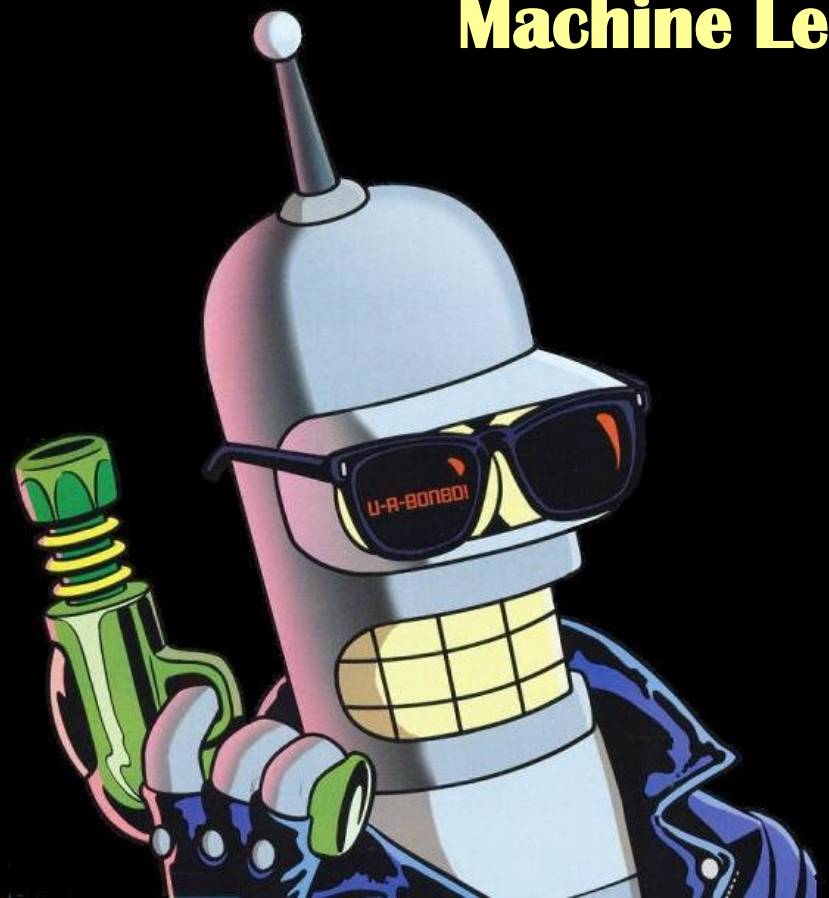
Kill All Humans... Bugs!

Machine Learning to the rescue of code review

Presented by Philippe Arteau



44CON London

Who am I ?

- Philippe Arteau
- Security Researcher at GoSecure Countertack
- Open-source developer
  - Find Security Bugs (SpotBugs - Static Analysis for Java)
  - Security Code Scan (Roslyn – Static Analysis for .NET)
  - Burp and ZAP Plugins (Retire.js, CSP Auditor, Reissue Request Scripter, ..)
- Machine Learning Enthusiast .. *not a professsional*

Agenda

1. Static-analysis concepts
   - Symbolic execution
   - False positives
   - Inter-procedural data-flow
   - Obstacles
2. Applying Machine Learning
   - Identifying attributes
   - Graph creation
   - Overview of the Machine Learning algoritms
   - Results
3. Demonstrations
4. Lessons learned (Conclusion)

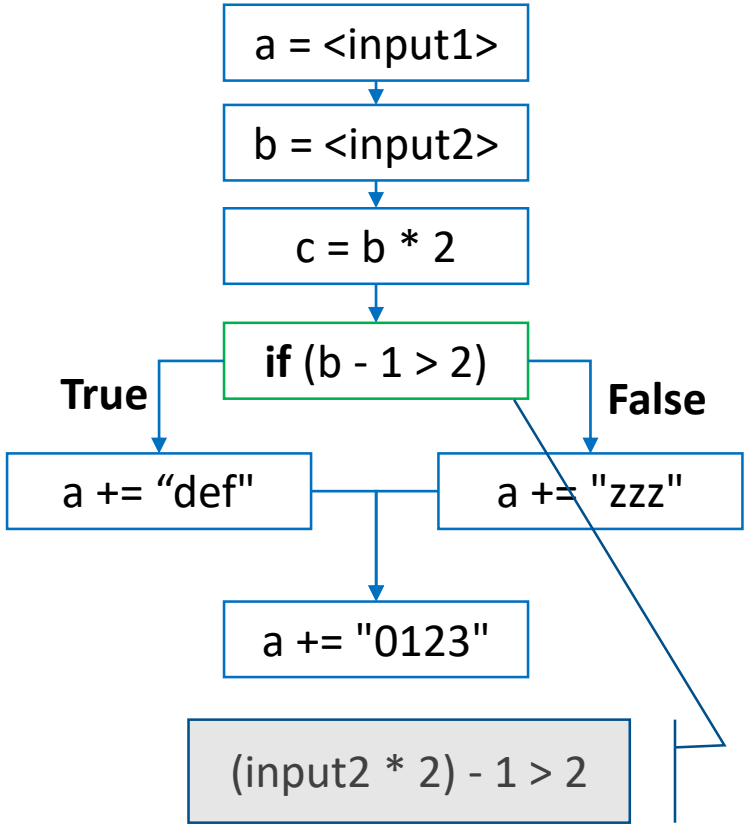
Static-analysis

Symbolic execution

Simulating the code execution using expression rather than concrete data

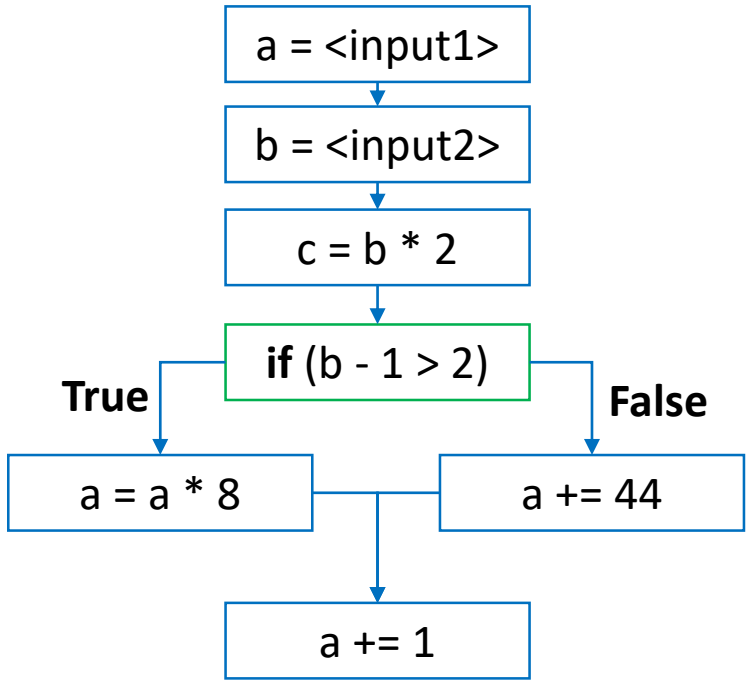
It can be use :

- To resolve conditions
- To evaluate the states of variable as it evolves during execution



Reference : [Symbolic Execution for Software Testing: Three Decades Later]

Symbolic execution (in action)



a	b	c
input1		
input1	input2	
input1	input2	input2*2

(b - 1 > 2) == true

a	b	c
input1*8	input2	input2*2
input1*8+1	input2	input2*2

(b - 1 > 2) == false

a	b	c
input1+44	input2	input2*2
input1+45	input2	input2*2

Symbolic execution

- Symbolic execution mainly focuses on resolving input values to **reach a specific path**

- Many vulnerabilities analyzers need to monitor **validation state** of variables.

Taint analysis

Safe

```
a = "userId = "
b = Config.ADMIN_USER_ID
c = a + b

User.applyFilter(c)
```

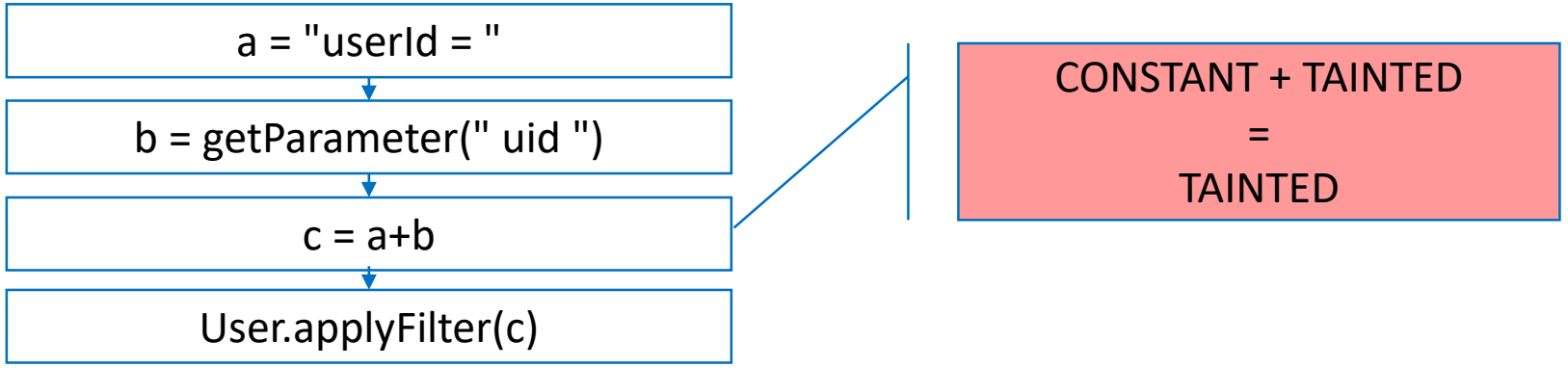
Unsafe

```
a = "userId = "
b = getHttpParameter("uid")
c = a + b

User.applyFilter(c)
```

How to avoid reporting an issue for the left code sample?

Taint analysis (in action)

Taint analysis as implemented in Find Security Bugs



a	b	c
CONSTANT		
CONSTANT	TAINTED	
CONSTANT	TAINTED	TAINTED

Demonstration

{} Find Security Bugs

Obstacles

Obstacles

Obstacles that static analyzers must consider:

 Reflection

 Dependency injection

 Second order vulnerability

 Encapsulation

 Custom framework and utility

Two blind spots from FSB

1. Implements Inter-procedural data flow

But..

- Inter-procedural data flow can missed some cases
  - See : Reflection, Incomplete code, DI, Second Order, etc.
- Inter-procedural data flow can identify exploitable path from source to sink

2. **Proprietary** framework and utility functions are not taken into account by Find Security Bugs

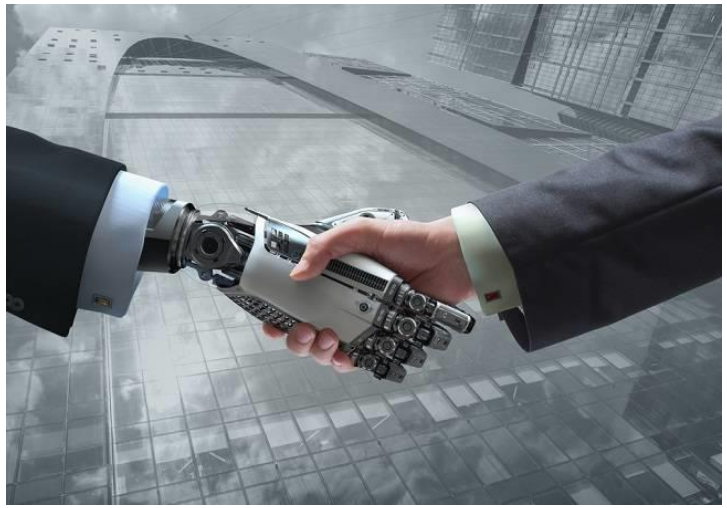
Objective :

Prioritize new vulnerabilities based on previously classified bug patterns

Applying Machine Learning

What is Machine Learning?

"Machine learning is a field of computer science that uses **statistical techniques** to give computer systems the ability to **"learn" with data**, **without** being **explicitly programmed**."



- https://en.wikipedia.org/wiki/Machine_learning

Supervised vs Unsupervised

Supervised

Unsupervised

Supervised classification vs regression

Supervised classification

Attempt to predict the **right answer** from a discrete number of possibilities

Prediction Features ->

- Recommend or not to user (binary)
- Type of species (finite list)

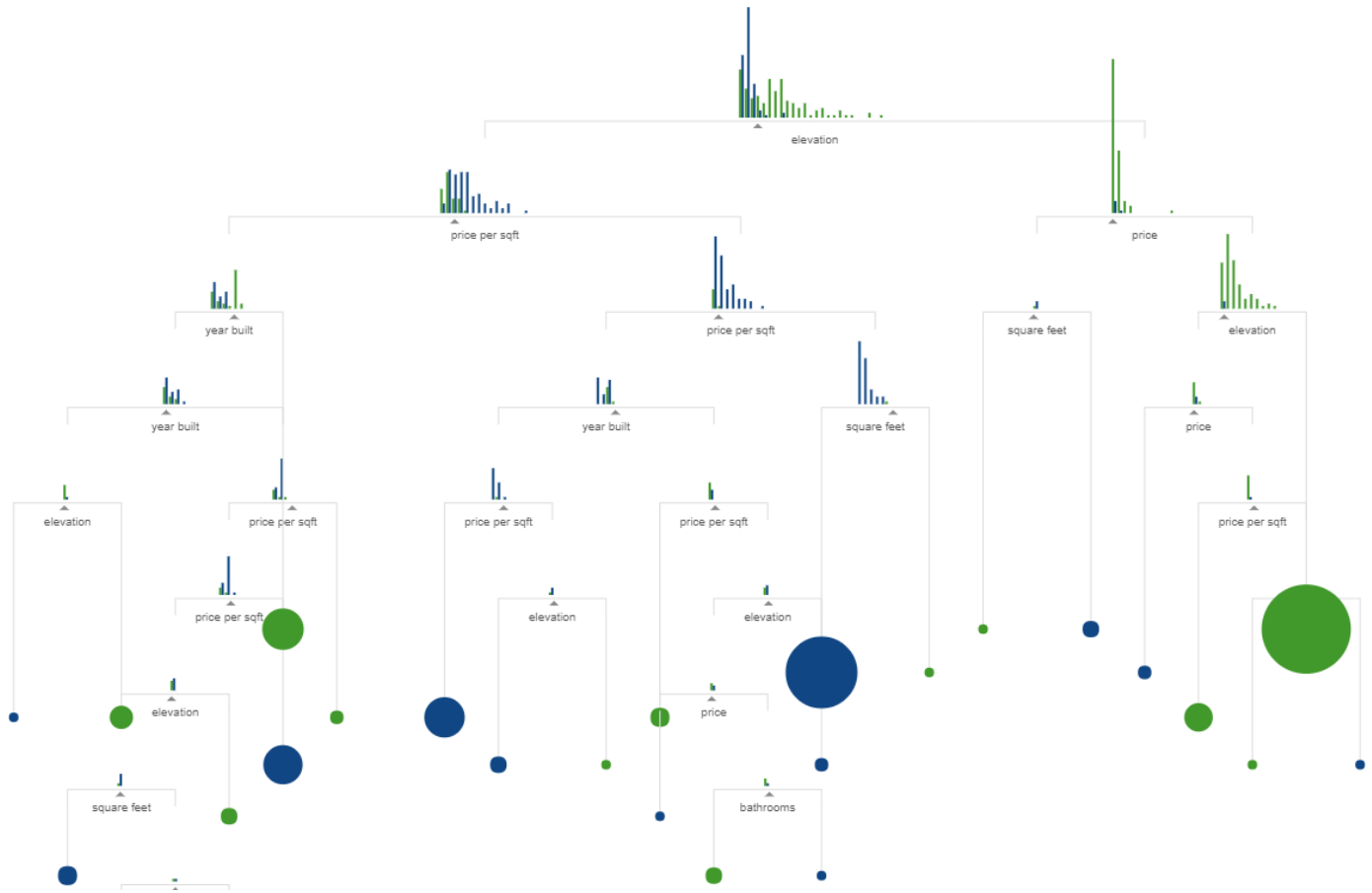
Supervised regression

Attempt to predict a **continuous value**

Prediction Features ->

- House pricing (10k to 10M)
- Time of recovery from incident

Algorithm?



Generated can take many forms:

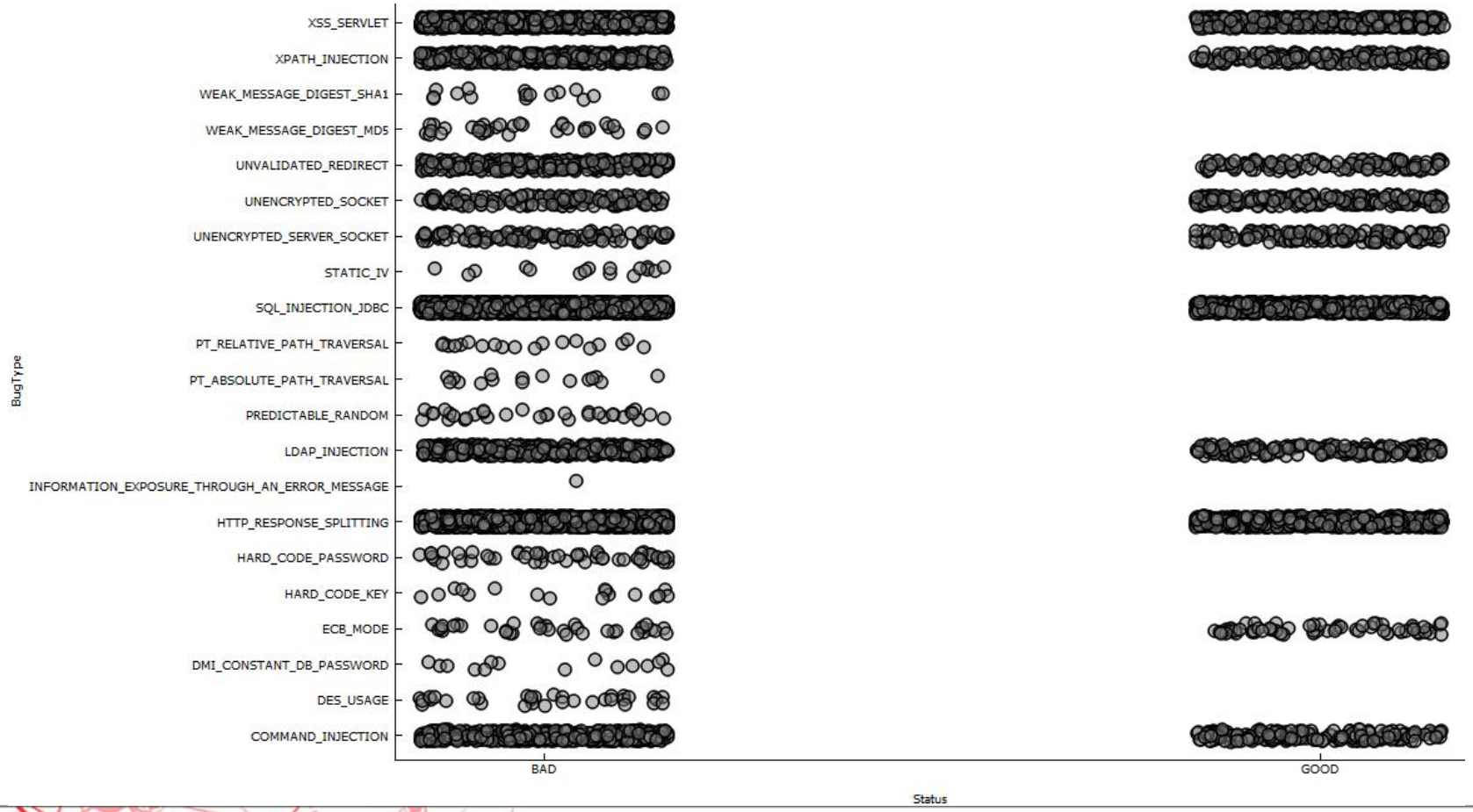
- Graph (Naive Bayes)
- Series of condition (tree)
- Neural network

The algorithm are **not explicitly programmed.**

http://www.r2d3.us/visual-intro-to-machine-learning-part-1/

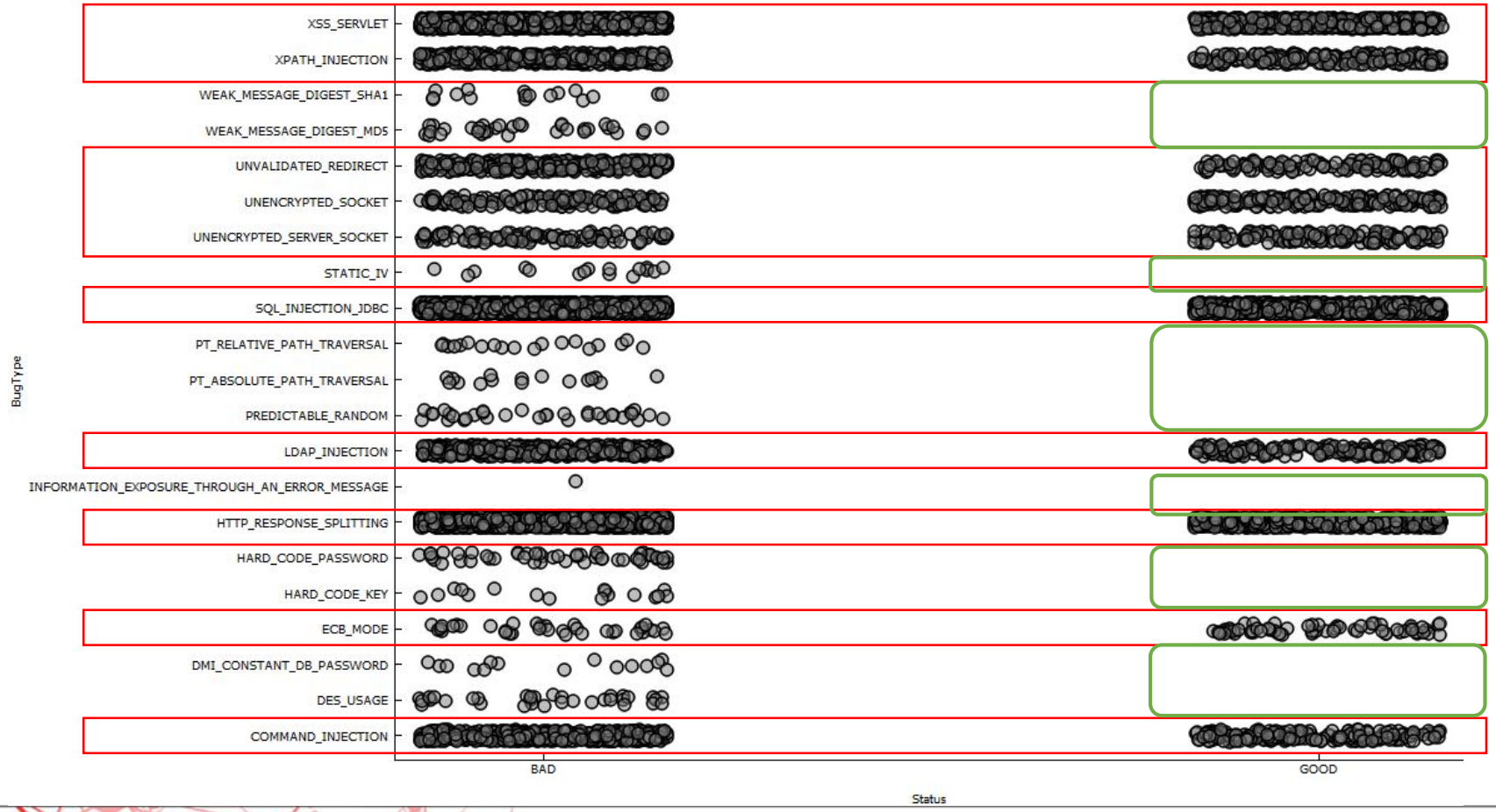
Identifying Attributes

- Visualization of the data is crucial
  - Some bug types do not have false positive (on the Juliet Test Suite)



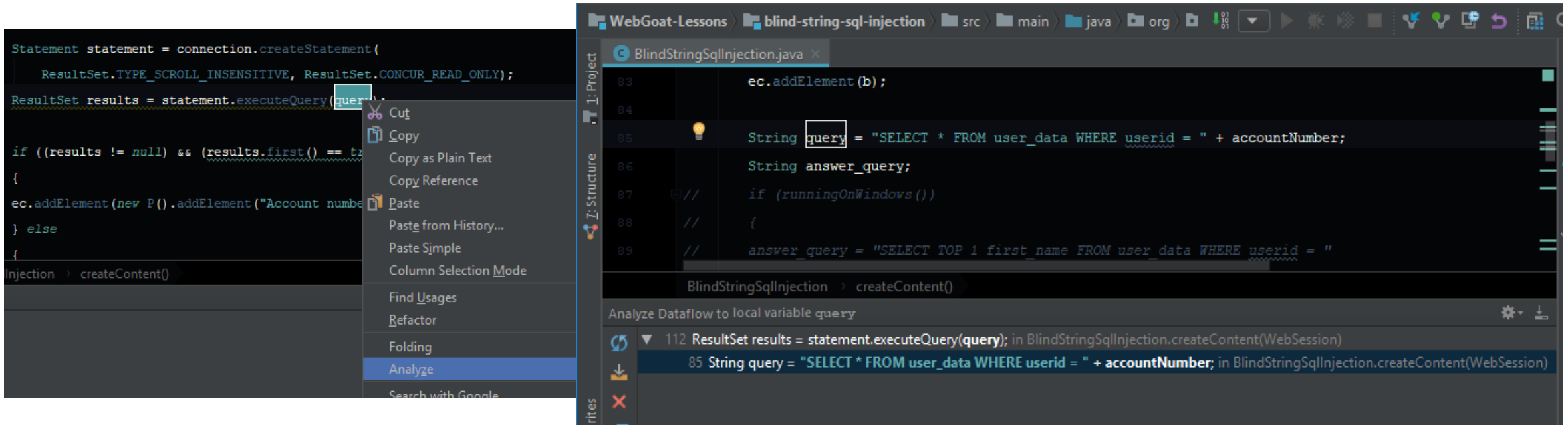
Identifying Attributes

- Visualization of the data is crucial
  - Some bug types do not have false positive (on the Juliet Test Suite)



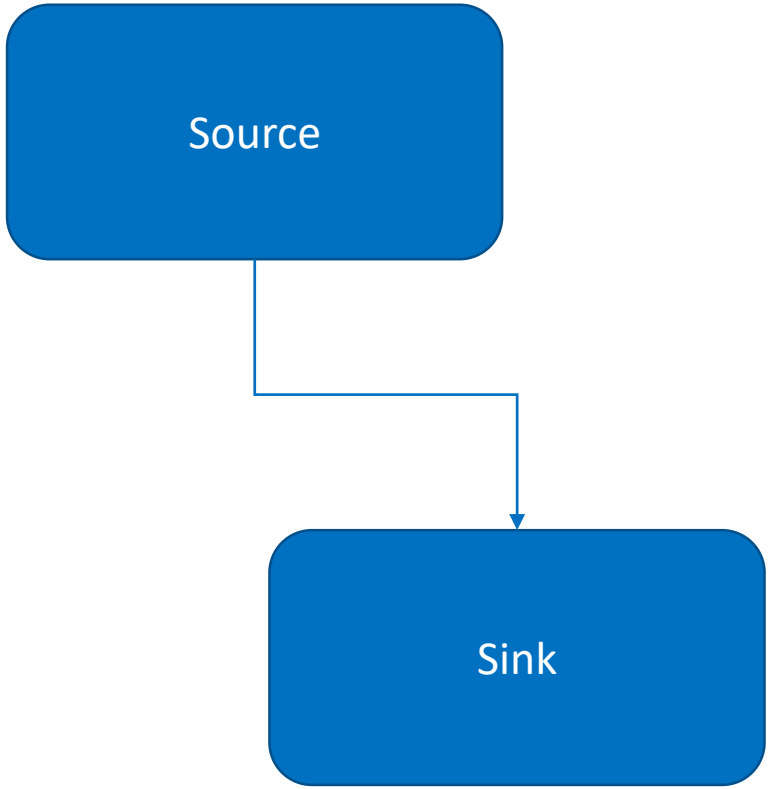
Identifying attributes

- Quality attributes are key
- Our attribute are based on contextual information that a human would use



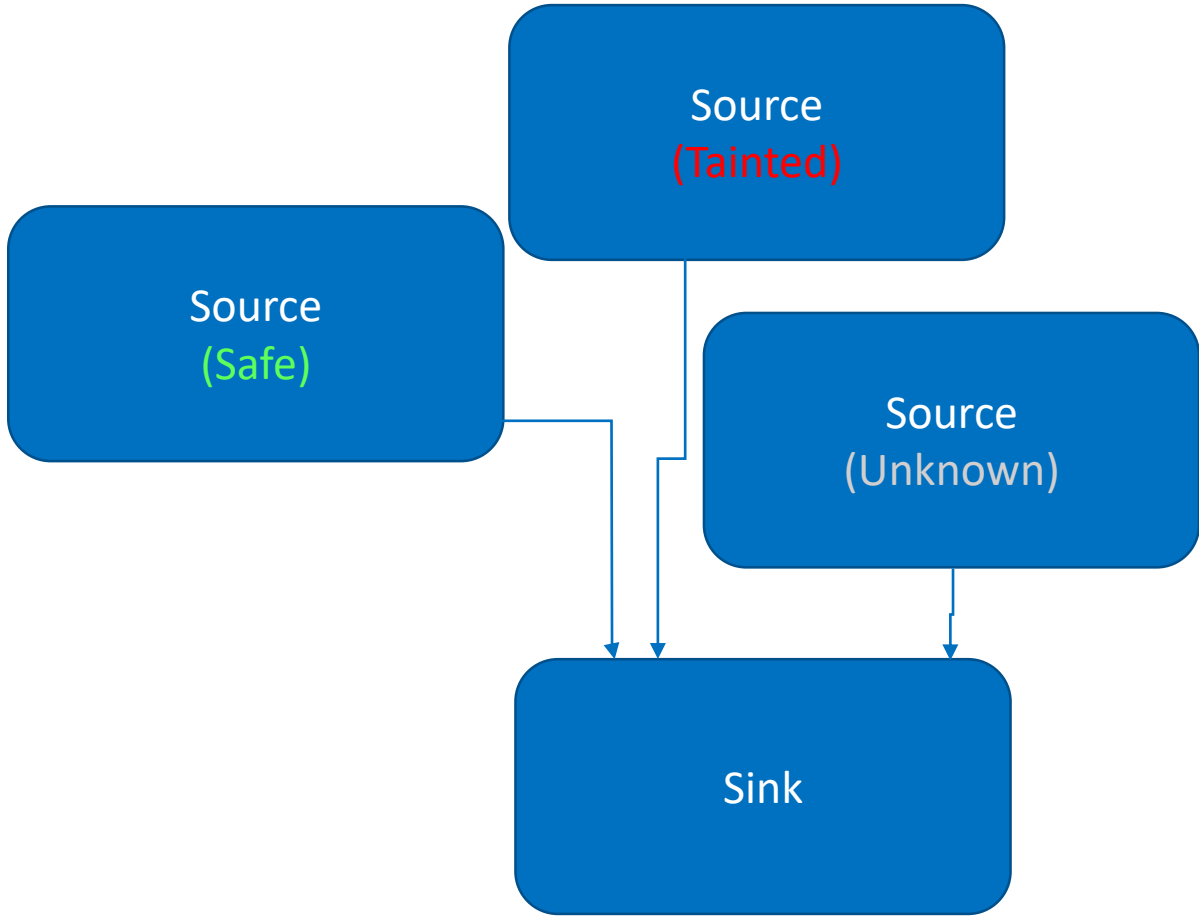
New attributes created (part 1)

- Extracting the signature of the **source**
  - ProprietaryEncoder.encodeHTML(String)
  - RequestWrapper.getParameter(String)
  - Configuration.getParameter(String)
- Extracting the signature of the sink
  - Statement.executeUpdate(String)
  - File(String)



New attributes created (part 2)

- Extracting the state of **sources**
  - Does the injectable parameter is affected by one **safe** source
  - […] by one **tainted** source
  - […] by one **unknown** source
- Additional localization attributes
  - Filename
  - Module name



Modelizing inter-procedural dataflow

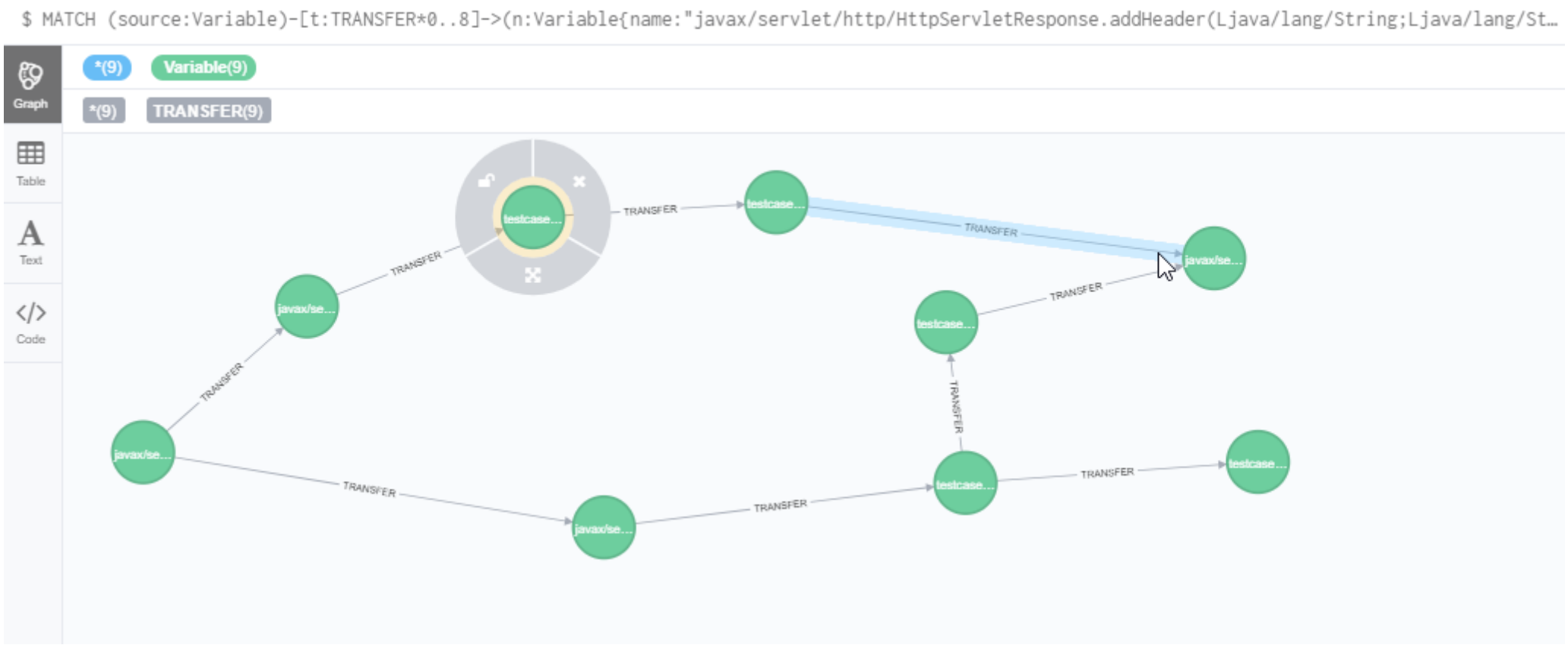
```
public void loadUser(SessionFactory sessionFactory, String input) {

    Session session = sessionFactory.openSession();

    //HQL injection

    session.createQuery("select t from UserEntity t where id = " + input);

[…]

public Query createQuery(String query) {
[…]
```

Node1

Node2

Node3

Graph representation



In order to do taint analysis accross the entire web application, a graph was built

Cypher query

```
MATCH (source:Variable)-[r1:TRANSFER*0..8]->
   (node:Variable)-[r:TRANSFER]->(sink:Variable)
WHERE
  source.state IN ["UNKNOWN", "SAFE", "TAINTED"] AND
  sink.name = _sink_** AND
  node.source = _source_**
RETURN source,sink,r1,node,r;
```

** Full method signature (including parameter index)

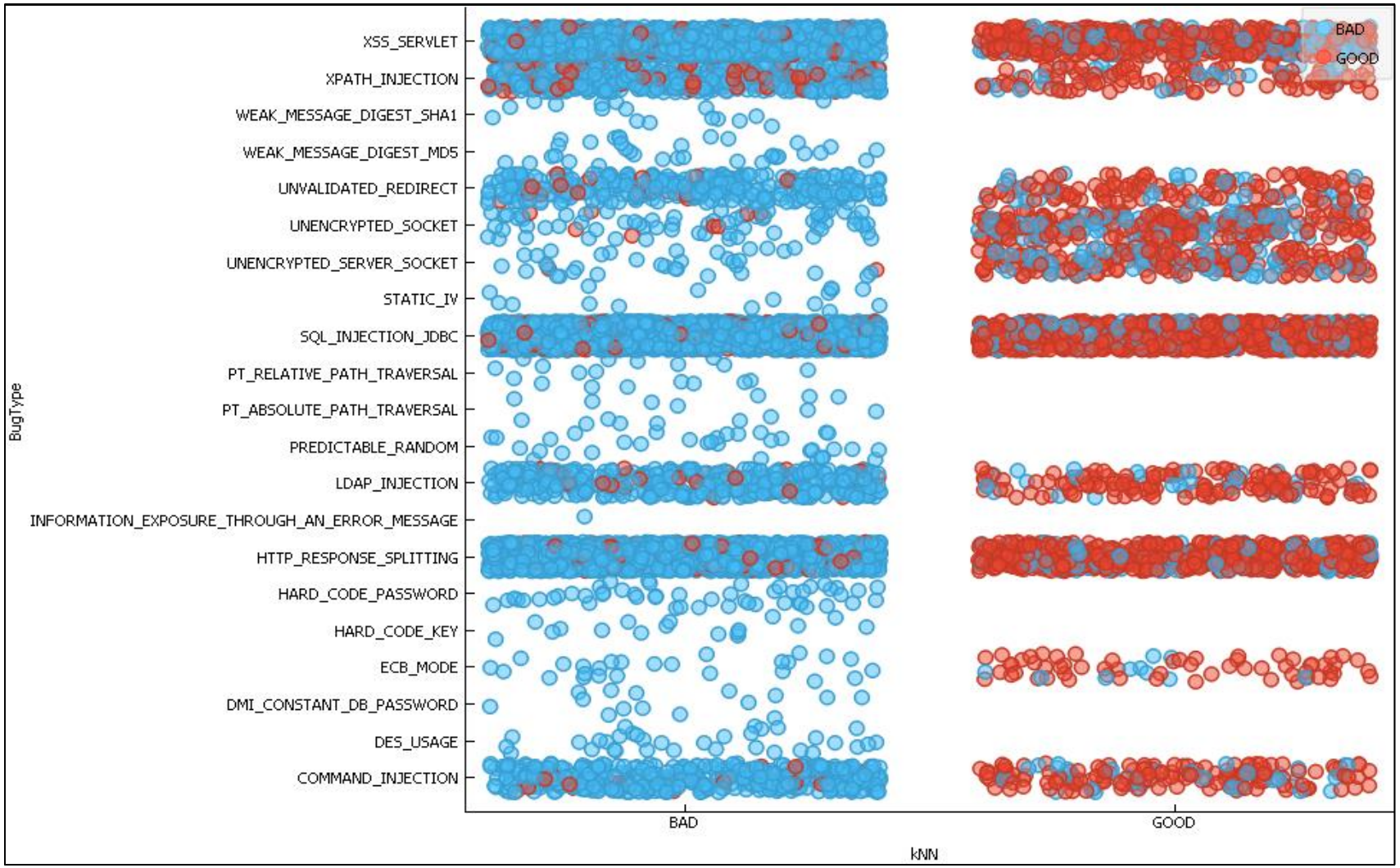
Demonstration

Find Security Bugs

sonarqube

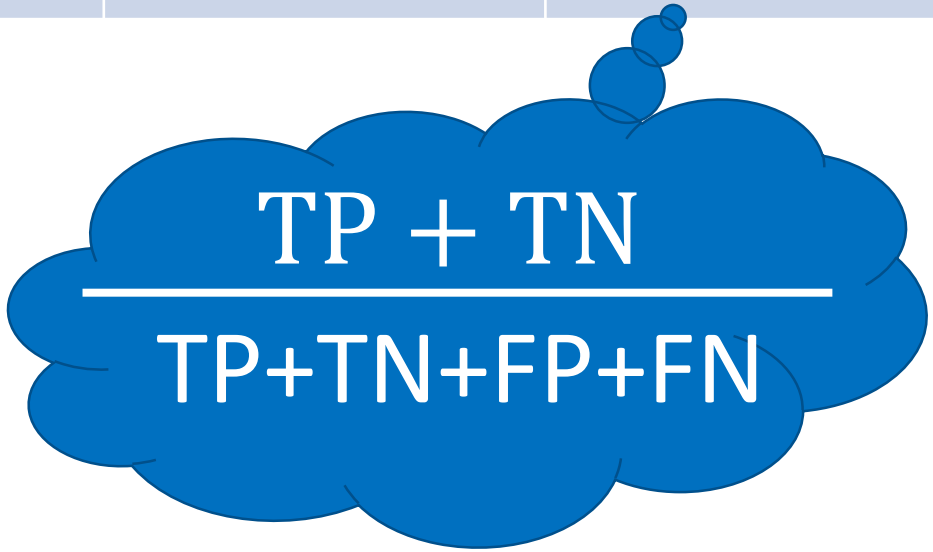
Vulnerabilities prioritization

Results (Juliet Dataset)



Results (Spring Framework)

	Recall	Precision	F-Measure	Accuracy
Naïve Bayes	0.761	1.00	0.614	72.7 %
K-NN (k=2)	0.973	0.966	0.970	94.2 %
C4.5	1.00	0.955	0.977	95.5 %
Random Forest	1.00	0.964	0.982	96.4 %
SVM	1.00	0.955	0.977	95.5 %



Lessons learned

Important of attributes

- **Quality attributes** are more important than the choice of ML algorithms
- Training set need to be close to the test set
  - Method signature attributes don’t work well when comparing two differents librairies
- Machine Learning benefits mainly large codebase (1000+ vulnerabilities)
  - Analysis of most web application, frameworks or libraries can be done in few hours
- Aggregating datasets from various librairies did not work well

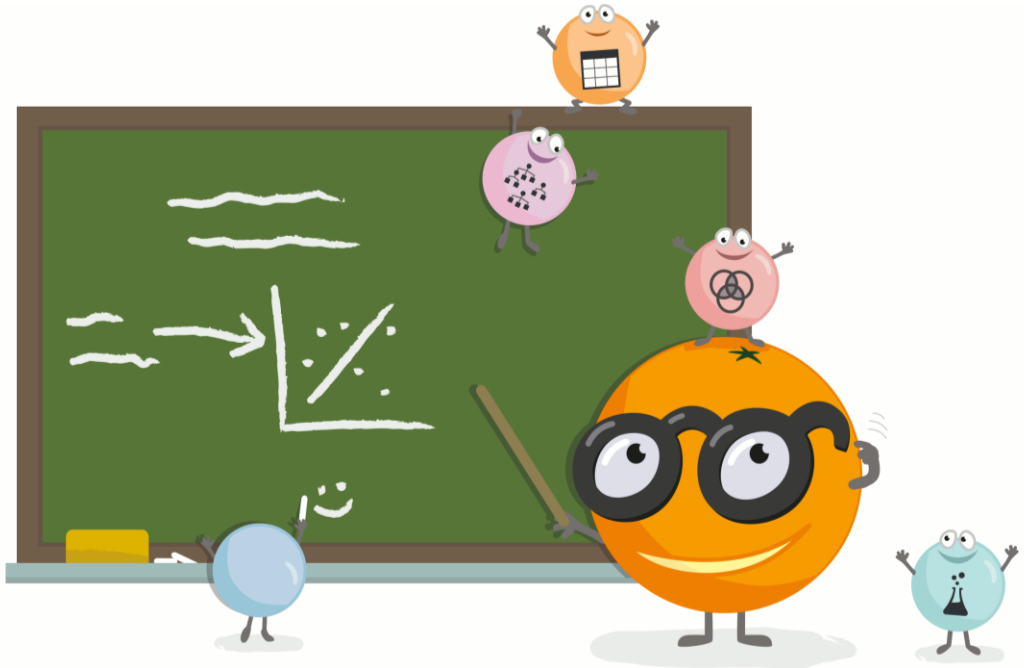
Future for FindSecBugs and ML..

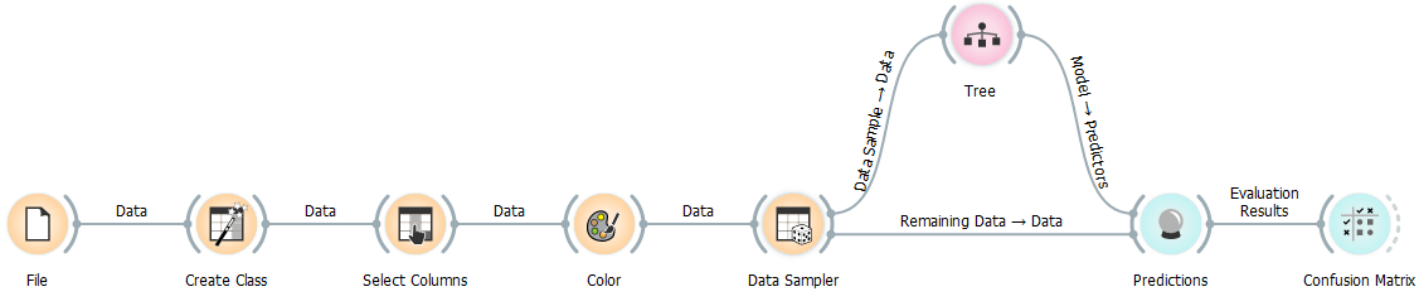
- At the moment, still consider experimental

- Improving FSB from the statistical analysis
  - Detect **bug type** that have **a very high false positive rate**
  - Detect **source** method that have **a very high false positive rate**
    - Suggest addition of encoding method
  - Detect **sink** method that have **a very high false positive rate**
    - Improve the heuristic of the detector

Introduction to ML with Orange

- Hand-on introduction
- Step-by-step exercises
- Supervised learning using
  - Titanic Dataset
  - Static-Analysis Dataset (taken from the project presented today)
- **When: September 13th (Tomorrow)**





Questions ?

Contact

parteau@gosecure.ca

gosecure.net/blog/

@h3xStream @GoSecure_Inc

References

References (Online course)

Machine Learning Coursera

- https://www.coursera.org/learn/machine-learning

Introduction to Machine Learning

- https://developers.google.com/machine-learning/crash-course/ml-intro

References Code Graph and Symbolic Execution

- Symbolic Execution for Software Testing: Three Decades Later by Cristian Cadar and Koushik Sen: https://courses.cs.washington.edu/courses/cse484/14au/reading/symbolic-execution-testing.pdf
- Mining for Bugs with Graph Database Queries by Fabian Yamaguchi: https://hacktivity.com/en/downloads/archives/405
- Modeling and Discovering Vulnerabilities with Code Property Graphs : http://ieeexplore.ieee.org/ielx7/6954656/6956545/06956589.pdf