Securing Energy Metering Software with Automatic Source Code Correction

Ibéria Medeiros\textsuperscript{1}, Nuno F. Neves\textsuperscript{1}, Miguel Correia\textsuperscript{2}

University of Lisboa. \textsuperscript{1}Faculty of Sciences, LaSIGE. \textsuperscript{2}IST, INESC-ID
Motivation

Energy Metering Software
Motivation

Energy Metering Software

Web Applications
Motivation

Energy Metering Software

Web App

*Input validation vulnerabilities*
Motivation

Energy Metering Software

- violation of user privacy
- countering the benefits of metering
- attack other user software

Web App

Input validation vulnerabilities
Motivation

Energy Metering Software

Web App

Input validation vulnerabilities

WAP

Web Application Protection:
- searches vulnerabilities in source code
- removes them automatically
- protects the source code with fixes
Motivation

**Energy Metering Software**

Web App

**Protected**

**Web Application Protection:**
- searches vulnerabilities in source code
- removes them automatically
- protects the source code with fixes

attacks
Motivation

Securing Energy Metering Software with Automatic Source Code Correction

Web Application Protection:
- searches vulnerabilities in source code
- removes them automatically
- protects the source code with fixes
Outline

1. Energy Metering Software
2. The WAP Approach and Tool
3. Vulnerabilities Discovered
4. Conclusions
What is it?

- Software that permits monitoring and analysis of electricity consumption information provided by energy power meters.

- Energy power meters: measure the amount of electricity consumption, and can transmit it to remote places to be processed by energy metering software (EMS).

- EMS (in this work) can be a web application with web interfaces for the user to:
  - view the information analyzed.
  - insert data to be stored into database management system (DBMS), such as MySql.

- emoncms and measureit are examples EMS that implement these functionalities and are written mostly in PHP language.
Vulnerabilities and Attacks

- EMS can be vulnerable to common web application vulnerabilities: SQL injection (SQLI), cross site scripting (XSS), etc.
- These vulnerabilities allow an attacker modify the behavior of the application or tamper with its data.
2. THE WAP APPROACH AND TOOL
Web Application Vulnerabilities

What are they?
- Bugs in the application source code.
- An input malformed can explore these bugs.
- Resulting an unexpected behavior of the web application.

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

Others:
- Remote file inclusion
- Local file inclusion
- Directory path / Traversal path
- Source code disclosure
- OS command injection
- Eval injection

How to avoid input validation vulnerabilities:
- Doing proper validation or sanitization of user input before using it in some sensitive sink (function susceptible to be explored).
SQL Injection

Technical features

- Direct victim: server
- Input malformed (with SQL metacharacters) is inserted in SQL query and executed in the database without any validation
SQL Injection

Technical features

- Direct victim: server
- Input malformed (with SQL metacharacters) is inserted in SQL query and executed in the database without any validation

PHP vulnerable example

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

Technical features

- Direct victim: server
- Input malformed (with SQL metacharacters) is inserted in SQL query and executed in the database without any validation

PHP vulnerable example

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

![Password input field with SQL injection example](image)
SQL Injection

Technical features

- Direct victim: server
- Input malformed (with SQL metacharacters) is inserted in SQL query and executed in the database without any validation

PHP vulnerable example

```php
$u = "administrator' -- ";
$p = "any";
$q = "SELECT * FROM users WHERE user='$u' AND pass='$p'";
$r = mysql_query($q);
```
SQL Injection

Technical features
- Direct victim: server
- Input malformed (with SQL metacharacters) is inserted in SQL query and executed in the database without any validation

PHP vulnerable example

```php
$u = "administrator" -- ";
$p = "any";
$q = "SELECT * FROM users WHERE user='administrator'-- ' AND pass='any"; 
$r = mysql_query($q);
```
The WAP Approach and Tool

Input Validation Vulnerabilities

**SQL Injection**

Technical features
- Direct victim: server
- Input malformed (with SQL metacharacters) is inserted in SQL query and executed in the database without any validation

PHP vulnerable example

```php
$u = "administrator' -- ";
$p = "any";
$q = "SELECT * FROM users WHERE user='administrator'";
$r = mysql_query($q);
```
SQL Injection

Technical features

- Direct victim: server
- Input malformed (with SQL metacharacters) is inserted in SQL query and executed in the database without any validation

PHP vulnerable example

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

Technical features
- Direct victim: server
- Input malformed (with SQL metacharacters) is inserted in SQL query and executed in the database without any validation

PHP vulnerable example

```
$u = mysql_real_escape_string($_POST['user']);
$p = mysql_real_escape_string($_POST['password']);
$q = "SELECT * FROM users WHERE user='$u' AND pass='$p';"
$r = mysql_query($q);
```
Cross Site Scripting - XSS

Technical features

- Direct victim: client
- Running a malicious script in the browser of the victim (e.g. JavaScript) without any validation or output encoding
Cross Site Scripting - XSS

Technical features

- Direct victim: client
- Running a malicious script in the browser of the victim (e.g. JavaScript) without any validation or output encoding

PHP vulnerable example

```php
$user = $_POST['user'];
echo Welcome . $user;
```
Cross Site Scripting - XSS

Technical features

- Direct victim: client
- Running a malicious script in the browser of the victim (e.g. JavaScript) without any validation or output encoding

PHP vulnerable example
$user = $_POST['user'];

```php
echo Welcome . $user;
```

Alice `<script>alert('XSS attack');</script>`
Cross Site Scripting - XSS

Technical features

- Direct victim: client
- Running a malicious script in the browser of the victim (e.g. JavaScript) without any validation or output encoding

PHP vulnerable example

```php
$user = "Alice <script>alert('XSS attack')</script>";
echo Welcome Alice <script>alert('XSS attack')</script>;
```
Cross Site Scripting - XSS

Technical features

- Direct victim: client
- Running a malicious script in the browser of the victim (e.g. JavaScript) without any validation or output encoding

PHP vulnerable example

```
$user = "Alice <script>alert('XSS attack')</script>; 
Echo Welcome Alice <script>alert('XSS attack')</script>;
```

![Image of XSS attack alert box]
Cross Site Scripting - XSS

Technical features

- Direct victim: client
- Running a malicious script in the browser of the victim (e.g. JavaScript) without any validation or output encoding

PHP vulnerable example

```php
$user = $_POST['user'];
echo 'Welcome . ' . $user;
```
Cross Site Scripting - XSS

Technical features

- Direct victim: client
- Running a malicious script in the browser of the victim (e.g. JavaScript) without any validation or output encoding

PHP vulnerable example

```
$user = htmlentities($_POST['user']);
echo Welcome . $user;
```
**WAP (Web Application Protection)** is a tool that does:

1. **Analysis**
   - Analyzes the source code of a PHP web application
   - Searches for input validation vulnerabilities (presented above)
WAP (Web Application Protection) is a tool that does:

1. Analysis
   - Analyzes the source code of a PHP web application
   - Searches for input validation vulnerabilities (presented above)

2. Correction
   - Removes the discovered vulnerabilities
   - Inserts fixes in the source code (instructions that validate the input, called sanitization functions)
   - Outputs a corrected version of the web application.
The WAP Approach and Tool

**WAP** (Web Application Protection) is a tool that does:

1. **Analysis**
   - Analyzes the source code of a PHP web application
   - Searches for input validation vulnerabilities (presented above)

2. **Correction**
   - Removes the discovered vulnerabilities
   - Inserts fixes in the source code (instructions that validate the input, called sanitization functions)
   - Outputs a corrected version of the web application.

3. **Teaching**
   - Reports the vulnerabilities detected and how they were corrected
   - Teaches the programmer how to avoid inserting similar vulnerabilities and how to build secure software.
Architecture

The WAP Approach and Tool

1. Code Analyzer
   - tree generator
     - lexer
     - parser
     - AST
   - taint analysis
     - tree walkers
     - entry points
     - sensitive sinks
     - untainted data
     - PHP sanitization functions
     - tainted symbol table
     - tainted exec. path tree

2. Code Corrector
   - tainted exec. path tree
   - tainted symbol table
   - vulnerable code
   - PHP & WAP sanitization functions

3. Protected source code
### Architecture

**Code Analyzer**
- Does static analysis, more precisely taint analysis.
- **Tree generator:**
  - parses the source code and generates an abstract syntax tree (AST) that represents that code.
  - includes a lexer and a parser to create the AST.
- **Taint analyser:**
  - tree walkers for walks through the ASTs to identify the input validation vulnerabilities.

**Code Corrector**

**PHP source code**
Architecture

Code Analyzer

- Does static analysis, more precisely taint analysis.
- Tree generator:
  - parses the source code and generates an abstract syntax tree (AST) that represents that code.
  - includes a lexer and a parser to create the AST.
- Taint analyser:
  - tree walkers for walks through the ASTs to identify the input validation vulnerabilities.

Code Corrector

- Identifies the fix to insert for each vulnerability found.
- Identifies the place in the source code where the fix needs to be inserted.
- Modifies the file where that place is.
- Reports the vulnerabilities detected and how they were corrected.
Taint analysis

Main purpose:
- Track the user inputs (without proper validation) to see if they reach a sensitive sink.

   *How:*
   - starting from an entry point (e.g., \$_POST)
   - follows the code by walking through the AST
   - until reaches a sensitive sink (e.g., mysql_query)

- If such a case is found, it is a vulnerability

Propagate or remove taintedness:
- The user inputs are considered tainted (not trusted, compromised)
- **Propagate:** a variable receives an user input not sanitized. The variable is **tainted**.
- **Remove:** a variable receives an user input sanitized by a sanitization function (e.g., mysql_real_escape_string). The variable is **untainted**.
3. **Vulnerabilities Detected**
Summary of the EMS analysis

- WAP tool analyzed *emoncms* and *measureit* applications.
- PHP files are the most of the interaction with the users, i.e., attack surface of the applications.

<table>
<thead>
<tr>
<th>webapp / vuln. files</th>
<th>files</th>
<th>app. LOCs</th>
<th>file LOCs</th>
<th>SQLI</th>
<th>XSS</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>emoncms</em> - modules</td>
<td>7</td>
<td>1,089</td>
<td>63</td>
<td>-3</td>
<td>-3</td>
</tr>
<tr>
<td>- igraph3.php</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>emoncms</em> - extras</td>
<td>7</td>
<td>291</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>- embed.php</td>
<td></td>
<td></td>
<td>48</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>- kwhdstacked.php</td>
<td></td>
<td></td>
<td>47</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>- kwhdzoomer.php (old)</td>
<td></td>
<td></td>
<td>44</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>- kwhdzoomer.php</td>
<td></td>
<td></td>
<td>72</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td><em>emoncms</em> - examples</td>
<td>62</td>
<td>5,496</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- user.php</td>
<td></td>
<td></td>
<td>177</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td><em>measureit</em> v.1.14</td>
<td>2</td>
<td>967</td>
<td></td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>- measureit_functions.php</td>
<td></td>
<td></td>
<td>915</td>
<td></td>
<td></td>
</tr>
<tr>
<td>total</td>
<td>78</td>
<td>7,843</td>
<td>1,366</td>
<td>3</td>
<td>17</td>
</tr>
</tbody>
</table>

*emoncms*: 2 SQLI, 13 XSS (3 of them are possible false positives)

*measureit*: 1 SQLI, 4 XSS
**emoncms: reflected XSS**

File *kwhdzoomer.php*

---

Vulnerability n.: 1

Vulnerable code:
18: `$kwhd = $_GET['kwhd'];`
69: `echo $kwhd;`

Corrected code:
18: `$kwhd = san_out($_GET['kwhd']);`
69: `echo $kwhd;`

---

Vulnerability n.: 2

Vulnerable code:
17: `$power = $_GET['power'];`
70: `echo $power;`

Corrected code:
17: `$power = san_out($_GET['power']);`
70: `echo $power;`

**san_out**, a WAP sanitization function that calls functions of the OWASP PHP Anti-XSS Library
emoncms: reflected XSS

File kwhdzoomer.php

Vulnerabilities Detected

Vulnerability n.: 1
Vulnerable code:
18: $kwhd = $_GET['kwhd'];
69: echo $kwhd;

Corrected code:
18: $kwhd = san_out($_GET['kwhd']);
69: echo $kwhd;

Vulnerability n.: 2
Vulnerable code:
17: $power = $_GET['power'];
70: echo $power;

Corrected code:
17: $power = san_out($_GET['power']);
70: echo $power;

san_out, a WAP sanitization function that calls functions of the OWASP PHP Anti-XSS Library
**emoncms: reflected XSS**

File *kwhdzoomer.php*

---

**Vulnerability n.: 1**

**Vulnerable code:**

```php
18: $kwhd = $_GET['kwhd'];
69: echo $kwhd;
```

**Corrected code:**

```php
18: $kwhd = \texttt{san\_out}($_GET['kwhd']);
69: echo $kwhd;
```

---

**Vulnerability n.: 2**

**Vulnerable code:**

```php
17: $power = $_GET['power'];
70: echo $power;
```

**Corrected code:**

```php
17: $power = \texttt{san\_out}($_GET['power']);
70: echo $power;
```

---

\texttt{san\_out}, a WAP sanitization function that calls functions of the OWASP PHP Anti-XSS Library

---

**VSL:** violate user privacy

- accessing to user data in the server.
- sending it to some server controlled by the attacker.
Vulnerabilities Detected

**emoncms**: reflected XSS

File `kwhdzoomer.php`

<table>
<thead>
<tr>
<th>Vulnerability</th>
<th>Code</th>
<th>Corrected Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>18: <code>$kwhd = $_GET['kwhd']</code>; 69: echo $kwhd;</td>
<td>18: <code>$kwhd = san_out($_GET['kwhd'])</code>; 69: echo $kwhd;</td>
</tr>
<tr>
<td>2</td>
<td>17: <code>$power = $_GET['power']</code>; 70: echo $power;</td>
<td>17: <code>$power = san_out($_GET['power'])</code>; 70: echo $power;</td>
</tr>
</tbody>
</table>

---

**violate user privacy**
- accessing to user data in the server.
- sending it to some server controlled by the attacker.

**counter the benefits of metering**
- sending a request to the server.
- causing the modification of the data stored there.

---

`san_out`, a WAP sanitization function that calls functions of the OWASP PHP Anti-XSS Library
**emoncms: reflected XSS**

File *kwhdzoomer.php*

= = = = Vulnerability n.: 1 = = = =
Vulnerable code:
18: `$kwhd = $_GET['kwhd']`;
69: echo $kwhd;

Corrected code:
18: `$kwhd = san_out($_GET['kwhd'])`;
69: echo $kwhd;

= = = = Vulnerability n.: 2 = = = =
Vulnerable code:
17: `$power = $_GET['power']`;
70: echo $power;

Corrected code:
17: `$power = san_out($_GET['power'])`;
70: echo $power;

---

**Vulnerabilities Detected**

- **violates user privacy**
  - accessing to user data in the server.
  - sending it to some server controlled by the attacker.

- **counter the benefits of metering**
  - sending a request to the server.
  - causing the modification of the data stored there.

- **attack the software user**
  - stealing user cookies and sending them to some server controlled by the attacker.
  - This user can be an engineer or an administrator of a company.
  - This can be used as platform for another attack.

---

`san_out`, a WAP sanitization function that calls functions of the OWASP PHP Anti-XSS Library
Vulnerabilities Detected

**emoncms: SQL Injection**

File **user.php**

Vulnerable code:

```php
140: $username = $_POST['username'];
144: $result = db_query("SELECT id,password, salt FROM users WHERE username = '"$username"');
16: return $result = mysql_query($query);
   (/home/iberiam/Desktop/Grib/emoncms_1/emoncms_examples-master/feed01/includes/db.php)
```

Corrected code:

```php
140: $username = mysql_real_escape_string($_POST['username']);
144: $result = db_query("SELECT id,password, salt FROM users WHERE username = '"$username"');
16: return $result = mysql_query($query);
   (/home/iberiam/Desktop/Grib/emoncms_1/emoncms_examples-master/feed01/includes/db.php)
```
**emoncms: SQL Injection**

File `user.php`

Vulnerable code:

```php
140: $username = " OR 1=1 INTO OUTFILE '/var/www/html/vulnsite/login-info.html' -- " ;
144: $result = db_query("SELECT id,password, salt FROM users WHERE username = '"$username'"" );
16: return $result = mysql_query($query);
    (/home/iberiam/Desktop/Grib/emoncms_1/emoncms_examples-master/feed01/includes/db.php)
```

Corrected code:

```php
140: $username = mysql_real_escape_string($_POST['username']);
144: $result = db_query("SELECT id,password, salt FROM users WHERE username = '"$username'"" );
16: return $result = mysql_query($query);
    (/home/iberiam/Desktop/Grib/emoncms_1/emoncms_examples-master/feed01/includes/db.php)
```
Vulnerabilities Detected

**emoncms: SQL Injection**

File `user.php`

Vulnerable code:

```php
140: $username = "' OR 1=1 INTO OUTFILE '/var/www/html/vulnsite/login-info.html' -- ";
144: $result = db_query("SELECT id,password, salt FROM users WHERE username = "'$username'"PLETED);
16: return $result = mysql_query($query);
    (/home/iberiam/Desktop/Grib/emoncms_1/emoncms_examples-master/feed01/includes/db.php)
```

Corrected code:

```php
140: $username = mysql_real_escape_string($_POST['username']);
144: $result = db_query("SELECT id,password, salt FROM users WHERE username = '$username';
16: return $result = mysql_query($query);
    (/home/iberiam/Desktop/Grib/emoncms_1/emoncms_examples-master/feed01/includes/db.php)
```

SELECT id, password, salt FROM users
WHERE username = "' OR 1=1
INTO OUTFILE '/var/www/html/vulnsite/login-info.html' -- '
emoncms: SQL Injection

File user.php

Vulnerable code:

140: $username = "" OR 1=1 INTO OUTFILE '/var/www/html/vulnsite/login-info.html' -- ".
144: $result = db_query("SELECT id,password, salt FROM users WHERE username = '$username'");
16: return $result = mysql_query($query);
   (/home/iberiam/Desktop/Grib/emoncms_1/emoncms_examples-master/feed01/includes/db.php)

Corrected code:

140: $username = mysql_real_escape_string($_POST['username']);
144: $result = db_query("SELECT id,password, salt FROM users WHERE username = '$username'");
16: return $result = mysql_query($query);
   (/home/iberiam/Desktop/Grib/emoncms_1/emoncms_examples-master/feed01/includes/db.php)

SELECT id, password, salt FROM users
WHERE username = "" OR 1=1
INTO OUTFILE '/var/www/html/vulnsite/login-info.html' -- "

SELECT id, password, salt FROM users

Vulnerabilities Detected

**emoncms**: SQL Injection

File *user.php*

Vulnerable code:

```php
140: $username = $_POST['username'];
144: $result = db_query("SELECT id,password, salt FROM users WHERE username = "'$.username'"考虑到);
16: return $result = mysql_query($query);

(/home/iberiam/Desktop/Grib/emoncms_1/emoncms_examples-master/feed01/includes/db.php)
```

Corrected code:

```php
140: $username = mysql_real_escape_string($_POST['username']);
144: $result = db_query("SELECT id,password, salt FROM users WHERE username = '$_POST['username']"考虑到); 
16: return $result = mysql_query($query);

(/home/iberiam/Desktop/Grib/emoncms_1/emoncms_examples-master/feed01/includes/db.php)
```

```sql
SELECT id, password, salt FROM users WHERE username = " OR 1=1
INTO OUTFILE '/var/www/html/vulnsite/login-info.html' --
```
Vulnerabilities Detected

emoncms: SQL Injection

File user.php

Vulnerable code:

140: $username = $_POST['username'];
144: $result = db_query("SELECT id,password, salt FROM users WHERE username = '$username'" );
16: return $result = mysql_query($query);
   (/home/iberiam/Desktop/Grib/emoncms_1/emoncms_examples-master/feed01/includes/db.php)

Corrected code:

140: $username = mysql_real_escape_string($_POST['username']);
144: $result = db_query("SELECT id,password, salt FROM users WHERE username = '$username'" );
16: return $result = mysql_query($query);
   (/home/iberiam/Desktop/Grib/emoncms_1/emoncms_examples-master/feed01/includes/db.php)

SELECT id, password, salt FROM users
WHERE username = '' OR 1=1
INTO OUTFILE '/var/www/html/vulnsite/login-info.html' --

violate user privacy

counter the benefits of metering
Stored XSS is a variant of reflected XSS, where:
1. attacker inserts the malicious script in the application’s database
2. script is sent to one or more users; user application accesses to the DB

```php
$db->query("INSERT INTO measure_sensors (sensor_id, sensor_title) VALUES ('$params[sensor_id]', '$params[sensor_name]')");
```
**measureit: stored XSS**

Stored XSS is a variant of reflected XSS, where:

1. attacker inserts the malicious script in the application’s database
2. script is sent to one or more users; user application accesses to the DB

```php
$db->query("INSERT INTO measure_sensors (sensor_id, sensor_title) VALUES ('$params[sensor_id]', '$params[sensor_title]')");
```

Sensor 1 `<script>alert('Sensor 1 – XSS');</script>`
measureit: stored XSS

Stored XSS is a variant of reflected XSS, where:
1. attacker inserts the malicious script in the application’s database
2. script is sent to one or more users; user application accesses to the DB

```php
$db->query("INSERT INTO measure_sensors (sensor_id, sensor_title) VALUES ('$params[sensor_id]', '$params[sensor_name]')");
```

VALUES (`1`, `Sensor 1 <script>alert('Sensor 1 – XSS');</script>`)
**measureit: stored XSS**

Stored XSS is a variant of reflected XSS, where:

1. attacker inserts the malicious script in the application’s database
2. script is sent to one or more users; user application accesses to the DB

```php
$db->query("INSERT INTO measure_sensors (sensor_id, sensor_title)
VALUES (".$params['sensor_id']." , ".$params['sensor_name'].'"));
```

![Sensor 1 XSS Alert](image)
measureit: stored XSS

Stored XSS is a variant of reflected XSS, where:
1. attacker inserts the malicious script in the application’s database
2. script is sent to one or more users; user application accesses to the DB

```php
$db->query("INSERT INTO measure_sensors (sensor_id, sensor_title)
VALUES (".$params[sensor_id]."), ",".$params[sensor_name]."))
```

Stored XSS violates user privacy and counter the benefits of metering. Attack the software user.
## 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>Vulnerable files</th>
<th>Vulnerab. found</th>
<th>Vulnerab. corrected</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>
<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>
<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>
<td>4</td>
</tr>
<tr>
<td>currentcost</td>
<td>3</td>
<td>270</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>dmoz2mysql</td>
<td>6</td>
<td>1,000</td>
<td>0</td>
<td>0</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>
<td>7</td>
</tr>
<tr>
<td>emoncms</td>
<td>76</td>
<td>6,876</td>
<td>6</td>
<td>6</td>
<td>15</td>
<td>12</td>
</tr>
<tr>
<td>Ghost</td>
<td>16</td>
<td>398</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>gilbitron-PIP</td>
<td>14</td>
<td>328</td>
<td>1</td>
<td>0</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>
<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>
<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>
<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>
<td>5</td>
</tr>
<tr>
<td>Mfm 0.13</td>
<td>7</td>
<td>5,859</td>
<td>6</td>
<td>1</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>Mutilildae 1.3</td>
<td>18</td>
<td>1,623</td>
<td>6</td>
<td>10</td>
<td>19</td>
<td>19</td>
</tr>
<tr>
<td>Mutilildae 2.3.5</td>
<td>578</td>
<td>102,567</td>
<td>63</td>
<td>7</td>
<td>10</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>
<td>0</td>
</tr>
<tr>
<td>OWASP Vicnum</td>
<td>22</td>
<td>814</td>
<td>2</td>
<td>7</td>
<td>18</td>
<td>14</td>
</tr>
<tr>
<td>paCRUD 0.7</td>
<td>100</td>
<td>11,079</td>
<td>11</td>
<td>0</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>
<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>
<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>
<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>
<td>5</td>
</tr>
<tr>
<td>PhpCrud</td>
<td>6</td>
<td>612</td>
<td>3</td>
<td>0</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>
<td>0</td>
</tr>
<tr>
<td>PHPFusion</td>
<td>633</td>
<td>27,000</td>
<td>40</td>
<td>0</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>
<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>
<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>
<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>
<td>0</td>
</tr>
<tr>
<td>PHPMyAdmin 2.6.3-phil</td>
<td>287</td>
<td>143,171</td>
<td>105</td>
<td>0</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>
<td>0</td>
</tr>
<tr>
<td>WebCalendar</td>
<td>122</td>
<td>30,173</td>
<td>12</td>
<td>0</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>
<td>14</td>
</tr>
<tr>
<td>ZiPEC 0.32</td>
<td>10</td>
<td>735</td>
<td>2</td>
<td>1</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>2854</strong></td>
<td><strong>470,496</strong></td>
<td><strong>473</strong></td>
<td><strong>107</strong></td>
<td><strong>294</strong></td>
<td><strong>266</strong></td>
</tr>
</tbody>
</table>
4. Conclusions
Conclusions

- Energy metering software (EMS) can be web application and can have input validation vulnerabilities.

- We present an approach and a tool called WAP to automatically identify and correct these vulnerabilities.

- WAP tool analyzed two open EMS, emoncms and measureit, and:
  - identified and corrected 17 vulnerabilities: 3 SQLI and 14 XSS.
  - identified other 3 XSS vulnerabilities in emoncms that may be false positives since we did not manage to attack them.

- WAP identified around 300 vulnerabilities in around 35 applications
Thank you!

Software
Build it secure
Make it safe