Discovering vulnerabilities using data-flow analysis and machine learning

ARES'18

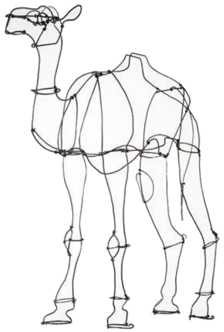
August 27th 2018

Jorrit Kronjee, Arjen Hommersom, and Harald Vranken

Open Universiteit
www.ou.nl

Introduction

WIRECAML – a tool that combines machine learning and features extracted from CFGs to detect SQLi and XSS vulnerabilities in PHP software

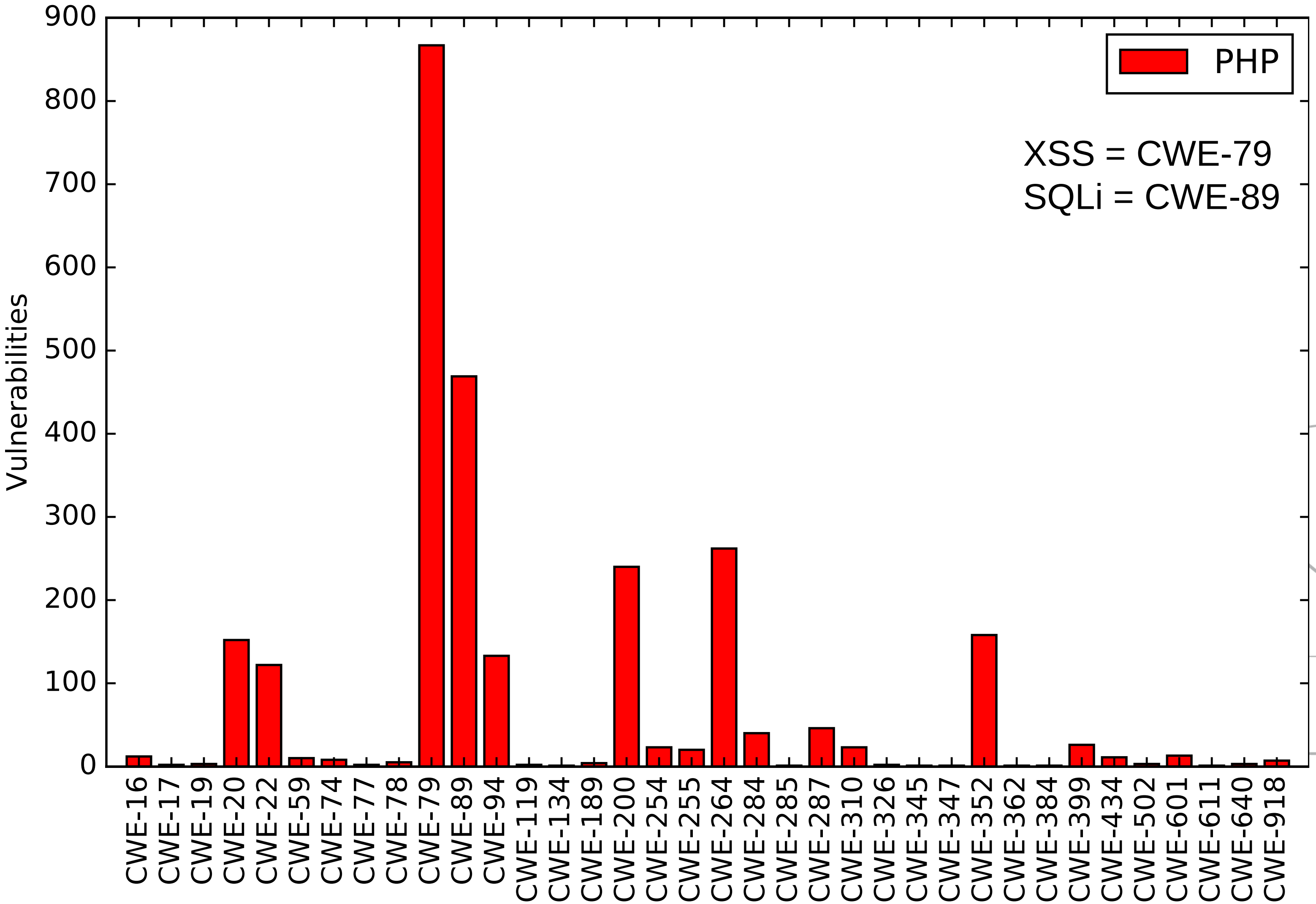


- Motivation for this research
- Description of the WIRECAML method
- Predictive performance of classifiers and features
- Comparison to other tools
- Conclusions

Open Universiteit
www.ou.nl

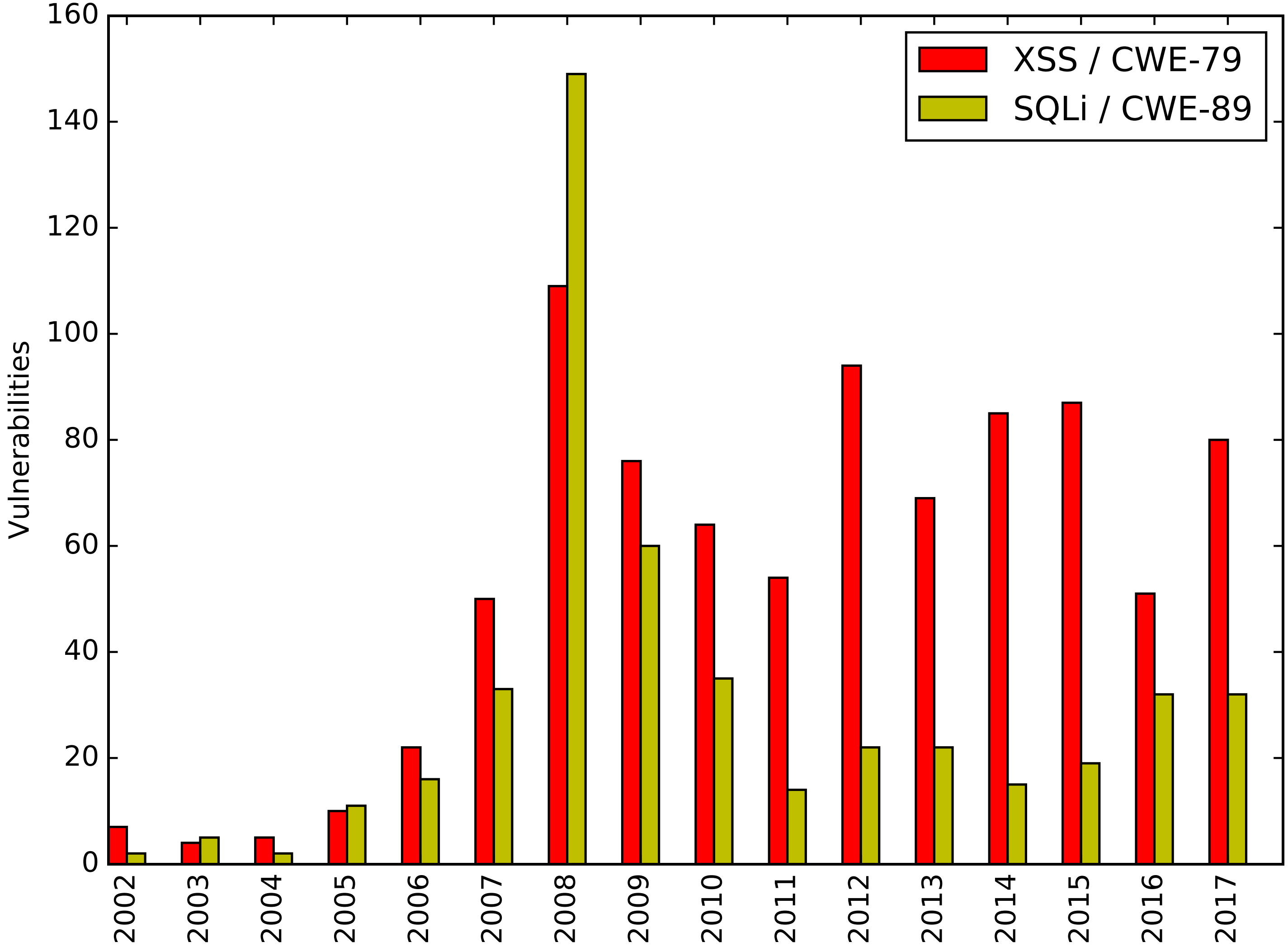
Sculpture by Michael van Hout

Motivation for this research (1/2)



Source: NVD 2002 - 2017

Motivation for this research (2/2)



Source: NVD 2002 - 2017

Example of XSS

phpmyadmin (<4.2.6) vulnerability (CVE-2014-4954)

```
$browse_table_label = '<a href="sql.php?' . $tbl_url_query
    . '&amp;pos=0" title="' . $current_table['TABLE_COMMENT'] . '">'
    . $truename . '</a>';
```

where `$current_table['TABLE_COMMENT']` is an unfiltered variable.

Example of SQLi

phpmyadmin (<3.5.8.2 and <4.0.4.2) vulnerability (CVE-2013-5003)

```
$sql = "REPLACE INTO " . $pma_table . " (db_name, table_name,
pdf_page_number, x, y) SELECT db_name, table_name, " . $pdf_page_number
. ", ROUND(x/" . $scale . ") , ROUND(y/" . $scale . ") y FROM " .
$pmd_table . " WHERE db_name = '" . $db . "'";

PMA_query_as_controluser($sql,TRUE,PMA_DBI_QUERY_STORE);
```

where `$scale` is an unescaped POST variable.

Data sets

National Vulnerability Database (NVD)

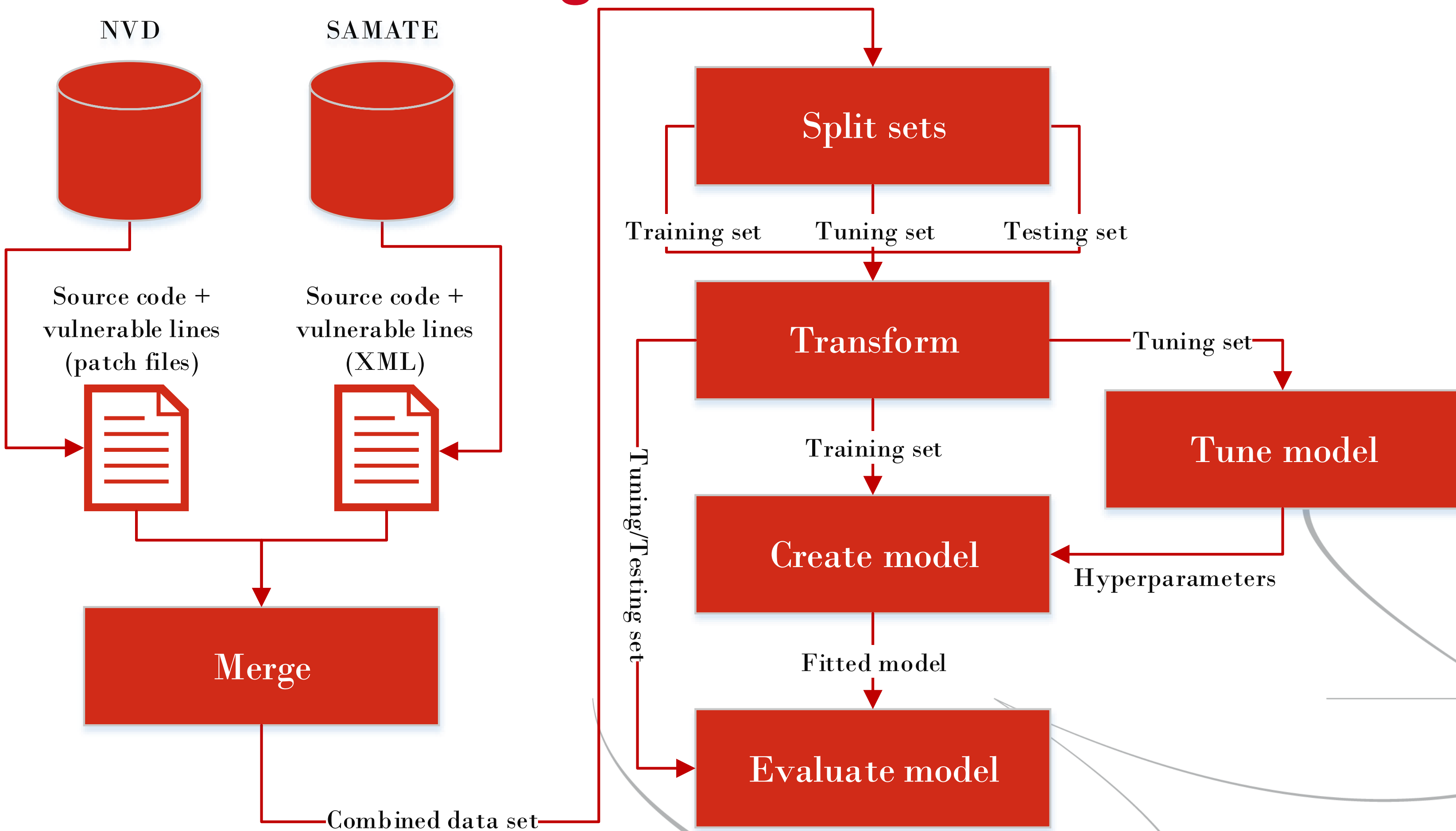
- Database containing vulnerabilities including classification
- Mine repositories to find affected code and fix

Software Assurance Metrics And Tool Evaluation (SAMATE)

- Data set from NIST
- Generated test cases by Bertrand Stivalet containing safe and unsafe examples

Open Universiteit
www.ou.nl

WIRECAML design



Taint Analysis

```
$student_id = $_GET['student_id'];
```

tainted source

```
$student_id = filter_var($student_id,
FILTER_SANITIZE_NUMBER_INT);

$query = "SELECT * FROM student where id='.$student_id.'";

$res = mysql_query($query);
```

Open Universiteit
www.ou.nl

Taint Analysis

```
$student_id = $_GET['student_id'];
```

tainted source

```
$student_id = filter_var($student_id,
FILTER_SANITIZE_NUMBER_INT);
```

sanitization function

```
$query = "SELECT * FROM student where id='.$student_id.'";

$res = mysql_query($query);
```

Open Universiteit
www.ou.nl

Taint Analysis

```
$student_id = $_GET['student_id'];
```

tainted source

```
$student_id = filter_var($student_id,
FILTER_SANITIZE_NUMBER_INT);
```

sanitization function

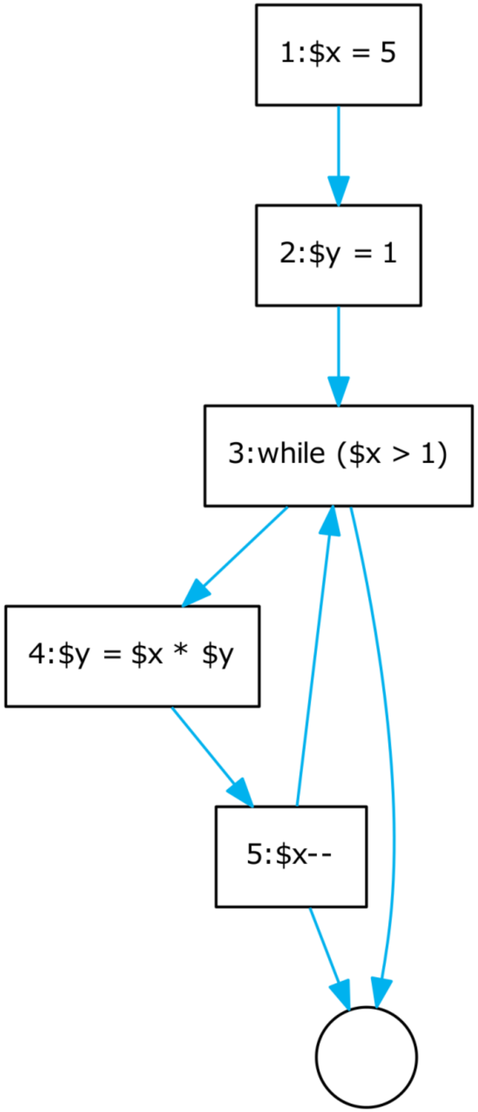
```
$query = "SELECT * FROM student where id='.$student_id.'";
```

```
$res = mysql_query($query);
```

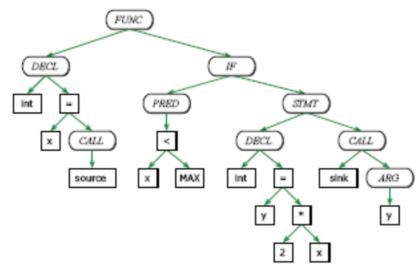
potentially vulnerable function or 'sink'

Open Universiteit
www.ou.nl

Control Flow Graphs

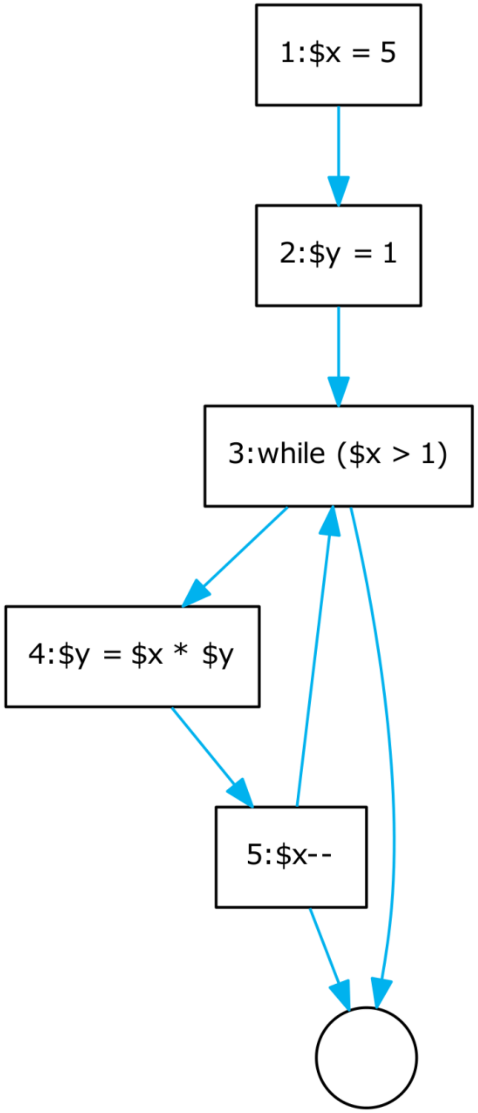



```
$x = 5;
$y = 1;
while ($x > 1) {
    $y = $x * $y;
    $x--;
}
```



AST

Reaching Definitions



$IN[1] = \emptyset$
$OUT[1] = \{x_1\}$

$IN[2] = \{x_1\}$
$OUT[2] = \{x_1, y_2\}$

$IN[3] = \{x_1, x_5, y_2, y_4\}$
$OUT[3] = \{x_1, x_5, y_2, y_4\}$

$IN[4] = \{x_1, x_5, y_2, y_4\}$
$OUT[4] = \{x_1, x_5, y_4\}$

$IN[5] = \{x_1, x_5, y_4\}$
$OUT[5] = \{x_5, y_4\}$

$GEN[1] = \{x_1\}$
$KILL[1] = \{x_5\}$

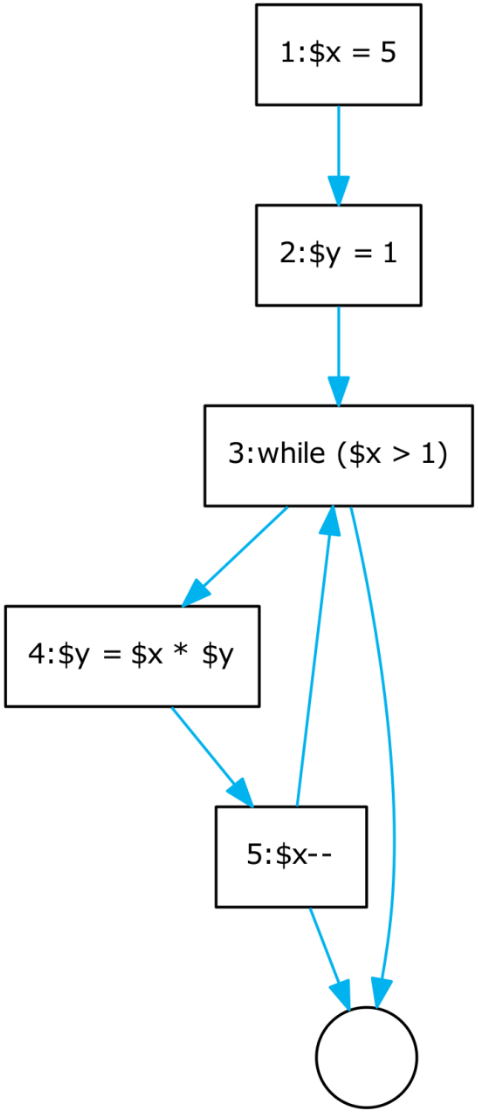
$GEN[2] = \{y_2\}$
$KILL[2] = \{y_4\}$

$GEN[3] = \emptyset$
$KILL[3] = \emptyset$

$GEN[4] = \{y_4\}$
$KILL[4] = \{y_2\}$

$GEN[5] = \{x_5\}$
$KILL[5] = \{x_1\}$

Reaching Definitions



$IN[1] = \emptyset$
$OUT[1] = \{x_1\}$

$GEN[1] = \{x_1\}$
$KILL[1] = \{x_5\}$

$IN[2] = \{x_1\}$
$OUT[2] = \{x_1, y_2\}$

$GEN[2] = \{y_2\}$
$KILL[2] = \{y_4\}$

$IN[3] = \{x_1, x_5, y_2, y_4\}$
$OUT[3] = \{x_1, x_5, y_2, y_4\}$

$GEN[3] = \emptyset$
$KILL[3] = \emptyset$

$IN[4] = \{x_1, x_5, y_2, y_4\}$
$OUT[4] = \{x_1, x_5, y_4\}$

$GEN[4] = \{y_4\}$
$KILL[4] = \{y_2\}$

$IN[5] = \{x_1, x_5, y_4\}$
$OUT[5] = \{x_5, y_4\}$

$GEN[5] = \{x_5\}$
$KILL[5] = \{x_1\}$

Use	UD chain
y_4	y_2, y_4
x_5	x_1, x_5

Open Universiteit
www.ou.nl

Data matrix

line	echo	mysql_ close	mysql_ connect	mysql_fetch_array	mysql_ query	mysql_ select_db	print_r	vulnerable
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	1	0	0	0	0	0
4	0	0	0	0	0	1	0	0
5	1	0	0	0	0	0	0	0
6	0	0	0	0	1	0	0	1
7	0	0	0	1	1	0	0	1
8	0	0	0	1	1	0	1	1
9	1	0	0	0	0	0	0	0
10	0	1	1	0	0	0	0	0

Open Universiteit
www.ou.nl

Classification (1/2)

We use probabilistic classifiers, such as:

- *Logistic Regression*
- *Decision Tree*
- *Random Forest*
- *Naïve Bayes*
- *Tree-augmented Naïve Bayes*

Probabilistic classifiers output ‘probabilities’. We need binary classification, so we use:

$$Y = Z > c$$

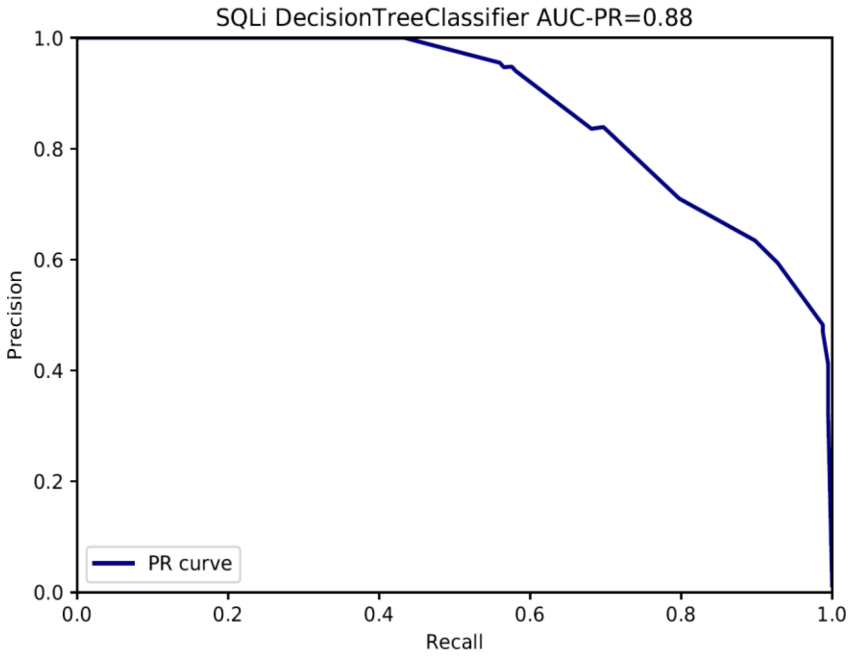
where Z is the set of output probabilities and c is a threshold value.

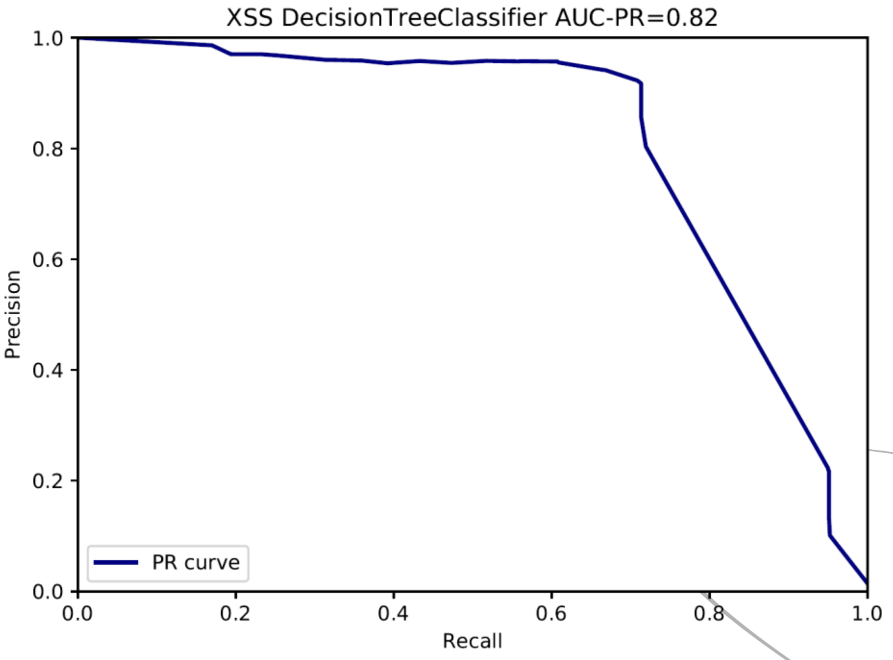
Open Universiteit
www.ou.nl

Classification (2/2)

- Recall is the fraction of vulnerabilities that were correctly identified among all vulnerabilities ($Recall = \frac{TP}{TP+FN}$)
- Precision is the fraction of vulnerabilities that were correctly identified among all identified vulnerabilities ($Precision = \frac{TP}{TP+FP}$)
- We use AUC-PR (=Area Under Precision-Recall-Curve) to evaluate the models.

Classifier performance (1/2)





Data set = NVD + SAMATE

Open Universiteit
www.ou.nl

Classifier performance (2/2)

SQLi samples

	AUC-PR
Decision Tree	**0.88**
Logistic Regression	0.87
Random Forest	0.85
TAN	0.75
Naive Bayes	0.64
Dummy	0.51

XSS samples

	AUC-PR
Decision Tree	**0.82**
Random Forest	**0.82**
TAN	0.81
Logistic Regression	0.79
Naive Bayes	0.69
Dummy	0.51

Data set = NVD + SAMATE

Feature performance (1/2)

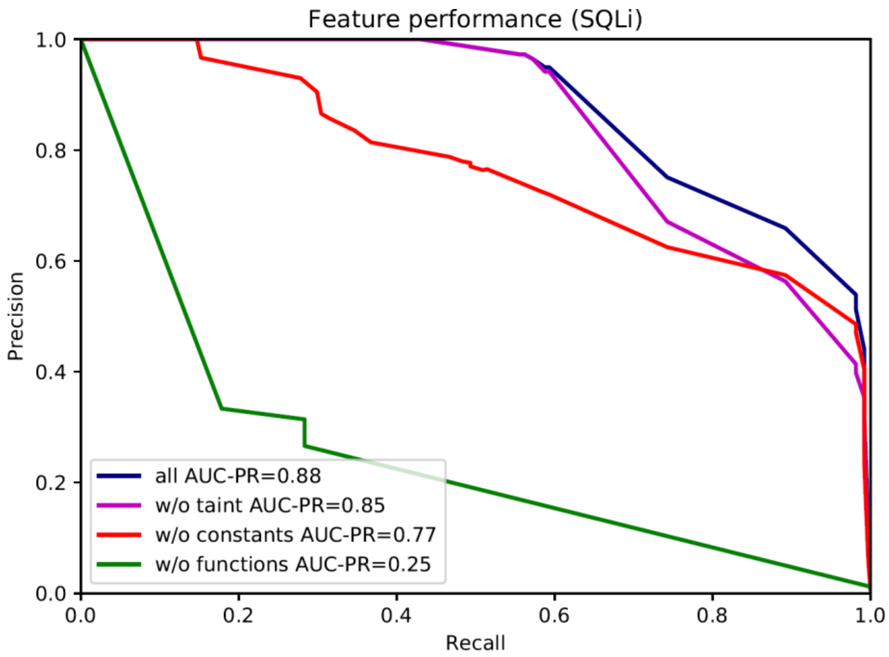
We use three types of features (all based on Reaching Definitions):

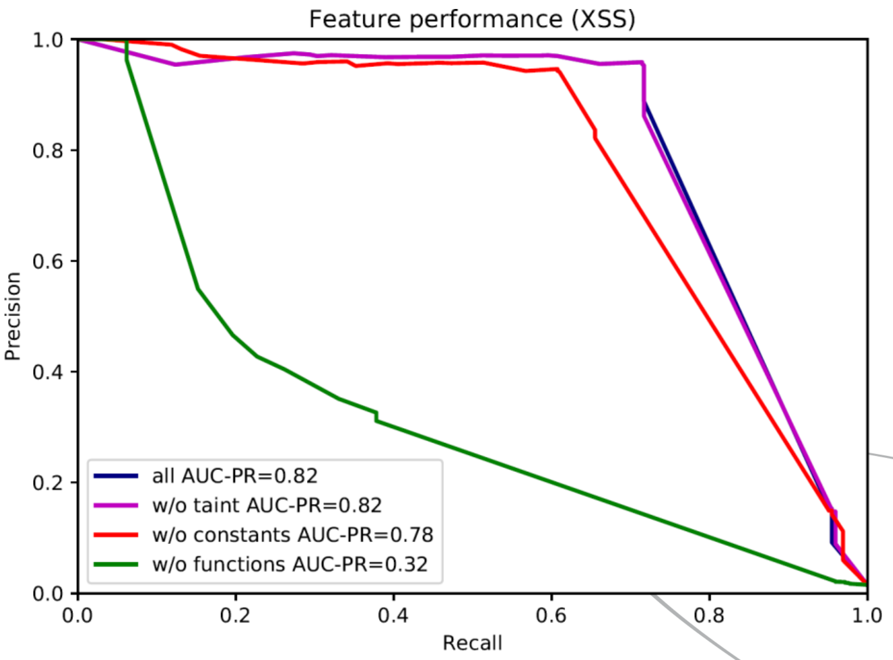
- *Functions*
- *Constants*
- *Taint*

A variable is considered untainted when its type is *float*, *int*, *double* or *bool*.

Open Universiteit
www.ou.nl

Feature performance (2/2)





Data set = NVD + SAMATE
Model = Decision Tree

Open Universiteit
www.ou.nl

Comparing against other tools (1/2)

Comparing against 4 OSS vulnerability scanners:

- *Pixy*
- *RIPS* (free version)
- *WAP*
- *Yasca*

Using F_1 score for evaluation:

$$F_1 = 2\frac{Precision * Recall}{Precision + Recall}$$

Open Universiteit
www.ou.nl

Comparing against other tools (2/2)

SQLi samples

	Precision	Recall	F_1-score
Our tool	**0.94**	**0.94**	**0.94**
Pixy	0.86	0.61	0.69
RIPS	0.83	0.80	0.82
WAP	0.83	0.84	0.83
Yasca	0.01	0.10	0.02

XSS samples

	Precision	Recall	F_1-score
Our tool	**0.79**	**0.71**	**0.71**
Pixy	0.61	0.61	0.61
RIPS	0.37	0.61	0.46
WAP	0.51	0.58	0.51
Yasca	0.24	0.25	0.24

Data set = SAMATE
Model = Decision Tree

Finding unknown vulnerabilities

Approach

- Train with our combined data set (NVD + SAMATE)
- Test against latest versions of OSS projects
- Examine all positives (or at least, as many as time allows us)

Result

We found an SQL injection in Piwigo 2.9.2, a photo-sharing web application (CVE-2018-6883)

Conclusions

- Extracted features from CFG and applied ML to find XSS/SQLi
- Shown performance of features and classifiers
- Best performance in comparison to other OSS tools
- Found previously unknown vulnerability

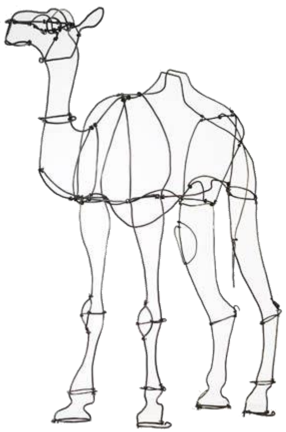
Future Work

- Other dynamic languages
- Other vulnerabilities
- Tracking arrays
- Use regular expressions as feature
- Deduplication instead of random sampling
- Different classifiers (such as RNN)
- Compare to commercial vulnerability scanners

Open Universiteit
www.ou.nl

Questions?

jj.kronjee@studie.ou.nl
https://github.com/jorkro/wirecaml



Open Universiteit
www.ou.nl