Prediction and Classification of Software Vulnerabilities

1. Problem Definition

Software vulnerabilities represent a significant security risk because they allow attackers to perform malicious actions, including data theft and taking control of systems.
The goal of this project is to develop an AI system that combines classification (to predict whether a given software is vulnerable or not) and regression (to predict the likelihood that a vulnerability will be exploited in the future).

2. Motivation

Software vulnerabilities pose a serious threat due to data breaches, financial losses, malware propagation, and national security concerns.
The developed AI system aims to assist security teams by helping them prioritize patches based on risk, reduce the time required for vulnerability analysis, and prevent security incidents more efficiently.

3. Datasets

Two main data sources are used:

- The **National Vulnerability Database (NVD)**, which contains information about CVE IDs, vulnerability descriptions, CVSS scores, vulnerability types, and software versions, with over 150,000 recorded vulnerabilities.
- The **Exploit Database (EDB)**, which links vulnerabilities to real-world exploits and enables estimation of exploit likelihood.

The target variable for classification is **vulnerable (1/0)**, while the target for regression is **exploit_probability**.

4. Data Preprocessing

The process includes parsing NVD JSON files to extract attributes, extracting CVE IDs from EDB CSV files, merging data by CVE ID, and creating a numerical column `num_exploits`. Numerical variables are normalized, categorical variables are encoded using one-hot encoding, and the complete dataset is saved in CSV/Parquet/Pickle format.

An advanced approach introduces **pretrained transformer models** (BERT, RoBERTa, SecBERT) instead of TF-IDF to generate embedding vectors that better capture the semantic meaning of vulnerability descriptions.

The model architecture combines a **textual stream** (vulnerability description → BERT embeddings → Dense layers) and a **tabular stream** (numerical and categorical features → Dense layers), which are then merged through additional Dense layers leading to two outputs — one for classification and one for regression.

5. Methodology

The problem-solving process includes data collection and cleaning, feature engineering with textual and numerical attributes, modeling using Logistic Regression, Random Forest, XGBoost, and Neural Networks for classification, and corresponding regressors for regression, followed by model evaluation and integration of results through a dashboard.

The model inputs include vulnerability attributes (CVSS score, description, software type, age, and vulnerability type), while outputs represent **classification (vulnerable/non-vulnerable)** and **exploit probability**.

Multi-task learning allows the model to learn both classification and regression tasks simultaneously, improving shared data representation learning.
Training uses **BCEWithLogitsLoss** for classification and **MSELoss** for regression, with the **AdamW optimizer** and **Linear LR warmup scheduler**.

The model captures the semantics of vulnerability descriptions, enables visual ranking of vulnerabilities by risk, and interprets key features through **SHAP/LIME** analysis.

6. Model Evaluation

The dataset is divided into a training set (70%), validation set (15%), and test set (15%). Classification model performance is measured using **accuracy**, **precision**, **recall**, and **F1-score**, while regression model performance is evaluated with **RMSE**, **MAE**, and **R^2 score**.

A feature importance analysis and result interpretation are planned to better understand which characteristics influence vulnerabilities and exploit probability.
Visualization will be done using **t-SNE/UMAP** for embeddings and to display similar vulnerabilities.

7. Technologies

The implementation is carried out in **Python** using:

- **pandas** and **numpy** for data processing,
- **scikit-learn** for machine learning,
- **XGBoost** for gradient boosting,
- **TensorFlow** or **PyTorch** for deep learning.

Data and result visualization are done using **matplotlib**, **seaborn**, and **plotly**, while the interactive dashboard is built with **Streamlit** or **Dash**.

8. Relevant Literature

The project is based on the research paper *"Predicting Exploitability of Software Vulnerabilities Using Machine Learning"*, the official **NVD documentation** (https://nvd.nist.gov), the **Exploit Database** (https://www.exploit-db.com/), and **Kaggle** datasets such as *"Vulnerability Detection & Exploit Prediction"* (https://www.kaggle.com/).