UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Informatica

Corso di Ingegneria, Gestione ed Evoluzione del Software

Final Report

TEAM MEMBER

Donia Daniele - 0522501575

La Marca Antonio - 0522501557

Somma Pasquale - 0522501543

Indice

CAPITOLO 1

Introduzione

Perseverance è un tool sviluppato in Python per analizzare repository utilizzando Pydriller, partendo da un dataset di progetti open source etichettati come vulnerabili o non vulnerabili. Il tool estrae dal codice Java, precedente ai commit di risoluzione delle vulnerabilità, tre diversi modelli di dati:

- **Text Mining**: crea un dizionario contenente le parole chiave presenti nel file e la loro frequenza all'interno del file stesso.
- **Metriche Software**: tramite il tool Understand, calcola nove metriche diverse per valutare la complessità del file.
- **Analisi Statica**: utilizza SonarQube per rilevare le vulnerabilità presenti nel file, con riferimento alle regole Java applicabili.

Questi modelli sono utilizzati per addestrare e valutare diversi classificatori di machine learning, tra cui Regressione Logistica, Naive Bayes, Support Vector Machine e Random Forest, che non sono inclusi nella versione originaria del sistema.

CAPITOLO 2

Esito delle modifiche

Il tool Perseverance ha subito significativi miglioramenti rispetto alla sua versione iniziale, concentrandosi sull'ottimizzazione per un utilizzo più intuitivo da parte dell'utente finale. Grazie all'introduzione dell'interfaccia con **CR8**, è stato possibile eliminare l'uso complesso di script da linea di comando, rendendo l'interazione con il tool semplice e diretta. È stata inoltre rimossa la dipendenza da strumenti esterni a pagamento come Understand: con l'implementazione di **CR5**, i calcoli sono ora eseguiti internamente al tool stesso. Parallelamente, la **CR6** ha introdotto la possibilità di comunicare automaticamente con altri strumenti come SonarQube, senza lcun lavoro da parte dell'utente. Tutto ciò, ha reso Perseverance più versatile e adattabile a vari dataset, senza vincoli su un'unica fonte di dati, per poter essere utilizzato da chiunque.

Un'altra innovazione significativa è l'integrazione di moduli di intelligenza artificiale, introdotti con la **CR7**, che permettono di predire la presenza di vulnerabilità all'interno dei file, sfruttando i dati estratti dal codice. La **CR4** ha migliorato la manutenibilità e l'organizzazione del tool, trasformando gli script in una struttura di classi organizzate, rendendo il codice più modulare e gestibile. Infine, le **CR1**, **CR2** e **CR3** hanno risolto errori critici che ne impedivano l'esecuzione, garantendo una maggiore affidabilità e stabilità del sistema.

CAPITOLO 3

Deviazioni da quanto pianificato

Durante l'esecuzione del tool, sono emerse alcune variazioni rispetto alla pianificazione iniziale che hanno richiesto un adattamento dell'approccio. Tali variazioni sono state rilevate a seguito di un test preliminare del tool.

Il problema principale riscontrato riguardava la presenza di bug nella lettura dei file, dovuti a caratteri speciali, spostamenti all'interno delle directory del progetto e l'assenza di alcuni controlli, che provocavano comportamenti imprevisti e, in rari casi, potenzialmente pericolosi. Inoltre, è emersa la necessità di integrare modelli di intelligenza artificiale, inizialmente non previsti, per consentire all'utente finale di utilizzare la principale funzionalità del tool: la previsione della presenza o assenza di vulnerabilità.

Nella versione aggiornata del sistema, sono stati mantenuti gli obiettivi originali considerati prioritari (**CR4**, **CR5**, **CR6** e **CR7**) e sono state aggiunte quattro nuove richieste (**CR1**, **CR2**, **CR3** e **CR8**), poiché la risoluzione dei bug e la fornitura di modelli di IA sono stati ritenuti fondamentali per il progresso del progetto.

CAPITOLO 4

Lessons Learned

Durante il processo di reengineering del sistema, abbiamo imparato molte lezioni preziose, in particolare:

- **L'importanza dei test**, che deve partire da un'ottima pianificazione, per individuare non solo possibili errori ma anche per comprendere a fondo il sistema in esame. É stata una parte fondamentale del nostro lavoro, se non quella che ha richiesto il maggior tempo e sforzo, dal momento che il sistema iniziale non includeva alcun tipo di test. È quindi essenziale riservare margini di tempo più ampi per i test, pianificando collaudi approfonditi e aggiornamenti dei casi di test per garantire la qualità e l'affidabilità del sistema nel lungo termine.

- Abbiamo potuto comprendere a fondo **l'importanza della manutenzione** dei sistemi software, sperimentandola direttamente durante l'analisi di un sistema che non avevamo sviluppato noi. Questo approccio ci ha permesso di approfondire le sfide e le necessità connesse alla comprensione e modifica di un codice preesistente, evidenziando l'importanza di una struttura organizzata e documentata per facilitare il lavoro di chi interviene successivamente.