAvg()**, and **adjustWorkers()** in an iterative manner.
2. **startWorkers()**: This method launches worker threads and assigns each thread a portion of the log statements to search for vulnerabilities. It iterates through the array of log statements and creates worker threads accordingly.
3. **updateCount()**: This method is responsible for updating the count of detected vulnerabilities. When a worker thread notifies the master thread about a detected vulnerability, this method increments the count in a thread-safe manner.
4. **calculateApproximateAvg()**: This method calculates the approximate average of detected vulnerabilities based on the current count and the total number of log statements processed so far.
5. **adjustWorkers()**: This method adjusts the number of worker threads based on the difference between the average and the approximate average. If the difference exceeds the threshold (20%), it increases the number of worker threads and updates the average.
6. **sleep()**: This method pauses the execution of the master thread for a specified period (2000 milliseconds) to allow time for worker threads to complete their tasks. This helps track the transition points where the number of launched worker threads is increased.

WorkerThread Class:

Attributes:

assignedLog: Assigned log statement to search for vulnerabilities.

Methods:

1. **run()**: This method is the main entry point for each worker thread. It executes the vulnerability detection process by calling other methods such as **searchVulnerability()** and **notifyMaster()**.
2. **searchVulnerability()**: This method searches for the vulnerability pattern within the assigned log statement. It iterates through the log statement and recursively passes substrings equal to the length of the vulnerability pattern to the **calculateLevenshteinDistance()** method.
3. **calculateLevenshteinDistance()**: This method calculates the Levenshtein distance between the substring and the vulnerability pattern using the provided **LevenshteinDistance** class. It determines whether the detected similarity is an acceptable change by comparing the change ratio to a predefined threshold (5%).
4. **notifyMaster()**: This method notifies the master thread when an acceptable change (i.e., a detected vulnerability) is found. It uses synchronization mechanisms to update the count of detected vulnerabilities in a thread-safe manner.

Program Flow:

1. The Main class initiates the vulnerability detection system by creating an instance of the MasterThread class and providing the file path and vulnerability pattern.
2. The MasterThread class loads the log statements from the file into an array of strings.
3. It iteratively launches worker threads and assigns each thread a portion of the log statements.
4. Each WorkerThread searches its assigned log statements for the vulnerability pattern using the Levenshtein Distance algorithm.
5. If a vulnerability pattern is found, the WorkerThread notifies the MasterThread about the detection.
6. After all worker threads complete their tasks, the MasterThread calculates the updated value of the approximate average.
7. Based on the difference between the average and the approximate average, the MasterThread adjusts the number of worker threads and sleeps if necessary.

Steps 3-7 are repeated iteratively until the termination condition is met.

Synchronization Method:

The synchronization method used in this system is based on thread-safe data access and coordination between the master and worker threads. To achieve this:

- The Count attribute in the MasterThread class is shared among worker threads. Synchronization is achieved by properly accessing and updating this attribute using synchronized methods.
- A mutex lock is applied when accessing shared resources to prevent race conditions and ensure data integrity.
- Atomic operations are used for updating shared attributes to ensure thread safety.

Rationale:

The chosen synchronization method ensures that the shared data (such as the count of detected vulnerabilities) is accessed and modified safely by multiple worker threads. By using synchronized methods and atomic operations, the system prevents data corruption and maintains consistency. Additionally, mutex locks prevent concurrent access to critical sections, reducing the likelihood of race conditions and ensuring the correctness of the vulnerability detection process.