Automated Management of Application Vulnerabilities

High Level Design

- **PROBLEM STATEMENT:**

  Currently there are several third party tools (sonarCube, BlackDuck, etc) that are used to detect vulnerabilities in a given project. The process of analyzing all the reports generated by these tools, and grouping similar vulnerabilities and checking if a vulnerability has already been reported and pushing them into JIRA as JIRA artifacts were all done manually, which in turn is a time-consuming and laborious process. The idea is to automate this process to help scrum team members quickly receive the vulnerability in an efficient manner. Thereby reducing the operation risk of the company.

- **Assumptions:**
  - ➔ Input of the vulnerability report is in the form of a CSV file (with dummy data).
  - ➔ Dealing with the following parameters in any vulnerability detected (as directed):
    ID, Date detected, Title, Desc, Location, Severity
  - ➔ Currently the application is built for one project, and one super user(as directed).
  - ➔ Only concerned with creation of bugs in Jira (as directed).
  - ➔ Assuming the due date for each Severity level (as directed)
    - ◆ LOW -> 30 days
    - ◆ MEDIUM -> 25 days
    - ◆ HIGH -> 20 days
    - ◆ EXTREME -> 15 days

- **PROBLEM ANALYSIS:**
  - When a user uploads a vulnerability report our app processes it and groups similar vulnerabilities based on their files of occurrence first then by their similarity (description)
  - Then the app creates a summary of the grouped vulnerabilities with the option of creating a jira artifact for each group(one artifact for each group)

- The App sets the due date of the respective artifacts based on the severity of the vulnerabilities in them, and also sets an estimated time required to complete the artifact based on the team's previous data.

- Can select a vulnerability tool configuration if he chooses to upload a new report.
- Can add a new configuration for report as well for that particular CSV file.
- The User gets a classified view of the vulnerabilities based on their description and location.
- Estimated time to complete can be generated by using previous data. (Can't be configured)
- Deadline to complete which can be configured by the user, based on severity is also generated.
- The user has the option to create and upload Jira artifacts.
- Reminder email is sent to the product owner based on the time.
- We can refresh our previous artifacts to update their status in our application.

- **High level design**

  ➔ **Flow Diagrams**

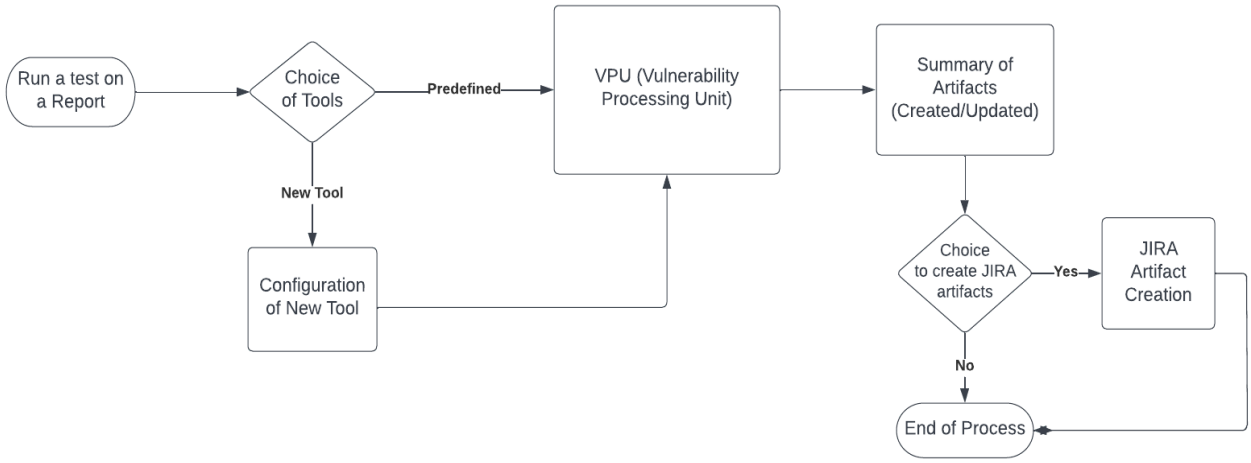

Fig.1: Main Flow Diagram

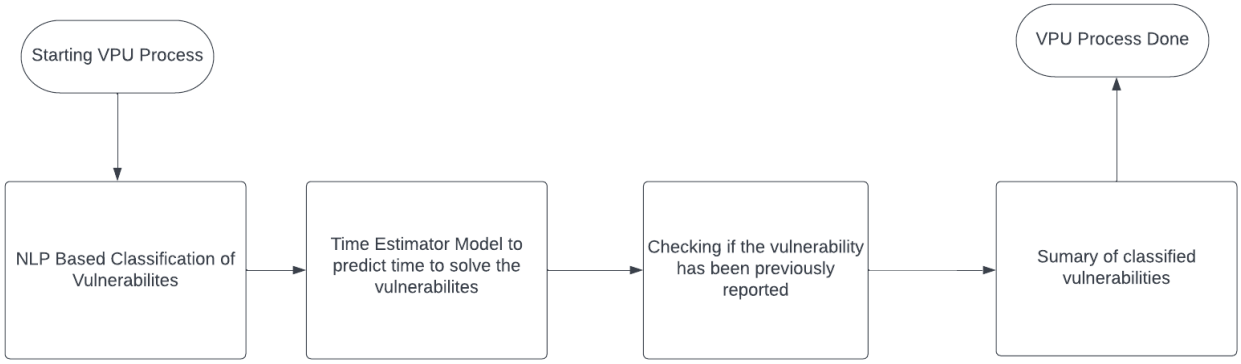

Fig.2: Vulnerability Processing Unit

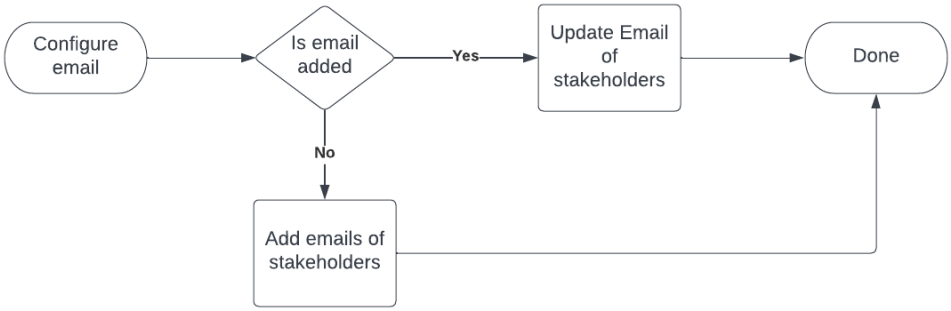

Fig.3: Email Configuration

➔ Tech Stack:

- Front End : ReactJS
- Backend : Django
- Database: PostGreSQL
- APIs: Jira REST API
- NLP/ML : BERT, Clustering Algorithms, Regression Models

➔ **Testing:**

Unit testing:

- **Frontend:** Proper functioning of the UI elements with JestJS
- **Backend:** API Testing using Postman,PyTest
- **NLP/ML:** The models shall be evaluated on test dataset using various metrics for eg. AUC-ROC score, F1-score etc.

Integration testing:

- Automated web testing of the entire application with all the units together can be implemented with pytest(or similar library).

➔ **Input Dataset:**

- The input dataset contains vulnerabilities, which have been extracted from the CVE (Common Vulnerability and Exposures) Dataset from kaggle (link).
- The data set has the following columns:

  (The location has been randomly generated as actual data was available)

ID	Title	Code	Description	Severity	Location
CVE-2013-1817	Information Exposure	89	MediaWiki before 1.19.4 and 1.20.x before 1.20.3 contains an error in the api.php script which allows remote attackers to obtain sensitive information.	MEDIUM	src/temp/file.php, line 31

- **Object Schema/Wireframes**

- **References:**

- https://developer.atlassian.com/server/jira/platform/rest-apis/
- https://djangopackages.org/grids/g/testing/
- https://www.kaggle.com/datasets/andrewkronser/cve-common-vulnerabilities-and-exposures