Configuration Manual

MSc Research Project
Data Analytics

Rajat Nagaraj Murdeshwar
Student ID: x22150927

School of Computing
National College of Ireland

Supervisor: Prof. Vladimir Milosavljevic

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Rajat Nagaraj Murdeshwar
Student ID:	x22150927
Programme:	Data Analytics
Year:	2023
Module:	MSc Research Project
Supervisor:	Prof. Vladimir Milosavljevic
Submission Due Date:	14/12/2023
Project Title:	Configuration Manual
Word Count:	814
Page Count:	10

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	13th December 2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Rajat Nagaraj Murdeshwar
x22150927

1 Introduction

This introduction provides an overview of the documentation for this research project. It highlights all the necessary requirements and outlines the steps needed to run the project, titled 'Harnessing Deep Learning for Proactive Detection of Security Threats in Android OS'.

2 System Configuration

All of the system configurations used for the research are listed in this section.

2.1 Hardware Requirements

- **OS**: macOS Ventura 13.5
- **Processor**: Apple M1
- **RAM**: 16GB

2.2 Software Requirements

- **Jupyter Notebook**: This the web based interactive tools used for all the python coding that is done in this project.
- **Google Colab**: This is a cloud-based tool by Google. This is used for python coding which requires higher computation power.
- **Gensim** This is NLP tool

3 Setting up the Environment

I installed the required software like pandas, numpy, seaborn, matplotlib and scikit-learn in Jupyter notebook using !pip install and some more software from conda environment. conda install -c anaconda torch_geometric,

conda install -c anaconda pytorch and conda install -c conda-forge gensim

4 Data Pre-processing

The following sections outlines the steps involved in implementation:

4.1 Extracting Data

Step 1: I created function get_all_cves which accepts argument as number as shown in code Figure 1. it is the content page number. In this function I called the API provided by the *National Vulnerability Database* (n.d.). It is called in loop for fetching the all JSON data from that URL with interval of 2000.

```
In [2]: #Created function with "name" as API call request type name and "number" as the pages
        def get_all_cves(number):
            #https://services.nvd.nist.gov/rest/json/cves/2.0?keywordSearch=android&startIndex=8001
            base_url = "https://services.nvd.nist.gov/rest/json/cves/2.0?keywordSearch=android&startIndex={}"
            acc_url = base_url.format(number)

            try:
                with urllib.request.urlopen(acc_url) as response:

                    return json.load(response)
            except urllib.error.URLError as e:
                print("The API URL is invalid.")
            except json.JSONDecodeError:
                print("Error decoding JSON response")
                return {}
```

Figure 1: Function to extract CVE from API

Step 2: In this, I created the function extract_vulnerabilities this function extracts the JSON data to dataframe. In this step I'm extracting only required data from JSON. I have created two function two extract two different format data as shown in code Figure 2 and Figure 3.

Step 3: In this, I created the function to extract_vulnerabilities this function extracts the JSON data to dataframe. In this step I'm extracting only required data from JSON. I have created two function two extract two different format data as shown in code Figure 4.

Step 4: I used the CWE data that was downloaded from *Common Vulnerabilities and Exposures* (n.d.) and mapped both the dataframe using left join on CWE-ID of first dataframe as shown in code Figure 5 and combined code Figure 6.

4.2 Preparing data for GNN

The cleaned data underwent preprocessing, during which the date column was formatted. Additionally, the base score, impact score, and exploitability score were encoded as shown in Figure 7.The CVE description was encode using NLP technique like TF-IDF and Word2Vec as shown in code snippet Figure 9 and Figure 11.

4.3 Preparing data for Random Forest

The same preprocessing was done for Random Forest with additional columns Severity column was encoded with the proper format. The CVE description was encode using NLP technique like TF-IDF and Word2Vec as shown in this code snippet Figure 12 Figure 13 and Figure 14.

```
In [4]: # Function to extract vulnerabilities information
def extract_vulnerabilities(data):
    vulnerabilities = data.get("vulnerabilities", [])
    vuln_list = []

    for vuln in vulnerabilities:
        cve = vuln.get("cve", {})

        # Extracting weaknesses information
        weaknesses_data = cve.get("weaknesses", [])
        weaknesses = ""
        for weakness in weaknesses_data:
            descriptions = weakness.get("description", [])
            for desc in descriptions:
                if desc.get("lang") == "en":
                    weaknesses = desc.get("value", "")
                    break

        # Extracting metrics information
        metrics_data = cve.get("metrics", {}).get("cvssMetricV2", [])
        baseScore, baseSeverity, exploitabilityScore, impactScore = "", "", "", ""
        if metrics_data:
            metrics = metrics_data[0].get("cvssData", {})
            baseScore = metrics.get("baseScore", "")
            baseSeverity = metrics_data[0].get("baseSeverity", "")
            exploitabilityScore = metrics_data[0].get("exploitabilityScore", "")
            impactScore = metrics_data[0].get("impactScore", "")

        vuln_dict = {
            "id": cve.get("id", ""),
            "published": cve.get("published", ""),
            "lastModified": cve.get("lastModified", ""),
            "vulnStatus": cve.get("vulnStatus", ""),
            "descriptions": " ".join([desc["value"] for desc in cve.get("descriptions", []) if desc["lang"] == "en"]
            "weaknesses": weaknesses,
            "baseScore": baseScore,
            "baseSeverity": baseSeverity,
            "exploitabilityScore": exploitabilityScore,
            "impactScore": impactScore
        }
        vuln_list.append(vuln_dict)

    return pd.DataFrame(vuln_list)
```

Figure 2: Function for extracting JSON data

```python
In [5]: # Function to extract vulnerabilities information
def extract_vulnerabilitiesV3(data):
    vulnerabilities = data.get("vulnerabilities", [])
    vuln_list = []

    for vuln in vulnerabilities:
        cve = vuln.get("cve", {})

        # Extracting weaknesses information
        weaknesses_data = cve.get("weaknesses", [])
        weaknesses = ""
        for weakness in weaknesses_data:
            descriptions = weakness.get("description", [])
            for desc in descriptions:
                if desc.get("lang") == "en":
                    weaknesses = desc.get("value", "")
                    break

        # Extracting metrics information
        metrics_data = cve.get("metrics", {}).get("cvssMetricV31", [])
        baseScore, baseSeverity, exploitabilityScore, impactScore = "", "", "", ""
        if metrics_data:
            metrics = metrics_data[0].get("cvssData", {})
            baseScore = metrics.get("baseScore", "")
            baseSeverity = metrics.get("baseSeverity", "")
            exploitabilityScore = metrics_data[0].get("exploitabilityScore", "")
            impactScore = metrics_data[0].get("impactScore", "")

        vuln_dict = {
            "id": cve.get("id", ""),
            "published": cve.get("published", ""),
            "lastModified": cve.get("lastModified", ""),
            "vulnStatus": cve.get("vulnStatus", ""),
            "descriptions": " ".join([desc["value"] for desc in cve.get("descriptions", []) if desc["lang"] == "en"
            "weaknesses": weaknesses,
            "baseScore": baseScore,
            "baseSeverity": baseSeverity,
            "exploitabilityScore": exploitabilityScore,
            "impactScore": impactScore
        }
        vuln_list.append(vuln_dict)

    return pd.DataFrame(vuln_list)
```

Figure 3: Function for extracting JSON data

```python
In [6]: # A list to hold all the resulting DataFrames
all_dataframes = []

for dat in np.arange(0, 6000, 2000):
    print("Fetching data starting from:", dat)
    api_response = get_all_cves(dat)
    if api_response:  # Check if the API response is not empty
        df = extract_vulnerabilities(api_response)
        all_dataframes.append(df)
    else:
        print("No data received for starting point:", dat)

Fetching data starting from: 0
Fetching data starting from: 2000
Fetching data starting from: 4000

In [8]: for dat in np.arange(6001, 10000, 2000):
    print("Fetching data starting from:", dat)
    api_response = get_all_cves(dat)
    if api_response:  # Check if the API response is not empty
        df = extract_vulnerabilitiesV3(api_response)
        all_dataframes.append(df)
    else:
        print("No data received for starting point:", dat)

# Concatenate all DataFrames into a single DataFrame
final_dataframe = pd.concat(all_dataframes, ignore_index=True)

Fetching data starting from: 6001
Fetching data starting from: 8001
```

Figure 4: Main function that call the API

```
In [9]: final_dataframe.head()
Out[9]:
```

	id	published	lastModified	vulnStatus	descriptions	weaknesses	baseScore	baseSeverity	exploitabilityScore	impactScore
0	CVE-2008-0985	2008-03-06T00:44:00.000	2018-10-15T22:04:08.043	Modified	Heap-based buffer overflow in the GIF library ...	CWE-119	6.8	MEDIUM	8.6	6.4
1	CVE-2008-0986	2008-03-06T00:44:00.000	2018-10-15T22:04:08.560	Modified	Integer overflow in the BMP::readFromStream me...	CWE-189	7.5	HIGH	10.0	6.4
2	CVE-2009-0606	2009-02-17T17:30:05.953	2018-10-10T19:29:55.763	Modified	The link_image function in linker/linker.c in ...	CWE-20	7.2	HIGH	3.9	10.0
3	CVE-2009-0607	2009-02-17T17:30:05.967	2018-10-10T19:29:56.093	Modified	Multiple integer overflows in malloc_leak.c in...	CWE-189	7.2	HIGH	3.9	10.0
4	CVE-2009-0608	2009-02-17T17:30:05.983	2018-10-10T19:29:56.653	Modified	Integer overflow in the showLog function in fa...	CWE-189	7.2	HIGH	3.9	10.0

Figure 5: Extracted dataframe

```
In [ ]: # Strip leading/trailing spaces and convert to string if necessary
        cve_df['CWE-ID'] = cve_df['CWE-ID'].astype(str)
        cwe_df['CWE-ID'] = cwe_df['CWE-ID'].astype(str)


        # Merge df1 with df2. This will map each 'CWE-ID' in df1 to its corresponding entry in df2.
        df_merged = pd.merge(cve_df, cwe_df, on='CWE-ID', how='left')

        df_merged.head()

In [ ]: most_frequent_string = cve_df['CWE-ID'].value_counts().idxmax()

In [ ]: most_frequent_string

In [ ]: csv_file_path = "/Users/rajatmurdeshwar/Downloads/cve_cwe_updated_with_6.csv"
        # Use the to_csv method to save the DataFrame to the CSV file with tab separator and UTF-8 encoding
        df_merged = pd.read_csv(csv_file_path)
```

Figure 6: Combine dataframe

```
In [ ]: # Function to generate severity columns
        def generate_severity_columns(row):
            if pd.isna(row['CVE Base Score']):
                return [0, 0, 0, 0]  # Assuming NA values are represented as zeros in all severity columns
            elif row['CVE Base Score'] < 4:
                return [1, 0, 0, 0]  # Low
            elif 4 <= row['CVE Base Score'] < 6:
                return [0, 1, 0, 0]  # Medium
            elif 6 <= row['CVE Base Score'] < 9:
                return [0, 0, 1, 0]  # High
            else:
                return [0, 0, 0, 1]  # Critical

        # Apply the function and split the results into four new columns
        df_merged[['Severity_LOW', 'Severity_MEDIUM', 'Severity_HIGH', 'Severity_CRITICAL']] = pd.DataFrame(df_merged.apply

In [ ]: from sklearn.preprocessing import LabelEncoder

        # Initialize the label encoder
        le = LabelEncoder()

        # Label encode 'CVE Vulnerability Status'
        df_merged['CVE Vulnerability Status Encoded'] = le.fit_transform(df_merged['CVE Vulnerability Status'].fillna('Unkn

        # Label encode 'CVE Base Severity'
        df_merged['CVE Base Severity Encoded'] = le.fit_transform(df_merged['CVE Base Severity'].fillna('Unknown'))
```

Figure 7: Encoded data

4.4 Preparing data for SVM

The same preprocessing was done for SVM as Random Forest as both requires similar format. The CVE description was encode using NLP technique like TF-IDF and Word2Vec. Figure 13

5 Upload Data to Google Drive

The preprocessed data was uploaded to Google Drive, which facilitated its integration with Google Colab. This setup was utilized for building and running the model, leveraging Google Colab's powerful computing resources and seamless access to data stored on Google Drive. This approach allowed for efficient model development and testing, taking advantage of Colab's collaborative features and cloud-based environment to optimize the machine learning workflow.

6 Implemented Models

In this implementation step, three different models were utilized. For each model, two NLP techniques were employed: the first being TF-IDF and the second being Word2Vec.

6.1 Implementation of GNN

The final dataframe, integral to our analysis, was utilized in the code provided below. This code is responsible for generating the nodes and edges essential for the Graph Neural Network (GNN). Specifically, nodes and edges were created based on the relationship of each Common Vulnerabilities and Exposures (CVE) entry to its corresponding Common Weakness Enumeration (CWE-ID). As shown in below code Figure 8 and Graph Convolutional Network (GCN) model type and is configured structured with a 16-dimensional hidden space and Adam optimizer is used with a learning rate of 0.01. The loss function is binary cross-entropy with logits (BCEWithLogitsLoss), which combines a sigmoid layer and the BCE loss in one single class as mentioned in code Figure 10

```
In [14]: # Then, when you add nodes and edges to your graph, you will be working with integer IDs:
G = nx.Graph()
# Add nodes with CVE attributes
for _, row in df_merged.iterrows():
    # Convert CWE ID to string and check if it's numeric
    cwe_id_str = str(row['Weakness CWE-ID'])
    cwe_id = int(cwe_id_str) if cwe_id_str.isdigit() else None

    # Add the node to the graph
    G.add_node(row['CVE-ID'],
               cve_score=row['CVE Base Score'],
               cve_severity=row['Severity_HIGH'],
               cwe_id=cwe_id)

# Add edges between CWEs using the new function
for _, row in df_merged.iterrows():
    if isinstance(row['CWE Related Weaknesses'], str):
        edges = extract_relationships(row['CWE Related Weaknesses'])
        for edge in edges:
            # Ensure the nodes for the edges exist before adding the edge
            if not G.has_node(edge[0]):
                G.add_node(edge[0])
            if not G.has_node(edge[1]):
                G.add_node(edge[1])
            G.add_edge(edge[0], edge[1])
```

Figure 8: GNN

```
In [16]: # Initialize the TF-IDF Vectorizer
tfidf_vectorizer = TfidfVectorizer(max_features=500)

# Fit and transform the CVE Descriptions into TF-IDF vectors
tfidf_matrix = tfidf_vectorizer.fit_transform(df_merged['CVE Descriptions'].values.astype('U'))

# Create the edge_index tensor for PyTorch Geometric
edge_index = torch.tensor(list(map(list, G.edges())), dtype=torch.long).t().contiguous()

# Create the feature tensor from the TF-IDF matrix
features_tensor = torch.FloatTensor(tfidf_matrix.todense())

# Create the GNN Data object
data = Data(x=features_tensor, edge_index=edge_index)
```

Figure 9: GNN NLP TF-IDF

```
In [17]: from torch_geometric.nn import GCNConv
import torch.nn.functional as F

class GNNModel(torch.nn.Module):
    def __init__(self, num_node_features, num_classes):
        super(GNNModel, self).__init__()
        # Define GNN layers
        self.conv1 = GCNConv(num_node_features, 16)
        self.conv2 = GCNConv(16, num_classes)

    def forward(self, x, edge_index):
        # First GNN layer
        x = self.conv1(x, edge_index)
        x = F.relu(x)
        x = F.dropout(x, training=self.training)

        # Second GNN layer
        x = self.conv2(x, edge_index)

        return x
```

```
In [18]: # Set a random seed for reproducibility
torch.manual_seed(0)
np.random.seed(0)

# Instantiate the model
model = GNNModel(num_node_features=features_tensor.size(1), num_classes=1)

# Assume binary labels for each graph
labels = torch.tensor(df_merged['Severity_HIGH'].values, dtype=torch.float).unsqueeze(1)
```

Figure 10: GNN Model

```
In [16]: tokenized_texts = [word_tokenize(description.lower()) for description in df_merged['CVE Descriptions'].values.astyp

word2vec_model = Word2Vec(tokenized_texts, vector_size=100, window=5, min_count=1, workers=4)
```

```
In [17]: def get_word2vec_embedding(words, model):
    word_embeddings = [model.wv[word] for word in words if word in model.wv]
    if not word_embeddings:
        return np.zeros(model.vector_size)
    return np.mean(word_embeddings, axis=0)
```

```
In [18]: # Creating embeddings for each CVE description
embeddings_matrix = np.array([get_word2vec_embedding(words, word2vec_model) for words in tokenized_texts])

features_tensor = torch.FloatTensor(embeddings_matrix)

# Create the edge_index tensor for PyTorch Geometric
edge_index = torch.tensor(list(map(list, G.edges())), dtype=torch.long).t().contiguous()

# Create the GNN Data object
data = Data(x=features_tensor, edge_index=edge_index)
```

Figure 11: GNN word2vec NLP

6.2 Implementation of Random Forest

The final dataframe, In this implementation the data was cleaned and preprocessed so in the below code I did combine the CVE description and related weakness ID to make it a single text to pass through word2vec or TD-IDF process. Random Forest is set with 100 trees balancing computational efficiency with the ability to capture diverse patterns in the data as shown in code Figure 13.

```
In [6]: import re

        # Function to extract CWE IDs from the 'CWE Related Weaknesses' column
        def extract_cwe_ids(cwe_string):
            if pd.isnull(cwe_string):
                return []
            cwe_ids = re.findall(r'CWE ID:(\d+)', cwe_string)
            return [int(id) for id in cwe_ids if id.isdigit()]

        df_merged['CWE Related Weakness IDs'] = df_merged['CWE Related Weaknesses'].apply(extract_cwe_ids)

        # Example of converting CWE ID list to a string (to use in TF-IDF)
        df_merged['CWE Related Weakness IDs'] = df_merged['CWE Related Weakness IDs'].apply(lambda ids: ' '.join(['CWE_ID_'

In [7]: df_merged['combined_text'] = df_merged['CVE Descriptions'] + ' ' + df_merged['CWE Related Weakness IDs'].fillna('')
```

Figure 12: Random Forest Extract Data

```
In [8]: import gensim
        from gensim.models import Word2Vec

        tokenized_texts = [word_tokenize(description.lower()) for description in df_merged['combined_text'].values.astype('

        word2vec_model = Word2Vec(tokenized_texts, vector_size=100, window=5, min_count=1, workers=4)
```

Figure 13: Random Forest word2vec NLP

```
In [10]: # Combine the vectorized text with other features
         df_final = pd.concat([df_merged[['Weakness CWE-ID', 'CVE-ID', 'CVE Base Score', 'Severity_HIGH']], vector_df], axis

In [11]: # Handle categorical data (CWE-ID, CVE-ID)
         label_encoder = LabelEncoder()
         df_final['Weakness CWE-ID'] = label_encoder.fit_transform(df_final['Weakness CWE-ID'].astype(str))
         df_final['CVE-ID'] = label_encoder.fit_transform(df_final['CVE-ID'])

In [12]: # Scale numerical data (CVE Base Score)
         scaler = StandardScaler()
         df_final['CVE Base Score'] = scaler.fit_transform(df_final[['CVE Base Score']])
```

Figure 14: Random Forest Encoding

6.3 Implementation of SVM

In building SVM model, I made similar pre processing as Random Forest and the train data and test data was exactly same then I trained as shown in this code below at Figure 16. In SVM kernel is set to Linear because it aligns with nature of our data, ensuring optimal separation and accuracy. I have also added the SVM performance and it's evaluation metrics results Figure 17.

```
In [12]: # Scale numerical data (CVE Base Score)
         scaler = StandardScaler()
         df_final['CVE Base Score'] = scaler.fit_transform(df_final[['CVE Base Score']])

In [13]: df_final = df_final.fillna(0)
         df_final = df_final.replace([np.inf, -np.inf], np.nan).fillna(0)

         max_val = np.finfo(np.float32).max
         df_final = df_final.clip(upper=max_val)

In [14]: # Prepare data for training
         X = df_final.drop('Severity_HIGH', axis=1)
         y = df_final['Severity_HIGH']


         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

In [15]: rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
```

Figure 15: Random Forests Model

```
In [14]: # Prepare data for training
         X = df_final.drop('Severity_HIGH', axis=1)
         y = df_final['Severity_HIGH']


         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

In [15]: svm_model = SVC(kernel='linear')

In [16]: svm_model.fit(X_train, y_train)

Out[16]: SVC(kernel='linear')
```

Figure 16: SVM Model

```
In [18]: from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
         from sklearn.metrics import roc_auc_score, confusion_matrix, matthews_corrcoef
         from sklearn.metrics import classification_report

         # Evaluation metrics
         accuracy = accuracy_score(y_test, y_pred)
         precision = precision_score(y_test, y_pred)
         recall = recall_score(y_test, y_pred)
         f1 = f1_score(y_test, y_pred)
         roc_auc = roc_auc_score(y_test, y_pred)
         conf_matrix = confusion_matrix(y_test, y_pred)
         mcc = matthews_corrcoef(y_test, y_pred)

         print("Accuracy:", accuracy)
         print("Precision:", precision)
         print("Recall:", recall)
         print("F1 Score:", f1)
         print("ROC AUC:", roc_auc)
         print("Confusion Matrix:\n", conf_matrix)
         print("Matthews Correlation Coefficient:", mcc)
         print("\nClassification Report:\n", classification_report(y_test, y_pred))

         Accuracy: 0.7836710369487485
         Precision: 0.6619385342789598
         Recall: 0.56
         F1 Score: 0.6067172264355363
         ROC AUC: 0.7193039049235994
         Confusion Matrix:
          [[1035  143]
          [ 220  280]]
         Matthews Correlation Coefficient: 0.46199998478331955
```

Figure 17: SVM Results

References

Common Vulnerabilities and Exposures (n.d.). https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=android. Accessed: Dec 11, 2023.

National Vulnerability Database (n.d.). https://nvd.nist.gov/vuln/search?results_type=overview&query=android+os&search_type=all&form_type=Basic&isCpeNameSearch=false. Accessed: Dec 11, 2023.