CodeBERT-Based LM for Vulnerability Detection in Static Analysis

Xinyi Liu

Software Vulnerability

- Vulnerabilities in source code can lead to:
  - **Data Breaches**: Unauthorized access to sensitive information.
  - **Malware Infections**: Exploitation of insecure code.
  - **System Failures**: Critical bugs undermining system integrity.
- **Static Analysis**
  - **Definition**: A technique to analyze source code without executing it.
  - **Purpose**:
    - Identify potential issues early in the development lifecycle.
    - Reduce downstream costs and improve software reliability.

Traditional Static Analysis Tools

- **Manual Reviews**:
  - Time-consuming and prone to human error.
- **Rule-Based Tools**:

Tool	Focus	Languages Supported	Customization	Cost
SonarQube	General Quality & Security	Multiple	Moderate	Free
Checkmarx	Security	Multiple	Low	Paid
Bandit	Python Security	Python	Moderate	Free
Fortify SCA	Security	Multiple	Low	Paid

  - High false positive rates.
  - Lack adaptability to new coding patterns and vulnerabilities.

ML Approach

Approach	Description	Limitations
RoBERTa (Do et al., 2024)	Detect vulnerabilities in source code written in C and C++ with NLP	• Cannot generalize to other languages
Large Language Model for Vulnerability Detection (Zhou et al. 2024)	Detecting software vulnerabilities with GPT-3.5 and GPT-4	• Lacks code-specific pretraining
CodeBert (Feng et al.2020)	Pre-trained transformer designed for both programming languages and natural language tasks	• Requires fine-tuning for specific tasks like vulnerability detection

- Current LM models lack scalability, structural insights, or adaptability

CodeBert

- **Transformer-Based Model**
  - Pre-trained using **masked language modeling (MLM)** and **replaced token detection (RTD)** to capture both syntax and semantics of code
- **Multi-Language Support**
  - Covers **multiple languages**, including Python, Java, JavaScript, C++, etc.
- **Pre-Trained on Source Code**
  - Trained on a **large-scale dataset** of programming languages and natural language documentation
  - Specifically optimized to understand **code semantics**, **structural relationships**, and **context**, far beyond simple text analysis

Refine CodeBert

- Adapted CodeBERT to binary classification (**safe** vs. **vulnerable**) tailored to enhance **static code analysis**
- Datasets
  - **CVEFixes**
    - 15K labeled code snippets
    - Contains paired vulnerable code and labels for language and safety
  - **DiverseVul**

code Source Code	language Programming language	safety Safe or vulnerable
404: Not Found 5% <?php namespace... 0% Other (29717) 95%	c 28% Other 20% Other (16440) 53%	2 unique values
package org.bouncycastle.jca jce.provider.asymmet ric.dsa; import java.security.Invali dAlgorithmPar...	java	vulnerable
<?php /** * ownCloud - user_ldap * * @author Dominik Schmidt * @copyright 2011 Dominik Sc...	php	vulnerable
#!/usr/bin/env python from __future__ import division, absolute_import, print_function __all__ =...	py	safe

Refine CodeBert

- Adapted CodeBERT to binary classification (**safe** vs. **vulnerable**) tailored to enhance **static code analysis**
- Datasets
  - **CVEFixes**
  - **DiverseVul**
    - Enriched with structural vulnerability details
    - Broader language and vulnerability types coverage


```
{
"func": "static char *make_filename_safe(const char *filename
TSRMLS_DC)\n{\n\tif (*filename && strncmp(filename, \":memory:\",
sizeof(\":memory:\")-1)) {\n\t\tchar *fullpath =
expand_filepath(filename, NULL TSRMLS_CC);\n\n\t\tif (!fullpath)
{\n\t\t\treturn NULL;\n\t\t}\n\n\t\tif (PG(safe_mode) && (!
php_checkuid(fullpath, NULL, CHECKUID_CHECK_FILE_AND_DIR)))
{\n\t\t\tefree(fullpath);\n\t\t\treturn NULL;\n\t\t}\n\n\t\tif
(php_check_open_basedir(fullpath TSRMLS_CC))
{\n\t\t\tefree(fullpath);\n\t\t\treturn NULL;\n\t\t}\n\t\treturn fullpath;
\n\t}\n\treturn estrdup(filename);\n}",
"target": 1,
"cwe": ["CWE-264"],
"project": "php-src",
"commit_id": "055ecbc62878e86287d742c7246c21606cee8183",
"hash": 211824207069112513181516095447837228041,
"size": 22,
"message": "Improve check for :memory: pseudo-filename in SQlite"
}
```

Refine CodeBert

- **Training Pipeline**
  - **Preprocessing**
    - Tokenized code snippets for compatibility with CodeBERT

```
Example Input IDs:
tensor([[    0, 49051, 50140,  ...,  1215,   534,     2],
        [    0, 42326, 15753,  ...,  4238,     4,     2]])

Example Attention Masks:
tensor([[1, 1, 1,  ..., 1, 1, 1],
        [1, 1, 1,  ..., 1, 1, 1]])

Example Labels:
tensor([1, 0])
```

Refine CodeBert

- **Training Pipeline**
  - **Preprocessing**
  - **Fine-Tuning**
    - Adapted CodeBERT to binary classification using labeled data

```
training_args = TrainingArguments(
    output_dir="./results",
    evaluation_strategy="epoch",
    save_strategy="epoch",
    learning_rate=2e-5,
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    num_train_epochs=3,
    weight_decay=0.01,
    logging_dir="./logs",
    logging_steps=10,
    save_total_limit=2,
    load_best_model_at_end=True,
    metric_for_best_model="f1",
    greater_is_better=True,
)
```

Refine CodeBert

- **Training Pipeline**
  - **Preprocessing**
  - **Fine-Tuning**
  - **Evaluation**
    - Assessed model performance on a **held-out test set** to validate accuracy and other metrics

Initial Results

- **Key Metrics**
  - **Accuracy**: 48.8%
  - **Precision:** 0%
  - **Recall:** 0%
  - **F1**: 0%
  - **Evaluation Loss**: 0.691 (near-random predictions; underfitting observed)

Initial Results

- **Key Metrics**
  - **Accuracy**: 48.8%
  - **Precision:** 0%
  - **Recall:** 0%
  - **F1**: 0%
  - **Evaluation Loss**: 0.691 (near-random predictions; underfitting observed)

- Label distribution

	Training	Validation	Testing
0	7928	1933	2486
1	7148	1836	2226

```
training_args = TrainingArguments(
    output_dir="./results",
    evaluation_strategy="epoch",
    save_strategy="epoch",
    learning_rate=2e-5,
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    num_train_epochs=3,
    weight_decay=0.01,
    logging_dir="./logs",
    logging_steps=10,
    save_total_limit=2,
    load_best_model_at_end=True,
    metric_for_best_model="f1",
    greater_is_better=True,
)
```

```
training_args = TrainingArguments(
    output_dir="./results",
    evaluation_strategy="epoch",
    save_strategy="epoch",
    learning_rate=1e-5,
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    num_train_epochs=5,
    weight_decay=0.01,
    logging_dir="./logs",
    logging_steps=10,
    save_total_limit=2,
    load_best_model_at_end=True,
    metric_for_best_model="f1",
    greater_is_better=True,
)
```

Initial Results

- **Key Metrics**
  - **Accuracy**: 48.8% → 52.7%
  - **Precision:** 0% → 47.3%
  - **Recall:** 0% → 71.2%
  - **F1**: 0% → 56.8%
  - **Evaluation Loss**: 0.691 (near-random predictions; underfitting observed)

Compare with Baseline

- **Baseline (CodeBERT)**
  - **Accuracy**: 52.8%
  - **Precision:** 0%
  - **Recall:** 0%
  - **F1**: 0%
- **Fine-tuned CodeBERT**
  - **Accuracy**: 52.8%
  - **Precision:** 47.3%
  - **Recall:** 71.2 %
  - **F1**: 56.8 %

Challenges

- **Dataset Limitations**
  - Code snippets may not fully represent the context of vulnerabilities.
  - Emerging vulnerabilities lack labeled data
- **Model Complexity**
  - Capturing semantic and structural nuances in code is challenging
- **Training Constraints**
  - Limited computational resources require careful optimization (e.g., batch size, gradient accumulation)

Potential Future Work

- **Enhancements**
  - Incorporate structural representations like Abstract Syntax Trees (ASTs) or control flow graphs (CFGs)
  - Experiment with ensemble models combining CodeBERT and GNNs
- **Broader Applications**
  - Extend to multi-class classification (e.g., vulnerability types or severity levels)
  - Deploy in real-time CI/CD pipelines for continuous vulnerability scanning

Conclusion

- This project extend CodeBERT for detecting vulnerabilities in source code as part of static analysis
- Demonstrated the potential of transformer-based models in secure software development
- With further fine-tuning and enhancements, this approach could significantly reduce the reliance on manual reviews and traditional tools

Thank you!

Questions?