Towards a Prototype Based Explainable JavaScript Vulnerability Prediction Model

Balázs Mosolygó, Norbert Vándor,
Gábor Antal, Dr. Péter Hegedűs
Introduction

- Systems are increasingly complex → manual vulnerability checking is not efficient enough
  - Solution: Automatic software analysis tools
    - Machine learning based approaches offer many benefits
  - Issues:
    - Prediction models are „black-boxes”
      - No explanation provided for the prediction
    - Prediction context is too coarse-grained (file or function)
      - Not actionable by developers

- Our proposed solution for vulnerability detection
  - Works at line level
  - Provides explanation for the prediction (the code line in the training set that is the most "similar")
  - Light-weight and extendable
Approach Overview

VLR: vulnerable lines repository

SUT: system under test
Approach

- Apply NLP based embedding to code
  - Tokenization was done with JS Tokens
  - Word2vec (a vector representation for each token type)
- Token types + non-default tokens
  - JS language keywords (if, for, while etc.) are kept
  - Special string literals (SQL, HTML etc.) are added
VLR

- The Vulnerable Line Repository forms the basis of vulnerability detection
  - It consists of known vulnerable lines
    - And their tokenized forms
  - Derived from vulnerability fix patches collected from public repositories [1] and data mining
  - Filtered manually
    - To remove false positives

Detection of Vulnerable Lines

Vulnerability likelihood for one line in SUT:

1. Get the line from VLR with the minimal cosine distance to SUT line
2. Calculate complexity of the SUT line
   \[ 1 - \frac{1}{\text{uniqueTokenCount}} \]
3. Get the average of the distance and the complexity measure
4. If the average is above a predefined threshold, mark it as vulnerable
Results

- We split the patches in the dataset into two sets in a 90%-10% ratio
  - We used the 10% dev set to define the threshold needed to derive a prediction
  - On the remaining 90% of the data, we applied 10-fold cross-validation
    - Split the data for train and test and used lines in train as VLR
    - Tested on lines in test

<table>
<thead>
<tr>
<th>file_lines</th>
<th>flagged_lines</th>
<th>vuln_lines</th>
<th>flagged_vuln_lines</th>
<th>%_flagged</th>
<th>%_is_vuln</th>
<th>%_vuln_flagged</th>
<th>flagged_ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANR</td>
<td>4639.4</td>
<td>2084.2</td>
<td>12.2</td>
<td>10.1</td>
<td>44.49</td>
<td>82.11</td>
<td>1.84</td>
</tr>
<tr>
<td>ACR</td>
<td>4639.4</td>
<td>576.8</td>
<td>12.2</td>
<td>7.2</td>
<td>12.43</td>
<td>59.01</td>
<td>4.74</td>
</tr>
<tr>
<td>MiNR</td>
<td>15139</td>
<td>6195</td>
<td>38</td>
<td>18</td>
<td>40.92</td>
<td>47.36</td>
<td>1.15</td>
</tr>
<tr>
<td>MiCR</td>
<td>11044</td>
<td>2326</td>
<td>26</td>
<td>17</td>
<td>21.06</td>
<td>65.38</td>
<td>3.1</td>
</tr>
<tr>
<td>MaNR</td>
<td>2686</td>
<td>1046</td>
<td>7</td>
<td>7</td>
<td>38.94</td>
<td>100.0</td>
<td>2.56</td>
</tr>
<tr>
<td>MaCR</td>
<td>2614</td>
<td>228</td>
<td>6</td>
<td>6</td>
<td>8.72</td>
<td>100.0</td>
<td>11.46</td>
</tr>
</tbody>
</table>
Results – a Sample

```javascript
- tail = normalizeString(path.slice(rootEnd), !isAbsolute, '\'');
+ tail = normalizeString(path.slice(rootEnd), !isAbsolute, '\', isPathSeparator);

IdentifierName = IdentifierName ( IdentifierName . IdentifierName ( IdentifierName ), ! IdentifierName , StringLiteral );

- related = path.parse(path.join(__dirname, './assets', relative));
+ related = decodeURIComponent(path.join(__dirname, './assets/styles.css'));

IdentifierName = IdentifierName . IdentifierName ( IdentifierName . IdentifierName ( IdentifierName , StringLiteral , IdentifierName ) );
```
Conclusion

- Our method can produce useful results
- Our complexity rule brought considerable improvement
- Method flags ~13% of all lines, which contain 60% of all vulnerable lines
- Possible improvements
  - More sophisticated rules
  - Improving score aggregation method (AVG and 1-1/x are both somewhat arbitrary)
Acknowledgement

The presented work was carried out within the SETIT Project (2018-1.2.1-NKP-2018-00004) and supported by the Ministry of Innovation and Technology NRDI Office within the framework of the Artificial Intelligence National Laboratory Program (MILAB).

Furthermore, Péter Hegedűs was supported by the Bolyai János Scholarship of the Hungarian Academy of Sciences and the ÚNKP-20-5-SZTE-650 New National Excellence Program of the Ministry for Innovation and Technology.