UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Informatica

Corso di Ingegneria, Gestione ed Evoluzione del Software

Initial Report

TEAM MEMBER

Donia Daniele - 0522501575

La Marca Antonio - 0522501557

Somma Pasquale - 0522501543

Indice

CAPITOLO 1

Perseverance

1.1 Descrizione del tool

Perseverance è un tool sviluppato in Python che è in grado di minare repository tramite l'utilizzo di Pydriller, a partire da un dataset di progetti open source vulnerabili. Combina tre modelli per la predizione di vulnerabilità quali: metriche del software, mining del codice sorgente, analisi statica automatica.

Le performance (i.e., Precision, Recall e F1-score) di tali modelli sono valutate utilizzando i seguenti classificatori di machine learning: Regressione Logistica, Naive Bayes, Support Vector Machine, Random Forest.

1.2 Utilizzo del tool

Una volta clonato il progetto GitHub nel proprio workspace e installate le librerie necessarie (Python 3.4 o versioni successive e PyDriller 1.15), si può procedere con l'utilizzo del tool per la predizione del codice vulnerabile. Di seguito vengono descritte le diverse fasi operative.

1.2.1 Mining delle Repository

Per iniziare, bisogna navigare nella cartella Dataset2 > RepoMining ed eseguire lo script divide_Dataset.py. Successivamente, si deve eseguire il comando: python3 main_repo_mining.py.

Al termine di questa esecuzione, i risultati saranno disponibili nella cartella Dataset2 > mining_results.

1.2.2 Metriche del Software

Per ottenere il dataset delle metriche software, è stato utilizzato lo strumento Understand di SciTools. Questo strumento estrae varie metriche per ciascun file analizzato, come il numero di linee di codice, il numero di classi dichiarate, il numero di funzioni dichiarate, e diverse misure di complessità del codice. I risultati delle metriche software sono suddivisi in due dataset:

- mining_results_neg: contiene le metriche per i file non vulnerabili
- mining_results_pos: contiene le metriche per i file vulnerabili

È disponibile anche un dataset completo chiamato mining_results_sm_final.

1.2.3 Text Mining

Per eseguire l'analisi di text mining, bisogna navigare nella cartella Dataset2 > Text_Mining e seguire questi passaggi:

- Eseguire lo script text_mining.py
- Eseguire lo script dict_generator.py
- Eseguire lo script less_element_text_mining.py
- Eseguire lo script creator_csv_for_TextMining.py

Il risultato finale di questa analisi sarà disponibile nel file csv_mining_final.csv.

1.2.4 Analisi automatica

L'analisi statica automatica viene effettuata utilizzando SonarQube con il plugin CNESReport. Ogni file del dataset è analizzato rispetto a 19 regole specifiche. I risultati dell'analisi SonarQube sono salvati in due file:

- RepositoryMining_ASAResults_neg.csv: per i file non vulnerabili
- RepositoryMining_ASAResults_pos.csv: per i file vulnerabili

Per completare questa fase, bisogna eseguire i seguenti script nell'ordine indicato:

1. ASA_vulnerability_dict_generator.py
2. rules_dict_generator_ASA.py
3. creator_csv_for_ASA.py

Il file risultante sarà csv_ASA_final.csv.

1.2.5 Combinazione delle Tecniche

Una volta completate le singole analisi, è possibile combinare i risultati per migliorare le prestazioni del modello di predizione. Le combinazioni possibili sono le seguenti: text mining con metriche del software; text mining con analisi statica automatica; metriche del software con analisi statica automatica; combinazione completa tra text mining, metriche del software ed analisi statica automatica.

Per combinare il text mining con le metriche software bisogna navigare alla cartella dataset2 > Union > Union_TM_SM ed eseguire lo script Union.py. Il risultato sarà salvato nel file union_SM_TM.csv.

Per combinare il text mining con l'analisi statica automatica bisogna navigare alla cartella dataset2 > Union > Union_TM_ASA ed eseguire lo script Union_TMwithASA.py

Per combinare le metriche software con l'analisi statica automatica bisogna navigare alla cartella dataset2 > Union > Union_SM_ASA ed eseguire lo script Union_SMwithASA.py. Il risultato sarà salvato nel file union_SM_ASA.csv.

Per effettuare la combinazione completa, bisogna eseguire prima i passaggi di combinazione di text mining con analisi statica automatica. Successivamente bisogna navigare alla cartella dataset2 > Union > Total_Combination ed eseguire lo script 3Combination.py. Il risultato finale sarà salvato nel file 3Combination.csv.

1.3 Output

Come accennato nella sezione 1.2, i vari script eseguiti genereranno diversi output intermedi, che verranno utilizzati nelle fasi successive:

- **mining_results.csv** contiene i file .java corrispondenti alle BFIM (before image) delle classi introdotte o modificate con il commit risolutivo. Tali classi sono suddivise in base alla repository di appartenenza.

- **csv_mining_final.csv** contiene per ogni file .java analizzato, la frequenza di ogni parola chiave individuata e un'etichetta che indica se il file è soggetto a vulnerabilità o meno.
- **csv_ASA_final.csv**, tramite i risultati ottenuti da SonarQube, contiene per ogni file .java la frequenza delle varie vulnerabilità individuate tramite l'analisi statica.
- **union_SM_TM.csv** contiene per ogni file .java l'unione delle metriche software calcolcate e la frequenza delle parole chiave al loro interno.
- **union_SM_ASA.csv** contiene per ogni file .java l'unione delle metriche software calcolcate e dell'analisi statica con la frequenza delle varie vulnerabilità individuate.
- **3Combination.csv** contiene per ogni file .java l'unione delle metriche software calcolcate, dell'analisi statica con la frequenza delle varie vulnerabilità individuate e la frequenza delle parole chiave ottenute col text mining.

CAPITOLO 2

Comprensione del codice

Di seguito è riportato il contenuto e il ruolo principale dei pacchetti, visibili anche nel diagramma in Figura 2.4, che costiuiscono il sistema.

- Divided_dataset: contiene i file .csv con i link delle repository github e il relativo commit da analizzare. Il dataset iniziale è diviso in 35 file di 50 repository ciascuno.
- mining_result: contiene i file .java corrispondenti alle BFIM (before image) delle classi introdotte o modificate con il commit risolutivo. Tali classi sono suddivise in base al commit di appartenenza in cartelle nominate 'RepositoryMining', a cui si aggiunge il numero del file .csv corrispondente. Tali cartelle contengono anche il file 'ERROR.txt' che tiene traccia degli errori sorti durante il mining e 'CHECK.txt' che mostra un riepologo dell'attività di mining. Sono presenti, inoltre, i file .txt risultanti dal processo di text mining e un file .csv che tiene traccia delle repository non esistenti.
- mining_result_asa: contiene i file .cvs con i risultati dell'analisi statica, effettuata tramite SonarQube, e gli script che prendono in input tali file e producono un file .csv in cui ad ogni file è associato il numero di istanze di ogni vulnerabilità individuata. Vi è, inoltre, lo script 'Test_Union_TMwithASA.pyTest_Union_TMwithASA.py' che verifica l'output dell'unione dei risultati dell'analisi statica e del text mining; non è utilizzato all'interno del sistema.

- Old_Software_Metrics_Mining_Result: presenta vecchie versioni (e non più utilizzate) dei file contenenti le metriche software estratte dai file relativi alle repository analizzate. Il package non viene più utilizzato all'interno del sistema.

- RepoMining: contiene lo script 'divide_dataset.py', per la divisione del dataset iniziale in file da 50 commit ciascuno, e gli script 'main_repo_mining.py' e 'repo_Mining.py' che interagiscono con il package 'Dataset_Divided' e sono responsabili dell'estrazione dei file, salvati nel package 'mining_repo', introdotti o modificati per ciascun commit risolutivo contenuto nel dataset.

- Results: contiene i documenti relativi ai risultati ottenuti dai modelli di machine learning addestrati sui diversi dati estratti dai file analizzati. Il package non viene utilizzato durante l'esecuzione del tool.

- Software_Metrics: contiene le metriche software relative ai diversi file analizzati, ottenute tramite il tool Understand. In particolare, sono presenti:

  - il numero di dichiarazioni di classi presenti;
  - il numero di dichiarazioni di funzioni presenti;
  - il numero di righe di codice che rappresentano dichiarazioni;
  - la somma delle complessità essenziali delle funzioni presenti;
  - il valore massimo della complessità essenziale tra le funzioni presenti;
  - la somma delle complessità ciclomatiche delle funzioni presenti;
  - il valore massimo della complessità ciclomatica delle funzioni presenti;
  - la massima profondità di annidamento.

  Il file "add_classes_type.py" è utilizzato, una volta ottenute le metriche, per aggiungere manualmente al .csv finale il tipo di file analizzato (ovvero positivo o negativo) alle vulìnerabilità.

- Text_Mining: contiene file che interagiscono molto col package 'mining_results', infatti i risultati degli script, sia intermedi che finali, vengono proprio salvati all'interno del suddetto package. Il file 'text_mining.py' crea un dizionario di tutte le parole utilizzate per ogni file .java analizzato, che viene salvato nella relativa directory, con le relative frequenze. Il file 'dict_generator.py' va a creare un unico dizionario a partire da quelli relativi a ogni file. Il file 'less_element_text_mining.py' è responsabile del

text processing delle chiavi del dizionario e dell'aggiornamento delle frequenze. Infine, 'creator_csv_for_TextMining.py' genera un dataset .csv che contiene la frequenza delle parole chiave per ogni file Java, e classifica i file come "pos" o "neg" in base al numero del repository. Contiene, inoltre, altre versioni dei risultati ottenuti tramite, probabilmente, precedenti esecuzioni degli script.

- Union: contiene ulteriori quattro package, ognuno contenente il relativo script necessario per combinare i risultati ottenuti dalle analisi. Il package interagisce quindi con i package di 'Software_Metrics', 'Text_Mining' e 'mining_results_asa'. Stranamente, infatti, non vengono utilizzati i risultati del text mining salvati nel package 'mining_results', ma altri risultati presenti nel package 'Text_Mining'.

2.1 Mining delle repository

Lo script 'repo_Mining.py' ha una funzione 'main' che chiama la funzione 'inizialize' per accedere ai file CSV del dataset e leggerne i dati dei commit e delle repository scritti al suo interno. Questi verranno raccolti in un dizionario e passati alla funzione 'startMiningRepo'. Quest'ultima verifica l'esistenza delle repository e dei commit del dataset tramite richieste HTTP. Per ogni commit esistente, si utilizza PyDriller per analizzare le modifiche apportate ai file Java e poi si estrae il codice sorgente precedente in nuove directory strutturate in base all'ID CVE e all'ID del commit. Dopo aver analizzato un commit, lo script registra lo stato del processo in due file di log, 'CHECK.txt' ed 'ERRORS.txt', indicando se il commit è stato trovato e analizzato correttamente o se ci sono stati problemi come la non esistenza del commit o del repository o la presenza di un commit non definito. Il flusso di esecuzione è descritto nel flowchart in figura 2.1.

2.2 Text Mining

Lo script 'text_mining.py' ha una funzione 'main' che itera attraverso le repository precedentemente minate e attraverso le classi java estratte per ciascuna repository. Per ciascuna classe, crea un dizionario di parole, eseguendo la funzione 'takeJavaClass', per poi salvarlo in un file .txt nella stessa directory della classe analizzata. La funzione 'takeJavaClass' prende in input il nome della classe da analizzare, in modo da poterne leggerne il contenuto. Successivamente, chiama la funzione 'removeComments' per rimuovere eventuali commenti multi-line. Il testo filtrato viene tokenizzato escludendo qualsiasi stringa letterale e commenti

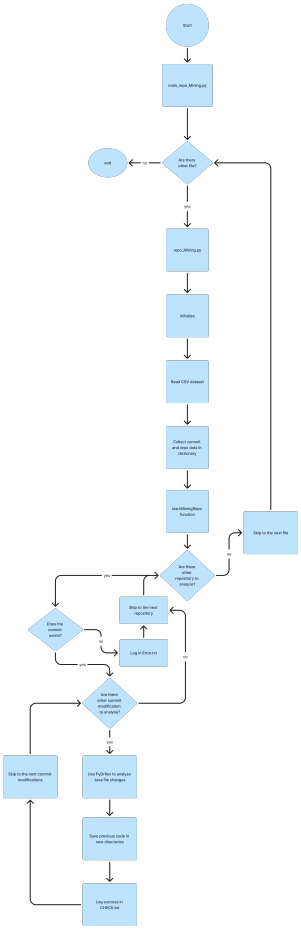

Figura 2.1: Flowchart Mining Repository - (link)

single-line, tramite la funzione 'stringTokenizer', la quale si serve a sua volta della funzione 'removeNotAlpha' per rimuovere i caratteri non alfabetici. Con i token definiti, infine, si crea il dizionario sopracitato in cui a ogni token corrisponde la frequenza nella classe analizzata.

Lo script 'dict_generator.py' presenta un'unica funzione main che itera attraverso le directory delle repository e legge il dizionario creato per ciascuna classe analizzata. Tali dizionari sono poi uniti in un singolo dizionario, salvato nel file 'text_mining_dict.txt'.

Lo script 'less_element_text_mining.py' presenta una funzione 'main' che inizializza il dizionario finale tramite una chiamata alla funzione 'initialize', per poi salvarlo attraverso la funzione 'WriteToFile' in un nuovo file .txt chiamato 'FilteredTextMining.txt'. La funzione 'initialize' legge il dizionario memorizzato nel file 'text_mining_dict.txt' ed esegue lo splitting CamelCase sulle chiavi del dizionario chiamando la funzione 'splitCamelCase'. Nello specifico, tale funzione separa le chiavi che si basano sullo stile CamelCase e ne aggiorna la frequenza nel dizionario, se presenti nel dizionario, altrimenti le inserisce.

Lo script 'creator_csv_for_Text_Mining.py' genera un file .csv contenente le frequenze per le parole trovate nei file di text mining delle classi Java. Per ciascuna classe, elabora i dati di mining presenti nel dizionario generato dallo script 'text_mining.py' e divide le parole in base al CamelCase. Alle chiavi del dizionario memorizzato in FilteredTextMining sono quindi associate le frequenze del dizionario relativo alla singola classe. Infine si crea il file .cvs strutturato, in cui a ogni classe sono associate le frequenze delle stringhe, ordinate alfabeticamente, e la categorizzazione (positiva o negativa).

Il flusso di esecuzione è descritto nel flowchart in figura 2.2

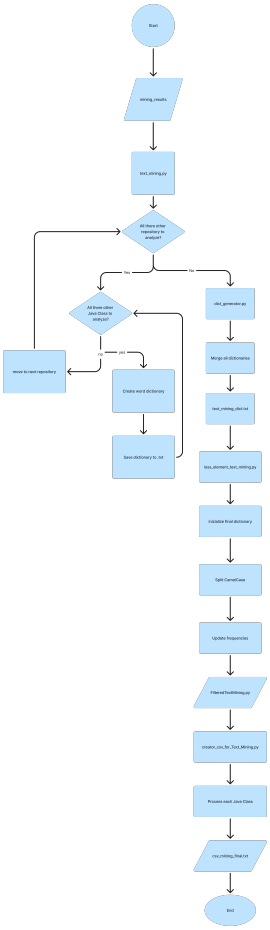

Figura 2.2: Flowchart text mining - (link)

2.3 Rifinitura dati dell'analisi statica

Lo script 'ASA_vulnerability_dict_generator' contiene un'unica funzione 'main' che estrae dalle problematiche rilevate dall'analisi statica per le repository sia negative che positive,

unicamente quelle di tipo "VULNERABILITY" relative ad ogni file .java esaminato, nonchè la relativa regola e le salva in un dataset in formato .csv. Lo script 'rules_dict_generator' prende anch'esso in input le metriche estratte dall'analisi statistica per entrambi i tipi di repository, e la sua unica funzione Main va a creare e salvare in un file .csv un semplice dizionario delle regole di tipo "VULNERABILITY". L'ultimo script, 'creator_csv_for_ASA.py', utilizza, all'interno della sua unica funzione 'main', i due dataset creati dai precedenti script e salva in un dataset, per ciascun file .java, il nome del file e il numero di vulnerabilità per ogni regola. Se una regola non ha vulnerabilità, viene scritto "0". Viene anche scritta la classe del file Java.

Il flusso di esecuzione è descritto nel flowchart in figura 2.3:

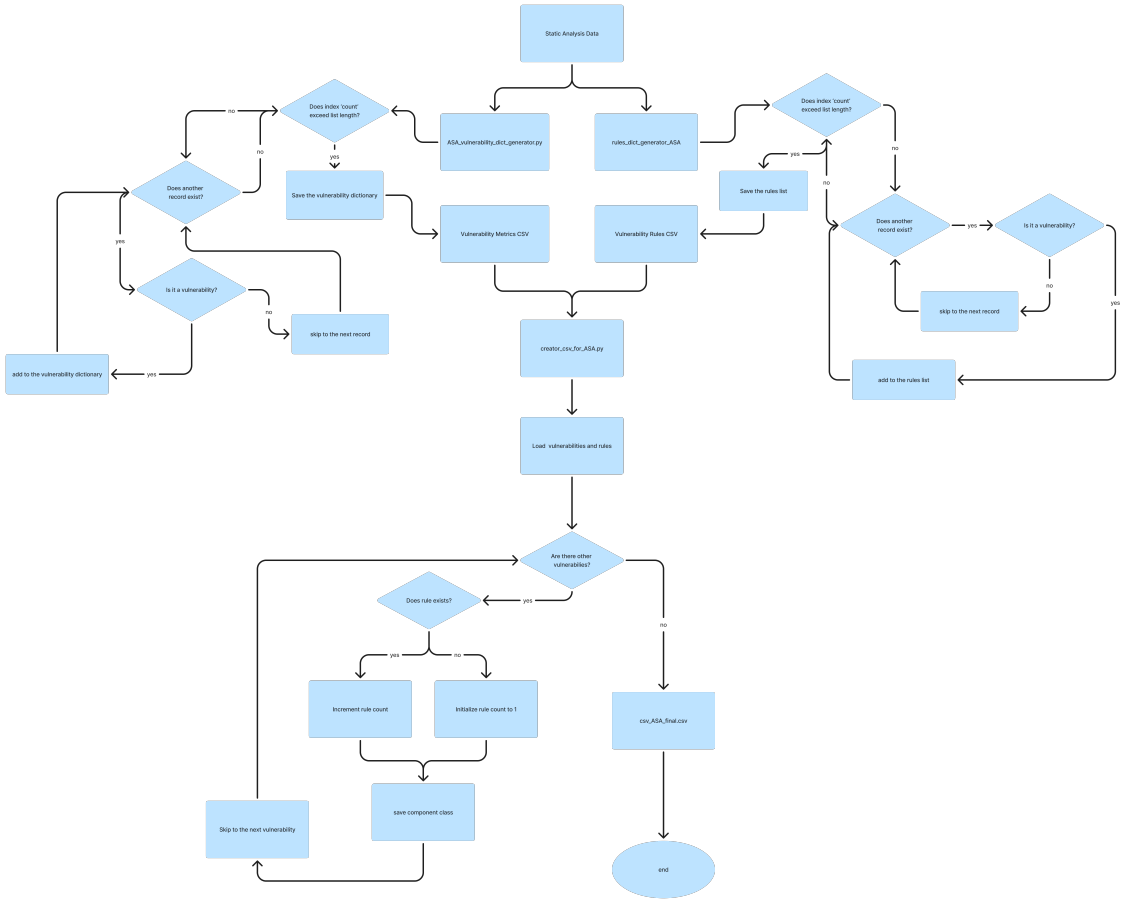

Figura 2.3: Flowchart analisi statica - (link)

2.4 Combinazione dei risultati ottenuti dall'analisi

Lo script 'Union.py' ha una funzione 'main' che chiama la funzione 'inizialize' passando i nomi dei due dataset da unire ed il file di destinazione per la combinazione dei risultati. Questa funzione cambia la directory corrente per accedere in lettura ai file CSV di text mining e delle metriche del software. Successivamente legge singolarmente le righe del dataset di text mining e se è la prima riga, combina i nomi delle metriche dei due dataset e li scrive come intestazione del file destinazione 'union_SM_TM.csv'. Per tutte le altre righe, viene creata una riga corrispondente nel file di destinazione basandosi sul nome del file e sul commit ID. Se viene trovata una corrispondenza, vengono combinate le metriche delle due righe tramite la funzione 'another_option' e scrivendo il risultato del file di destinazione. Se non viene trovata una corrispondenza, la riga viene ignorata. La funzione another_option rielabora le righe dei dataset per rimuovere il class_element e prepararle per la combinazione, mentre la funzione getClass estrae il class_element (pos o neg) da una riga del dataset. Alla fine del processo, 'initialize' stampa il numero di file letti e scritti e ritorna alla funzione main, che completa l'esecuzione dello script. Il risultato finale è un nuovo file CSV che combina i dataset.

Gli altri tre script hanno un funzionamento pressochè identico, utilizzando le medesime funzioni. Differiscono unicamente per i dataset che vanno ad unire.

Lo script Union_TM_ASA.py utilizzando come dati di partenza l' analisi statica automatica e il dataset di Text Mining e fornendo come risultato il file CSV union_TM_ASA. Differentemente dallo script Union.py, quando non viene trovata una corrispondenza tra le righe dei due dataset, vengono generati valori di default (zero) per le metriche di ASA.

Lo script 'TotalCombination.py' crea un unico dataset contenente per ogni file .java tutte e tre le metriche, utilizzando come input il file creato dallo script 'Union_TM_ASA' e il dataset finale sulle metriche del software.

Lo script 'Union_SMwithASA' utilizza come input il file creato dallo script il dataset finale sulle metriche del software e il dataset ottenuto rifinendo i dati estratti dall'analisi statistica. Anch'esso, se non trova una corrispondenza tra le righe dei due dataset, genera valori di default (zero) per le metriche di ASA.

Come già detto, lo script Union.py presenta un comportamento simile agli altri script nel progetto inerenti alla combinazione dei risultati. Di seguito è rappresentato il flusso di esecuzione dettagliato dello script, evidenziando i principali passi e decisioni prese durante l'elaborazione.

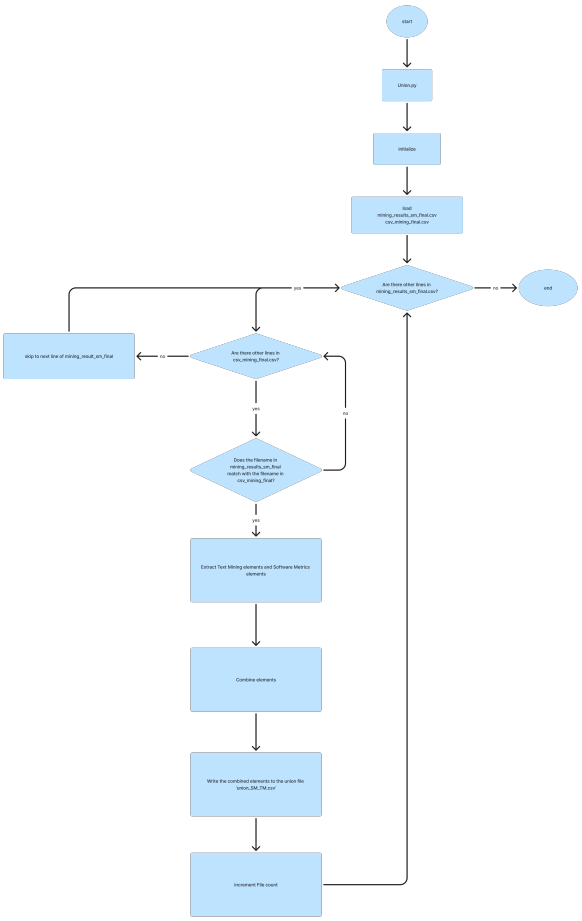

Figura 2.4: Flowchart combinazione dei risultati - (link)