Using Machine Learning to Identify Security Issues in Open-Source Libraries

Asankhaya Sharma
Yaqin Zhou
SourceClear
Outline

- Overview of problem space
- Unidentified security issues
- How Machine Learning can help
- Results
- WOPR Demo
Open-Source Library Growth

![Graph showing the growth of open-source libraries from 2011 to 2017. The y-axis represents the total number of libraries, ranging from 0 to 1,500,000, and the x-axis represents the years from 2011 to 2017. The graph shows a steady increase in the number of libraries over the years.]
Projection: > 400M Libraries by 2026
Complexity of Libraries has exploded

For every 1 Java library you add to your projects, 4 others are added.

For every one library you add to a Node.js project, 9 others are added.
The Code Cocktail

Open-Source = 90%

- Open-source code (~70%)
- Custom Code (~10%)
- Open-source code (~20%)
Vulnerabilities in Open-Source Libraries

- Known Sources
  - CVEs / NVD
  - Advisories
  - Mailing list disclosures

- Unidentified issues
  - Commit logs
  - Bug reports
  - Change logs
  - Pull Requests

Security Issues are often not reported or publically mentioned.
Mining for unidentified vulnerabilities

Number of days a vulnerability is unidentified

Data reviewed over 2016-2017

- GitHub Fix Commits
- GitHub Issues
- GitHub Pull Requests
- Bugzilla Entries
- JIRA Tickets
WOPR: Tool for Reviewing Unidentified Issues

Research Tool

- Vulnerability Knowledgebase
- Workflow Engine
- Queue

Data Source

- Bugzilla
- Changelog
- NVD CVE
- JIRA
- GitHub
- Misc.
- Reserved CVE
- Commit Watcher
- Mailing List

REST API

Machine Learning
Machine Learning for Identifying Vulnerabilities

“do machine learning like the great engineer you are, not like the great machine learning expert you aren’t.”


System overview
ML Pipeline

Generating more triaged data, driving better models
Data collection

- Regular expression to filter out security-unrelated issues
  - Rule sets cover almost all possible expressions related to security issues
- Tracked 8536 projects in 6 languages
  - Tracked languages: Java, Python, Ruby, JavaScript, Objective C, and Go
- Ground truth datasets
  - Professional security researchers label all data, and create vulnerability reports
  - Available at SourceClear Registry

<table>
<thead>
<tr>
<th>Source</th>
<th># of tracked projects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Github</td>
<td>5002</td>
</tr>
<tr>
<td>JIRA</td>
<td>1310</td>
</tr>
<tr>
<td>Bugzilla</td>
<td>2224</td>
</tr>
</tbody>
</table>
## Datasets

### Highly imbalanced

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Size</th>
<th># vulnerability_related</th>
<th>Imbalanced ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commit</td>
<td>12409</td>
<td>1303</td>
<td>10.50%</td>
</tr>
<tr>
<td>GitHub bug reports</td>
<td>10414</td>
<td>612</td>
<td>5.88%</td>
</tr>
<tr>
<td>JIRA bug reports</td>
<td>11145</td>
<td>204</td>
<td>1.83%</td>
</tr>
<tr>
<td>Bugzilla bug reports</td>
<td>2629</td>
<td>1089</td>
<td>41.42%</td>
</tr>
<tr>
<td>Mails</td>
<td>4499</td>
<td>2721</td>
<td>60.48%</td>
</tr>
</tbody>
</table>

Commits & bug reports initial training data: Jan. 2012 - Feb. 2017
Mails initial training data: Feb. 2017 - Aug. 2017
Samples

Noisy, diverse, mixed with urls, directories, variable names...
Features

Commits
- Commit messages
- Comments
  - Most null
- Project name
  - Might impact prediction on projects not in training data
- Name of author
  - Common names and changed names etc

Bug reports
- Title
- Description
- Comments, number of comments
- Number of attachments
- Labels
- Created date and Last edited date

Mails
- Subject
- Content
- Sender
Text feature-Word embedding

- **Word embedding**
  - Map words to vectors so that computers can understand

- **Word2vec**
  - A word embedding method that uses a shallow 2-layer neural network to learn vector representation of words based on similarity in context

```python
>>> word2vec['xss']
array([-0.06691808, 0.01889833, 0.08988539, 0.03727728, 0.09463213,
        0.04498576, 0.02401953, 0.01821383, -0.04510168, ... , -0.00888534], dtype=float32)
```

```python
>>> word2vec.most_similar('xss')
[(u'vulnerability', 0.6009132862091064), (u'attacks', 0.5554373860359192), (u'forgery',
0.4951219856739044), (u'spoofing', 0.49092593789100647), (u'dos', 0.4852156937122345), (u'prevention',
0.48259809613227844), (u'clickjacking', 0.48095956444740295), (u'protection', 0.4675652980804434),
(u'csrf', 0.457594096660614), (u'vuln', 0.4533842206001282)]
```

**Built word2vec model based on 3 million unfiltered commits**
First training attempts-random forest

How Random Forest works?

- **Training**
  - Generate a forest of binary decision trees through randomly sampling a subset of train set and fitting

- **Prediction**
  - Each data sample traverses each tree until it reaches a leaf
  - At the leaf node, each tree creates a vote, the proportion of related votes is the prediction for the data sample

Sample 1
- 2 votes related,
- 1 vote not related,
- Vulnerability risk = 0.67
First training attempts - SVM

How SVM works?

- Mapping data to a high-dimensional feature space so that data points can be categorized
- Kernel - Mathematical function used for transformation
  - Linear
  - Polynomial
  - RBF (Radial basis function)
Unfortunately, these basic binary classifiers, even with best tuning parameters, failed us...
K-fold stacking

1. Split training data into K folds.
2. In each iteration k in [1,K], test the kth fold and train the rest K-1 folds.
3. Feed the 6 basic classifiers: Random Forest, Gaussian Naïve Bayes, K-nearest neighbors, SVM, Gradient Boosting, and Adaboost.
4. Output 6 sets of test results:
   - Random Forest
   - Gaussian Naïve Bayes
   - K-nearest neighbors
   - SVM
   - Gradient Boosting
   - Adaboost
5. Each classifier has K folds of test results.
6. Feed to optimize ensemble of basic classifiers.
7. Logistic regression.
Evaluation-metrics

- **Precision rate**
  - Helps us focus on true vulnerabilities and save manual work on false positives

  \[
  \text{Precision} = \frac{\text{true positive}}{\text{true positive} + \text{false positive}}
  \]

- **Recall rate**
  - Indicates the coverage of existing vulnerabilities

  \[
  \text{Recall rate} = \frac{\text{true positive}}{\text{true positive} + \text{false negative}}
  \]

- Probability threshold of vulnerability to control the tradeoff between two metrics

<table>
<thead>
<tr>
<th>Commits (Total)</th>
<th>Commits (Positive)</th>
<th>Commits (Negative)</th>
<th>True positive</th>
<th>False positive</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>100</td>
<td>900</td>
<td>70</td>
<td>35</td>
</tr>
</tbody>
</table>

Totally (70+35) = 105 shown to researchers
- Precision rate = 70/ (70+35) = 66.67%
- Recall rate = 70/ 100 = 70%
- Filtered commits: 895, 89.5%
Evaluation-test results of commits

Figure: Identification performance of our stacking approach under commits

Table: Comparison with basic classifiers under the same recall rate in commits

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Recall rate</th>
<th>Precision (compared classifier vs. stacking)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear SVM</td>
<td>0.72</td>
<td>0.22 vs. 0.34</td>
</tr>
<tr>
<td>Logistic Regression</td>
<td>0.76</td>
<td>0.22 vs. 0.31</td>
</tr>
<tr>
<td>Random Forest</td>
<td>0.76</td>
<td>0.19 vs. 0.31</td>
</tr>
<tr>
<td>Gaussian Naive Bayes</td>
<td>0.77</td>
<td>0.14 vs. 0.28</td>
</tr>
</tbody>
</table>
Production observation

• The initial 3-months observation from commit watcher
  • Observation period
    • 03/2017 – 05/2017
  • Deployed Model
    • 12-fold stacking with probability threshold 0.75
    • Test precision 0.44 and recall rate 0.62
  • Added ~3000 new projects
    • 2070 -> 5002
  • Precision 0.83 and recall rate 0.74

<table>
<thead>
<tr>
<th>Commits (Total)</th>
<th>Commits (Positive)</th>
<th>Commits (Negative)</th>
<th>True positive</th>
<th>False positive</th>
</tr>
</thead>
<tbody>
<tr>
<td>2268</td>
<td>215</td>
<td>2053</td>
<td>160</td>
<td>32</td>
</tr>
</tbody>
</table>
Production observation

- Track vulnerabilities at large scale and low cost in real time
  - Increased number of projects, e.g., for Github, 4 times more

<table>
<thead>
<tr>
<th>Sources</th>
<th>GitHub</th>
<th>JIRA</th>
<th>Bugzilla</th>
</tr>
</thead>
<tbody>
<tr>
<td>#Projects</td>
<td>10113</td>
<td>1310</td>
<td>2224</td>
</tr>
</tbody>
</table>

- Accelerate vulnerability identification
  - When we firstly added go projects from Github in May, by May 29, 2017*
    - 87 go artefacts created from commit watcher
    - 33 go artefacts created from Github Issues

- Current Github/Jira issues can spot vulnerabilities at the first time
Demo

Daily Research Queue System

Commit Watcher Entries
Thanks!