Automatic Detection and Correction of Web Application Vulnerabilities using Data Mining to Predict False Positives

Ibéria Medeiros¹, Nuno F. Neves¹, Miguel Correia²

University of Lisboa. ¹Faculty of Sciences, LaSIGE. ²IST, INESC-ID
PayPal vulnerability finally closed

Robert Kugler, 17, found a cross-site scripting flaw on the payment processing firm's website before claiming a reward under PayPal's bug bounty programme.

The hole was a critical one: it allowed attackers to inject arbitrary JavaScript code into the PayPal site, potentially enabling them to harvest users' access credentials.
Botnet forces infected Firefox users to hack the sites they visit
"Advanced Power" automates the process of finding sites vulnerable to data theft.

Advanced Power Botnet Uses Infected Computers to Seek Vulnerabilities
The botnet malware conducts SQL injection attacks on websites that infected users visit. So far, Advanced Power has detected more than 1,800 websites that are vulnerable to the attacks.
Motivation...

Big American retail stores have become a top target of cybercriminals, but the retail industry has very little incentive to beef up its security.

But retailers also see more to gain from collecting consumer information than protecting it. That magnetic stripe shares your name, bank and card information.

Hackers that once targeted banks exclusively now aim at retailers. In 2013, they recorded the highest number of data breaches in a decade, according to the Open Security Foundation.
Report: Cyberthreat Detection Lacking
Critical Infrastructure Security Incidents Go Unnoticed

Common methods used to infiltrate critical infrastructure organizations include attacks, spear phishing and SQL injection, according to the ICS-CERT report.
A British man has been charged with hacking into US Federal Reserve computer servers and stealing the personal information of users.

Love **used a SQL attack** to infiltrate the bank’s servers, hacking into the Federal Reserve Bank’s servers and **stealing names, email addresses and other personal information of the bank’s computer users.**
Motivation...

Web applications are exposed to malicious user inputs
Web applications with vulnerabilities are insecure
Web applications lack software security!

Web applications are mostly written in PHP... more than 77%
1. Taint Analysis: detect vulnerabilities
2. Data Mining: predict false positives
3. WAP Tool:
   - Taint analysis
   - Data mining
   - Code correction
4. Evaluation
5. Conclusion
Outline

1. Taint Analysis: detect vulnerabilities

2. Data Mining: predict false positives

3. WAP Tool:
   - Taint analysis
   - Data mining
   - Code correction

4. Evaluation

5. Conclusion
Vulnerability: SQL Injection example...

```php
$u = $_POST['user'];
$p = $_POST['password'];
$q = "SELECT * FROM users WHERE user='$u' AND pass='$p';"
$r = mysql_query($q);
```

SQL Injection exploited
$u = $_POST['user'];
$p = $_POST['password'];
$q = "SELECT * FROM users WHERE user='$u' AND pass='$p';"
$r = mysql_query($q);
$u = \$_POST[\'user\'];
$p = \$_POST[\'password\'];
$q = \"SELECT * FROM users WHERE user='$u' AND pass='$p'\";
$r = mysql_query($q);

If we could track the user inputs and verify if they reach some functions, then we could detect the vulnerability...
Taint Analysis: vulnerabilities detected

If we could track the user inputs and verify if they reach some functions, then we could detect vulnerabilities...

...Taint Analysis

Source Code Static Analysis

Vulnerabilities detected:

Most exploited:
- SQL Injection
- Cross Site Scripting (XSS)

Others:
- Remote file inclusion
- Local file inclusion
- Directory Traversal / Path Traversal
- Source code disclosure
- OS command injection
- PHP code injection
Taint Analysis: taintedness

How?

- taints all entry points (user inputs, e.g., \$_POST)
- follows the code propagating its taintedness
- until it reaches a sensitive sink (some functions, e.g., mysql_query)

```php
$u = \$_POST['user'];
$p = \$_POST['password'];
$q = "SELECT * FROM users WHERE user='\$u' AND pass='\$p'";
$r = mysql_query($q);
```

**SQL Injection detected**

<table>
<thead>
<tr>
<th>T</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>$u, $p</td>
</tr>
<tr>
<td>1</td>
<td>$q</td>
</tr>
</tbody>
</table>

T: taintedness
0: untaint, 1: taint
D: depends of..
PHP has *sanitization functions* to:
- sanitize the user inputs
- invalidate the exploiting of vulnerabilities

**Taint Analysis:**
- handles this
- does not propagate the taintedness

```php
$u = $_POST['user'];
$p = $_POST['password'];
$uu = mysql_real_escape_string($u);
$pp = mysql_real_escape_string($p);
$q = "SELECT * FROM users WHERE user='$uu' AND pass='$pp'";
$r = mysql_query($q);
```

<table>
<thead>
<tr>
<th>T</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>0</td>
<td>$u</td>
</tr>
<tr>
<td>0</td>
<td>$p</td>
</tr>
<tr>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>0</td>
<td>-</td>
</tr>
</tbody>
</table>

*T: taintedness*
(0: untaint, 1: taint)

*D: depends of..*
Taint Analysis: false positives

Taint analysis tends to generate **false positives**
(detected non-vulnerabilities)

How to avoid them?
- Not propagate taintedness when sanitization functions are present
- Propagate taintedness between: function calls and include files

**But... what happens when...**

```
$u = $_POST['user'];
$p = $_POST['password'];
$uu = substr($u, 2, 6);
$uu = trim($uu);
$q = "SELECT * FROM users WHERE user='$uu' AND pass='$p'";
$r = mysql_query($q);
```

---

**False Positive**

<table>
<thead>
<tr>
<th>T</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>$u</td>
</tr>
<tr>
<td>1</td>
<td>$uu</td>
</tr>
<tr>
<td>1</td>
<td>$uu, $p</td>
</tr>
<tr>
<td>1</td>
<td>$q</td>
</tr>
</tbody>
</table>

**T**: taintedness (0: untaint, 1: taint)

**D**: depends of...

---

**We have a problem!!! How to solve it?**
What are the symptoms of the possible existence of a false positive?

If in the code slice between the entry point and the sensitive sink the user input is:

- changed
  - string manipulation functions (e.g., `substr`)
  - concatenation operations

- validated
  - type checking functions (e.g., `isset`, `is_string`)
  - white and black lists

- manipulated in SQL queries
  - aggregate functions (e.g., `avg`, `sum`)
  - complex query
  - FROM clause

**The presence of each of these symptoms is an attribute**

If we could verify the presence of these attributes in the slice, then we could classify the vulnerabilities as being a FP or real
Approach to resolve false positives

If we could verify the presence of these attributes in the slice, then we could classify the vulnerabilities as being a FP or real

Coding this knowledge
- hard and complex task
- a lot of if's statements
- can incur in logic errors

if with these attributes we could retrieve and discover information then we could classify the vulnerabilities

Discover the knowledge

Data Mining
Outline

1. Taint Analysis: detect vulnerabilities

2. Data Mining: predict false positives

3. WAP Tool
   - Taint analysis
   - Data mining
   - Code correction

4. Evaluation

5. Conclusion
What do we have...

Taint Analysis

detects

Candidate Vulnerabilities

What do we want...

Real Vulnerabilities

confirm

False Positives

predicts

Data Mining
What do we need to apply data mining?

**A set of attributes that characterize a false positive**
- string manipulation, validation, manipulation in SQL queries

**Class values to classify an instance in the class**
- is a FP (Y); is not a FP (N = real Vulnerability)

**A data set of instances with FP and real vulnerabilities to train a classifier**
- 76 instances: 32 false positives; 44 real vulnerabilities. Data set was obtained manually by a hard and tedious process

**A machine learning classifier to classify new instances**

We define a **process** to evaluate machine learning classifiers to choose the best that classify our instances with high accuracy and precision
Composition of the data set used

- 76 instances: 32 false positives + 44 real vulnerabilities
- 15 attributes: 14 to characterize a false positive + 1 to classify it
Evaluation of classifiers

Using WEKA to:

- evaluate 10 machine learning classifiers
  - 5 Graphical/symbolic algorithms (decision trees)
  - 3 Probabilistic algorithms (NB, K-NN, LR)
  - 2 Neural network algorithms (MLP, SVM)

- train and test the classifiers with 10-fold cross validation estimator
  - divides the data into 10 buckets, trains the classifier with 9 of them and tests it with the 10th. This process is repeated 10 times to test every bucket with the classifier trained with the rest

- get 10 metrics to evaluate the classifiers performance
  - 3 for false positives prediction
  - 3 for real vulnerabilities detection
  - 2 global metrics (accuracy and precision)
  - 2 global tests
Evaluation of classifiers

Metrics are based in a confusion matrix

<table>
<thead>
<tr>
<th>Predicted</th>
<th>Observed</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Yes (FP)</td>
<td>No (not FP)</td>
</tr>
<tr>
<td>Yes (FP)</td>
<td></td>
<td>match</td>
</tr>
<tr>
<td>No (not FP)</td>
<td>not match</td>
<td></td>
</tr>
</tbody>
</table>

Logistic Regression

Accuracy = 92.1%
Precision = 92.5%

Whenever more data is included in the data set, we can redo the process to define a better classifier
Outline

1. Taint Analysis: detect vulnerabilities
2. Data Mining: predict false positives
3. WAP Tool
   - Taint analysis
   - Data mining
   - Code correction
4. Evaluation
5. Conclusion
Approach

**WAP (Web Application Protection)**

**Analysis**
- searches for candidate input validation vulnerabilities in the source code of a PHP web application

**Prediction**
- predicts if a candidate vulnerability is a false positive or a real vulnerability

**Correction**
- inserts fixes in the source code to remove the vulnerabilities

**Feedback**
- reports the real vulnerabilities detected and how they were corrected
- outputs a corrected version of the web application
- reports the false positives predicted
Vulnerabilities are removed:
Correction of source code by insertion of fixes in the right places

Fixes:
What do they do?
- Do proper validation or sanitization of user input before it is used in some sensitive sink

Where are they inserted?
- In sensitive sinks or close to them
- Do not compromise the behavior of the web application

What are they?
- Small PHP functions developed by us
- Some do sanitization and others do validation
Architecture

PHP Source Code

Lexer | Parser ➔ AST

Tree Generator

Code Analyzer

Entry Points ➔ Sensitive Sinks ➔ PHP Sanitization Functions ➔ Taint analysis ➔ Slice of candidate vulnerabilities

False Positive Information

Identifying the right places ➔ Correcting the source code ➔ Fixes

Protected Source Code

Code Corrector

False Positives Predictor

False positives ➔ Predicting false positives ➔ Slice of real vulnerabilities

Collecting attributes

Collecting attributes ➔ Training data set ➔ Logistic Regression classifier
Challenges of implementing WAP

Global, interprocedural and context-sensitive analysis

- To avoid false positives and false negatives the taintedness propagation has to be:
  - interprocedural: between functions or methods calls
  - global: even if the functions/methods belong to different modules
  - context-sensitive: to the point of the program where the function call was made

Environment variables

- Resolve the name of the include files to perform the corrected global analysis

Class analysis

- Handles taintedness propagation between objects and methods calls

Uncertainty about PHP's syntax

- PHP is weakly typed and not formally specified. Frequent use of poorly documented features that can break the parser
Challenges of implementing WAP

Need of both top-down and bottom-up approaches

- navigates in the AST using the top-down approach to taint the entry points, then following the bottom-up approach to propagate its taintedness to its parent
- identifies the vulnerable path and the right places to insert the fixes using the bottom-up approach
- collects the attributes and performs the correction of the source code using the top-down approach
Outline

1. Taint Analysis: detect vulnerabilities
2. Data Mining: predict false positives
3. WAP Tool
   - Taint analysis
   - Data mining
   - Code correction

4. Evaluation

5. Conclusion
Comparison of WAP vs Pixy

- **Pixy** is a static analysis tool that performs taint analysis to detect SQL injection and XSS vulnerabilities.

- Taint analysis comparative evaluation between WAP and **Pixy** in analysis of 10 open source code packages:

<table>
<thead>
<tr>
<th>Webapp</th>
<th>WAP-TA</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>Pixy</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>WAP (complete)</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>Corrected</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SQLI</td>
<td>XSS</td>
<td>FP</td>
<td>FN</td>
<td>SQLI</td>
<td>XSS</td>
<td>FP</td>
<td>FN</td>
<td>SQLI</td>
<td>XSS</td>
<td>FP</td>
<td>FN</td>
<td>SQLI</td>
<td>XSS</td>
<td>FP</td>
<td>FN</td>
<td>Corrected</td>
<td></td>
</tr>
<tr>
<td>CurrentCost</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>5</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DVWA 1.0.7</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>emoncms</td>
<td>2</td>
<td>6</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Measureit 1.14</td>
<td>1</td>
<td>7</td>
<td>7</td>
<td>0</td>
<td>1</td>
<td>16</td>
<td>16</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>7</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mfm-0.13</td>
<td>0</td>
<td>8</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>8</td>
<td>3</td>
<td>0</td>
<td>5</td>
<td>3</td>
<td>0</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multilidae 2.3.5</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SAMATE</td>
<td>3</td>
<td>11</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>11</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>11</td>
<td>0</td>
<td>0</td>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vicnum15</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wackopicko</td>
<td>3</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>3</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ZiPEC 0.32</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>7</td>
<td>8</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>22</td>
<td>46</td>
<td>21</td>
<td>0</td>
<td>20</td>
<td>53</td>
<td>41</td>
<td>5</td>
<td>14</td>
<td>33</td>
<td>21</td>
<td>0</td>
<td><strong>47</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- 68 vuln.: 21 are FP
- 0 false negatives
- Same 11 FP than Pixy

- 73 vuln.: 41 are FP
- 5 false negatives
- Same 11 FP than WAP
- + 30 FP than WAP

- Without data mining

- With data mining

47 real vulnerabilities
- 21 predicted false positives
- 0 false negatives
- 47 vulnerabilities corrected
Comparison of WAP vs PhpMinerII

- PhpMinerII is a tool that predicts the presence of SQLI/XSS vulnerabilities in PHP applications. On the contrary to WAP, it does not identify where the vulnerabilities are, only predicts their existence.

- PhpMinerII does data mining of slices that end at a sensitive sink, independently of data being propagated through them starting at an entry point or not.

### Observed

<table>
<thead>
<tr>
<th></th>
<th>Yes (Vuln.)</th>
<th>No (not Vuln.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes (Vuln.)</td>
<td>48</td>
<td>5</td>
</tr>
<tr>
<td>No (not Vuln.)</td>
<td>5</td>
<td>20</td>
</tr>
</tbody>
</table>

### Logistic Regression

- Accuracy = 87.2%
- Precision = 85.2%

The 48 vulnerabilities can contain false positives

- 47 real vulnerabilities
- 21 predicted false positives
- 0 false negatives
- 47 vulnerabilities corrected

WAP with data mining
## Full comparative evaluation

<table>
<thead>
<tr>
<th>Metric</th>
<th>WAP</th>
<th>Pixy</th>
<th>PhpMinerII</th>
</tr>
</thead>
<tbody>
<tr>
<td>accuracy</td>
<td>92.1%</td>
<td>44.0%</td>
<td>87.2%</td>
</tr>
<tr>
<td>precision</td>
<td>92.5%</td>
<td>50.0%</td>
<td>85.2%</td>
</tr>
</tbody>
</table>
### WAP analysis to all vulnerabilities

<table>
<thead>
<tr>
<th>Webapp</th>
<th>Detected taint analysis</th>
<th>Detected data mining</th>
<th>Corrected</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SQLI</td>
<td>RFI, LFI DT/PT</td>
<td>SCD</td>
</tr>
<tr>
<td>currentcost</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DVWA 1.0.7</td>
<td>4</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>emoncms</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Measureit 1.14</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Mfm 0.13</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Mutillidae 2.3.5</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>OWASP Vicnum</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SRD(1)</td>
<td>3</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>Wackopico</td>
<td>3</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>ZiPEC 0.32</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>22</td>
<td>11</td>
<td>0</td>
</tr>
</tbody>
</table>
## Summary of the WAP analysis

<table>
<thead>
<tr>
<th>Web application</th>
<th>Files</th>
<th>Lines of code</th>
<th>Analysis time (s)</th>
<th>Vuln. files</th>
<th>Vulner. found</th>
</tr>
</thead>
<tbody>
<tr>
<td>adminer-1.11.0</td>
<td>45</td>
<td>5,434</td>
<td>27</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Butterfly insecure</td>
<td>16</td>
<td>2,364</td>
<td>3</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>Butterfly secure</td>
<td>15</td>
<td>2,678</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>currentcost</td>
<td>3</td>
<td>270</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>dmoz2mysql</td>
<td>6</td>
<td>1,000</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DVWA 1.0.7</td>
<td>310</td>
<td>31,407</td>
<td>15</td>
<td>12</td>
<td>15</td>
</tr>
<tr>
<td>emoncns</td>
<td>76</td>
<td>6,876</td>
<td>6</td>
<td>6</td>
<td>15</td>
</tr>
<tr>
<td>Ghost</td>
<td>16</td>
<td>398</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>glibitron-PIP</td>
<td>14</td>
<td>328</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>GTD-PHP</td>
<td>62</td>
<td>4,853</td>
<td>10</td>
<td>33</td>
<td>111</td>
</tr>
<tr>
<td>Hexjector 1.0.6</td>
<td>11</td>
<td>1,640</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Lithuanian-7.02.05-v1.6</td>
<td>132</td>
<td>3,790</td>
<td>24</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Measureit 1.14</td>
<td>2</td>
<td>967</td>
<td>2</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>Mfm 0.13</td>
<td>7</td>
<td>5,859</td>
<td>6</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>Mutillidae 1.3</td>
<td>18</td>
<td>1,623</td>
<td>6</td>
<td>10</td>
<td>19</td>
</tr>
<tr>
<td>Mutillidae 2.3.5</td>
<td>578</td>
<td>102,567</td>
<td>63</td>
<td>7</td>
<td>10</td>
</tr>
<tr>
<td>ocsvg-0.2</td>
<td>4</td>
<td>243</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>OWASP Vicnum</td>
<td>22</td>
<td>814</td>
<td>2</td>
<td>7</td>
<td>18</td>
</tr>
<tr>
<td>paCRUD 0.7</td>
<td>100</td>
<td>11,079</td>
<td>11</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Peruggia</td>
<td>10</td>
<td>988</td>
<td>2</td>
<td>6</td>
<td>22</td>
</tr>
<tr>
<td>PHP X Template 0.4</td>
<td>10</td>
<td>3,009</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PhpBB 1.4.4</td>
<td>62</td>
<td>20,743</td>
<td>25</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Phpcms 1.2.2</td>
<td>6</td>
<td>227</td>
<td>2</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>PhpCrud</td>
<td>6</td>
<td>612</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PhpDiary-0.1</td>
<td>9</td>
<td>618</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PHPFusion</td>
<td>633</td>
<td>27,000</td>
<td>40</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>phpldapadmin-1.2.3</td>
<td>97</td>
<td>28,601</td>
<td>9</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PHPLib 7.4</td>
<td>73</td>
<td>13,383</td>
<td>35</td>
<td>3</td>
<td>14</td>
</tr>
<tr>
<td>PHPMyAdmin 2.0.5</td>
<td>40</td>
<td>4,730</td>
<td>18</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PHPMyAdmin 2.2.0</td>
<td>34</td>
<td>9,430</td>
<td>12</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PHPMyAdmin 2.6.3-pl1</td>
<td>287</td>
<td>143,171</td>
<td>105</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Phpweather 1.52</td>
<td>13</td>
<td>2,465</td>
<td>9</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>WebCalendar</td>
<td>122</td>
<td>30,173</td>
<td>12</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>WebScripts</td>
<td>5</td>
<td>391</td>
<td>4</td>
<td>2</td>
<td>14</td>
</tr>
<tr>
<td>ZiPEC 0.32</td>
<td>10</td>
<td>765</td>
<td>2</td>
<td>1</td>
<td>7</td>
</tr>
</tbody>
</table>

**Total** | **2854** | **470,496** | **473** | **107** | **294**
Conclusion

- Web applications can have input validation vulnerabilities

- We present an approach and a tool called WAP to automatically identify and correct these vulnerabilities and to predict false positives using data mining

- The WAP tool was compared with Pixy and PhpMinerII, analyzing 10 open source code packages, and:
  - had better performance in detection of SQLI and XSS vulnerabilities with and without data mining

- WAP analyzed 35 applications (~470K LOC in ~2900) and identified ~300 vulnerabilities