Static Detection of Vulnerabilities in Modern PHP Applications

Johannes Dahse

HackPra ’14, November 26th, Ruhr-University Bochum, Germany
1.1 Background

- @FluxReiners / websec.wordpress.com
- IT-Security Student, Ruhr-University Bochum (2006 - 2012)
- Co-founder CTF team *FluxFingers* (2007)
- Co-founder *HackerPraktikum*, *BadBank* Developer (2009)
- Penetration Tester / Code Auditer
- *RIPS 0.5* – Static Code Analyzer (2009 - 2011)
- *RIPS 1.0* – New rewritten prototype (2012 – today, 24/7)
- PhD Student at Chair for Systems Security, RUB (2013 - today)
1.2 Why PHP?

(23% of all websites)

(wordPress, Joomla!)

(3% of all websites)

http://w3techs.com

All CVE entries

PHP related: 29%

Other: 71%

http://www.coelho.net/php_cve.html

<table>
<thead>
<tr>
<th>Server-side language</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>PHP</td>
<td>81.8%</td>
</tr>
<tr>
<td>ASP.NET</td>
<td>17.9%</td>
</tr>
<tr>
<td>Java</td>
<td>2.7%</td>
</tr>
<tr>
<td>ColdFusion</td>
<td>0.8%</td>
</tr>
<tr>
<td>Perl</td>
<td>0.6%</td>
</tr>
<tr>
<td>Ruby</td>
<td>0.5%</td>
</tr>
<tr>
<td>Python</td>
<td>0.2%</td>
</tr>
<tr>
<td>JavaScript</td>
<td>0.1%</td>
</tr>
</tbody>
</table>

CTF services analyzed by FluxFingers

Python: 22%
PHP: 19%
C: 14%
Java: 9%
Binary: 9%
Perl: 8%
Ruby: 5%

http://www.coelho.net/php_cve.html

1. Introduction
2. Static Code Analysis
3. Modern Vulnerabilities
4. Open Challenges
1.3 Motivation

- SQL Injection in BadBank

1. $id = \$_GET['id'];
2. $sql = "SELECT id, titel, hinweis
   FROM hinweise WHERE id = ".$id;
3. $result = mysql_query($sql);

- Cross-Site Scripting in BadBank

1. $order = \$_GET['order'];
2. $html = " (nach " . $order . ")\n"
3. echo $html;
1.4 Taint-style Vulnerabilities

user input
$_GET
$_POST
$_COOKIE
$_REQUEST
$_FILES
$_SERVER
...

sensitive sink
print()
mysql_query()
include()
eval()
system()
...

vulnerability type
Cross-Site Scripting
SQL Injection
File Inclusion
Code Execution
Command Execution
...

Note: Logical Flaws do not follow such a general concept and are harder to detect
Static Detection of Vulnerabilities in Modern PHP Applications

1. Introduction
2. Static Code Analysis
3. Modern Vulnerabilities
4. Open Challenges

RIPS 0.5
BadBank Demo Scan
Static Detection of Vulnerabilities in Modern PHP Applications

1. Introduction
2. Static Code Analysis
3. Modern Vulnerabilities
4. Open Challenges

it doesn’t work, but it’s fast

theprofoundprogrammer.com
1.4 Taint-style Vulnerabilities

- User input: 
  - $_GET
  - $_POST
  - $_COOKIE
  - $_REQUEST
  - $_FILES
  - $_SERVER
  - ...

- Sensitive sink: 
  - print()
  - mysql_query()
  - include()
  - eval()
  - system()
  - ...

- Vulnerability type: 
  - Cross-Site Scripting
  - SQL Injection
  - File Inclusion
  - Code Execution
  - Command Execution
  - ...

Note: Logical Flaws do not follow such a general concept and are harder to detect.
1.4 Taint-style Vulnerabilities

user input

$_GET
$_POST
$_COOKIE
$_REQUEST
$_FILES
$_SERVER
...

+ sanitation

htmlentities()
addslashes()
basename()
(int)
escapeshellarg()...

+ sensitive sink

print()
mysql_query()
include()
eval()
system()...

= vulnerability

XSS
SQL Injection
File Inclusion
Code Exec
Cmd Exec
...

Refined concept of previous prototype and current tools on the market
1.4 Taint-style Vulnerabilities

- User input: $_GET, $_POST, $_COOKIE, $_REQUEST, $_FILES, $_SERVER...
- Sanitization: htmlentities(), addslashes(), basename(), (int), escapeshellarg(), ...
- Markup: HTML, SQL, File Path, PHP, OS Command, ...
- Sensitive sink: print(), mysql_query(), include(), eval(), system(), ...
- Vulnerability: XSS, SQL Injection, File Inclusion, Code Exec, Cmd Exec, ...

Refined concept of new prototype
1.5 Lessons Learned

1. It's easy to build a static code analysis tool that detects **simple vulnerabilities**
1.5 Lessons Learned

1. It is easy to build a static code analysis tool that detects **simple vulnerabilities**.

2. It is **challenging** to build a static code analysis tool that detects **sophisticated vulnerabilities**.
1.5 Lessons Learned

1. Its easy to built a static code analysis tool that detects **simple vulnerabilities**

2. Its *challenging* to built a static code analysis tool that detects **sophisticated vulnerabilities**

3. Its *hard* to built a static code analysis tool that detects **sophisticated vulnerabilities in large applications** with acceptable **performance** and a **low false positive rate**
RIPS 1.0
BadBank Demo Scan
Static Detection of Vulnerabilities in Modern PHP Applications

1. Introduction
2. Static Code Analysis
3. Modern Vulnerabilities
4. Open Challenges
2. Static VS Dynamic Code Analysis

<table>
<thead>
<tr>
<th></th>
<th>Static analyze code without execution</th>
<th>Dynamic analyze code while execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Coverage</td>
<td>full</td>
<td>Single execution path</td>
</tr>
<tr>
<td>Data Coverage</td>
<td>Compile-time data</td>
<td>Runtime data (valid for environment)</td>
</tr>
<tr>
<td>Decidability</td>
<td>Halting Problem</td>
<td>Real data</td>
</tr>
</tbody>
</table>
## 2. Static Code Analysis

<table>
<thead>
<tr>
<th></th>
<th>Static analyze code without execution</th>
<th>Dynamic analyze code while execution</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Code Coverage</strong></td>
<td>full</td>
<td>Single execution path</td>
</tr>
<tr>
<td><strong>Data Coverage</strong></td>
<td>Compile-time data</td>
<td>Runtime data (valid for environment)</td>
</tr>
<tr>
<td><strong>Decidability</strong></td>
<td>Halting Problem</td>
<td>Real data</td>
</tr>
</tbody>
</table>

### Code Coverage
- **Static**: full
- **Dynamic**: Single execution path

### Data Coverage
- **Static**: Compile-time data
- **Dynamic**: Runtime data (valid for environment)

### Decidability
- **Static**: Halting Problem
- **Dynamic**: Real data
2.1 Overview

- Load all PHP files
- Tokenize PHP code and build an Abstract Syntax Tree (AST)
- Split AST into Basic Blocks
- Connect Basic Blocks to a Control Flow Graph (CFG)
- Analyze data flow through CFG
- RIPS uses *block* and *function summaries*
2.2 Abstract Syntax Tree

- Tokenize Code
- Parse tokens according to PHP syntax
- Structure tokens into a tree representation
- AST allows to parse semantics without dealing with punctuation or delimiters

```php
$cookie = $_COOKIE['text'];
```
2.2 Abstract Syntax Tree

- Tokenize Code
- Parse tokens according to PHP syntax
- Structure tokens into a tree representation
- AST allows to parse semantics without dealing with punctuation or delimiters

```php
$cookie = $_COOKIE['text'];
```

<table>
<thead>
<tr>
<th>Variable</th>
<th>Variable</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>$cookie</td>
<td>$COOKIE</td>
<td>'text'</td>
</tr>
</tbody>
</table>
2.2 Abstract Syntax Tree

- Tokenize Code
- Parse tokens according to PHP syntax
- Structure tokens into a tree representation
- AST allows to parse semantics without dealing with punctuation or delimiters
2.2 Abstract Syntax Tree

- Tokenize Code
- Parse tokens according to PHP syntax
- Structure tokens into a tree representation
- AST allows to parse semantics without dealing with punctuation or delimiters

```
$cookie = $_COOKIE['text'];
```

```
$cookie
Assign
var
expr

$_COOKIE['text']
```
2.2 Abstract Syntax Tree

- Tokenize Code
- Parse tokens according to PHP syntax
- Structure tokens into a tree representation
- AST allows to parse semantics without dealing with punctuation or delimiters

```php
$cookie = $_COOKIE['text'];
```

```
Assign
  var
    $cookie variable
  expr
    $_COOKIE array
  dim
    'text' string
```
2.3 Basic Block

- Split AST at *jump nodes*
- *Single input, single output* code block
- Simulate data flow in basic block
- Precisely model PHP built-in features
- Summarize data flow

```php
if(isset($_COOKIE['text'])) {
    $cookie = $_COOKIE['text'];
    $s = $cookie;
} else {
    $cookie = trim($default);
    $s = $cookie;
}
```
2.3 Basic Block

- Split AST at *jump nodes*
- *Single input, single output* code block
- Simulate data flow in basic block
- Precisely model PHP built-in features
- Summarize data flow
2.3 Basic Block

- Split AST at *jump nodes*
- *Single input, single output* code block
- Simulate data flow in basic block
- Precisely model PHP built-in features
- Summarize data flow

```php
if(isset($COOKIE['text'])) {
    $cookie: $COOKIE['text'];
    $s: $COOKIE['text'];
}
else {
    $cookie: $default;
    $s: $default;
}
```
2.4 Control Flow Graph

- Connect basic blocks to CFG
- Edges are jump conditions
- Represents all code paths
- Efficient data flow analysis on block summaries

```php
if
    $cookie: $_COOKIE['text'];
    $s:     $_COOKIE['text'];
else
    $cookie: $default;
    $s:     $default;
```

...
2.4 Control Flow Graph

- Connect basic blocks to CFG
- Edges are jump conditions
- Represents all code paths
- Efficient data flow analysis on block summaries

```php
$cookie: $_COOKIE['text'];
$s:   $_COOKIE['text'];
```

```php
$cookie: $default;
$s:   $default;
```
2.5 Inter-procedural Analysis

- Call to user-defined function invokes intra-procedural analysis of this function.
- Effects of function are stored in function summary.

```php
function getText($default) {
    $cookie: $_COOKIE['text'];
    $s:     $_COOKIE['text'];
    $cookie: $default;
    $s:     $default;
    return $s;
}
```
2.5 Inter-procedural Analysis

- Call to user-defined function invokes intra-procedural analysis of this function.
- Effects of function are stored in function summary.
2.5 Inter-procedural Analysis

• Call to user-defined function invokes intra-procedural analysis of this function

• Effects of function are stored in function summary

```php
function getText($arg1)
return: $_COOKIE['text'] | $arg1
```

```php
1 ...  
2 $text = getText('foo');  
3 ... 
```
### 2.5 Inter-procedural Analysis

- Call to user-defined function invokes intra-procedural analysis of this function.
- Effects of function are stored in *function summary*.

```php
function getText($arg1)
return: $_COOKIE['text'] | $arg1
```

```php
$text: $_COOKIE['text']|'foo'
```
2.6 Taint Analysis

- Identify a configured set of sensitive sinks
- Resolve sensitive arguments from previous block summaries
- Issue vulnerability report if argument is resolved to a source
2.7 Context-sensitive String Analysis

- Resolve all strings (markup) of a sensitive argument
- Replace sources with a placeholder
- Invoke markup parser
- Evaluate sanitization of each source regarding its markup context

```php
$s = addslashes($_GET['s']);
if($mode == 1) {
    $where = "name = '$s'";
} else {
    $where = "id = $s";
}
mysql_query('SELECT * FROM users ' . $where);
```
2.7 Context-sensitive String Analysis

- Resolve all strings (markup) of a sensitive argument
- Replace sources with a placeholder
- Invoke markup parser
- Evaluate sanitization of each source regarding its markup context
2.7 Context-sensitive String Analysis

- Resolve all strings (markup) of a sensitive argument
- Replace sources with a placeholder
- Invoke markup parser
- Evaluate sanitization of each source regarding its markup context
2.7 Context-sensitive String Analysis

- Resolve all strings (markup) of a sensitive argument
- Replace sources with a placeholder
- Invoke markup parser
- Evaluate sanitization of each source regarding its markup context

```php
$s : $_GET['s']

$where: "name = '$s'"
$where: "id = "$s"

mysql_query('SELECT * FROM users WHERE ' . $where);
```

SELECT * FROM users WHERE name = 'S'
SELECT * FROM users WHERE id = 'S'

SQL SQ, SQL, DQ

 resolving all strings (markup) of a sensitive argument

 resolving sources with a placeholder

 invoking markup parser

 evaluating sanitization of each source regarding its markup context
3. Detecting *Modern* Vulnerabilities

- Taint-style concept remains: source → sink
- Data flow is more complex
  - Dynamic language features
  - Array Handling
  - PHP built-in features
- Input sanitization + validation
- Object-Oriented Programming
- More LOC (100k-200k)
3.1 Analyze Exceptional Sources

- Often sanitized: \$_GET, \$_POST, and \$_REQUEST
- Often overlooked: \$_SERVER, \$_FILES, and \$_COOKIE
- Examples:

<table>
<thead>
<tr>
<th>PHP Source</th>
<th>Developer Assumption</th>
<th>Exploit Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>$_SERVER['PHP_SELF']</td>
<td>/index.php</td>
<td>/index.php/&quot;&lt;svg+onload=alert(1)&gt;</td>
</tr>
<tr>
<td>$_SERVER['REQUEST_URI']</td>
<td>/index.php?payload=%22%27</td>
<td>GET /?a=’or(1)=1-- HTTP/1.0 or IE</td>
</tr>
<tr>
<td>$_SERVER['HTTP_HOST']</td>
<td>localhost</td>
<td>Host: ' or (1)=1-- -</td>
</tr>
<tr>
<td>$_FILES['picture']['name']</td>
<td>alphanumeric.jpg</td>
<td>'or (1)=1-- -.jpg</td>
</tr>
</tbody>
</table>

- RIPS analyzes sources context-sensitively
3.2 Precise Array Handling

- PHP superglobals ($_GET, $_POST, etc.) are arrays
- RIPS analyzes built-ins precisely
- Array keys are tainted too (Drupal)

```
function export($type, $records) {
    $records = explode('',' ', $records);
    $records = "'" . implode('','" ', $records) . "'";
    $where = "{$focus->table_name}.id in ($records)"

    export(clean($_REQUEST['module']), $_REQUEST['uid']);
}
```
3.2 Precise Array Handling

- PHP superglobals ($_GET, $_POST, etc.) are arrays
- RIPS analyzes built-ins precisely
- Array keys are tainted too (Drupal)

```
function confirm_delete_users($users) {
    foreach($_POST['allusers'] as $key => $val) {
        echo "<input name='user[]' value='{$val}'/>
";
    }
}

allusers[0]='<script>alert('found by RIPS');</script>
```
3.3 Object-Oriented Programming

```php
class Upload {
    function parse($name) {
        $this->set_filename($_FILES[$name]['name']);
    }

    function set_filename($filename) {
        $this->filename = $filename;
    }
}
```

```php
$sql_file = new Upload();
if ($sql_file->parse('sql_file') == true) {
    $read_from = $sql_file->filename;
}
tep_db_query("insert into " . TABLE_CONFIG . " values ('DB_RESTORE', '', $read_from . '', '6', '');
```
3.3 Object-Oriented Programming

Assist backwards-directed data flow analysis with forwards-directed data propagation.
3.3 Object-Oriented Programming

Assist backwards-directed data flow analysis with forwards-directed data propagation

```php
<?php

class Upload {
    function parse($name) {
        $this->set_filename($_FILES[$name]['name']);
    }
    function set_filename($filename) {
        $this->filename = $filename;
    }
}

$sql_file = new Upload();
if ($sql_file->parse('sql_file') == true) {
    $read_from = $sql_file->filename;
}
	ep_db_query("insert into " . TABLE_CONFIG . " values ('DB_RESTORE', '' . $read_from . '' , '6', '')");

?>
```
3.3 Object-Oriented Programming

### osCommerce 2.3.4

```php
class Upload {
    function parse($name) {
        $this->set_filename($_FILES[$name]["name"]);
    }
    function set_filename($filename) {
        $this->filename = $filename;
    }
}

$sql_file = new Upload();
if ($sql_file->parse('sql_file') == true) {
    $read_from = $sql_file->filename;
}
tep_db_query("insert into " . TABLE_CONFIG . " values ('DB_RESTORE', '', $read_from, '6', '')");
```
3.3 Object-Oriented Programming

Assist \textit{backwards-directed} data flow analysis with \textit{forwards-directed} data propagation.

```php
class Upload {
    function parse($name) {
        $this->set_filename($_FILES[$name]['name']);
    }
    function set_filename($filename) {
        $this->filename = $filename;
    }
}

$sql_file = new Upload();
if ($sql_file->parse('sql_file') == true) {
    $read_from = $sql_file->filename;
}
tep_db_query("insert into " . TABLE_CONFIG . " values ('DB_RESTORE', '' . $read_from . '' , '6', '')");
```
3.3 Object-Oriented Programming

osCommerce 2.3.4

```
class Upload {
    function parse($name) {
        $this->set_filename($_FILES[$name]['name']);
    }
    function set_filename($filename) {
        $this->filename = $filename;
    }
}

$sql_file = new Upload();
if ($sql_file->parse('sql_file') == true) {
    $read_from = $sql_file->filename;
}

tep_db_query("insert into " . TABLE_CONFIG . " values ('DB_RESTORE', '' . $read_from . '', '6', '')");
```
3.4 Insufficient Sanitization

- Sanitization tags
- Encoding stack
- Decoding stack
- Escaping level

**Pligg CMS 2.0.2**

```
233 $parent = substr(addslashes($_REQUEST['parent']),9,100);
245 $sql = "update .table_categories." set category_parent = " . $parent . " where category_id=" . $id . ";";
246 $db->query($sql);
```
3.4 Insufficient Sanitization

- Sanitization tags
- Encoding stack
- Decoding stack
- Escaping level

```
Pligg CMS 2.0.2 /admin/admin_categories.php
233 $parent = substr(addslashes($_REQUEST['parent']), 9, 100);
245 $sql = "update .table_categories." set category_parent = " . $parent . " where category_id=" . $id . ";";
246 $db->query($sql);
```

12345678901234567890'
3.4 Insufficient Sanitization

- Sanitization tags
- Encoding stack
- Decoding stack
- Escaping level

```
Pligg CMS 2.0.2 /admin/admin_categories.php

233 $parent = substr(addslashes($_REQUEST['parent']),9,100);
245 $sql = "update " . $table_categories . " set category_parent = " . $parent . " where category_id=" . $id . ";";
246 $db->query($sql);
```

12345678901234567890"
3.4 Insufficient Sanitization

- Sanitization tags
- Encoding stack
- Decoding stack
- Escaping level
3.4 Insufficient Sanitization

- Sanitization tags
- Encoding stack
- Decoding stack
- Escaping level

```php
Pligg CMS 2.0.2
/admin/admin_categories.php
233 $parent = substr(addslashes($_REQUEST['parent']), 9, 100);
245 $sql = "update .table_categories." set category_parent = " . $parent . " where category_id=" . $id . ";";
246 $db->query($sql);
```

1, category_title=(SELECT password ...
3.4 Insufficient Sanitization

- Sanitization tags
- Encoding stack
- Decoding stack
- Escaping level

**Pligg CMS 2.0.2**

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>233</td>
<td>$parent = substr(addslashes($_REQUEST['parent']),9,100);</td>
</tr>
<tr>
<td>245</td>
<td>$sql = &quot;update \table_categories.\ set category_parent = &quot; . $parent . &quot; where category_id=&quot; . $id ;&quot;;</td>
</tr>
<tr>
<td>246</td>
<td>$db-&gt;query($sql);</td>
</tr>
</tbody>
</table>

**Mambo CMS 4.6.2**

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>88</td>
<td>$mosmsg = mosGetParam($_REQUEST, 'mosmsg', '');</td>
</tr>
<tr>
<td>90</td>
<td>$mosmsg = addslashes($mosmsg);</td>
</tr>
<tr>
<td>91</td>
<td>echo &quot;\n&lt;div class=&quot;message&quot;&gt;&quot;.$mosmsg&lt;/div&quot;;</td>
</tr>
</tbody>
</table>
3.4 Insufficient Sanitization

- Sanitization tags
- Encoding stack
- Decoding stack
- Escaping level

**Pligg CMS 2.0.2**  
/admin/admin_categories.php

```php
233 $parent = substr(addslashes($_REQUEST['parent']),9,100);
245 $sql = "update .table_categories." set category_parent = " . $parent . " where category_id=" . $id . ";
246 $db->query($sql);
```

**Mambo CMS 4.6.2**  
/includes/cmtclasses.php

```php
88 $mosmsg = mosGetParam($_REQUEST, 'mosmsg', '');
90 $mosmsg = addslashes($mosmsg);
91 echo "\n<div class="message">$mosmsg</div>";
```

**Couch CMS 1.4**  
/couch/includes/fileuploader/io.php

```php
603 echo '<script type="text/javascript">';
621 $rpl = array( '' => '\\\', '"' => '\\"' );
623 echo 'OnUploadCompleted("'. strtr($msg, $rpl) . '");';
623 echo '</script>';
```
3.4 Insufficient Sanitization

- Sanitization tags
- Encoding stack
- Decoding stack
- Escaping level

---

**Pligg CMS 2.0.2**
/admin/admin_categories.php

```
233 $parent = substr(addslashes($_REQUEST['parent'])),9,100);
245 $sql = "update " .table_categories." set category_parent = " . $parent . " where category_id=" . $id . ";";
246 $db->query($sql);
```

**Mambo CMS 4.6.2**
/includes/cmtclasses.php

```
88 $mosmsg = mosGetParam($_REQUEST, 'mosmsg', '');
90 $mosmsg = addslashes($mosmsg);
91 echo "\n\n\n$mosmsg</div>";
```

**Couch CMS 1.4**
/couch/includes/fileuploader/io.php

```
603 echo '<script type="text/javascript">'</;
621 $rpl = array( '\\' => '\\\\', '"' => '\\"' );
623 echo 'OnUploadCompleted("" . strtr($msg, $rpl) . '"');';
623 echo '</script>';<script>alert(1)</script>";
```

---
3.5 Second-Order Vulnerabilities

„First-Order“ SQL Injection:

```
1 $name = $_POST['name'];
2 $pwd = md5($_POST['pwd']);
3 $query = "INSERT INTO users VALUES('$name','$pwd');
4 $result = mysql_query($query);```

```
user input
send
application
```
3.5 Second-Order Vulnerabilities

„First-Order“ SQL Injection (sanitized):

```php
1  $name = mysql_real_escape_string($_POST['name']);
2  $pwd = md5($_POST['pwd']);
3  $query = "INSERT INTO users VALUES('$name','$pwd')";
4  $result = mysql_query($query);
```
3.5 Second-Order Vulnerabilities

Database Write:

1. $name = mysql_real_escape_string($_POST['name']);
2. $pwd = md5($_POST['pwd']);
3. $query = "INSERT INTO users VALUES('$name','$pwd')";
4. $result = mysql_query($query);

Diagram:
- Input: User input
- Flow: Send
  - Application: Application
    - Operation: Write
  - Database: Database
3.5 Second-Order Vulnerabilities

Database Read:

1. $query = "SELECT * FROM users WHERE id = 1";
2. $result = mysql_query($query);
3. $user = mysql_fetch_assoc($result);
4. echo $user['name'];
3.5 Second-Order Vulnerabilities
3.5 Second-Order Vulnerabilities

PDS

δ*

Temporary Vulnerability Report
users[name] XSS

Reads

tainted?

Writes

source

δ

PDS'

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>pass</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>δ</td>
<td></td>
</tr>
</tbody>
</table>
3.5 Second-Order Vulnerabilities

PDS

δ*
sink

Reads

Temporary Vulnerability Report
users[name]
XSS

δ* sanitized?

Writes

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>pass</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>δ</td>
<td></td>
</tr>
</tbody>
</table>

source

δ

PDS'}
3.5 Second-Order Vulnerabilities

- Reads
  - Temporary Vulnerability Report
  - users[name] XSS

- Writes
  - Users
    | id | name | pass |
    |----|------|------|
    |    | δ    |      |

- Sink
- Source

- PDS
- PDS'
3.6 Multi-Step Exploits

„First-Order“ SQL Injection:

```
1  $name = $_POST['name'];
2  $pwd = md5($_POST['pwd']);
3  $query = "INSERT INTO users VALUES($name,'$pwd');"
4  $result = mysql_query($query);
```
3.6 Multi-Step Exploits

Exploit „First-Order“ SQL Injection to taint database:

```
1. $name = $_POST['name'];       // ', 'payload')-- -
2. $pwd = md5($_POST['pwd']);
3. $query = "INSERT INTO users VALUES('$name','$pwd');
4. $result = mysql_query($query);
```
3.6 Multi-Step Exploits

Data from tainted database used in sensitive sink:

1. $query = "SELECT * FROM users WHERE id = 1";
2. $result = mysql_query($query);
3. $user = mysql_fetch_assoc($result);
4. file_put_contents($user['pwd'] . '.txt', $data);
3.6.1 Multi-Step Exploit – osCommerce 2.3.4

- **Inject JavaScript (pXSS)**
- **Admin triggers payload**
- **Remote Command Execution**

**SQL Injection (INSERT)**

<table>
<thead>
<tr>
<th>id</th>
<th>key</th>
<th>value</th>
<th>gid</th>
<th>function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>COUNTRY</td>
<td>223</td>
<td>6</td>
<td>get_country</td>
</tr>
<tr>
<td>2</td>
<td>ZONE</td>
<td>18</td>
<td>6</td>
<td>get_zone</td>
</tr>
<tr>
<td>3</td>
<td>PAYMENT</td>
<td>Paypal</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>DB_RESTORE</td>
<td>id</td>
<td>6</td>
<td>system</td>
</tr>
</tbody>
</table>

*Example error message:* 1064 - You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'sql', 'Last database restore file', '6', '0', null, now(), '', '' at line 1

*Example configuration values:*
- id: 33
- gid: 33
- groups: 33

*Module Options:*
- Configuration
- System
- Administers
- Cache
- Customer Details
- Download
3.6.2 Multi-Step Exploit – OpenConf 5.30

1. upload
2. escalate
3. reconfigure OC_headerFile
4. include

PDF File Upload

Pre-auth SQLi

Second-Order LFI

Remote Code Execution
# 3.7 Exceptional Vulnerability Types

| ✗ Authorization Bypass | ✔ File Inclusion | ✔ PHP Object Injection |
| ✗ Cross-Site Request Forgery | ✔ File Overwrite | ✔ Reflection/Autoload Injection |
| ✔ Cross-Site Scripting | ✔ File System Manipulation | ✔ Resource Contention |
| ✔ Code Execution | ✔ File Upload | ✔ Server-Side JavaScript Injection |
| ✔ Command Execution | ✔ HTTP Response Splitting | ✔ Server-Side Request Forgery |
| ✔ Connection String Injection | ✔ Information Leakage | ✔ Session Fixation |
| ✔ Denial of Service | ✔ LDAP Injection | ✔ SQL Injection |
| ✔ Directory Listing | ✔ Log Forgery | ✔ Variable Manipulation |
| ✔ Execution After Redirect | ✔ Mass Assignment | ✔ Weak Cryptography |
| ✔ File Delete | ✔ Memcached Injection | ✔ XML/XXE Injection |
| ✔ File Disclosure | ✔ Open Redirect | ✔ XPath Injection |
4. Open Challenges

✗ Authorization Bypass
✗ Cross-Site Request Forgery
✓ Cross-Site Scripting
✓ Code Execution
✓ Command Execution
✓ Connection String Injection
✓ Denial of Service
✓ Directory Listing
✓ Execution After Redirect
✓ File Delete
✓ File Disclosure
✓ File Inclusion
✓ File Overwrite
✓ File System Manipulation
✓ File Upload
✓ HTTP Response Splitting
✓ Information Leakage
✓ LDAP Injection
✓ Log Forgery
✓ Mass Assignment
✓ Memcached Injection
✓ Open Redirect
✓ PHP Object Injection
✓ Reflection/Autoload Injection
✓ Resource Contention
✓ Server-Side JavaScript Injection
✓ Server-Side Request Forgery
✓ Session Fixation
✓ SQL Injection
✓ Variable Manipulation
✗ Weak Cryptography
✓ XML/XXE Injection
✓ XPath Injection
4.1 Loops

```php
function insert_query($table, $array) {
    foreach($array as $key => $value) {
        $fields .= $key . ",";
        $values .= "'" . $value . "',";
    }
    ...
    $this->write_query("INSERT INTO {$table} (" . $fields . ")
                          VALUES (" . $values . ")");
    return $this->insert_id();
}

$new_profile_field = array(
    "name" => $db->escape_string($mybb->input['name'])),
    "description" => $db->escape_string($mybb->input['description'])
);

$fid = $db->insert_query("profile", $new_profile_field);
```
4.1 Loops

```
function insert_query($table, $array) {
    foreach($array as $key => $value) {
        $fields .= $key . ",";
        $values .= "," . $value . ",";
    }
    ...
    $this->write_query("INSERT INTO {$table} (" . $fields . ")
    VALUES (" . $values . ")");
    return $this->insert_id();
}

$new_profile_field = array(
    "name" => $db->escape_string($mybb->input['name']),
    "description" => $db->escape_string($mybb->input['description'])
);
(fid = $db->insert_query("profile", $new_profile_field);
```
4.1 Loops

```php
function insert_query($table, $array) {
    foreach ($array as $key => $value) {
        $fields .= $key . "",";
        $values .= "'" . $value . "',";
    }
    ...
    $this->write_query("INSERT INTO {$table} (" . $fields . ")
VALUES (" . $values . ")");
    return $this->insert_id();
}

$new_profile_field = array(
    "name" => $db->escape_string($mybb->input['name']),
    "description" => $db->escape_string($mybb->input['description'])
);

$fid = $db->insert_query("profile", $new_profile_field);
```
4.1 Loops

```php
function insert_query($table, $array) {
    foreach($array as $key => $value) {
        $fields .= $key . ",";
        $values .= "'" . $value . "',";
    }
    ...
    $this->write_query("INSERT INTO \{$table\} (" . $fields . ")
    VALUES ('" . $values . ")");
    return $this->insert_id();
}

$new_profile_field = array(
    "name" => $db->escape_string($mybb->input['name'])),
    "description" => $db->escape_string($mybb->input['description'])
);

$fid = $db->insert_query("profile", $new_profile_field);

INSERT INTO profile (name, description) VALUES ('$1', '$2')
```
4.2 Frameworks

- Hard to analyze components lead to false negatives
- Query builders
  ```php
  $db->select('users')->where(array('id', $var));
  ```
- Template engines
  ```php
  $template->assign('B', $var)
  ```
- Reflection logic in configuration file
  ```php
  $controller();
  ```
- Partial solution: Framework-specific configuration
- Problem: high maintainance overhead
- Drupageddon
4.3 Path-sensitivity

- Infeasable paths lead to false positives
- Partial solution: *satisfiability solvers* (*Z3-str, S3*)
- Problem: Performance

```php
... 
2    if(!is_numeric($_GET['id'])) {
3        $error = true;
4    } 
5 
6    if(!$error) {
7        echo $_GET['id'];
8    }
```
4.3 Path-sensitivity

- Infeasable paths lead to false positives
- Partial solution: satisfiability solvers (Z3-str, S3)
- Problem: Performance

```php
$numeric = is_numeric($_GET['id']);
...
if(!$numeric) {
    $error = true;
}

if(!$error) {
    echo $_GET['id'];
}
```
If you think you have a stupid question, just remember NASA engineers once asked Sally Ride if 100 tampons were enough for a 7 day mission.
References

- **Simulation of Built-in PHP features for Precise Static Code Analysis**
  Johannes Dahse, Thorsten Holz
  Annual Network & Distributed System Security Symposium (NDSS), San Diego, CA, USA, February 2014

- **Static Detection of Second-Order Vulnerabilities in Web Applications**
  Johannes Dahse, Thorsten Holz
  23rd USENIX Security Symposium, San Diego, CA, USA, August 2014

- **Code Reuse Attacks in PHP: Automated POP Chain Generation**
  Johannes Dahse, Nikolai Krein, Thorsten Holz
  21st ACM Conference on Computer and Communications Security (CCS), Scottsdale, AZ, USA, November 2014
Join the RIPS project!

(€ / CP)

johannes.dahse@rub.de