UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
UNIVERSITY OF SZEGED
Department of Software Engineering

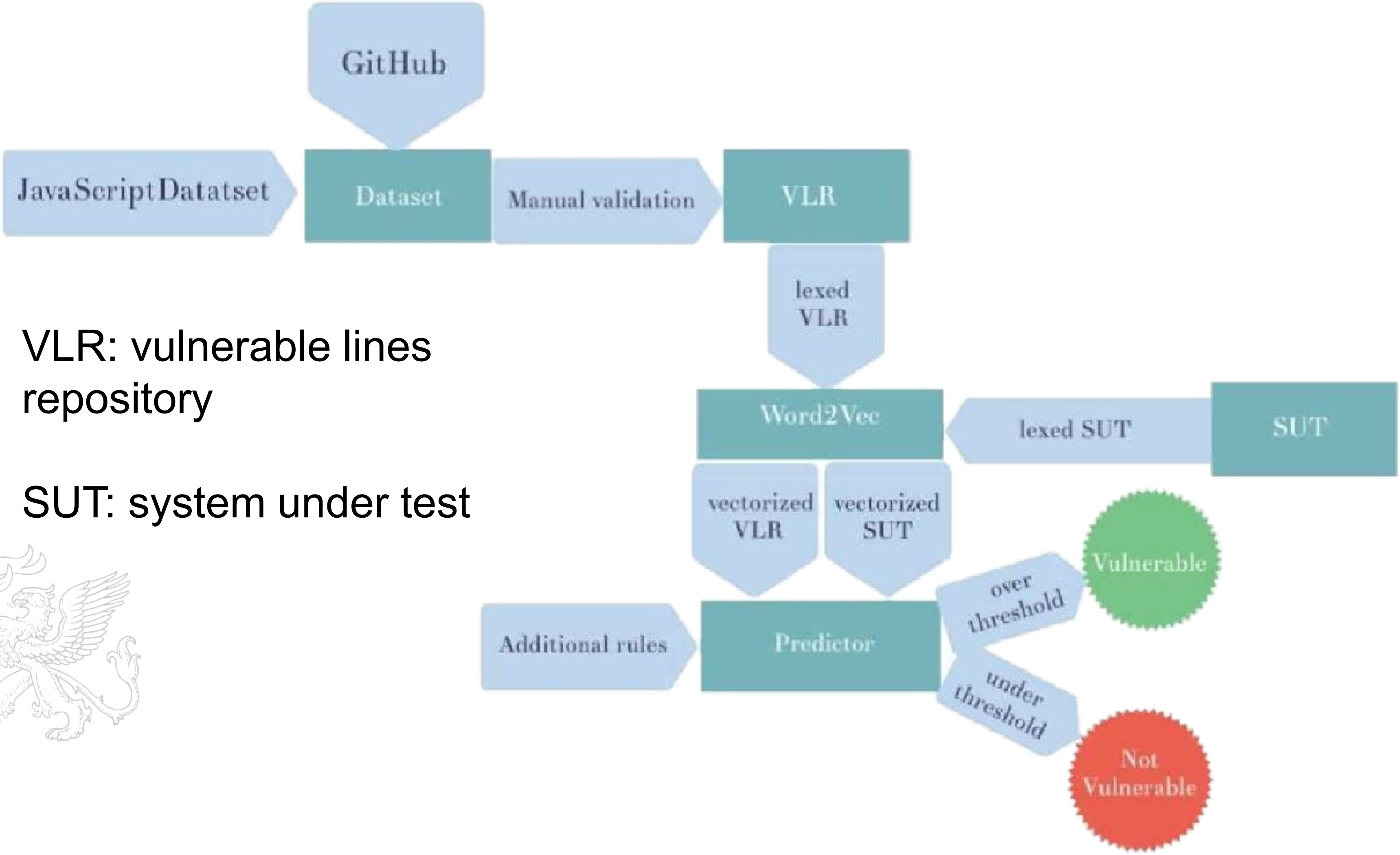
Towards a Prototype Based Explainable JavaScript Vulnerability Prediction Model

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

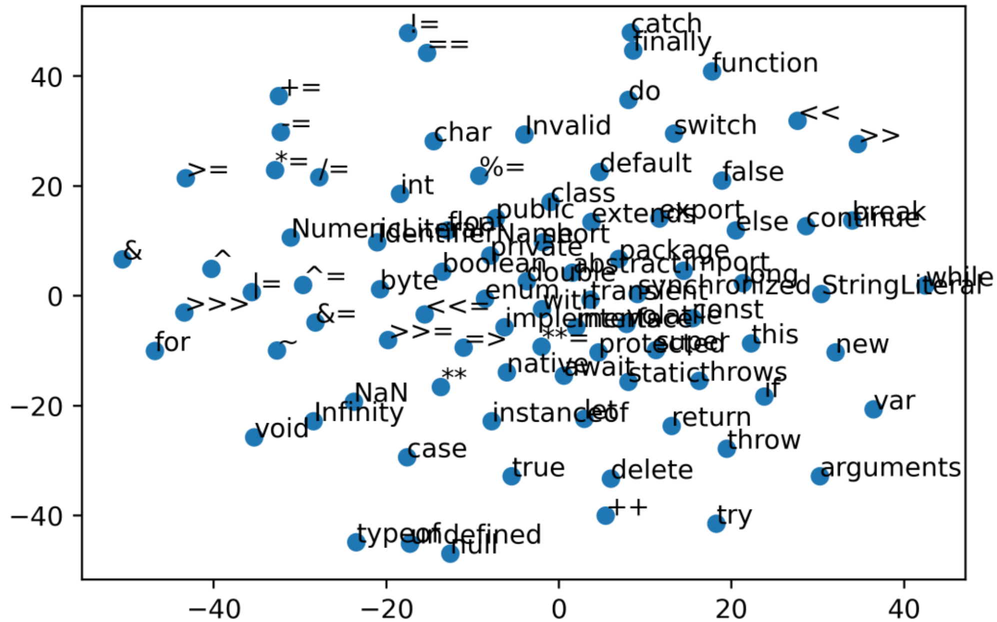


VLR: vulnerable lines repository

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

[1] Rudolf Ferenc, Péter Hegedűs, Péter Gyimesi, Gábor Antal, Dénes Bán, and Tibor Gyimóthy. Challenging Machine Learning Algorithms in Predicting Vulnerable JavaScript Functions. In Proceedings of the 7th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering. 2019. IEEE Press, 8–14.

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
UNIVERSITY OF SZEGED
Department of Software Engineering

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 – 1/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

	file_lines	flagged_lines	vuln_lines	flagged_vuln_lines	%_flagged	%_is_vuln	%_vuln_flagged	flagged_ratio
ANR	4639.4	2084.2	12.2	10.1	44.49	0.48	82.11	1.84
ACR	4639.4	576.8	12.2	7.2	12.43	1.24	59.01	4.74
MiNR	15139	6195	38	18	40.92	0.29	47.36	1.15
MiCR	11044	2326	26	17	21.06	0.73	65.38	3.1
MaNR	2686	1046	7	7	38.94	0.66	100.0	2.56
MaCR	2614	228	6	6	8.72	2.63	100.0	11.46

Results – a Sample

```
-       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)

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
UNIVERSITY OF SZEGED
Department of Software Engineering

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.