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
„First-Order“ Vulnerabilities

- SQL injection

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

- SQL injection (prevented)

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

![Diagram showing user input → send → application]
Second-Order Vulnerability (1)

- **Database Write**

```php
<?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
Second-Order Vulnerability (2)

- Database Read

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

![Diagram showing the process of database access]

**Diagram Notes:**
- User input is sent to the application.
- The application reads data from the database.
- The database is written by the application.
- The database is read by the application.
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
- ...
Second-Order Vulnerabilities

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

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

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

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

- Static Code Analysis for PHP applications
- Analyze *reads* and *writes* to persistent data stores
- Connect input and output points at the end of the analysis to detect second-order and multi-step vulnerabilities
Welcome to the USENIX Security '14 submissions site. Sign in to submit or review papers. For general information about USENIX Security '14, see the conference site.

Email

Password

Sign me in
- I forgot my password
- I’m a new user and want to create an account using this email address

Sign in

Submissions: The deadline for registering new papers has passed.

Conference information
- Deadlines
- Program committee
- Conference site

67 papers were accepted out of 352 submitted.
Static Detection of Second-Order Vulnerabilities in Web Applications

2. Implementation

(Overview)
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'];

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.');
```

Vulnerability Report
POST[name] SQLi
Second-Order Taint Analysis (write)

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

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

![Diagram showing second-order taint analysis](image)
Multi-Step Taint Analysis (write)

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

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

<table>
<thead>
<tr>
<th>users</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
</tr>
<tr>
<td>0</td>
</tr>
</tbody>
</table>
Second-Order Taint Analysis (read)

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

$row = mysql_fetch_assoc($res);

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

sink

PDS

δ^*
```
Second-Order Taint Analysis (read)

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

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

Diagram:
- `PDS`
- `δ*` sink
- `$res = mysql_query('select name from users');`
- `$row = mysql_fetch_assoc($res);`
- `echo('Hi ' . $res['name'] . ' !');`
Second-Order Taint Analysis (read)

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

$row = mysql_fetch_assoc($res);

echo('Hi ' . $res['name'] . ' !');
```
Second-Order Taint Analysis (read)

```php
$res = mysql_query('select name from users);
$row = mysql_fetch_assoc($res);
echo('Hi ' . $res['name'] . ' !');
```

Temporary Vulnerability Report
users[name] XSS

PDS
Second-Order Taint Decision

PDS

\[ \delta^* \]

Temporary Vulnerability Report
users[name] XSS

sink

\[ \delta \]

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>

Reads

connect

Writes

source
Second-Order Taint Decision

PDS

δ*

sink

δ*

Temporary Vulnerability Report
users[name] XSS

δ*

Reads

Writes

tainted?

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>
</tbody>
</table>

PDS'

δ

source
Second-Order Taint Decision

Second-Order Taint Decision

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

- Writes
  - sanitized?
  - users
    - id
    - name
    - pass

- sink

- source

- PDS

- PDS'

- \( \delta^* \)

- \( \delta \)
Second-Order Taint Decision

PDS

\[ \delta^* \]

Temporary Vulnerability Report
users[name] XSS

sink

Reads

\[ \delta^* \]

Write

users

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>pass</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>[\delta]</td>
</tr>
</tbody>
</table>

Second-Order Vulnerability Report
XSS

source

\[ \delta \]

PDS'

sink
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)
- 43 False Positives (21%)
  - Root cause: Path-sensitive sanitization
  - E.g., store only valid email
Multi-Step Exploits

- 14 True Positives (93%)
  - 2 based on file upload
  - 12 based on SQLi

- 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

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

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

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

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

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

PDS

sink

source

PDS

PDS
Multi-Step Exploitation in OpenConf

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

File Upload

SQLi or XSS

Second-Order LFI

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
  ▪ Prepared Statements
  ▪ Improve SQL parser
Questions

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