Object Flow Analysis in PHP

Extended with include algorithm and annotation analysis

Ruud van der Weijde

April 30, 2014, 28 pages

Supervisor: Jurgen Vinju
Host organisation: Werkspot, http://werkspot.nl
Host supervisor: Winfred Peereboom

Universiteit van Amsterdam
Faculteit der Natuurwetenschappen, Wiskunde en Informatica
Master Software Engineering
http://www.software-engineering-amsterdam.nl

Contents

Abstract

testx

Chapter 1

Introduction

According to the Tiobe Index[TIO14], PHP is on the 6th place of the list with the most populair programming languages.

This paper is unique because it uses Object Flow Analysis on PHP.

In this paper an object oriented approach is used to find vulnerabilities.

In the next chapter 3 the research method is explained. Chapter 2 contains background and context information about related work.

Chapter 2

Background and context

This section should contain two main subjects: Difficulties in analysing PHP and related work. But also Rascal, M3, OFL.

2.1 PHP Analysis

Explain that there is a static and dynamic analysis. Explain them both briefly.

Dynamic analysis The advantage of dynamic analysis is that you are able to analyse all executed code. The downside is that not all lines are executed, and so they are not analysed.

Static analysis The advantage of static analysis is that you can cover all lines of code. The downside is that some expressions in php depend on runtime values. More on this in the section difficulties in PHP.

2.2 Rascal

The language of choice is Rascal, which is developed at CWI for analysing and transforming software.

M3 M3 ref to M3 paper is a generic analysis model which can be used for different kind of analysis.

2.3 OFL

Object Flow Language (OFL) is explained in "Reverse Engineering Object Oriented Code". In this book the writers describe a method to construct UML diagrams from source code. UML diagrams are mainly written when designing the architecture or implementation of the software. When the software is written, the diagrams may not be the same as the implementation anymore. This can also occur when the system evaluates, the diagrams may become outdated.

2.4 From literatur study

The goal of the PHP analysis is to be able to parse PHP code and extract useful information out of it, which then can be analyzed to find vulnerabilities. The main difficulties in analyzing PHP are the dynamic includes, type inference and alias analysis. More details about them can be found below.

Dynamic includes The PHP functions `include[_once]` and `require[_once]` include other script pages in the current script. The location of these files are given as parameter and can be either static or dynamic and are resolved at run time. When a full file path is given, this file will be included. If the location contains a relative path, PHP will try to resolve the file by checking the include path and if

needed the working directory. The location may also contain variables which will be parsed at run time to determine a path to be resolved.
The analysis by Hills, Klint, and Vinju [HKV12], tries to resolve dynamic includes using `__FILE__`, `__DIR__`, and similar variables, constants (defines in PHP), and tries to find files using path matching. Path matching is done by creating a regex for unresolved includes and includes the file if there is a unique match. The results of the analysis show that on average about 80% of the includes can be resolved. The ZendFramework is doing great, about 81%. However, the other frameworks I will probably have to deal with at WerkSpot will be Symfony and Doctrine, which have only 43% and 66% of dynamic includes resolved. I want to see if I can add a layer in the analysis to be able to resolve more includes. This should be possible if I can find a pattern that is used to include dynamic files. Son and Shmatikov [SS11] asked a human to resolve includes if there is more than once match. This might be something to implement, because it can be useful when focusing on one application.

Alias analysis Another part that might be difficult to analyze are referenced objects. In PHP you can refer to an object using `&`, like `$a = &$b`. Now when you modify `$a`, `$b` will also be modified, and the other way around, because they point to the same memory location. During the analysis, I need to focus on how to keep track of aliases. Pixy [JKK06] performs aliases analysis by keeping track of referenced object.

Reflection PHP has many dynamic functions like object constructs containing variables, methods calls can contain variables, and even variables can contain variables. There are also very dynamic functions like `eval` and `call_user_func`. Many people doing static analysis faced the problems of these had to analyze constructs. Facebooks HipHop compiler [Zha+12] tries to resolve dynamic names by keeping track of a global system table.

Chapter 3

Research Method

Steps taken from project plan

- Implement in PHP-Parser/Rascal in the following steps:
- Find out how the compiler works; check control flow graphs, check limitations (part of literature study).
- Convert AST to M3 model for PHP.
- Convert M3 to OFL.
- First OFL step is context-insensitive.
  - Optional: make OFL context-sensitive.
- Apply flow propagation algorithm, as described in the book Reverse Engineering of Object Oriented Code
- Define and add taint restrictions (what can be labeled as tainted?)
- Define and add untain restrictions (how will tainted values be untainted)
- Add application based settings to optimize results
- 'Include' optimization (try to resolve more includes using the study of mark hills)
- Object construction optimization using annotations.
- YAML optimizations to improve analysis results.
  - YAML can be used to scope the number of input flows.
- Validation of the analysis results:
  - Use pre-defined files analyzed in different tools.
  - Use analyzed tools in similar research that is still available for download.
  - Compare results similar existing tools: Pixy, RIPS, SAFERPHP. (All these tools focus on SQLi and XSS).
  - - Optional: Try to run the code in the sinks using a PHP interpreter.
- Produce final results (this step will be iterated)

Chapter 4

Global Analysis

Object flow analysis without optimisations.

4.1 Research

4.2 Results

4.3 Discussion and limitations

Chapter 5

Includes Analysis

The includes analysis done by Mark Hills will be added into the analysis to gain more precise results.

5.1 Research

5.2 Results

5.3 Discussion and limitations

Chapter 6

Annotations Analysis

In the annotations analysis, annotations are used to gain more detailed results of the Object Flow Analysis.

6.1 Research

6.2 Results

6.3 Discussion and limitations

Chapter 7

Conclusion

Summary of the whole work, with conclusions.
T.B.A.

Chapter 8

Examples

Note: this section will be removed/excluded later. It is just included to test code examples, images and tables.

8.0.1 Code example

Hello world php example:

```
echo "Hello World!";
exit;
```

```
// connect to database
$id = $_POST['id'];
mysql_query("select * from table where id = $id");
```

test

8.0.2 Algorithms

Algorithm 1: Pseudocode of the flow propagation algorithm.

foreach *node* $n \in OFG$ **do**
    $\text{in}[n] = \emptyset$
    $\text{out}[n] = \text{gen}[n] \cup (\text{in}[n] \setminus \text{kill}[n])$
end
while *any in[n] or out[n] changes* **do**
    foreach *node* $n \in OFG$ **do**
        $\text{in}[n] = \cup_{p \in pred(n)} \text{out}[p]$
        $\text{out}[n] = \text{gen}[n] \cup (\text{in}[n] \setminus \text{kill}[n])$
    end
end

$pred(n)$ is the set of predecessors of node n.

8.0.3 Image example

This is dummy text around the image.

Figure 8.1: Demo Image

8.0.4 Table example

Some tables from http://en.wikibooks.org/wiki/LaTeX/Tables.

1	2	3
4	5	6
7	8	9

Table 8.1: Demo Table1

Another table to test.

7C0	hexadecimal
3700	octal
11111000000	binary
1984	decimal

Table 8.2: Another demo table

Bibliography

[HKV12] Mark Hills, Paul Klint, and Jurgen J. Vinju. "Program Analysis Scenarios in Rascal". In: *Proceedings of the 9th International Conference on Rewriting Logic and Its Applications*. WRLA'12. Tallinn, Estonia: Springer-Verlag, 2012, pp. 10–30. ISBN: 978-3-642-34004-8. DOI: 10.1007/978-3-642-34005-5_2. URL: http://dx.doi.org/10.1007/978-3-642-34005-5_2.

[JKK06] Nenad Jovanovic, Christopher Kruegel, and Engin Kirda. "Pixy: A Static Analysis Tool for Detecting Web Application Vulnerabilities (Short Paper)". In: *Proceedings of the 2006 IEEE Symposium on Security and Privacy*. SP '06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 258–263. ISBN: 0-7695-2574-1. DOI: 10.1109/SP.2006.29. URL: http://dx.doi.org/10.1109/SP.2006.29.

[SS11] Sooel Son and Vitaly Shmatikov. "SAFERPHP: Finding Semantic Vulnerabilities in PHP Applications". In: *Proceedings of the ACM SIGPLAN 6th Workshop on Programming Languages and Analysis for Security*. PLAS '11. San Jose, California: ACM, 2011, 8:1–8:13. ISBN: 978-1-4503-0830-4. DOI: 10.1145/2166956.2166964. URL: http://doi.acm.org/10.1145/2166956.2166964.

[TIO14] TIOBE. *TIOBE Index for March 2014*. Mar. 2014. URL: http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html.

[Zha+12] Haiping Zhao et al. "The HipHop Compiler for PHP". In: *SIGPLAN Not.* 47.10 (Oct. 2012), pp. 575–586. ISSN: 0362-1340. DOI: 10.1145/2398857.2384658. URL: http://doi.acm.org/10.1145/2398857.2384658.

Appendices can be found in the next pages. The current ones will be removed in the final versions.

Appendix A

Project Plan

Plan for a Master's Project

Title	Data flow analysis to detect sql-injection vulnerabilities using Rascal
Document version	0.4.0 (fourth draft version)
Student	Ruud van der Weijde | 10453857
Host organization	Werkspot | www.werkspot.nl | Technology | Amsterdam
Contact person	Winfred Peereboom | Technology Director | winfred.peereboom@werkspot.nl | 06-46446888
Project summary (short version)	In my thesis, I will do a static source code analysis to find sql-injection vulnerabilities on the codebase of Werkspot. The implementation will be done in Rascal. It will contain the following main steps: • Parse PHP code to AST (using PHP-Parser & php-analysis) • Convert the AST to Object Flow Language (OFL) (in Rascal) ◦ OFL is used to get the data flows of an OO-program • Optimize the analysis (in Rascal) ◦ By reading annotations to find object information. ◦ Optional: By using the PHP Include algorithm of Mark Hills. • Propagate Object Flows Structure to find `sinks`. • Validation of the results in iterating form. The analysis will first focus on finding SQL injection (SQLi) and afterwards on finding XSS vulnerabilities. The scope of the research will be the legacy part of the system, which is using ZendFramework1 and Symfony1 (both first versions of the frameworks), and doctrine12 (all deprecated frameworks).
Research question	- How can vulnerabilities be found in PHP using Rascal, having as precise results as possible.
Research method	Implement in PHP-Parser/Rascal in the following steps: • Find out how the compiler works; check control flow graphs, check limitations (part of literature study). • Convert AST to M3 model for PHP. • Convert M3 to OFL. • First OFL step is context-insensitive. • [Optional: make OFL context-sensitive.] • Apply flow propagation algorithm, as described in the book "Reverse Engineering of Object Oriented Code" • Define and add taint restrictions (what can be labeled as tainted?) • Define and add untain restrictions (how will tainted values be untainted) • Add application based settings to optimize results • `Include` optimization (try to resolve more includes using the study of mark hills) • Object construction optimization using annotations. • YAML optimizations to improve analysis results. ◦ YAML can be used to scope the number of input flows. • Validation of the analysis results: ◦ Use pre-defined files analyzed in different tools. ◦ Use analyzed tools in similar research that is still available for download.

	◦ Compare results similar existing tools: Pixy, RIPS, SAFERPHP. (All these tools focus on SQLi and XSS). ◦ [Alternative: Try to run the code in the sinks using a PHP interpreter.] ◦ Produce final results (this step will be iterated)
Expected results	Thesis, analysis tool, human readable results of analysis.
Required expertise	PHP, Rascal, taint + static code analysis, PHP compiler, OFL, SSA
Global timeline	• Jan: ◦ Finish the project plan. • Feb: ◦ Thesis: background, problem description, motivation. ◦ Rascal: basic proof of concept in Rascal • Mrt: ◦ Thesis: describe analysis method. ◦ Define taint and untaint definitions. ◦ Rascal: Extend the proof of concept to a working prototype. And compare the results with existing programs. • Apr: ◦ Thesis: update analysis method, write about result analysis. ◦ Rascal: Improve prototype using dynamic include resolving. • May: ◦ Thesis: update thesis, add first results to thesis. ◦ Rascal: Add annotations to analysis, for doctrine/symfony. • Jun: ◦ Thesis: add results to thesis. ◦ Rascal: Handle type hinting annotations out of doctrine/symfony. • Jul: ◦ Thesis: Finish, write conclusion, future work, limitations • Aug: ◦ Room to fix delays in the progress.
Main risks	• Too ambitious: provide better scope on the project. ◦ Added scope in vulnerabilities: SQLi first, maybe XSS later. ◦ Added scope to source code, only old website code. • Unclear project goals: define exactly what is expected. ◦ Human readable output is expected. ◦ This can be done by providing sinks, which are readable. • Analysis is not measuring the right data. ◦ Use comparable tools to look for similar results. • Too many false positives. ◦ Add scoping in input values using routing.yml files. ◦ Apply include algorithm to reduce the number of options.

Appendix B

Literature study

1 Relevant Literature

The main structure of my literature study is:

- PHP Dynamics.
  - Dynamic includes.
  - Alias analysis.
  - Reflection.
- Finding Vulnerabilities.
  - Static analysis.
  - Dynamic analysis.
  - Existing tools.
- Literature Log

1.1 PHP Dynamics

The goals of the PHP analysis is to be able to parse PHP code and extract useful information out of it, which then can be analyzed to find vulnerabilities. The main difficulties in analyzing PHP are the dynamic includes, type inference and alias analysis. More details about them can be found below.

Dynamic includes The PHP functions `include[_once]` and `require[_once]` include other script pages in the current script. The location of these files are given as parameter and can be either static or dynamic and are resolved at run time. When a full file path is given, this file will be included. If the location contains a relative path, PHP will try to resolve the file by checking the include path and if needed the working directory. The location may also contain variables which will be parsed at run time to determine a path to be resolved.
The analysis by Hills, Klint, and Vinju [HKV12], tries to resolve dynamic includes using `__FILE__`, `__DIR__`, and similar variables, constants (defines in PHP), and tries to find files using path matching. Path matching is done by creating a regex for unresolved includes and includes the file if there is a unique match. The results of the analysis show that on average about 80% of the includes can be resolved. The ZendFramework is doing great, about 81%. However, the other frameworks I will probably have to deal with at WerkSpot will be Symfony and Doctrine, which have only 43% and 66% of dynamic includes resolved. I want to see if I can add a layer in the analysis to be able to resolve more includes. This should be possible if I can find a pattern that is used to include dynamic files. Son and Shmatikov [SS11] asked a human to resolve includes if there is more than once match. This might be something to implement, because it can be useful when focusing on one application.

Alias analysis Another part that might be difficult to analyze are referenced objects. In PHP you can refer to an object using `&`, like `$a = &$b`. Now when you modify `$a`, `$b` will also be modified, and the other way around, because they point to the same memory location. During the analysis, I need to focus on how to keep track of aliases. Pixy [JKK06] performs aliases analysis by keeping track of referenced object.

Reflection PHP has many dynamic functions like object constructs containing variables, methods calls can contain variables, and even variables can contain variables. There are also very dynamic functions like `eval` and `call_user_func`. Many people doing static analysis faced the problems of these had to analyze constructs. Facebooks HipHop compiler [Zha+12] tries to resolve dynamic names by keeping track of a global system table..

1.2 Finding Vulnerabilities

Many studies use taint analysis to find possible vulnerabilities. I will list a few tools in the paragraph below.

Static analysis Many researches [MLA06; JKK06; SS11; HKV12; HKV13] used static analysis to analyze PHP and some tried to find vulnerabilities. The PHP is parsed by a compiler, mostly using an open source external parser. The limitations of the parser are not always fully described, but since PHP is very hard to statically analyze, the can probably not fully analyze PHP.
A fairly common used method to find vulnerabilities in PHP code is done using **taint analysis**. Taint analysis keeps track of input variables which can be manipulated by an outsider. For example all `$_POST` and `$_GET` can easily be modified. Variables that have tainted values will be tagged as tainted. The variables stay tainted until an untaint function is called (which can be defined by the analyzer, for instance: `htmlentities`, `addslashes`, and `mysql_real_escape_string`). Variables modified by untaint functions will no longer be tagged as tainted and will be 'safe'. The analysis on finding vulnerabilities will then be performed by checking possible vulnerable constructs which contain tainted variables, like for instance SQL strings.
An algorithm using taint analysis using Extended Static Singe Assignment is proposed in a paper by Rimsa et al. [Rim+14]

Dynamic analysis Many researchers use static analysis to find security vulnerabilities. I have two papers I still want to read, which use dynamic analysis, or combine them with static analysis to gain more precise results. These two papers are: SANER (2008) and WAFA (2009).
The paper about Saner, written by Balzarotti et al. [Bal+08], reuses static analysis from Pixy [JKK06]. Here are possible candidate security vulnerabilities tagged. The dynamic part of the analysis will then evaluate the custom

functions to create untaint variables using a predefined set of input test variables.
I will add up information about WAFA here later...

Existing tools There have been researchers with the same goal to find vulnerabilities in PHP programs. Two examples are Pixy and SAFERPHP and are described below. More will be added.
Pixy [JKK06] is a tool presented in 2006. Their tools uses taint analysis to find XSS vulnerabilities and SQL- and command injection. The major limitation of this tool is that it is build for PHP4 and does not support object oriented features.
SAFERPHP [SS11] is presented in 2011 and also uses taint analysis. It focuses on finding possible infinite loops that can be caused by tainted input, unauthorized access to sensitive operations, SQLi, misuse of uninitialized variables, and tainted input in vulnerable native PHP functions. The first and last can lead to DoS (denial of service) by an outsider, which can be used to make a website unavailable by consuming all resources of the webserver. The others are security vulnerabilities, which can be used to gain sensitive information from the web- or database server.

1.3 Literature Log

Title "SAFERPHP: Finding Semantic Vulnerabilities in PHP Applications"

Author(s) Son and Shmatikov [SS11]

Summary SAFERPHP is a PHP analysis tool that is able to find five types of vulnerabilities: DoS by infinite loops, unprotected script pages, SQLi, misuse of uninitialized variables, and DoS due to allowing user input in native PHP function calls. SAFERPHP will ask the user to provide information on dynamic includes, to be able to resolve as many as possible) and then creates a call graph. Using taint analysis, the authors are able to find whether a variable at a given point can be manipulated by an untrusted source.

Difference The difference is that the authors mainly look for loops which are DoS attack prone.

Useful results Statical tainted analysis of PHP code. Tainted variable are the root of many problem. In the paper the authors define what type of variables are categorized as tainted.

Open questions Variable function calls are still not covered, like: $foo(); Besides that, in PHP there are no return types defined. So It is hard to define the result of the function, especially when you expect to receive an Object of type X or Y, like: $foo = $bar(); $foo->bar(); Another point is that the analysis has several sources of false positives.

Rejected? No. Paper not rejected.

Title "An Empirical Study of PHP feature usage: a static analysis perspective"

Author(s) Hills, Klint, and Vinju [HKV13]

Summary This paper describes the usage of the native PHP features. This is done by analyzing a list of programs written in PHP. PHP files are parsed and analyzed using Rascal-MPL.

Difference This papers differs because its main goal differs. They only investigate how far you can get by analyzing dynamic PHP code. I would like to jump more into the security part.

Useful results The open source PHP analysis tool made by CWI and the results of how much of PHP information is available.

Open questions How to resolve the dynamic includes, function callbacks, and the `eval` function.

Rejected? No. Not rejected. Will serve as a good basis for PHP static analysis.

Title "Program Analysis Scenarios in Rascal"

Author(s) Hills, Klint, and Vinju [HKV12]

Summary This paper explains how Rascal can be used to analyze and transform source code. They provide information on the ongoing research of analyzing PHP code.

Difference The difference with my approach is that this paper tries to rewrite existing code. My intention is not to do this.

Useful results The analysis of PHP using Rascal (and PHP-Parser) is really useful. They also explain the difficulties of analyzing PHP code (Includes, Type Inference, and Alias Analysis.

Open questions Handling variable constructs, memory consuming for bigger projects, not fully integrated for PHP5 features.

Rejected? No. Useful information on PHP analysis.

Title "Insider and Ousider Threat-Sensitive SQL Injection Vulnerability Analysis in PHP"

Author(s) Merlo, Letarte, and Antoniol [MLA06]

Summary The authors use an approach based on static analysis to find statements the could be vulnerable to SQLi. The paper describes an algorithm that will determine the authorization levels for the PHP code by static inter-procedural flow analysis. Once this information is available, another check will use the manual configured security levels to find vulnerabilities.

Difference This one differs from my approach because I do not want to avoid manual configuration of authorization levels.

Useful results The part of the statical analysis to find the intern flow is useful. Also the fact that they do not only focus on the outside treats but also on the inside treats (errors made by the programmers).

Open questions More a limitation, manual configuration of access/authorization levels

Rejected? No. Paper not rejected.

Title "SQL-Injection Security Evolution Analysis in PHP"

Author(s) Merlo, Letarte, and Antoniol [MLA07b]

Summary This paper is a case study on the evolution of security vulnerabilities using the technique of the previous paper by Merlo: "Insider and Ousider Threat-Sensitive SQL Injection Vulnerability Analysis in PHP".

Difference Difference from my approach is that this paper focuses on the evolution of security threats.

Useful results Only useful result is that they show the previous algorithm in practice.

Open questions No open questions for this paper.

Rejected? Yes. Rejected because it's kind of a case study on the previous paper by Merlo. The previous one contains information on the implantation, the interesting part of this research.

Title "Automated Protection of PHP Applications Against SQL-injection Attacks"

Author(s) Merlo, Letarte, and Antoniol [MLA07a]

Summary This article analysis PHP code by combining static and dynamic analysis. The goals is to find database calls (SQL) and insert model-based guards using prepare statements to prevent SQLi.

Difference This item differs from my approach because I am not planning to automatically refactor legacy code.

Useful results The article parses PHP and creates an AST using JavaCC.

Open questions The used method is only applicable for simple similar SQL statements.

Rejected? Partly. The analysis of PHP is something to look into.

Title "A comparison of the efficiency and effectiveness of vulnerability discovery techniques"

Author(s) Austin, Holmgreen, and Williams [AHW13]

Summary This paper compares different vulnerability techniques (XSS, SQLi, dangerous function, path manipulation, error information leak, HTTPonly attribute, hidden field manipulation, command injection, nonexistent access control, auditing, trust boundary validation, dangerous file upload, and uncontrolled resource consumption). A case study is done on three software products (2 JAVA, 1 PHP) using (semantic)manual/automatic penetration tests, and static analysis. Static analysis found the most vulnerabilities, but also many false positives which are were time consuming to find out.

Difference This is a comparison of techniques. My goal will be to pick one technique, although they advice to use multiple.

Useful results One technique may not be sufficient to find all vulnerabilities.

Open questions Too many false positives were found using statical analysis.

Rejected? No. Not fully rejected, might be useful for background information when explaining available techniques.

Title "Efficient static checker for tainted variable attacks"

Author(s) Rimsa et al. [Rim+14]

Summary The writers propose an algorithm for tainted analysis on (for example) PHP code using e-SSA (Extended Static Single Assignment). e-SSA created unique entries for each assign statement.

Difference This is a general approach, I would like to see if I can add a layer to provide better solution for specific applications. My goal is not to check any PHP application.

Useful results The algorithm is (claimed to be) pretty fast.

Open questions The algorithm is based on the PHC (open source PHP compiler), and this compiler has limitations in analyzing PHP code.

Rejected? No. I want to do tainted analysis, so this paper might be useful afteral.

Title "Pixy: A Static Analysis Tool for Detecting Web Application Vulnerabilities (Short Paper)"

Author(s) Jovanovic, Kruegel, and Kirda [JKK06]

Summary In this paper a tool called Pixy is presented. The tool tries to find vulnerabilities from tainted input/variables using uses data flow analysis, which is based on CFG's. The PHP code is parsed using JFlex (lexer) and Cup (parser).

Difference It is different to my approach because I will focus on object oriented structures.

Useful results The analysis based on tainted input (and untainted) to find vulnerabilities. When doing analysis, it detects aliases (referenced objects) and tags them as tainted as well.

Open questions This tool is not able to analyze object oriented structures. Also some includes are not handled correctly, they are ignored.

Rejected? No. The analysis used in this paper may be useful.

Title "The HipHop Compiler for PHP"

Author(s) Zhao et al. [Zha+12]

Summary HipHop is created by Facebook developers is a static compiler which compiles PHP code to C++ code. It parses the PHP code creating an AST and then run optimizations and create C++ code.

Difference The difference is that they focus and optimize for performance reasons. I'm interested in the security aspects.

Useful results In the transformation of PHP to C++, Hiphop tries to deal with dynamic name binding and function initialization, it keeps track of an global system table which contain unique names for variable and redefined classes and methods.

Open questions Not all dynamic parts are covered.

Rejected? No. The methods to resolve dynamic name binding, redeclared functions/classes, and reflection are interesting

---

Title "Saner: Composing Static and Dynamic Analysis to Validate Sanitization in Web Applications"

Author(s) Balzarotti et al. [Bal+08]

Summary The writers present a tool called Saner, using static and dynamic analysis. The static analysis part is based on Pixy (tool listed in here as well). The dynamic analysis is done by stripping all unrelated information from the sanitization (=code that untaints variables) graph and tests the PHP code using predefined sets of test values. The main focus of the article is to check custom sanitization checks to see if they actually untaint variables, which is done by the dynamic part of the analysis.

Difference It differs because their focus is on the custom sanitization functions.

Useful results Lowering the false positives using an automated test procedure is very useful.

Open questions The tool is tested on small projects. I wonder if it is applicable on big (real life) systems.

Rejected? No. Nice way to reduce the number of false positives.

Title "Static Detection of Security Vulnerabilities in Scripting Languages"

Author(s) Xie and Aiken [XA06]

Summary The paper describes a statical analysis approach to analyze PHP to find security vulnerabilities. It is a three level analysis: on blocks, intraprocedural (over blocks) and interprocedural.

Difference This research does not focus on very dynamic parts of the language.

Useful results The three layer approach is a nice step towards finding more vulnerabilities.

Open questions This research does still not fully cover the PHP analysis problem.

Rejected? No, decent basis to work on. They have cleanly described the procedure they use to analyze.

Title "Evaluation of SQL Injection Detection and Prevention Techniques"

Author(s) Tajpour and JorJor Zade Shooshtari [TJ10]

Summary The paper describes nine types of SQLi and describes twenty-three tools/techniques. The paper maps the tools/techniques with the types of SQLi it covers (based on article information).

Difference The paper is different because it just gives an overview of the types of SQLi and a list of available tools.

Useful results It gives a decent overview several SQLi types and lists a number of tools which can be used to scan them. They talk about SAFELI, a statical monitor for ASP.NET. Might be something to lookup. Reference 19 (Pietraszek) and 20 (Nguyen-Tuong) modify a PHP interpreter to track precise per-character information. These papers should provide more information than this paper.

Open questions Open questions

Rejected? Yes. There are not enough details in the paper. Only about 2 to 3 lines per technique. It might be good to look up some of the tools, like SAFELI and AMNESIA.

Title "Fast Detection of Access Control Vulnerabilities in PHP Applications"

Author(s) Gauthier and Merlo [GM12]

Summary In this paper the authors created a mechanism to detect pages that are not guarded by permission checks. It first checks what links to pages are available and which are blocked, and then checks if these pages are accessible without rights.

Difference This article focuses on access control. This is not one of my/our security top priorities.

Useful results The authors used JavaCC to parse the PHP code. JavaCC creates a CFG of the source. This is something to look into.

Open questions This method uses static access checks. Would be nicer to have dynamic checks, but this is hard in PHP.

Rejected? Yes. Rejected because it is out of the scope. If focuses on access control, and I'm not planning to do that.

Title "Fault Localization for Dynamic Web Applications"

Author(s) Artzi et al. [Art+12]

Summary This paper explains a new created tool Apollo. It will give a prioritized list of bug candidates (execution failures or HTML faults), based on the most likely part of the source code that causes the bug. To narrow down the bugs, they use a method to automatically generate test cases, and use a constraint solver to generate input for the tests. The authors use 3 algorithms for fault localization: Tarantula, Ochiai, and Jaccard.

Difference This paper is searching for bugs and tries to narrow down the source. I will not do something similar.

Useful results The writers created a test generation tool, to generate test to narrow down the fault. Besides that, they used algorithms of other languages and applied them to PHP.

Open questions Apollo is only applied to small simple PHP projects. They also expect programmers to debug their code

Rejected? Yes. This project is out of the scope.

References

[AHW13] Andrew Austin, Casper Holmgreen, and Laurie Williams. "A comparison of the efficiency and effectiveness of vulnerability discovery techniques". In: *Information and Software Technology* 55.7 (2013), pp. 1279–1288. ISSN: 0950-5849. DOI: http://dx.doi.org/10.1016/j.infsof.2012.11.007. URL: http://www.sciencedirect.com/science/article/pii/S0950584912002339.

[Art+12] S. Artzi et al. "Fault Localization for Dynamic Web Applications". In: *Software Engineering, IEEE Transactions on* 38.2 (2012), pp. 314–335. ISSN: 0098-5589. DOI: 10.1109/TSE.2011.76.

[Bal+08] D. Balzarotti et al. "Saner: Composing Static and Dynamic Analysis to Validate Sanitization in Web Applications". In: *Security and Privacy, 2008. SP 2008. IEEE Symposium on*. 2008, pp. 387–401. DOI: 10.1109/SP.2008.22.

[GM12] F. Gauthier and E. Merlo. "Fast Detection of Access Control Vulnerabilities in PHP Applications". In: *Reverse Engineering (WCRE), 2012 19th Working Conference on*. 2012, pp. 247–256. DOI: 10.1109/WCRE.2012.34.

[HKV12] Mark Hills, Paul Klint, and Jurgen J. Vinju. "Program Analysis Scenarios in Rascal". In: *Proceedings of the 9th International Conference on Rewriting Logic and Its Applications*. WRLA'12. Tallinn, Estonia: Springer-Verlag, 2012, pp. 10–30. ISBN: 978-3-642-34004-8. DOI: 10.1007/978-3-642-34005-5_2. URL: http://dx.doi.org/10.1007/978-3-642-34005-5_2.

[HKV13] Mark Hills, Paul Klint, and Jurgen J. Vinju. "An Empirical Study of PHP feature usage: a static analysis perspective". In: *ISSTA*. Ed. by Mauro Pezzè and Mark Harman. ACM, 2013, pp. 325–335.

[JKK06] Nenad Jovanovic, Christopher Kruegel, and Engin Kirda. "Pixy: A Static Analysis Tool for Detecting Web Application Vulnerabilities (Short Paper)". In: *Proceedings of the 2006 IEEE Symposium on Security and Privacy*. SP '06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 258–263. ISBN: 0-7695-2574-1. DOI: 10.1109/SP.2006.29. URL: http://dx.doi.org/10.1109/SP.2006.29.

[MLA06] E. Merlo, D. Letarte, and G. Antoniol. "Insider and Ousider Threat-Sensitive SQL Injection Vulnerability Analysis in PHP". In: *Reverse Engineering, 2006. WCRE '06. 13th Working Conference on*. 2006, pp. 147–156. DOI: 10.1109/WCRE.2006.33.

[MLA07a] E. Merlo, D. Letarte, and G. Antoniol. "Automated Protection of PHP Applications Against SQL-injection Attacks". In: *Software Maintenance and Reengineering, 2007. CSMR '07. 11th European Conference on*. 2007, pp. 191–202. DOI: 10.1109/CSMR.2007.16.

[MLA07b] E. Merlo, D. Letarte, and G. Antoniol. "SQL-Injection Security Evolution Analysis in PHP". In: *Web Site Evolution, 2007. WSE 2007. 9th IEEE International Workshop on*. 2007, pp. 45–49. DOI: 10.1109/WSE.2007.4380243.

[Rim+14] Andrei Rimsa et al. "Efficient static checker for tainted variable attacks". In: *Science of Computer Programming* 80, Part A (Feb. 2014), pp. 91–105. ISSN: 0167-6423. URL: http://www.sciencedirect.com/science/article/pii/S0167642313000737.

[SS11] Sooel Son and Vitaly Shmatikov. "SAFERPHP: Finding Semantic Vulnerabilities in PHP Applications". In: *Proceedings of the ACM SIGPLAN 6th Workshop on Programming Languages and Analysis for Security*. PLAS '11. San Jose, California: ACM, 2011, 8:1–8:13. ISBN: 978-1-4503-0830-4. DOI: 10.1145/2166956.2166964. URL: http://doi.acm.org/10.1145/2166956.2166964.

[TJ10] A. Tajpour and M. JorJor Zade Shooshtari. "Evaluation of SQL Injection Detection and Prevention Techniques". In: *Computational Intelligence, Communication Systems and Networks (CICSyN), 2010 Second International Conference on*. 2010, pp. 216–221. DOI: 10.1109/CICSyN.2010.55.

[XA06] Yichen Xie and Alex Aiken. "Static Detection of Security Vulnerabilities in Scripting Languages". In: *Proceedings of the 15th Conference on USENIX Security Symposium - Volume 15*. USENIX-SS'06. Vancouver, B.C., Canada: USENIX Association, 2006. URL: http://dl.acm.org/citation.cfm?id=1267336.1267349.

[Zha+12] Haiping Zhao et al. "The HipHop Compiler for PHP". In: *SIGPLAN Not.* 47.10 (Oct. 2012), pp. 575–586. ISSN: 0362-1340. DOI: 10.1145/2398857.2384658. URL: http://doi.acm.org/10.1145/2398857.2384658.