CE 815 – Secure Software Systems

ML-Based Vulnerability Detection Methods (Hoppity)

Mohammad Haddadian/Mehdi Kharrazi
Department of Computer Engineering
Sharif University of Technology

Acknowledgments: Some of the slides are fully or partially obtained from other sources. A reference is noted on the bottom of each slide, when the content is fully obtained from another source. Otherwise a full list of references is provided on the last slide.
Introduction

• Vulnerability detection as first step
• Then, Vulnerability repair
Problem

Source-code analysis is:

- Undecidable
- Noisy
- Rules are hand written
- Tailored to specific code bases / bug patterns
Javascript Challenges

- Incorrect operators
- Incorrect identifiers
- Accessing undefined properties
- Mishandling variable scopes
- Type incompatibilities
Example

```javascript
function clearEmployeeListOnLinkClick() {
    document.querySelector("a").addEventListener("click",
        function(event) {
            document.querySelector("ul").innerHTML = "";
        });
}

if (matches) {
    return {
        episode: Number(matches.groups.episode),
        hosts: matches.groups.hosts.split(/([^,]+)\sand\s/).map(el => S(el).trim()).s
    };
}
```

(a) `innerHTML` should have been `innerHTML`.

```javascript
module.exports = function (grunt) {
    grunt.initConfig({
        executes: [...],
        copy: [...],
        checktextdomain: [...]
        wp_readme_to_markdown: [...],
        makepot: [...])
    ...
    grunt.registerTask('default', ['wp_readme_to_markdown', 'makepot', 'execute', 'checktextdomain']);
}
```

(b) Highlighted parentheses should have been removed.

```javascript
export default {
    computed: {
        level () {
            return dictMap.skillLevel[
                parseInt((this.value === 0 ? 1 : this.value)/20)];
        }
    },
}
```

(c) `copy` function should have also been included in the highlighted list.

(d) `parseInt` should have been removed because `===` implies `this.value` is an integer.

Fall 1402

CE 815 - Vulnerability Analysis

[Hoppity]
Solution

Leverage large amounts of Javascript fixes on Github to locate and repair bugs
Steps

- Represent source code
- Represent fixes
- Learning
Goal

1- function add(a) { a + b; }

2- function add(a, b) { a + b; }

3- function add(a, b) { return a + b; }
Source code representation

- AST
- ValueLink

```latex
function add(a) { a + b; }
```
Fix representation

• Graph Edits
Low level primitives

- Location
- Value
- Type
Graph edit operators

- add node
- del node
- replace type
- replace value
- No Action
Graph transformation
Inference

Learn

Request Code

Pool & Beam Search

Result

[Diagram showing a process of inference with nodes A, B, and C, and arrows indicating the flow of information.]
Dataset

- OneDiff (just one change)
- ZeroOneDiff (zero or one edit)
- ZeroOneTwoDiff (zero, one or two edits)

<table>
<thead>
<tr>
<th></th>
<th>ADD</th>
<th>REP_TYPE</th>
<th>REP_VAL</th>
<th>DEL</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>train</td>
<td>6,473</td>
<td>1,864</td>
<td>251,097</td>
<td>31,281</td>
<td>290,715</td>
</tr>
<tr>
<td>validate</td>
<td>790</td>
<td>245</td>
<td>31,357</td>
<td>3,957</td>
<td>36,349</td>
</tr>
<tr>
<td>test</td>
<td>796</td>
<td>233</td>
<td>31,387</td>
<td>3,945</td>
<td>36,361</td>
</tr>
</tbody>
</table>

Table 1: Statistic of OneDiff dataset. See appendix for more information of other dataset.
## Evaluation

<table>
<thead>
<tr>
<th></th>
<th>Total</th>
<th>Location</th>
<th>Operator</th>
<th>Value</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Top-3</td>
<td>Top-1</td>
<td>Top-3</td>
<td>Top-1</td>
<td>Top-3</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td>26.1</td>
<td>14.2</td>
<td>35.5</td>
<td>20.4</td>
<td>34.4</td>
</tr>
<tr>
<td>ADD</td>
<td>52.9</td>
<td>39.2</td>
<td>69.6</td>
<td>51.4</td>
<td>70.6</td>
</tr>
<tr>
<td>REP_VAL</td>
<td>23.4</td>
<td>11.9</td>
<td>33.3</td>
<td>18.5</td>
<td>31.7</td>
</tr>
<tr>
<td>REP_TYPE</td>
<td>71.7</td>
<td>52.4</td>
<td>73.0</td>
<td>52.8</td>
<td>79.4</td>
</tr>
<tr>
<td>DEL</td>
<td>39.6</td>
<td>24.8</td>
<td>44.0</td>
<td>27.5</td>
<td>45.8</td>
</tr>
<tr>
<td>Random</td>
<td>.08</td>
<td>.07</td>
<td>2.28</td>
<td>1.4</td>
<td>27.7</td>
</tr>
</tbody>
</table>

Table 2: Evaluation of model on the OneDiff dataset: accuracy (%).
Evaluation (cont.)

<table>
<thead>
<tr>
<th>Type</th>
<th>GGNN-Rep</th>
<th>GGNN-Cls</th>
<th>HOPPITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top-1</td>
<td>53.2%</td>
<td>99.6%</td>
<td>90.0%</td>
</tr>
<tr>
<td>Top-3</td>
<td>85.8%</td>
<td>99.6%</td>
<td>94.8%</td>
</tr>
</tbody>
</table>

Table 3: REP_TYPE accuracies with location+op.

<table>
<thead>
<tr>
<th>Value</th>
<th>GGNN-Rep</th>
<th>GGNN-RNN</th>
<th>HOPPITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top-1</td>
<td>63.8%</td>
<td>60.3%</td>
<td>69.1%</td>
</tr>
<tr>
<td>Top-3</td>
<td>67.6%</td>
<td>63.6%</td>
<td>73.4%</td>
</tr>
</tbody>
</table>

Table 4: REP_VAL accuracies with location+op.

<table>
<thead>
<tr>
<th>Bug Type</th>
<th>Amount</th>
<th>TAJJS</th>
<th>HOPPITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Undefined Property</td>
<td>7</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Functional Bug</td>
<td>11</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Refactoring</td>
<td>12</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>30</strong></td>
<td><strong>0</strong></td>
<td><strong>5</strong></td>
</tr>
</tbody>
</table>

Table 5: Overall OneDiff accuracy with location.

Table 6: Comparison with TAJJS.
Machine Learning already solved many problems in computer security
Unfortunately not... 😞
Motivation—Historical Examples

Network intrusion detection: The base rate fallacy

- Intrusion detectors should have low false positive rates (FPR)
- 'Low' FPR often still corresponds to large number of false positives

Android malware detection: Spatio-temporal bias inflating performance

- Models trained with access to 'future' information
- Unrealistic class balance inflates performance
Overview

1. Identification of common pitfalls
   - 10 subtle issues affecting ML for security
   - Recommendations for avoiding them

2. Survey on the prevalence of pitfalls
   - Review of 30 top papers in security
   - Pitfalls are widespread

3. Case studies demonstrating impact of pitfalls
   - Mobile malware detection
   - Vulnerability discovery
   - Source code authorship attribution
   - Network intrusion detection

[Arp]
ML Pipeline and Pitfalls

Data Collection and Labeling
- P1 Sampling bias
- P2 Label Inaccuracy

System Design and Learning
- P3 Data snooping
- P4 Spurious correlations
- P5 Biased parameters

Performance Evaluation
- P6 Inappropriate baselines
- P7 Inappropriate measures
- P8 Base rate fallacy

Deployment and Operation
- P9 Lab-only evaluation
- P10 Inappropriate threat model
**P1 – Sampling Bias.** The collected data does not sufficiently represent the true data distribution of the underlying security problem [1, 30, 33].

**P2 – Label Inaccuracy.** The ground-truth labels required for classification tasks are inaccurate, unstable, or erroneous, affecting the overall performance of a learning-based system [85, 144].

**P3 – Data Snooping.** A learning model is trained with data that is typically not available in practice. Data snooping can occur in many ways, some of which are very subtle and hard to identify [1].

**P4 – Spurious Correlations.** Artifacts unrelated to the security problem create shortcut patterns for separating classes. Consequently, the learning model adapts to these artifacts instead of solving the actual task.

**P5 – Biased Parameter Selection.** The final parameters of a learning-based method are not entirely fixed at training time. Instead, they indirectly depend on the test set.

**P6 – Inappropriate Baseline.** The evaluation is conducted without, or with limited, baseline methods. As a result, it is impossible to demonstrate improvements against the state of the art and other security mechanisms.

**P7 – Inappropriate Performance Measures.** The chosen performance measures do not account for the constraints of the application scenario, such as imbalanced data or the need to keep a low false-positive rate.

**P8 – Base Rate Fallacy.** A large class imbalance is ignored when interpreting the performance measures leading to an overestimation of performance.

**P9 – Lab-Only Evaluation.** A learning-based system is solely evaluated in a laboratory setting, without discussing its practical limitations.

**P10 – Inappropriate Threat Model.** The security of machine learning is not considered, exposing the system to a variety of attacks, such as poisoning and evasion attacks.
Prevalence Study

- **Sampling Bias**: Present (18), Partly Present (3), Discussed (6)
- **Label Inaccuracy**: Present (3), Partly Present (3), Discussed (6)
- **Data Snooping**: Present (17), Partly Present (5), Discussed (5)
- **Spurious Correlations**: Present (6), Partly Present (1)
- **Biased Parameters**: Present (3), Partly Present (2)
- **Inappropriate Baseline**: Present (6), Partly Present (2)
- **Inappropriate Measures**: Present (10), Partly Present (6), Discussed (6)
- **Base Rate Fallacy**: Present (3), Partly Present (6), Discussed (3)
- **Lab-Only Evaluation**: Present (14), Partly Present (2), Discussed (3)
- **Inappropriate Threat Model**: Present (5), Partly Present (1), Discussed (14)
Impact Analysis

Android Malware Detection
- P1: Sampling Bias
- P4: Spurious Correlations
- P7: Inappropriate Performance Measures

Authorship Attribution
- P1: Sampling Bias
- P4: Spurious Correlations

Vulnerability Discovery
- P2: Label Inaccuracy
- P4: Spurious Correlations
- P6: Inappropriate Baselines

Network Intrusion Detection
- P6: Inappropriate baselines
- P9: Lab-only evaluation

[Arp]
Acknowledgments