Static Detection of Second-Order Vulnerabilities in Web Applications

Johannes Dahse and Thorsten Holz
Ruhr-University Bochum

USENIX Security ’14, 20-22 August 2014, San Diego, CA, USA
Static Detection of Second-Order Vulnerabilities in Web Applications

„First-Order“ Vulnerabilities

• SQL injection

```php
<?php
    $name = $_POST['name']; // ', 1), (version(), 1)-- -
    $sql = "INSERT INTO users VALUES ('$name', '$pwd');
    mysql_query($sql);
?>
```

1. Introduction
2. Implementation
3. Evaluation
4. Conclusion

user input | send | application
Sanitization

• SQL injection (prevented)

```php
<?php
  $name = mysql_real_escape_string($_POST['name']);
  $sql = "INSERT INTO users VALUES ('$name', '$pwd')";
  mysql_query($sql);
?>
```
Second-Order Vulnerability (1)

- Database Write

```php
$name = mysql_real_escape_string($_POST['name']);
$sql = "INSERT INTO users VALUES ('$name', '$pwd');"
mysql_query($sql);
?>
```

Diagram:
- User input
- Application
- Database
- Send
- Write

1. Introduction
2. Implementation
3. Evaluation
4. Conclusion
Second-Order Vulnerability (2)

- **Database Read**

```php
<?php
    $result = mysql_query('SELECT * FROM users');
    $row = mysql_fetch_assoc($result);
    echo $row['name'];
?>
```

Diagram:
- User input
- Send
- Application
  - Write
  - Read
- Database
Multi-Step Exploit (1)

- First-Order SQL injection

```php
<?php
    $name = $_POST['name']; // 'payload'--
    $sql = "INSERT INTO users VALUES ('$name', '$pwd');
    mysql_query($sql);
?>
```
Multi-Step Exploit (1)

- **Exploit First-Order SQL injection**

```php
<?php
    $name = $_POST['name']; // 'payload')-- -
    $sql = "INSERT INTO users VALUES ('$name', '$pwd')";
    mysql_query($sql);
?>
```
Multi-Step Exploit (2)

- Second-Order Command Execution

```php
<?php
    $result = mysql_query('SELECT * FROM users');
    $row = mysql_fetch_assoc($result);
    system('htpasswd -b .htpasswd Admin '.$row['pwd']);
?>
```
Second-Order Vulnerabilities

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

1. Persistent Data Store (PDS)
- Databases
- File Names
- $_SESSION (File Content)
- ...

2. Sensitive Sink
- Cross-Site Scripting
- SQL Injection
- Code Execution
- File Inclusion
- File Disclosure
- ...

Static Detection of Second-Order Vulnerabilities in Web Applications
Second-Order Vulnerabilities

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

2. Persistent Data Store (PDS):
   - Databases
   - File Names
   - $_SESSION (File Content)
   ...

3. Sensitive Sink:
   - Cross-Site Scripting
   - SQL Injection
   - Code Execution
   - File Inclusion
   - File Disclosure
   ...

---

Static Detection of Second-Order Vulnerabilities in Web Applications

1. Introduction
2. Implementation
3. Evaluation
4. Conclusion

---

10
Our Approach

- Static Code Analysis (no access to environment)
- Analyze *writes* and *reads* to persistent data stores
- Connect input and output points at the end of the analysis to detect second-order and multi-step vulnerabilities
1. Introduction
2. Implementation
3. Evaluation
4. Conclusion

2. Implementation

(Overview)
First-Order Taint Analysis

$name = \$_POST[\'name\'];

mysql_query('insert into users values(null, \'$name\', \'$pwd\');
First-Order Taint Analysis

```php
$name = $_POST['name'];

mysql_query('insert into users values(null, '.$name.', '.$pwd.');
```
First-Order Taint Analysis

```php
$name = $_POST['name'];

mysql_query('insert into users
values(null, '$name', '$pwd');
```
First-Order Taint Analysis

```php
$name = $_POST['name'];

$db query = 'insert into users
values(null, $name', '$pwd');
```

Vulnerability Report
POST$\text{name}$ SQLi

Source

Sink
Second-Order Taint Analysis (write)

```php
$name = escape($_POST['name']);

mysql_query('insert into users values(null, "$name", "$pwd");
```

![Diagram showing taint analysis process]

**Users Table**

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>pass</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Multi-Step Taint Analysis (write)

```php
$name = $_POST['name'];

// Vulnerability Report
POST[name] SQLi

// Source

mysql_query('insert into users values(null, '$name', '$pwd');

// Sink

users
<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>pass</th>
</tr>
</thead>
<tbody>
<tr>
<td>δ</td>
<td>δ</td>
<td></td>
</tr>
</tbody>
</table>
```
Second-Order Taint Analysis (read)

```php
echo('Hi ' . $res['name'] . '!

$res = mysql_query('select name from users');

$row = mysql_fetch_assoc($res);

echo('Hi ' . $res['name'] . '!

PDS $\delta^*$ sink
```
Second-Order Taint Analysis (read)

```php
$res = mysql_query('select name from users');

$row = mysql_fetch_assoc($res);

echo('Hi ' . $res['name'] . '!
```

PDS

δ*

sink
Second-Order Taint Analysis (read)

```php
$res = mysql_query('select name from users');

$row = mysql_fetch_assoc($res);

echo('Hi ' . $res['name'] . ' !');
```

PDS

sink
Second-Order Taint Analysis (read)

```
$res = mysql_query('select name from users');
$row = mysql_fetch_assoc($res);
echo('Hi ' . $res['name'] . '氚');
```
Second-Order Taint Decision

PDS

\[ \delta^* \]

Temporary Vulnerability Report
users[name]
XSS

sink

\[ \delta^* \]

Reads

connect

Writes

source

\[ \delta \]

PDS'

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>pass</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( \delta )</td>
<td></td>
</tr>
</tbody>
</table>
Second-Order Taint Decision

- **PDS**
- **source**
- **sink**
- **users**

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

PDS

Reads

Temporary Vulnerability Report
users[name]
XSS

δ*

sink

Writes

users
id name pass

δ*
sanitized?

source

δ

PDS'

25
Second-Order Taint Decision

PDS

\(\delta^*\)

sink

Reads

Temporary Vulnerability Report
users[name] XSS

Second-Order Vulnerability Report
XSS

writes

source

\(\delta\)

users

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>pass</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(\delta)</td>
<td></td>
</tr>
</tbody>
</table>

PDS'
3. Evaluation
Selected Software

- osCommerce 2.3.3.4
- HotCRP 2.61
- OpenConf 5.30
- MyBloggie 2.1.4
- NewsPro 1.1.5
- Scarf 2007-02-27
PDS Usage and Coverage (first-order)

Manually counted PDS (841)
- Non-Taintable: 77%
- Taintable ""\<": 23%

Detected Taintable PDS
- False Positive: 6%
- True Positive: 71%
- False Negative: 29%
Second-Order Vulnerabilities

- 159 True Positives (79%)
  - 97% persistent XSS (database)
  - Missed by previous work

- 43 False Positives (21%)
  - Root cause: Path-sensitive sanitization
  - E.g., store only valid email
  - Failures in 1st step propagate to 2nd step
Multi-Step Exploits

- 14 True Positives (93%)
  - 2 based on file upload
  - 12 based on SQLi
  - Missed by previous work
- 1 False Positives (7%)
  - False positive SQLi

1. Introduction
2. Implementation
3. Evaluation
4. Conclusion
Second-Order LFI in OpenConf

```php
$r = mysql_query("select setting, value from " . OCC_TABLE_CONFIG);
while ($l = mysql_fetch_assoc($r)) {
    $config[$l['setting']] = $l['value'];
}

function printHeader($what, $function="0") {
    require $GLOBALS['pfx'] . $GLOBALS['config']['OC_headerFile'];
}
```
Second-Order LFI in OpenConf

```php
$r = mysql_query("select setting, value from \" . OCC_TABLE_CONFIG); while ($l = mysql_fetch_assoc($r)) {
    $config[$l["setting"]]) = $l["value"];}

function printHeader($what, $function="0") {
    require $GLOBALS["pfx"] . $GLOBALS["config"]['OC_headerFile'];
```
Second-Order LFI in OpenConf

$r = \text{mysql\_query(\"select setting, value from \" . OCC\_TABLE\_CONFIG\);}$
while ($l = \text{mysql\_fetch\_assoc($r)}) {
    $config[l['setting']] = l['value'];
}

\text{function printHeader($what, $function="0") {}
    require $GLOBALS['pfx'] . $GLOBALS['config']['OC\_headerFile'];
}

\text{function updateConfigSetting($setting, $value) {
    ocsql\_query("UPDATE \" . OCC\_TABLE\_CONFIG . ";
        SET `value`= " . safeSQLstr(trim($value)) . ";
        WHERE `setting`= " . safeSQLstr($setting) . ");
}

\text{foreach (array\_keys($\_POST) as $p) {
    if (preg\_match("/\^OC\_[\w-]+$/", $p)) {
        updateConfigSetting($p, $\_POST[$p]);
    }
}}
Multi-Step Exploitation in OpenConf

1. Introduction
2. Implementation
3. Evaluation
4. Conclusion

File Upload

SQLi or XSS

Remote Command Execution

All issues are fixed in version 5.31 and 6.01
4. Conclusion
Conclusion

- Static detection of second-order vulnerabilities is possible
  - Analyze and collect reads/writes to PDS (database, file names, session data)
  - Determine sensitive data flow at the end of analysis

- > 150 new vulnerabilities
  - Leading to RCE in NewsPro, Scarf, OpenConf, osCommerce
  - Overlooked problem in practice, missed in previous work

- Future work
  - Support prepared statements
  - Improve SQL parser
Thank you Facebook for the generous award
Questions

johannes.dahse@rub.de
Thank you!
Enjoy the conference.