Automated Detection of Complex Vulnerabilities with Static Code Analysis

Johannes Dahse, Dortmund, 10 Nov 2016
1. Introduction

2. Static Code Analysis

3. First-order Bug Detection

4. Second-order Bug Detection

5. Gadget Chain Detection
1.1 About

- Dr. Johannes Dahse
- CEO of RIPS Technologies
- Study/Ph.D. IT-Security, Ruhr-University Bochum
- Security Consultant
- CTF participant
- @FluxReiners, websec.wordpress.com
- Developer of RIPS
1.2 Research Timeline

- **2007 – 2009**: \textit{PHP Scanner} based on Regex used for CTF competitions
- **2009 – 2011**: \textit{RIPS 1\textsuperscript{st} Generation} based on Tokenizer open sourced during MOPS (2\textsuperscript{nd} place)
- **2012**: \textit{RIPS 2\textsuperscript{nd} Generation} based on AST and CFG subject of master thesis
- **2013 – 2015**: \textit{RIPS 3\textsuperscript{rd} Generation} subject of doctor thesis
- **2016**: \textit{RIPS (Standalone / Cloud)}
1.3 The Role of PHP in Security

- **82.2%** of the websites run PHP as server-side language
- Dynamic language, built-in features, oddities / pitfalls
- **25%** of all reported CVE vulnerabilities are related to PHP
- Sucuri Website Hacked Report: **97%** of hacked websites run PHP CMS
1.4 Security Vulnerability Demo
1.5 Goal

- Automated security analysis of PHP code
  - Analyze dynamic language
  - Support variety of language features
  - Detect common vulnerability types
  - Detect complex vulnerabilities
  - Scale to large applications
  - Non-annotation based
Automated Detection of Complex Vulnerabilities with Static Code Analysis

2. Static Code Analysis

- eval
- includes
- defaultParam
- superglobals

RIPS
- clone()
- RIPS_ArrayKey
- output_compress
- methodMerge
- RIPS_Concat
- Bounded Analysis
- second_order_grouping

Interprocedural Analysis
- interprocedural
- control_flow
- data_flow
- object_model

Taint Analysis
- taint_model
- source_code
- environment
- path_model

Sanitization Analysis
- type_sanitization
- process_sanitization
2.1 Overview

- Transform code into abstract syntax tree (AST)
- Split AST into basic blocks
- Analyze data flow within each basic block
- Summarize data flow in block and function summaries
- Connect basic blocks to a control flow graph (CFG)
- Perform backwards-directed taint analysis for each sensitive sink
Automated Detection of Complex Vulnerabilities with Static Code Analysis

3. First-order Bug Detection

- eval
- includes
- defaultParam
- superglobals

RIPS
- clone()
- RIPS_ArrayKey
- output compress
- methodMerge
- RIPS_Concat
- Bounded Analysis
- second-order grouping

Interprocedural Analysis
- tainted source
- tainted sink
- tainted path
Taint Analysis
- tainted-return
- tainted-argument
- tainted-param
Sanitization Analysis
3.1 Traditional Vulnerability Types

- Authorization Bypass
- Cross-Site Request Forgery
- Cross-Site Scripting
- Code Execution
- Command Execution
- Connection String Injection
- Denial of Service
- Directory Listing
- Environment Manipulation
- Execution After Redirect
- File Create
- File Delete
- File Disclosure
- File Inclusion
- File Write
- File System Manipulation
- File Upload
- HTTP Response Splitting
- Information Leakage
- LDAP Injection
- Library Injection
- Log Forge
- Mass Assignment
- Memcached Injection
- MongoDB Injection
- NoSQL Injection
- Open Redirect
- PHP Object Injection
- PHP Object Instantiation
- Reflection/Autoload Injection
- Server-Side JavaScript Injection
- Server-Side Request Forgery
- Session Fixation
- SQL Injection
- Variable Manipulation
- Weak Cryptography
- XML/XXE Injection
- XPath Injection
- Xquery Injection
Automated Detection of Complex Vulnerabilities with Static Code Analysis

3.2 Taint Analysis

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

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

XSS
SQL Injection
File Inclusion
Code Execution
Command Execution
...

1. Introduction
2. Static Code Analysis
3. First-order Bugs
4. Second-order Bugs
5. Gadget Chains
Automated Detection of Complex Vulnerabilities with Static Code Analysis

3.2 Taint Analysis (Refined)

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

Sanitization:
- htmlentities()
- addslashes()
- basename()
- (int)
- escapeshellarg()
- ...

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

Results:
- XSS
- SQL Injection
- File Inclusion
- Code Exec
- Cmd Exec
- ...

1. Introduction
2. Static Code Analysis
3. First-order Bugs
4. Second-order Bugs
5. Gadget Chains
3.3 Security Mechanisms

```php
$uri = htmlentities($_GET['id']);  " → "quot;
1 echo '<a href=""'. $url . '"</a>';  < → &lt;
2 echo '<a href='.$url'>click</a>";
3 echo '<a href='$url'>click</a>";
4 echo '<a href=""'. $url . '"'>click</a>";
```
3.3 Security Mechanisms

```php
$url = htmlentities($_GET['id']);  // → "quote;
echo '<a href="">' . $url . '</a>';  // < → &lt;
```
3.4 Taint Analysis (Context-Sensitive)

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()
- Code Exec
- XSS
- SQL Injection
- File Inclusion
- Cmd Exec
3.5 Context-Sensitive Taint Analysis

```php
$id = $_POST['id'];
if (...) {
    $id = (int)\$id;
}
else {
    $id = htmlentities($id);
}
echo "<div id='\$id'>";
```
3.5 Context-Sensitive Taint Analysis

$\text{id} = \$_\text{POST}['id']$;

$\text{id} = (\text{int})\text{id}$;

$\text{id} = \text{htmlentities}(\text{id})$;

echo "<div id='\text{id}'>";

Markup Context $\text{id}$:
HTML attribute single-quoted (SQ)
### 3.5 Context-Sensitive Taint Analysis

**Code**: $id = \$_POST['id'];

**Sanitized**: Integer only

$\text{echo} \ "<\text{div id='}\$id'\text{'>};$ $\text{htmlentities}(\$id);$ $\text{(int)}\$id;$ $\text{htmlentities}(\$id);$ $\text{echo} \ "<\text{div id='}\$id'\text{'>}";$

**Markup Context $id$:**
HTML attribute single-quoted (SQ)

---

**Diagram:***

1. **Code**
2. **AST**
3. **Basic Blocks**
4. **CFG**
5. **Report**

---

**1. Introduction**
**2. Static Code Analysis**
**3. First-order Bugs**
**4. Second-order Bugs**
**5. Gadget Chains**
Automated Detection of Complex Vulnerabilities with Static Code Analysis

3.5 Context-Sensitive Taint Analysis

$\_\text{POST}$

<table>
<thead>
<tr>
<th>$\text{id}$</th>
</tr>
</thead>
</table>

XSS <> Element

XSS DQ" Attribute

User input (no " < >)

Vulnerable!

$_\text{POST}$['id']

$id = (\text{int}) \text{id}$

$id = \text{htmlentities}($id$);

`echo "<div id='\$id'">";`

Markup Context $\text{id}$:

HTML attribute single-quoted (SQ)
# 3.6 Examples

<table>
<thead>
<tr>
<th>Software</th>
<th>Version</th>
<th>Vulnerability detected by RIPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wordpress</td>
<td>4.01</td>
<td>Cross-Site Scripting</td>
</tr>
<tr>
<td>phpBB</td>
<td>2.0.23</td>
<td>SQL Injection</td>
</tr>
<tr>
<td>phpMyAdmin</td>
<td>4.2.10</td>
<td>Local File Inclusion</td>
</tr>
<tr>
<td>CMS Made Simple</td>
<td>1.11.11</td>
<td>SQL Injection</td>
</tr>
</tbody>
</table>
Automated Detection of Complex Vulnerabilities with Static Code Analysis

4. Second-order Bug Detection
4.1 Second-order Vulnerabilities

Automated Detection of Complex Vulnerabilities with Static Code Analysis

1. Introduction
2. Static Code Analysis
3. First-order Bugs
4. Second-order Bugs
5. Gadget Chains
4.2 Persistent Data Stores

- 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
  - ...

Automated Detection of Complex Vulnerabilities with Static Code Analysis
4.3 First-order Taint Analysis

```php
$name = $_POST['name'];
if(...)
{
    $role = 'admin';
}
else {
    $role = 'user';
}
mysql_query("INSERT INTO users VALUES('$name', '$role')");
```
4.4 Second-order Taint Analysis

```php
$name = addslashes($_POST['name']);
if(...) {
    $role = 'admin';
}
else {
    $role = 'user';
}
mysql_query("INSERT INTO users VALUES('$_POST[name]', '$role')");
```

```
mysql_query("INSERT INTO users VALUES('$_POST[name]', 'admin')
INSERT INTO users VALUES('$_POST[name]', 'user')
```

```sql
users
<table>
<thead>
<tr>
<th>name</th>
<th>role</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

\( \delta \)
4.4 Second-order Taint Analysis

```php
$r = mysql_query('SELECT name FROM users');
if(...) {
    $row = mysql_fetch_assoc($r);
} else {
    die('error');
}
echo "Hi " . $row['name'];
```

Automated Detection of Complex Vulnerabilities with Static Code Analysis

4.5 Second-order Vulnerability Report

PDS

**Temp XSS**
users[name]

source

**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>

**Reads**

**Writes**

Second-Order XSS

$_POST[name]$
## 4.6 Examples

<table>
<thead>
<tr>
<th>Software</th>
<th>Version</th>
<th>Vulnerability detected by RIPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gallery3</td>
<td>3.0.4</td>
<td>Remote Code Execution</td>
</tr>
<tr>
<td>OpenConf</td>
<td>5.30</td>
<td>Remote Code Execution</td>
</tr>
<tr>
<td>osCommerce</td>
<td>2.3.4</td>
<td>Remote Command Execution</td>
</tr>
</tbody>
</table>
Automated Detection of Complex Vulnerabilities with Static Code Analysis

5. Gadget Chain Detection
5.1 PHP Object Injection + POP Chain

PHP Object Injection

Chaining existing code \((\text{gadgets})\)
5.2 PHP Serialization

```php
class Text {
    public function __construct($data) {
        $this->data = $data;
    }
}

$object1 = new Text('Ruhr');
$tmp = serialize($object1);

// O:4:"Text":1:{s:4:"data";s:4:"Ruhr";}

$object2 = unserialize($tmp);
echo $object2->data;
```
5.3 PHP Object Injection (POI)

class Text {
    public function __construct($data) {
        $this->data = $data;
    }
}

$object1 = new Text('Ruhr');
setcookie('tmp', serialize($object1));

// O:4:"Text":1:{s:4:"data";s:4:"Ruhr";}

$object2 = unserialize($_COOKIE['tmp']);
echo $object2->data;
5.3 PHP Object Injection (POI)

class Text {
    public function __construct($data) {
        $this->data = $data;
    }
}

$object1 = new Text('Ruhr');
setcookie('tmp', serialize($object1));

// O:4:"Text":1:{s:4:"data";s:4:"Ruhr";}
// O:8:"stdClass":1:{s:4:"data";s:4:"RIPS";}

$object2 = unserialize($_COOKIE['tmp']);
echo $object2->data;
5.4 Magic Methods

```php
class Text {
    public function __construct($d)
    {
        $this->data = $d;
    }
}

class File {
    public function __destruct()
    {
        unlink($this->filename);
    }
}
```

```php
// 0:4:"Text":1:{s:4:"data";s:4:"Ruhr";}
// 0:4:"File":1:{s:8:"filename";s:10:"config.php";}

$object2 = unserialize($_COOKIE['tmp']);
echo $object2->data;
```
5.5 Property-oriented Programming (POP)

```php
class File {
    public function __destruct(){
        $this->handler->close();
    }
}
```

```php
// O:4:"File":1:{s:7:"handler";O:3:"ABC":0:{}};
```

```php
$object2 = unserialize($_COOKIE['tmp']);
echo $object2->data;
```
5.5 Property-oriented Programming (POP)

```php
class File {
    public function __destruct() {
        $this->handler->close();
    }
}
```

```php
class Process {
    public function close() {
        system('kill '.$this->pid);
    }
}
```

```php
// O:4:"File":1:{s:7:"handler";O:7:"Process":0:{}}
```

```php
<object2 = unserialize($_COOKIE['tmp']);
echo $object2->data;
```
5.5 Property-oriented Programming (POP)

```php
class File {
    public function __destruct() {
        $this->handler->close();
    }
}

class Process {
    public function close() {
        system('kill '. $this->pid);
    }
}

$object2 = unserialize($_COOKIE['tmp']);
echo $object2->data;
```
5.5 Property-oriented Programming (POP)

```php
class File {
    public function __destruct()
    {
        $this->handler->close();
    }
}
```

```php
class Process {
    public function close() {
        system('kill .'.$this->pid);
    }
}
```

```php
$object2 = unserialize($_COOKIE['tmp']);

echo $object2->data;
```

```php
$s:6:"0;calc"
```

```php
$object2 = unserialize($_COOKIE['tmp']);

echo $object2->data;
```

```php
$s:6:"0;calc"
```

```php
$s:6:"0;calc"
```

```php
$s:6:"0;calc"
```
5.6 POI Detection

- Backwards-directed *taint analysis* for `unserialize()`
- If argument is resolved to user input, report POI vulnerability
- Vulnerable `unserialize()` call returns *tainted* object

```php
$tmp = $_COOKIE['tmp'];
$obj = unserialize($tmp);
```
5.6 POI Detection

- Backwards-directed \textit{taint analysis} for \texttt{unserialize()}
- If argument is resolved to user input, report POI vulnerability
- Vulnerable \texttt{unserialize()} call returns \textit{tainted} object
- Propagate \textit{tainted} object forward
5.7 POP Chain Detection

- Invoke **inter-procedural analysis** for all magic methods on POI
- For **unknown receivers**, combine analysis results of methods
5.7 POP Chain Detection

- Invoke inter-procedural analysis for all magic methods on POI
- For unknown receivers, combine analysis results of methods
- Arguments of a sensitive sink that are resolved to object properties are stored as the method's *sensitive properties*

```php
class File {
    public function __destruct() {
        $this->handler->close();
    }
}
```

```php
class Process {
    public function close() {
        system('kill '.$this->pid);
    }
}
```

```php
class Database {
    public function close() {
        mysql_close($this->db);
    }
}
```
5.7 POP Chain Detection

- Invoke **inter-procedural analysis** for all magic methods on POI.
- For **unknown receivers**, combine analysis results of methods.
- Arguments of a sensitive sink that are resolved to object properties are stored as the method's **sensitive properties**.
- Sensitive properties are applied to each **receiver** at call-site.

```php
class File {
    public function __destruct()
    {
        $this->handler->close();
    }
}
```

```php
class Process {
    public function close()
    {
        system('kill ' . $this->pid);
    }
}
```

```php
class Database {
    public function close()
    {
        mysql_close($this->db);
    }
}
```
5.8 POP Chain Report

- Sensitive properties are applied to the receiving object at call-site

- If receiving object is *tainted*, a POP gadget chain is reported and attached to the POI report

```php
$tmp = $_COOKIE['tmp'];
$obj = unserialize($tmp);
$this->handler->pid
$obj->handler->pid

POP Chain (Remote Command Execution)
```
### 5.9 Examples

<table>
<thead>
<tr>
<th>Software</th>
<th>Version</th>
<th>Vulnerability detected by RIPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joomla</td>
<td>3.3.4</td>
<td>PHP Object Injection</td>
</tr>
<tr>
<td>Magento</td>
<td>1.9.0.1</td>
<td>PHP Object Injection</td>
</tr>
<tr>
<td>Drupal</td>
<td>7.34</td>
<td>PHP Object Injection</td>
</tr>
</tbody>
</table>
6. Conclusion

- Requirements for SCA tools changed
  - Diverse language features
  - Applied security mechanisms
  - Complex vulnerability types
  - Growing code size

- SCA can automate bug detection
  - Quickly identify *traditional* vulnerabilities
  - Combine multiple bugs to detect complex bugs
  - Challenges for frameworks (reflection, template engines)
Automated Detection of Complex Vulnerabilities with Static Code Analysis

We are looking for PHP experts and UI designer

Join us building the superior PHP security analysis tool

contact@ripstech.com