Object Flow Analysis in PHP
Extended with include algorithm and annotation analysis

Ruud van der Weijde

April 24, 2014, 6 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 **2**

1 Introduction **3**

2 Background and context **4**

2.1 PHP Analysis 4

2.2 Rascal 4

2.3 OFL 4

2.4 From literatur study 4

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.

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 This can also occur when the system evaluates, the diagrams may become outdated.

$\langle \textit{statement} \rangle$ ::= $\langle \textit{ident} \rangle$ '=' $\langle \textit{expr} \rangle$

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.

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.