Automated Vulnerability Detection using Deep Learning Report

Name: Isha Sangpal

Email: sangpalisha@gmail.com

Title:

Automated Vulnerability Detection in Source Code Using Deep Learning
Category: Cybersecurity, Secure Software Development, AI for Security

---

Problem Statement

Software vulnerabilities pose a significant security risk, leading to potential exploits, data breaches, and system compromises. The growing complexity of modern software, coupled with rapid development cycles, makes it difficult for security teams to manually analyze and detect vulnerabilities in source code. Traditional static analysis tools often produce excessive false positives, leading to inefficient vulnerability detection and remediation efforts. This project aims to leverage deep learning techniques to automate vulnerability detection in source code, improving accuracy and efficiency while minimizing human intervention.

Objectives

- Develop an AI-based vulnerability detection model to analyze and classify security vulnerabilities in source code.
- Improve accuracy and efficiency over traditional static analysis methods.
- Reduce false positives and false negatives in vulnerability classification.
- Train a Convolutional Neural Network (CNN) model to detect vulnerabilities in source code based on known Common Weakness Enumeration (CWE) categories.
- Provide a scalable and automated approach for real-world security assessments.

Scope of Work

The project focuses on the automated detection of software vulnerabilities using deep learning. The scope includes:

- Analyzing large-scale datasets of vulnerable and non-vulnerable source code.
- Implementing a deep learning-based classification model using TensorFlow.

- Training and optimizing the model for multi-label classification of vulnerabilities.
- Validating the model using real-world vulnerability datasets.
- Deploying the trained model for inference on new, unseen code samples.
- Providing a report on model performance, including accuracy, precision, recall, and other relevant metrics.

Methodology & Approach

The approach to solving the problem is structured into five key phases:

1. Data Collection and Pre-processing

- The dataset used is the **VDISC dataset**, which contains labeled source code snippets categorized by vulnerability types.
- Pre-processing includes tokenization of source code, cleaning of irrelevant data, and conversion of text into numerical representations.
- The dataset is split into training, validation, and testing sets to ensure robust model evaluation.

2. Feature Engineering

- The project utilizes Natural Language Processing (NLP) techniques such as tokenization and sequence padding.
- Feature extraction is performed to transform raw source code into a format suitable for deep learning models.

3. Model Development

- A **Convolutional Neural Network (CNN)** architecture is implemented for multi-label classification.
- The model takes tokenized code as input and predicts vulnerability classes based on CWE categories.
- Optimization techniques such as dropout, batch normalization, and fine-tuning are applied to improve performance.

4. Model Training & Evaluation

- The model is trained using TensorFlow 2.0 with **GPU acceleration** to speed up computation.
- Performance is evaluated using metrics such as accuracy, precision, recall, F1-score, and AUC-ROC.
- Hyperparameter tuning is performed to optimize the model's effectiveness.

5. Deployment & Inference

- The trained model is deployed to analyze new code samples and detect vulnerabilities.

- An inference script is developed to test new code snippets and provide vulnerability predictions.
- The final results are validated against real-world datasets to measure the model's effectiveness.

Key Parameters

The success of the project is measured based on:

- **Performance Metrics:** Accuracy, Precision, Recall, F1-Score, AUC-ROC.
- **Dataset Used:** VDISC dataset (labeled software vulnerabilities).
- **Vulnerability Categories:**
  - CWE-119: Buffer Overflow
  - CWE-120: Improper Input Validation
  - CWE-469: Incorrect Pointer Scaling
  - CWE-476: NULL Pointer Dereference
  - CWE-Other: Miscellaneous vulnerabilities

Tools & Technologies

- **Programming Language:** Python
- **Deep Learning Framework:** TensorFlow 2.0
- **NLP Techniques:** Tokenization, Sequence Padding
- **Dataset Formats:** HDF5, Pickle
- **Hardware Requirements:**

  - **GPU Acceleration** (NVIDIA CUDA, cuDNN) for efficient model training and inference

---

Roles & Responsibilities

Since this project was conducted individually, the responsibilities were divided as follows:

- **Project Leader:** Defined the project scope, managed workflow, and ensured project goals were met.
- **Security Analyst:** Conducted vulnerability analysis and integrated security best practices into the detection model.
- **Developer/Researcher:** Developed and optimized deep learning models, implemented dataset preprocessing, and fine-tuned the model for better performance.

Expected Outcomes

The project aims to achieve the following outcomes:

- A trained deep learning model capable of detecting software vulnerabilities in source code automatically.
- Improved detection accuracy compared to traditional rule-based static analysis tools.
- Reduction in manual code review efforts through AI-powered automation.
- Deployment of the model for real-world security assessments.

Deliverables & Documentation

The final deliverables for the project include:

1. **Trained Model:** The deep learning model stored in .hdf5 format for further inference.
2. **Inference Script:** A Python script (run_inference.py) to detect vulnerabilities in new code samples.
3. **Dataset Preprocessing Scripts:** Scripts to clean, tokenize, and preprocess source code for model training.
4. **Project Report & Documentation:** A detailed report covering methodology, results, and analysis.

Conclusion

The proposed project provides an efficient and scalable solution for detecting vulnerabilities in source code using deep learning. The AI-powered approach aims to reduce the manual effort required for vulnerability analysis while improving detection accuracy. With further enhancements, this model can be integrated into real-world software development environments to assist security teams in identifying and mitigating vulnerabilities more effectively.