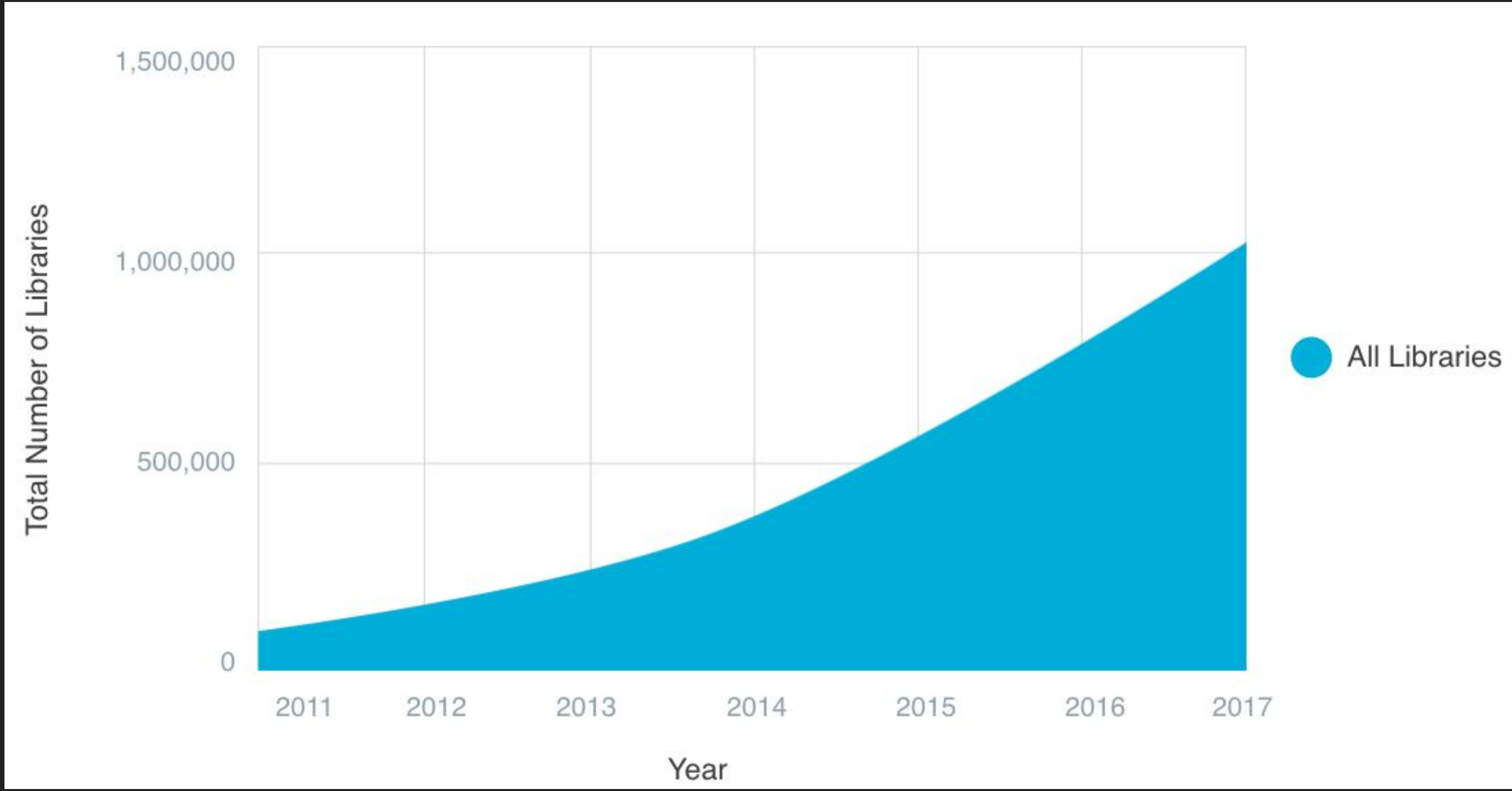
Using Machine Learning to Identify Security Issues in Open-Source Libraries

Asankhaya Sharma
Yaqin Zhou
SourceClear

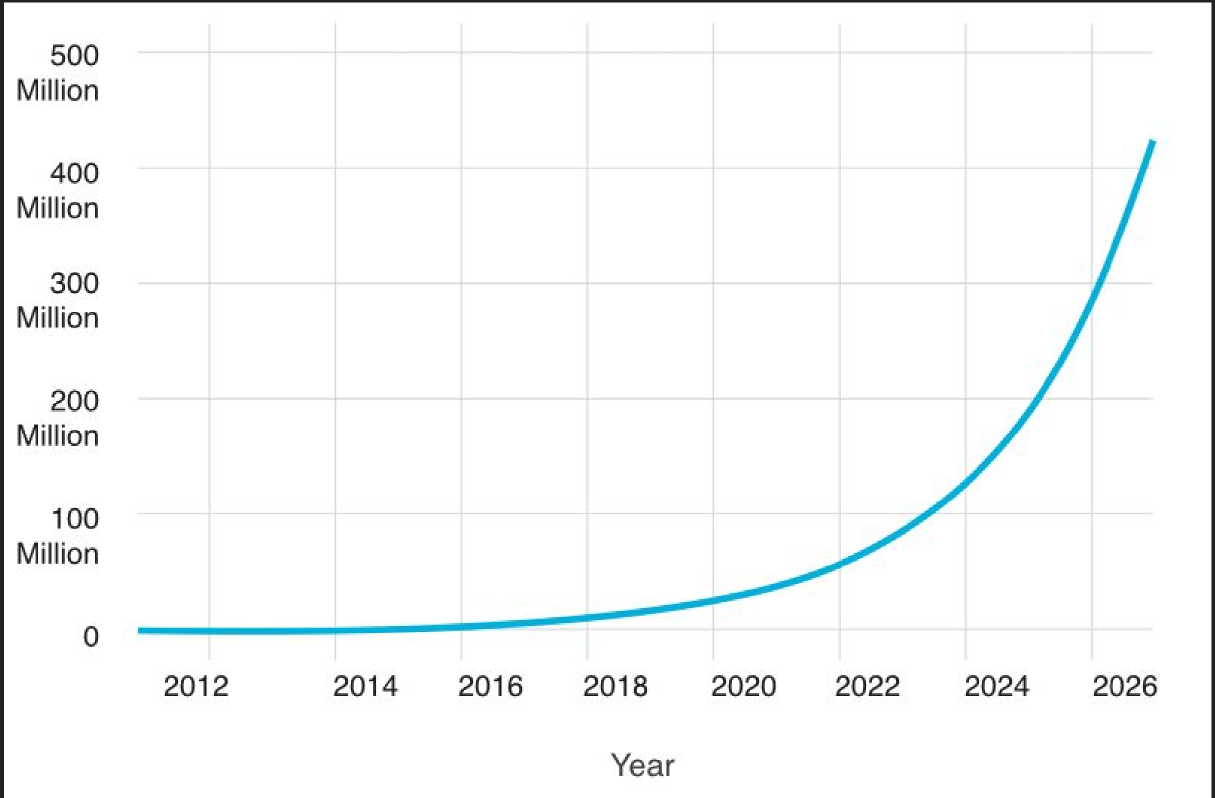
Outline

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

Open-Source Library Growth

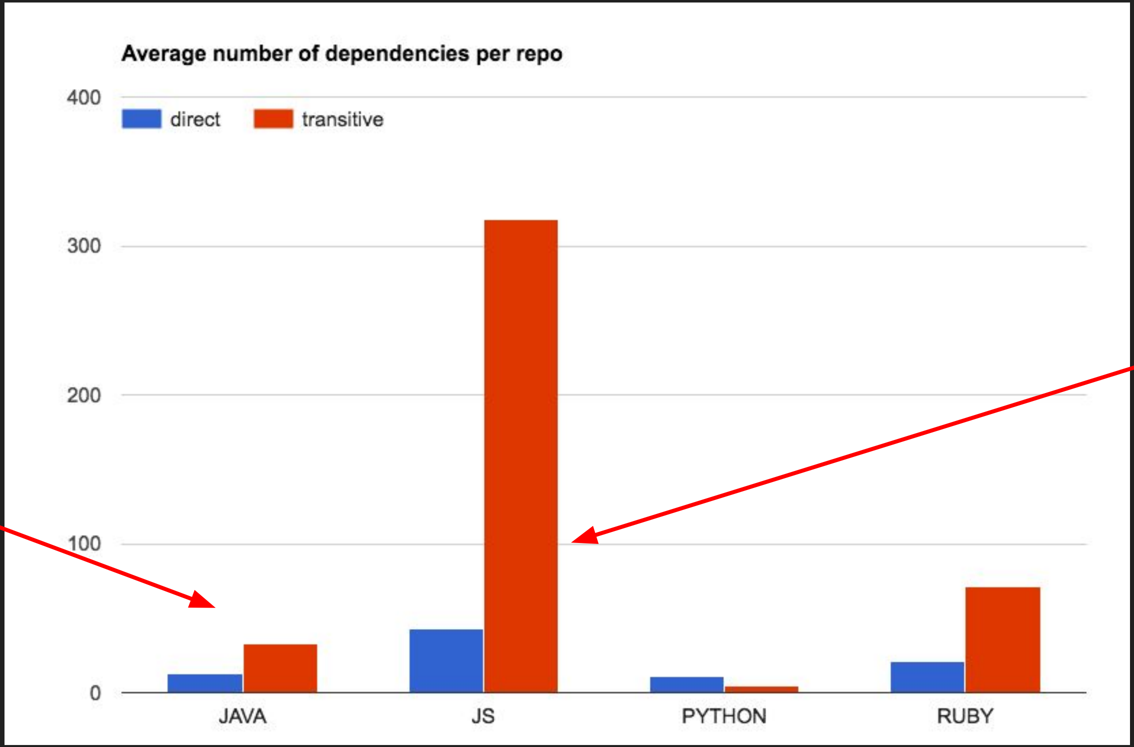


Projection: > 400M Libraries by 2026



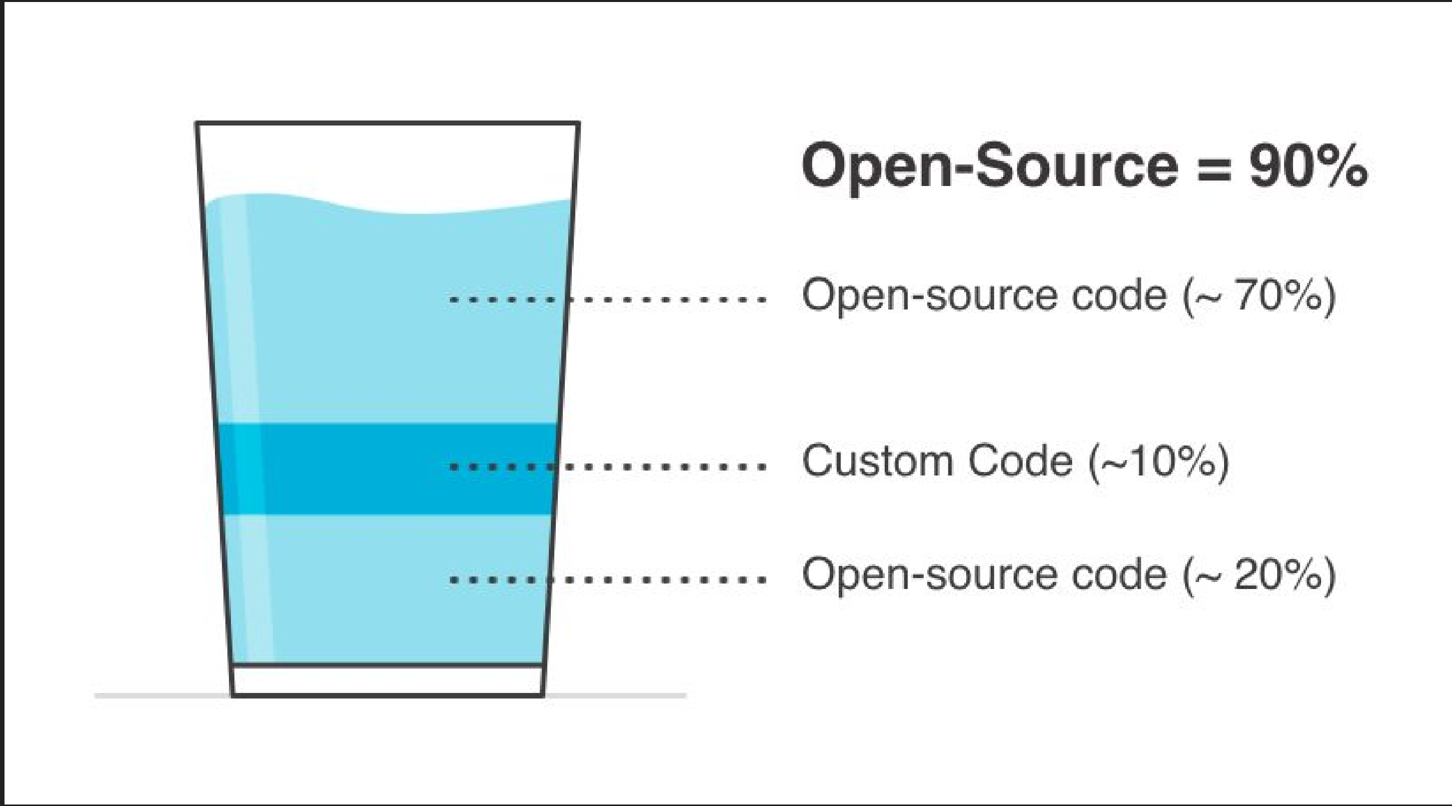
Complexity of Libraries has exploded

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



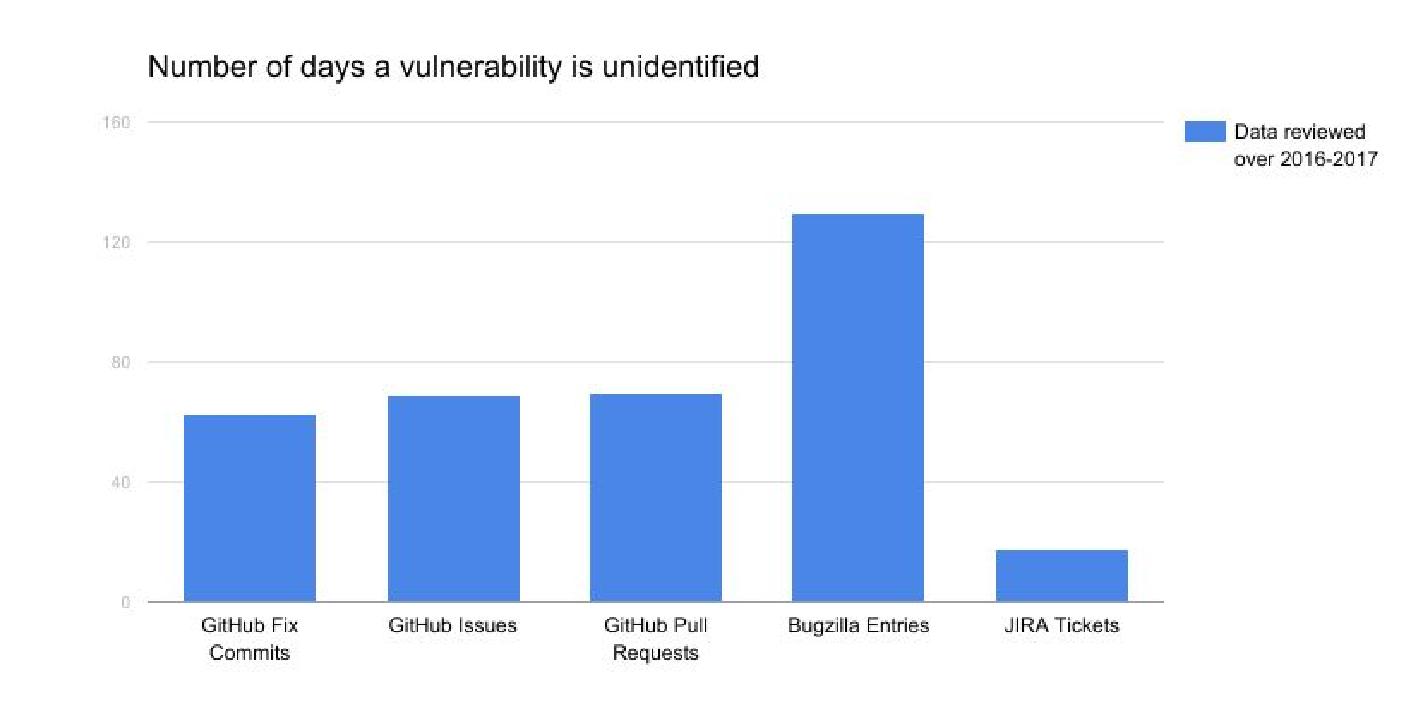
Vulnerabilities in Open-Source Libraries

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

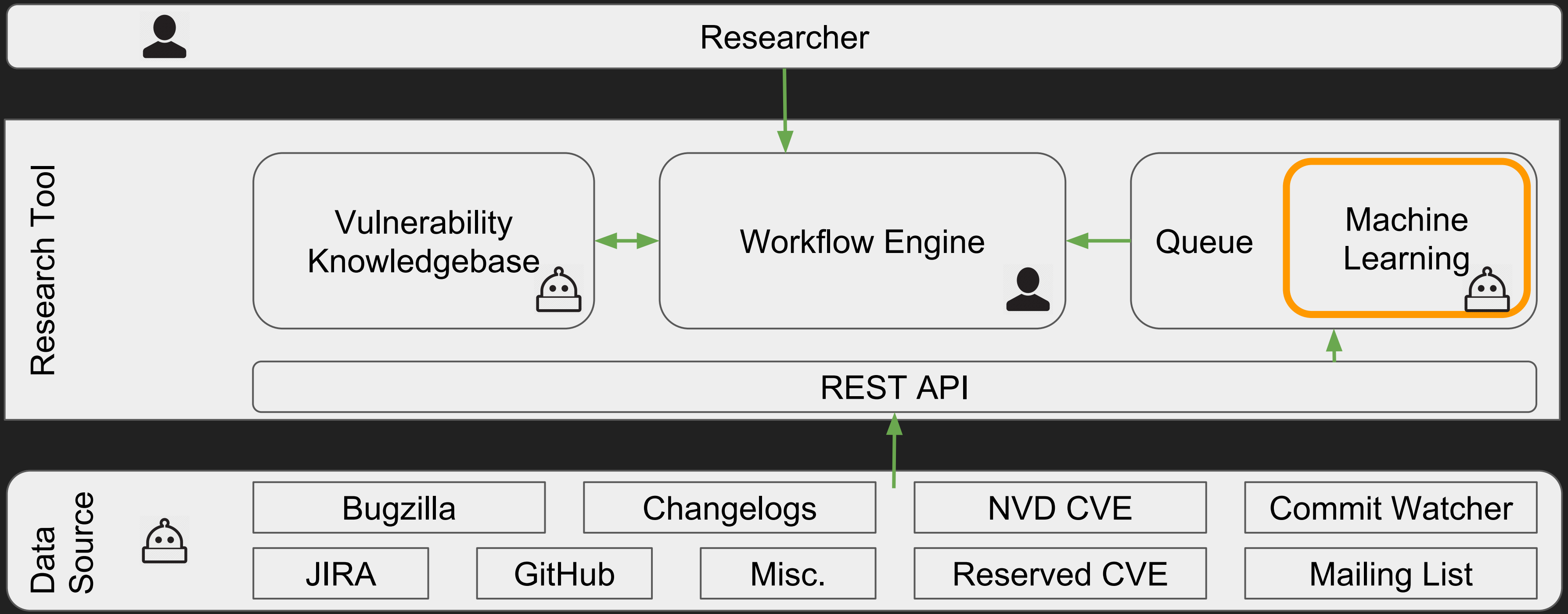
- Unidentified issues
  - Commit logs
  - Bug reports
  - Change logs
  - Pull Requests

Security Issues are often not reported or publically mentioned

Mining for unidentified vulnerabilities



WOPR: Tool for Reviewing Unidentified Issues

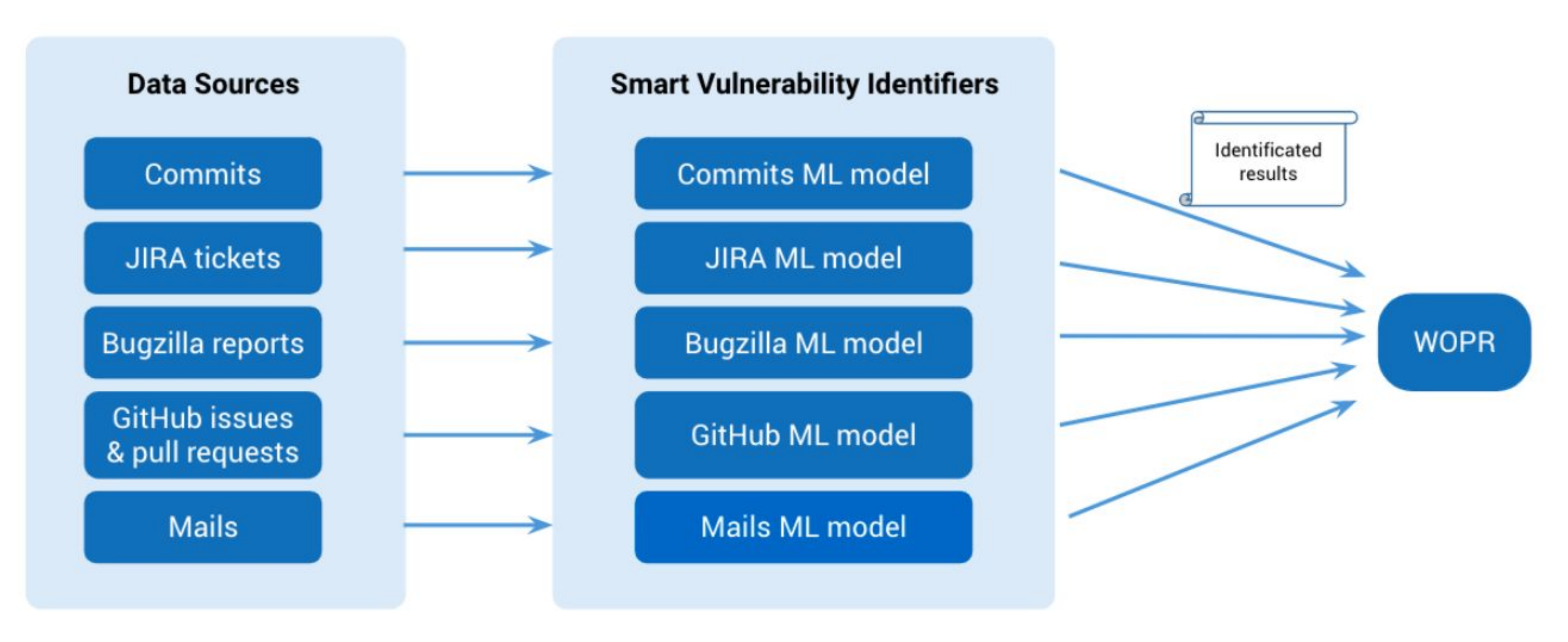


Machine Learning for Identifying Vulnerabilities

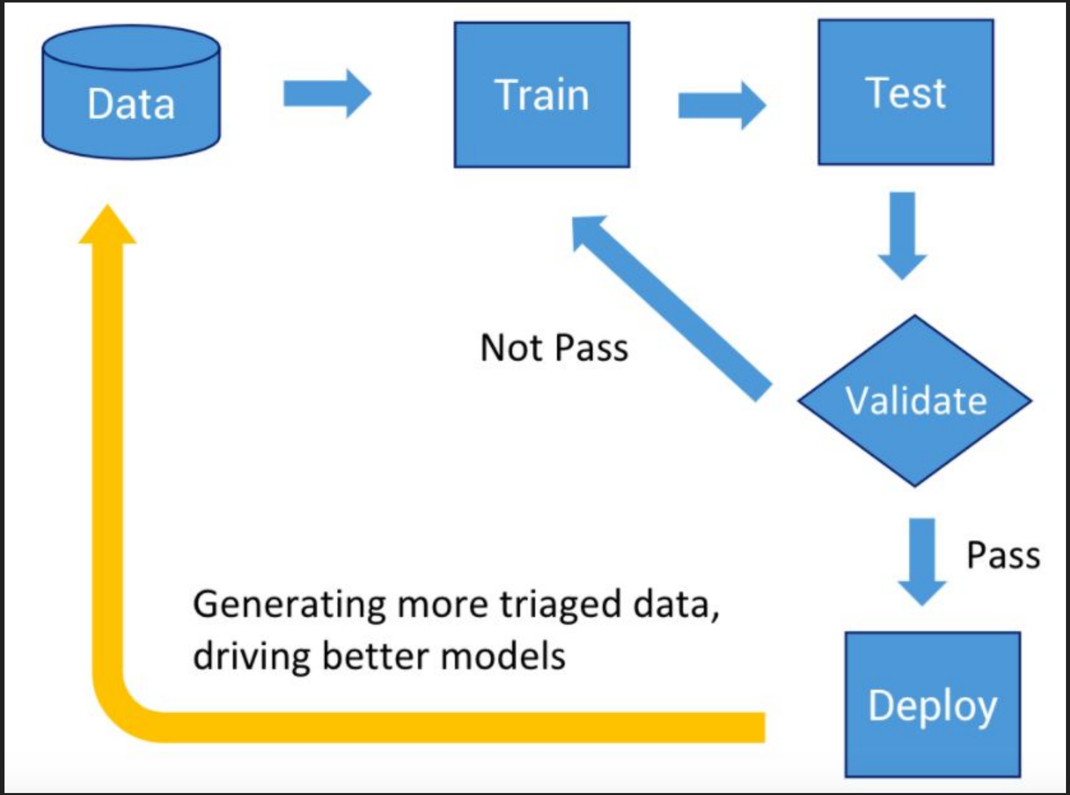
“do machine learning like the great engineer you are, not like the great machine learning expert you aren’t.”

Martin Zinkevich, Rules of Machine Learning: Best Practices for ML Engineering
http://martin.zinkevich.org/rules_of_ml/rules_of_ml.pdf

System overview



ML Pipeline


Data
Train
Test
Not Pass
Validate
Pass
Generating more triaged data,
driving better models
Deploy

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

Source	# of tracked projects
Github	5002
JIRA	1310
Bugzilla	2224

Datasets

Highly imbalanced

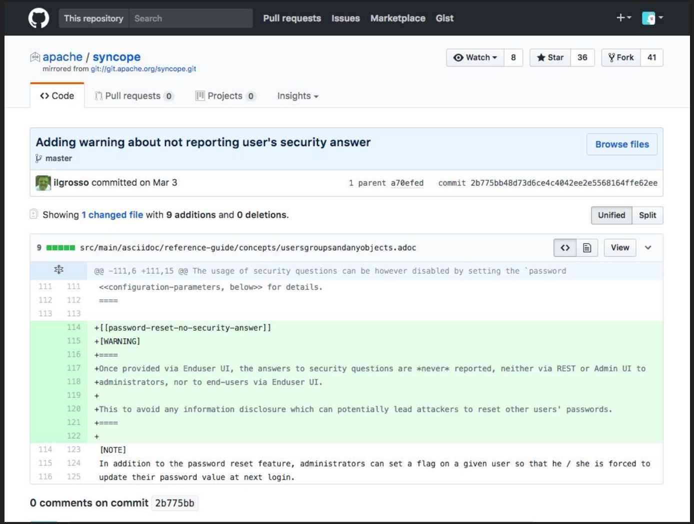
Dataset	Size	# vulnerability_related	Imbalanced ratio
Commit	12409	1303	10.50%
GitHub bug reports	10414	612	5.88%
JIRA bug reports	11145	204	1.83%
Bugzilla bug reports	2629	1089	41.42%
Mails	4499	2721	60.48%

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

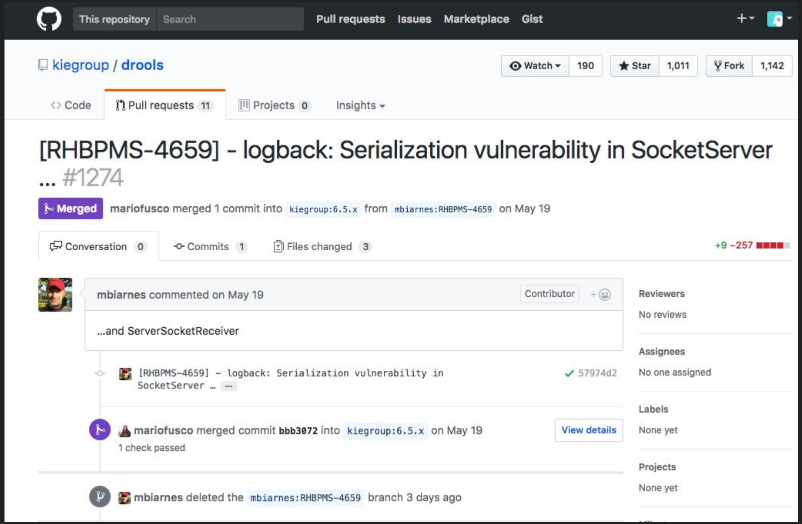
Samples

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

Commit



Bug report



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

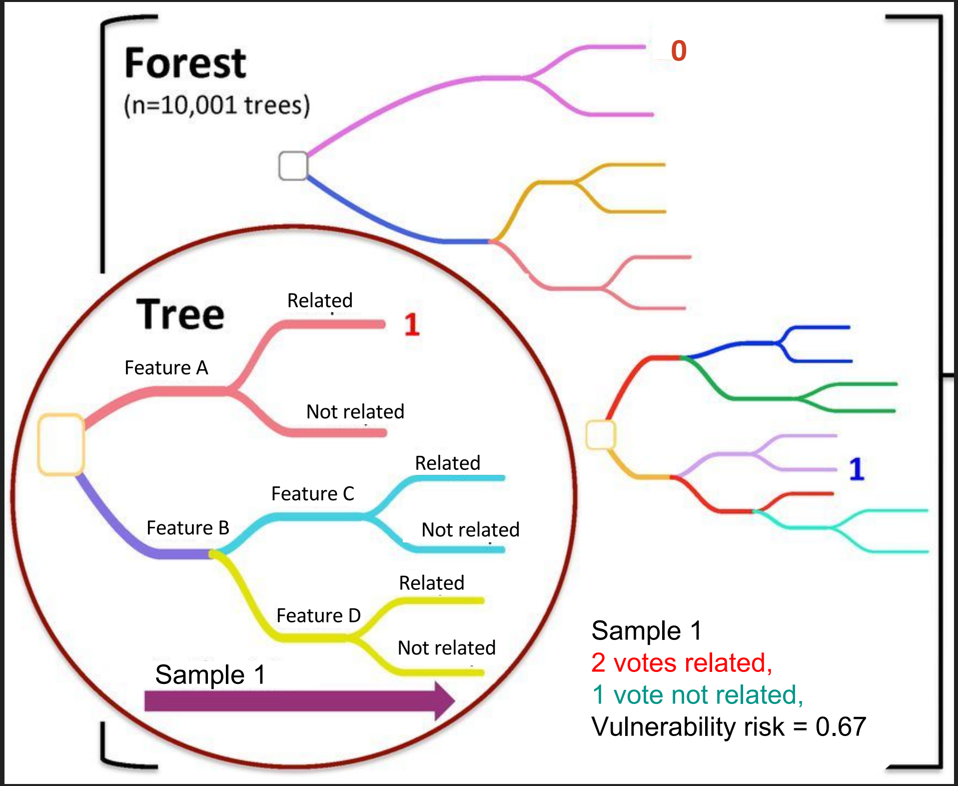
Text feature-Word embedding

- 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

```
>>> 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)
>>> 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', 0.46756529808044434),
(u'csrf', 0.457594096660614), (u'vuln', 0.4533842206001282)]
```

Built word2vec model based on 3 million unfiltered commits

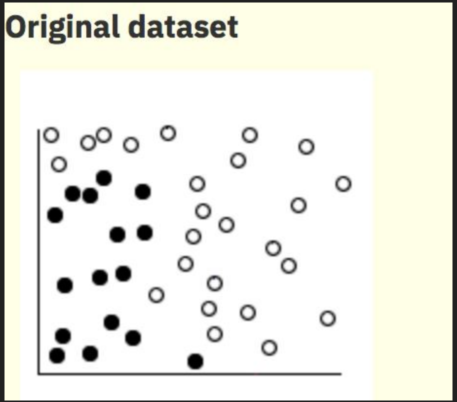
First training attempts-random forest

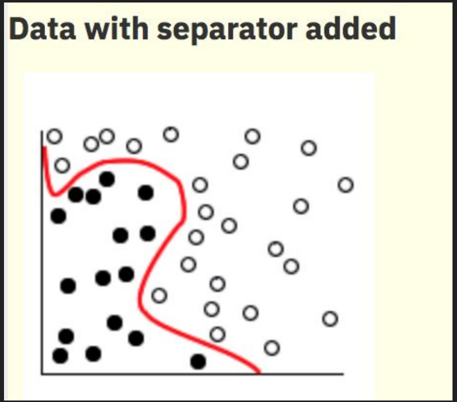


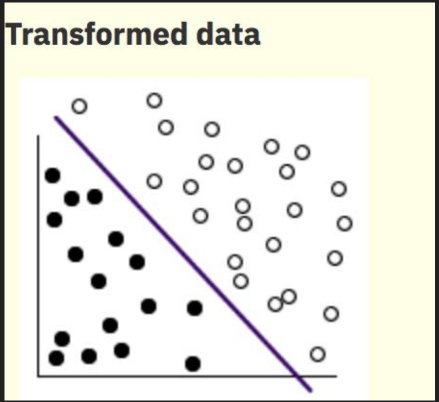
How Random Forest works?

- 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

First training attempts-SVM





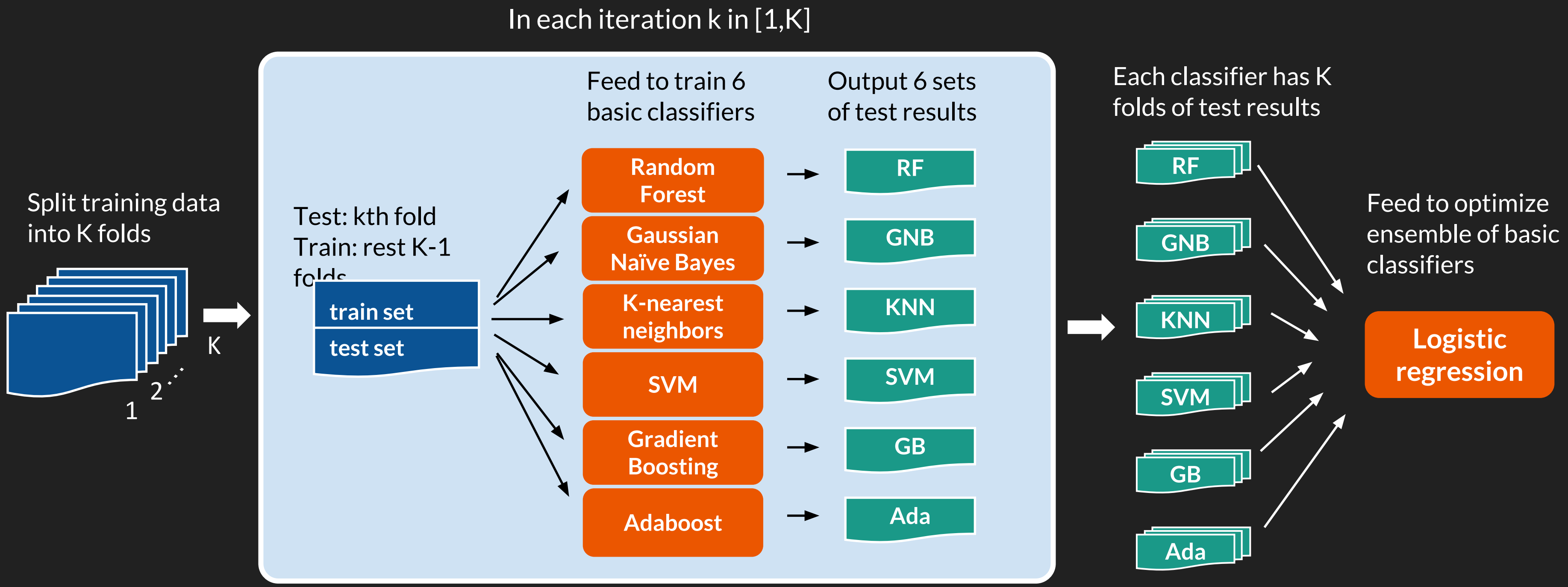


How SVM works?

- 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



Evaluation-metrics

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

$$Precision = \frac{true\ positive}{true\ positive + false\ positive}$$

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

$$Recall\ rate = \frac{true\ positive}{true\ positive + false\ negative}$$

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

Predicted positives

Commits (Total)	Commits (Positive)	Commits (Negative)	True positive	False positive
1000	100	900	70	35

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

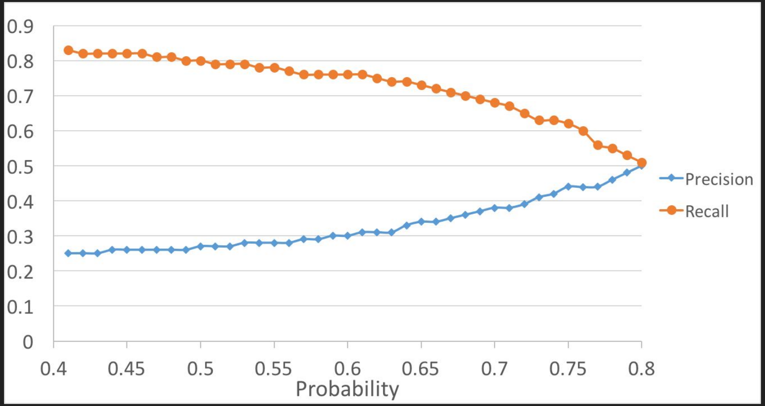


Figure: Identification performance of our stacking approach under commits

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

Classifier	Recall rate	Precision (compared classifier vs.stacking)
Linear SVM	0.72	0.22 vs. 0.34
Logistic Regression	0.76	0.22 vs. 0.31
Random Forest	0.76	0.19 vs. 0.31
Gaussian Naive Bayes	0.77	0.14 vs. 0.28

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**

Commits (Total)	Commits (Positive)	Commits (Negative)	True positive	False positive
2268	215	2053	160	32

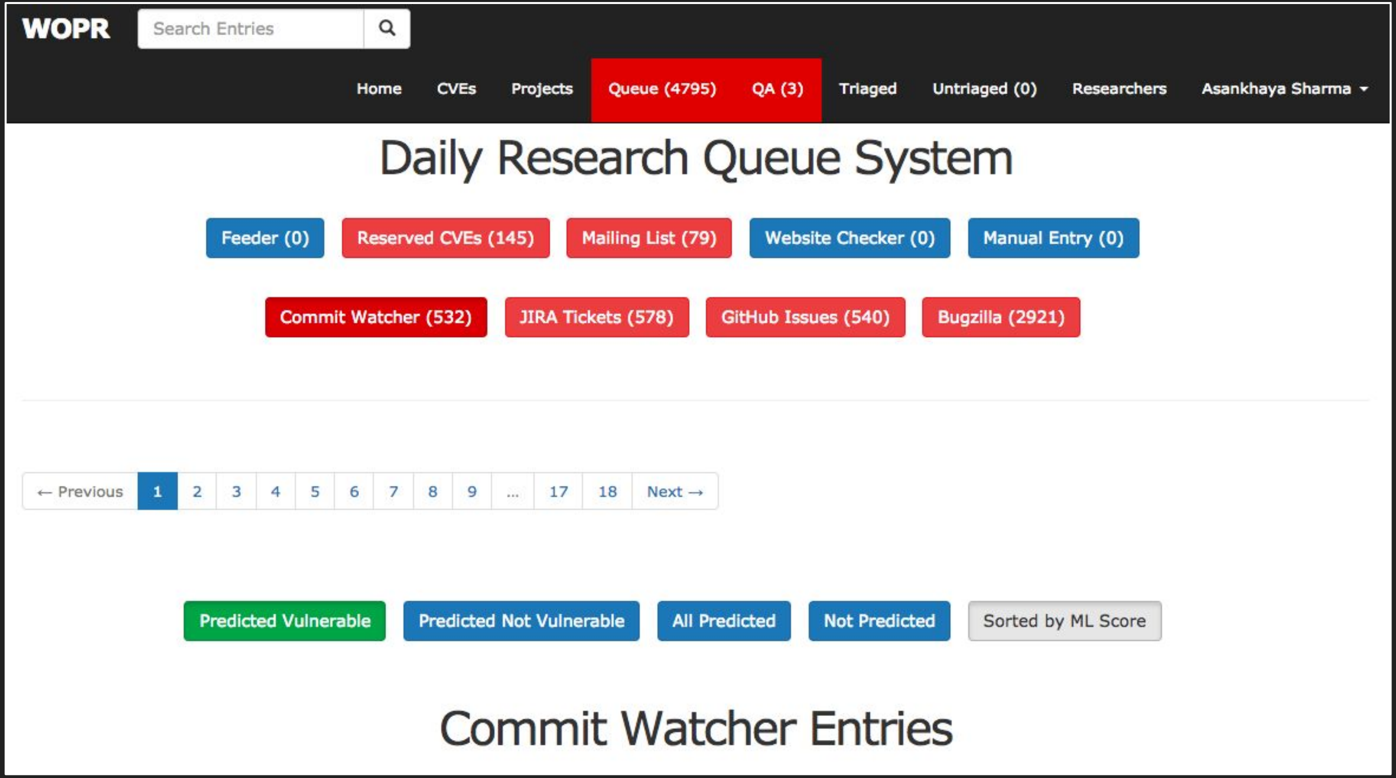
Production observation

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

Sources	GitHub	JIRA	Bugzilla
#Projects	10113	1310	2224

  - Accelerate vulnerability identification
    - When we firstly added go projects from Github in May, by May 29, 2017*
      - 87 go artifacts created from commit watcher
      - 33 go artifacts created from Github Issues
- Current Github/Jira issues can spot vulnerabilities at the first time

Demo



WOPR
Search Entries
Home
CVEs
Projects
Queue (4795)
QA (3)
Triaged
Untriaged (0)
Researchers
Asankhaya Sharma
Daily Research Queue System
Feeder (0)
Reserved CVEs (145)
Mailing List (79)
Website Checker (0)
Manual Entry (0)
Commit Watcher (532)
JIRA Tickets (578)
GitHub Issues (540)
Bugzilla (2921)
← Previous
1
2
3
4
5
6
7
8
9
...
17
18
Next →
Predicted Vulnerable
Predicted Not Vulnerable
All Predicted
Not Predicted
Sorted by ML Score
Commit Watcher Entries

Thanks!