Assessment of Software Vulnerability Contributing Factors using XAI (eXplainable AI) Techniques

Contributor: Ding Li, Prof. Yan Liu
Presenter: Ding Li

Department of Electrical and Computer Engineering,
Concordia University, Montreal, Canada
Presentation Overview

• Introduction
  • Background of software vulnerability detection
  • XAI Feature importance explanation

• Related work
  • Factors in Code Representation Techniques

• Research Questions

• Methodology
  • Text-based factors assessment
  • Graph-based factors assessment (This talk focus in Graph-based)

• Experiment results (Graph-based factors assessing).

• Conclusion, Contribution, Reference, Discussion.
Background — software vulnerability detection

Software vulnerability:
- Flaws or weaknesses in a software program.
- can be exploited to perform unauthorized actions, such as breaching data or disrupting services[1].

How to detect software vulnerabilities?

**Manual Detection**

**Pros:**
- In-depth understanding of the system's functionality

**Cons:**
- Time-consuming.
- Non scalable

**Static & Dynamic Analysis Tools**

**Pros:**
- Automated and scalable

**Cons:**
- High false positives
- Limited by rule sets

**Machine Learning Detection**

**Pros:**
- Automated and scalable
- Continue learning from data
- Reduces false positives

**Cons:**
- Computationally expensive
- Transparent concerns

Information summarized from [3]
Background – software vulnerability detection

Give a vulnerable code snippet, what are the contributing factors, and how to measure their features impact on the prediction results of machine learning based detection approach?

Factors that manual detection rely on:

- Semantic Tokens:
  - HashMap intHashMap, =
  - new, LinkedHashMap, data, ()
- Syntax Meanings:
  - LinkedHashMap call data
  - new init LinkedHashMap(data)
  - Hashmap decl LinkedHashMap(data)
  - LinkedHashMap(data) expr initHashMap

-> Memory Allocation with Excessive Size Value

What factors, and how the features affect the machine learning based detection decision?
Presentation Overview

• Introduction
  • Background of code vulnerability
  • XAI Feature importance explanation

• Related work
  • Factors in Code Representation Techniques

• Research Questions

• Methodology
  • Text-based factors assessment
  • Graph-based factors assessment

• Graph-based factors assessing results.

• Conclusion, Contribution, Reference, Discussion.
Background - XAI Feature Importance Explanation

XAI (eXplainable AI) feature importance explanation, as a branch of XAI method, helps user to understand the model’s predictions and specific influence of individual features contributing to these predictions[2].

Figure 2: XAI (EXplainable AI) feature importance explanation gives the quantified results of feature’s impact on model’s predictions.
Presentation Overview

• Introduction
  • Background of code vulnerability
  • XAI Feature importance explanation

• Related work
  • Factors in Code Representation Techniques

• Research Questions

• Methodology
  • Text-based factors assessment
  • Graph-based factors assessment

• Experiment results (Graph-based factors assessing).

• Conclusion, Contribution, Reference, Discussion.
Related work - Factors in Code Representation Techniques

Figure 3: The taxonomy of factors under various code feature representation techniques. Our contribution: extension to the feature factor graininess from work [4].
Related work

Text-based code vulnerability detection:
- primarily focus on refining processes and improving model for higher detection accuracy, transferring knowledge from nature language process (Transformer models[5,6,7], CodeBERT[8], etc).
- Factors Explanation:
  - **Token type**: Both code body and comments matters[9]; Transformer-based model also value separator symbols (commas, etc.)[10].
  - **Token Length**: Limiting token length leads to information loss [12] (max 512 tokens in [12]).
  - **Token Attention value**:
    - In NLP task, attention values potentially indicate token importance[13], however caution is needed for this conclusion[14].
    - In code vulnerability task, attention value explanations stay at individual code snippets level[15,16] by mapping attention value, lack of cross-validation with XAI methods for representing importance.
Related work

Graph-based code vulnerability detection:

- **Code Representation:**
  - Abstract Syntax Tree is majority of the exiting study [4], but combinations multiple graphs based on AST become recently trend [3].
  - State-of-the-art models Code2Vec(AST) [18], GraphCodeBert(DFG)[19], Devign (Combine)[20], GraphVecCode(AST)[21].
- **Factors explanation:**
  - Code2Vec[18] and MIL[22]techniques provide explainability at the AST path level, suggesting the importance of paths on individual code snippets.
  - Refer to syntactic constructs, *names, identifier, and parameter* play a significant role in vulnerability tasks, as highlighted by various studies[23,24,25].
  - CWE(Common Weakness Enumeration) developed the weakness type and gather similar types into a tree structure.

Despite insights on certain crucial identifiers, a gap exists in the complete evaluation of all syntactic constructs across different vulnerability types, suggesting the need for further exploration in this area.
Research Questions

RQ1. How do measure the code textual factors influence on the performance of transformer-based models in code vulnerability detection tasks?

*Three factors: Code Token Length, Code Token Type, Code Token Attention Value*

RQ2. How do syntactic constructs in Abstract Syntax Trees (AST) contribute to model’s prediction for different software vulnerability types?

*Aim to identify and quantify the impact of syntactic constructs linked to code vulnerabilities*

RQ3. How do the CWE similarity summarized by syntactic constructs’ importance explanations align with expert-defined results?

*To evaluate the effectiveness of similarity results from XAI approach with expert-defined baseline.*
Presentation Overview

• Introduction
  • Background of code vulnerability
  • XAI Feature importance explanation

• Related work
  • Factors in Code Representation Techniques

• Research Questions

• Methodology
  • Text-based factors assessment
  • Graph-based factors assessment

• Experiment results (Graph-based factors assessing).

• Conclusion, Contribution, Reference, Discussion.
To answer RQ2: How do syntactic constructs in Abstract Syntax Trees (AST) contribute to model’s prediction for different software vulnerability types?

**Dataset:** Juliet, OWASP, Draper benchmark projects.

**GraphCodeVec**[21]: novel sota model for creating a generalizable graph-based, task-agnostic code learning that leverages Graph Convolutional Networks (GCN)

**XAI methods:** SHAP[26], Mean-Centroid Preddiff[13].
Methodology
Graph-based factors assessment - CWE Similarity (RQ3)

To answer RQ3: How do the CWE similarity summarized by syntactic constructs' importance explanations align with expert-defined results?

Step 1 - Summarize CWE similarity from XAI explanation

Step 2 - Cross validation with baseline

Figure 5. The overall framework of XAI summarized CWE similarity validation with baseline

CWE Similarity Baseline: https://cwe.mitre.org/data/definitions/1000.html
To answer RQ3: How do the **CWE similarity** summarized by syntactic constructs' importance explanations align with expert-defined results?

**Step1 - Summarize CWE similarity from XAI explanation**

1. **Syntactic Construct Feature Importance Order of CWE_i**
2. **Ranking Distance Measurement**
3. **CWE Similarity Results from XAI**
4. **CWE Similarity Validation Results**

**Step2 - Cross validation with baseline**

We define three **metrics** to compare CWE similarity from our XAI approach and baseline.

- **TopN Hit Rate**: if CWE similarity pair in baseline is within the TopN similar of XAI results:
  \[
  \text{Top-N Hit Rate}_{CWE} = \begin{cases} 
  1, & \text{if hit condition} \\
  0, & \text{otherwise}
  \end{cases}
  \]

- **Avg Similarity score**: calculates the average normalized similarity score for all CWEs within a category in the baseline table
  \[
  \bar{S}_i = \frac{1}{|CWE_i\text{similar}|} \sum \frac{\rho(i, CWE_i\text{similar})}{\max \rho(i)}
  \]

- **Mean Reciprocal Rank**: calculates the reciprocal of the rank of the first correct answer, within the XAI ranking list.
  \[
  MRR = \frac{1}{|Q|} \sum_{i=1}^{\left\lvert Q \right\rvert} \frac{1}{\text{rank}_i}
  \]

**Figure 5. The overall framework of XAI summarized CWE similarity validation with baseline**

[CWE Similarity Baseline: https://cwe.mitre.org/data/definitions/1000.html](https://cwe.mitre.org/data/definitions/1000.html)
Presentation Overview

• Introduction
  • Background of code vulnerability
  • XAI Feature importance explanation

• Related work
  • Factors in Code Representation Techniques

• Research Questions

• Methodology
  • Text-based factors assessment
  • Graph-based factors assessment

• Experiment results (Graph-based factors assessing).

• Conclusion, Contribution, Reference, Discussion.
Experiment Results - Syntactic constructs feature explanation (RQ2)

Step 1-data preprocessing

Figure 6-1. Source Code: a code snippet of CWE789

Figure 6-2. AST structure code: extract AST information of the code snippet, includes code token node, and the AST path.

Figure 6-3. Masking AST path with syntactic construct (left unmarked, right marked)
Experiment Results - Syntactic constructs feature explanation (RQ2)

**Step 2 & 3 - model pre-training, syntactic constructs influence explanation**

From step 2, we observe GraphCodeVec + TextCNN perform consistent well.

From Figure 4

**Figure 7: syntactic constructs feature explanations results (Step 3) for all CWEs**
RQ2. How do syntactic constructs in Abstract Syntax Trees (AST) contribute to model’s prediction for different software vulnerability types?

Aim to identify and quantify the impact of syntactic constructs linked to code vulnerabilities

- The importance of syntactic constructs varies from CWE type, and the dataset.

- However, constructs such as statement, name, and parameters have a general high impact on code vulnerability types.
  - Similar findings that names, identifier(statement), and parameter play a significant role in vulnerability tasks, in studies[23,24,25]

- Several CWE type sharing high similarity based on feature importance order. (CWE 78, 79, 89)
  - As a motivation of RQ3
Experiment Results - CWE Similarity (RQ3)

**Step1: Summarize CWE similarity from XAI explanation**

- **CWE120 and CWE119 are more similar.**
- **CWE469 & CWE 476 are less similar with CWE 119&120.**

**Figure 8: CWE similarity distance value from syntactic construct feature importance based on XAI approach**

From Figure 5

**From Figure 5**

- **CWE119 and CWE120 are more similar.**
- **CWE469 & CWE 476 are less similar with CWE 119&120.**
Experiment Results - CWE Similarity (RQ3)

**Step2: CWE similarity results cross validation with baseline**

From Figure 5

<table>
<thead>
<tr>
<th>Category</th>
<th>Similar CWEs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Path traversal and resource management issues</td>
<td>CWE22, CWE23, CWE36</td>
</tr>
<tr>
<td>Trust boundaries and privilege management</td>
<td>CWE500, CWE501, CWE15</td>
</tr>
<tr>
<td>Buffer errors</td>
<td>CWE119, CWE120</td>
</tr>
<tr>
<td>Injection vulnerabilities</td>
<td>CWE78, CWE79, CWE89, CWE90, CWE643, CWE789</td>
</tr>
<tr>
<td>Cryptographic and sensitive data handling issues</td>
<td>CWE327, CWE328, CWE330, CWE614</td>
</tr>
<tr>
<td>Use of pointer subtraction to determine size</td>
<td>CWE469</td>
</tr>
<tr>
<td>NULL pointer dereference</td>
<td>CWE476</td>
</tr>
</tbody>
</table>

Table1: CWE categorized by baseline similarities

Table2: CWE Similarity Evaluation Results

<table>
<thead>
<tr>
<th>CWE</th>
<th>Top-1</th>
<th>Top-3</th>
<th>Top-5</th>
<th>MRR</th>
<th>$\delta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>CWE23</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.572</td>
<td>0.802</td>
</tr>
<tr>
<td>CWE327</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.736</td>
<td>0.628</td>
</tr>
<tr>
<td>CWE330</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.728</td>
<td>0.247</td>
</tr>
<tr>
<td>CWE79</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.728</td>
<td>0.250</td>
</tr>
<tr>
<td>CWE89</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0.630</td>
<td>0.328</td>
</tr>
<tr>
<td>CWE22</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.115</td>
<td>0.118</td>
</tr>
<tr>
<td>CWE78</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.687</td>
<td>0.622</td>
</tr>
<tr>
<td>CWE90</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.743</td>
<td>0.610</td>
</tr>
<tr>
<td>CWE501</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.767</td>
<td>0.774</td>
</tr>
<tr>
<td>CWE614</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.738</td>
<td>0.761</td>
</tr>
<tr>
<td>CWE643</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.738</td>
<td>0.761</td>
</tr>
<tr>
<td>CWE328</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.233</td>
<td>0.620</td>
</tr>
<tr>
<td>CWE36</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.122</td>
<td>0.661</td>
</tr>
<tr>
<td>CWE15</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CWE500</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CWE789</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CWE469</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>CWE476</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>CWE119</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>CWE120</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

| Average   | 0.778 | 0.833 | 0.889 | 0.696 | 0.677  |

Note: Top-1/3/5 represents the Top-N Hit rate, MRR represents Mean Reciprocal Rank, and $\delta$ represents the Average Normalized Similarity Score. CWE469 and CWE476 do not have a similar CWE in the datasets scope.

- Our CWE similarity summary from XAI effectively align with baseline with 77.8% Top1 Hit rate.
RQ3. How do the **CWE similarity** summarized by syntactic constructs' importance explanations align with expert-defined results?

To evaluate the effectiveness of similarity results from XAI approach with expert-defined baseline.

- Our CWE similarity evaluation method efficiently identifies related CWEs, achieving a hit rate of 77.8% for the most similar CWE (Top-1) and 88.9% for the top five similar CWEs (Top-5).

- In our evaluation, only two instances - CWE22 and CWE36 (2 out of 20) did not meet the baseline similarities.
Presentation Overview

• Introduction
  • Background of code vulnerability
  • XAI Feature importance explanation

• Related work
  • Factors in Code Representation Techniques

• Research Questions

• Methodology
  • Text-based factors assessment
  • Graph-based factors assessment

• Experiment results (Graph-based factors assessing).

• Conclusion Contribution, Reference, Discussion.
We extend the taxonomy of code representation techniques by examining them at the feature factor level.

Our study provides a comprehensive evaluation of the importance of all syntactic constructs, complementing previous studies that focused only on top-valued constructs.

By leveraging rankings of syntactic constructs, we effectively analyze and validate CWE similarity, comparing our results to expert-defined baselines to confirm the effectiveness of our XAI explanation approach.
## Syntactic Constructs and Categories in the software

**TABLE I: Syntactic Constructs in Abstract Syntax Tree**

<table>
<thead>
<tr>
<th>Meta Syntactic Constructs [84]</th>
<th>Syntactic Constructs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name, Base Elements</td>
<td>&lt;name&gt;, &lt;block_comment&gt;, &lt;literal&gt;, ...</td>
</tr>
<tr>
<td>Statements</td>
<td>&lt;assert&gt;, &lt;block&gt;, &lt;break&gt;, &lt;case&gt;, &lt;if_stmt&gt;, &lt;continue&gt;, &lt;default&gt;, &lt;do&gt;, &lt;empty_stmt&gt;, &lt;expr_stmt&gt;, &lt;for&gt;, &lt;if_stmt&gt;, &lt;label&gt;, &lt;return&gt;, &lt;switch&gt;, &lt;while&gt;, &lt;default&gt;, ...</td>
</tr>
<tr>
<td>Statement subelements</td>
<td>&lt;expr&gt;, &lt;condition&gt;, &lt;control&gt;, &lt;else&gt;, &lt;iftype=&quot;elseif&quot;&gt;, &lt;expr&gt;, &lt;if&gt;, &lt;incr&gt;, &lt;then&gt;, &lt;type&gt;, &lt;block_content&gt;, ...</td>
</tr>
<tr>
<td>Specifiers</td>
<td>&lt;specifier&gt;, ...</td>
</tr>
<tr>
<td>Declarations, Definitions, Initializations</td>
<td>&lt;lambda&gt;, &lt;function&gt;, &lt;decl_stmt&gt;, &lt;decl&gt;, &lt;init&gt;, &lt;new&gt;, ...</td>
</tr>
<tr>
<td>Classes, Interfaces, Annotations, and Enums</td>
<td>&lt;annotation&gt;, &lt;class&gt;, &lt;static&gt;, &lt;annotation_defn&gt;, ...</td>
</tr>
<tr>
<td>Expressions</td>
<td>&lt;call&gt;, ...</td>
</tr>
<tr>
<td>Exceptions</td>
<td>&lt;finally&gt;, &lt;throw&gt;, &lt;throws&gt;, &lt;try&gt;, &lt;catch&gt;, ...</td>
</tr>
</tbody>
</table>
Appendix 2 - Syntactic constructs feature explanation (RQ2)

Step 2 & 3 - model pre-training, syntactic constructs influence explanation

From step 2, we observe GraphCodeVec + TextCNN perform consistent well (88.4%, 89.9% on F1-Score for Juliet, Draper dataset) than GraphCodeVec + Random Forest or Transformer. We preform XAI based on GraphCodeVec + TextCNN.

Figure 7: Feature explanations results (Step 3) for CWE from OWASP dataset.