Defense of PHP: Method and Practice

Fan Runqi*
College of Cybersecurity
Sichuan University
Chengdu, Sichuan 610207
812633546@qq.com

Huo Yingxiao
School of Computer Science
University of Nottingham
Ningbo China
Ningbo, ZheJiang 315154
huoyingxiao@outlook.com

Liu Hanzhi
School of Electronic Information and Electrical Engineering
Shanghai Jiao Tong University
Shanghai, China 200240
misaka@sjtu.edu.cn

Xiao Ang
School of Cyber Science and Engineering
Huazhong University of Science and Technology
Wuhan, Hubei, China 430074
xiaoang@hust.edu.cn

Yan Yunxiang
College of Finance
Southern University of Science and Technology
Shenzhen, Guangdong
518055
11912525@mail.sustech.edu.cn

ABSTRACT

Nowadays, Web applications are developing rapidly, PHP become the most popular programming language on web applications. However, because of the defects of PHP itself and mistakes in the developing, PHP source code usually have lots of vulnerabilities. Aiming to detect and solve vulnerabilities in source code, there are many tools. In this paper, we introduced three static analysis tools used taint analysis: RIPS, phpSafe and pixy. First, we introduced the principle of taint analysis and these tools. Then we compared these tools use a sample source code with different vulnerabilities. Finally, based on analysis of the pros and cons of each tools, we modified and extend taint, and provided a new GUI of taint.

Categories and Subject Descriptors

C.2.0 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

Keywords

PHP, taint analysis, static analysis, vulnerabilities detection

1. INTRODUCTION

Nowadays, the development of web application is growing exponentially fast, web application is used everywhere in our life. In these web applications, PHP language is the most widely used server-side programming language. which is 81.4% of all recognized websites [1]. PHP have a large number of built-in features, weakly and dynamically typed syntax, so that even beginner can master it very fast. However, because of it complex features and characteristics, it is easy to have bugs or vulnerabilities during PHP application development. according to the MITRE CVE database [2], PHP language related vulnerabilities account for 29% in all vulnerabilities found in computer software. In various of vulnerabilities, SQL injection (SQLi) and Cross Site Scripting (XSS) are two most common vulnerabilities. Targeting these vulnerabilities, there are several industry PHP static analysis tools: such as Pixy, RIPS, PHPSafe, PHP-Sat and so on.

Pixy is an open-source PHP static detection tool which developed by Java, it use data flow analysis to detecting vulnerabilities, and it is mainly directed to XSS vulnerabilities, but it does not supports features and syntax of PHP5, it only supports PHP4 syntax.

RIPS (Research and Innovation to Promote Security) is a popular tool of PHP source code analysis. it supports detecting on 15 kinds of vulnerabilities, include SQLi, XSS, XPath injection, header injection and so on.

PHPSafe as a free and open-source PHP analysis tool support analysis of Object-Oriented Programming (OOP) web application plugins, it support to detect XSS and SQLi vulnerabilities.

Each of the above tools used taint analysis to detected vulnerabilities. Taint analysis method is converting source code into abstract syntax tree (AST), and then transformed into control flow graph (CFG), so that tools can use local file to analyse the code. The process of taint analysis can be divided into two stages: identify the taint source and propagate the tainted node. After that, programmer or maintainer can modify the source code based on the result.

In this paper, we based on three developed tools: RIPS, phpSafe and pixy, first explain and analyse the principle

*Author names are listed in alphabetical order.

of these four tools, and test their analytical performance. Finally, we will extend the capacity of taint and give it a new graphical user interface (GUI) on webpage.

2. TAINT ANALYSIS

2.1 Taint-Style Vulnerabilities

The PHP language is one of the most widely used web services programming languages in the world. However, it is prone to various vulnerabilities, such as SQL injection and XSS (cross-site scripting). According to the latest survey by OWASP (Open Web Application Security) in 2020, both SQL injection and XSS are listed as the top 10 web application security vulnerabilities. These two vulnerabilities are mainly triggered by the fact that user input data may be carefully constructed by an attacker, leading to undesired behavior, so these types of vulnerabilities are called tainted vulnerabilities.

2.2 Taint Analysis Algorithm

To conduct taint analysis, the PHP source code should be transformed into a data structure that is conductive to taint analysis. Specifically, the PHP source code is converted to AST (abstract syntax tree) with parser. Then, the AST is transformed into CFG (control flow graph) by CFG generation algorithm. Taint analysis can be performed on CFG.

Taint analysis is divided into the following two steps:

1) Identify the Taint Source: Taint source is the untrustworthy data inputted by users, including the user's direct input and MySQL sensitive functions. The user's direct input can be gotten by the super-global variables. The super-global variables (userTainted) comprise '$_GET', '$_POST', '$_COOKIE', and so on. MySQL sensitive functions (secretTainted) include 'mysql_query', 'mysql_fetch_array', and so on.

2) Propagate the Tainted Node: After we determine the taint source, the control flow graph can be traversed to determine all variables related to the taint source, and mark them as **userTainted**, or **secretTainted**. Furthermore, If the label is **userTainted**, there may be an XSS vulnerability. If the label is **secretTainted**, there may be a SQL injection vulnerability.

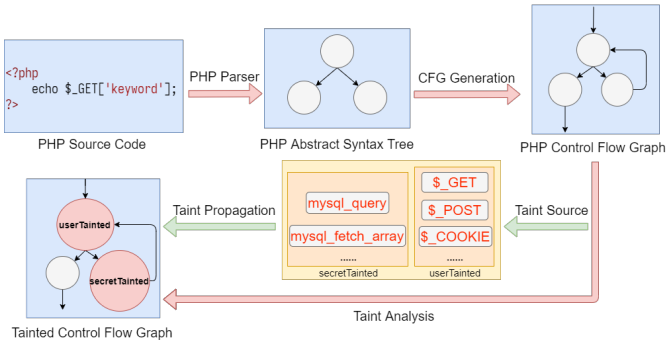

Figure 1: Flow Chart of Taint Analysis on PHP

3. TOOL BASED ON TAINT ANALYSIS

3.1 phpSAFE

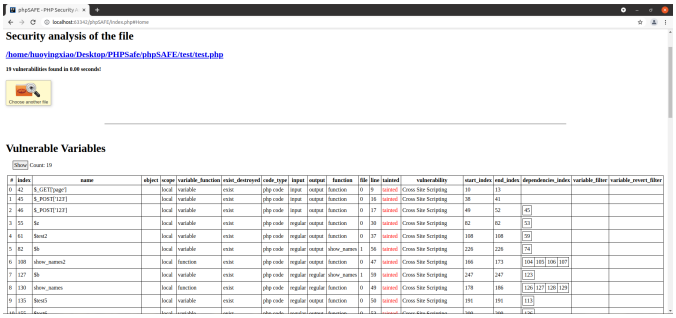

Figure 2: Working Interface of phpSafe

3.1.1 *Introduction*

This section will introduce a security analysis tool for web application plugins: phpSAFE. Different with some common tools on the market(eg. RIPS, Pixy), phpSAFE support for the analysis on Object Oriented Programming (OOP) code, which is commonly used for Content Management Systems (CMS) applications.

phpSAFE support for detection of Cross Site Scripting (XSS) and SQL injection (SQLi) vulnerabilities, which are two of the most common vulnerabilities on web application, and have a widespread use by organized crime and hackers [3][4]. phpSAFE use the technique of the static code analysis, which can permit the coverage of the source code, and use the concept of taint analysis to locate vulnerabilities. In phpSAFE, the code is modeled as a Control Flow Graph (CFG) and analyzed using some analysis technique: context sensitive data flow analysis, inter-procedural or global analysis, intra-procedural or local analysis, functions summaries, whole-program analysis [5].

3.1.2 *Workflow*

The analysis of phpSAFE have four phases: configuration, model construction analysis, results processing.

1) Configuration, in this stage, the configuration data are loaded, which are some php functions with several XSS and SQLi vulnerabilities. All configuration files is consisted of three files: *class-vulnerable-input.php*, *class-vulnerable-filter.php*, *class-vulnerable-output.php*[5]. By comparing the source code to these functions, so that phpSafe can locate the vulnerabilities in the source code.

2) Model construction, in this stage. the function token_get_all split the source code into many tokens, and phpSafe create Abstract Syntax Tree (AST) by these tokens. [5] Then clean the comments and whitespaces in AST.

3) Analysis, in this stage, phpSafe trace all tainted variables from where they enter the application or plugin to where they output. in order to ensure the 100 % code coverage, phpSafe starts analysis by executing functions which are not called from source code of application or plugins. [5] Then, analysis restarts from "main function", so that, every code snippets can be analyzed.

4) Result processing, after analysis, phpSafe provides variables which are tainted, and provides some resources: vulnerable, output and other variables, functions, tokens and the whole AST, and debug information.

3.2 RIPS

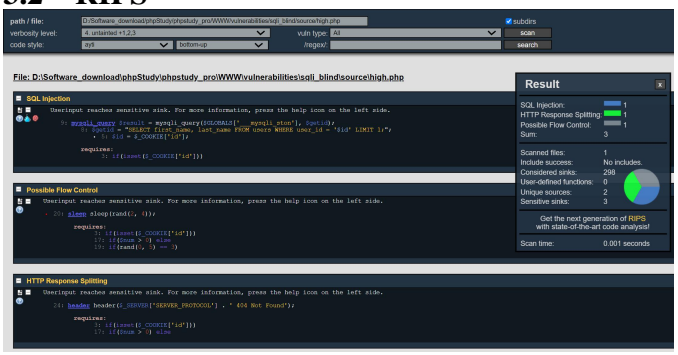

Figure 3: Working Interface of RIPS

3.2.1 Introduction

RIPS is short for Research and Innovation to Promote Security, which is a static code analysis tool for security vulnerabilities in PHP.[6]Its initial version is developed by Johannes Dahse. However, its open-source version stopped maintenance and the later version became a software product of RIPS Technologies, who was acquired by SonarSource in 2016.[7] The below discussion on the source code and implementation principle are all based on the latest opensource version: rips-0.55.[8]

3.2.2 Workflow

1)AST and CFG

RIPS first generates a main Abstract Syntax Tree(AST) for all the files in the project by removing the body of functions(who then form a separate AST). After that, RIPS transforms each main AST into a Control Flow Graph(CFG). To be specific, new block will be created when a conditional jump is encountered and a block edge is formed to connect the new block to the old one.

2) Simulation and Summary

Data flow is simulated with the blocks created previously and a backwards analysis is conducted on the data flows. As a result, block summary are generated by combining all the analysis results.

3)Taint Analysis

RIPS then conducts taint analysis on the simulated basic blocks. Note that it contains several novel techniques, such as assigning different sanitation tags for different vulnerability types, modeling a total of 952 built-in functions to recognize a variety of PHP-specific vulnerabilities and utilizing backwards-directed analysis to improve computing performances.[6]

3.3 Pixy

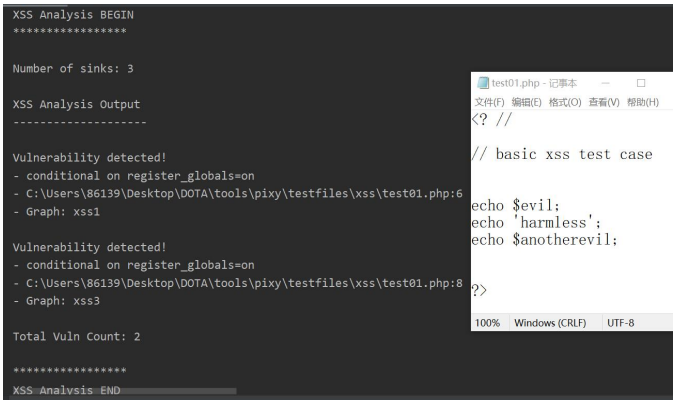

Figure 4: Working Interface of pixy

3.3.1 Introduction

Pixy is an open source PHP static detection tool developed by Java. It is very effective in detecting vulnerabilities through data flow analysis. But Pixy only supports PHP4 syntax, which does not support for PHP5 and later syntax features. The Pixy is mainly aimed at XSS vulnerabilities, and in contrast is not ideal for other vulnerabilities.[9]

3.3.2 Workflow

Pixy has two components, front-end and analysis back-end. In order to conduct static analysis, the input program has to be parsed and transformed into a form that makes the following analysis as easy as possible. These are implemented in the front-end, which is responsible for compiling the PHP input into a form suitable for the following data flow analysis. This first step includes the linearization of arbitrarily deep expressions in the original language as well as the reduction of various loop and branch constructs. The resulting intermediate representation, P-Tac, resembles the classical three-address code (TAC) presented in.

To improve the correctness and precision of taint analysis, Pixy conducted a supplementary alias analysis as well as literal analysis. All the analyses are interprocedural, context-sensitive and flow-sensitive for providing a high degree of precision and keeping the number of false positives low.

4. TAINT EXTENSION

Based on the taint framework, it is convenient to develop highly customized PHP taint analysis tools. In this part of the article, we will show how to use the taint framework to develop a PHP static analyzer that supports basic XSS and SQL vulnerability detection.

4.1 Framework for Taint Analysis: Taint

First, an example of the source code input to the taint framework is shown below,

```php
<?php
$a = "tainted string";
taint($a);
$b = "some string";
eval('$b = $a;');
die($b);
?>
```

Then the taint framework is used to analyze this program, and the output is as follows,

```
Warning: main() [exit]: Attempt to output a string
that might be tainted on line 6
```

Note that the taint source needs to be marked with `tainted()` in the input to the taint framework. Based on this framework, we can customize our own static analyzer.

4.2 Defense Based on Taint Framework

The core processing flow of taint framework is shown in Figure 8. It can perform basic taint propagation analysis on variables marked `taint()` as taint sources. This labelled source code can be considered an intermediate representation. This makes it convenient for us to customize the static analyzer.

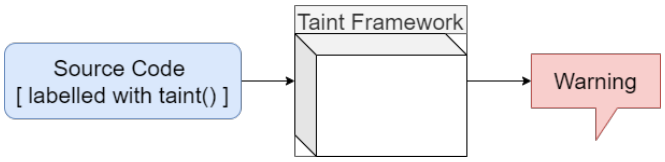

Figure 5: Taint Framework

The main idea is very simple: customize a transpiler, convert the source code into an intermediate representation (labelled source code), and submit it to the taint framework for analysis. The whole process is shown in Figure 6.

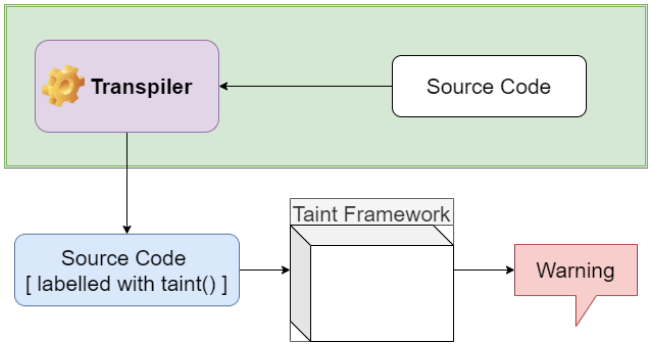

Figure 6: Taint Framework with Transpiler

The specific implementation algorithm of the transpiler has the following two steps:

1) Identify the Taint Source: This step can be customized and expanded, which is also one of the advantages of development based on taint framework. For the simple example here, regular expressions are used to match all variable assignments based on $_GET calls.

2) Replace the matched taint source: Mark the matched taint source through the `taint()` function. If multiple taint sources are matched, multiple IR files are generated respectively.

For example, for this PHP program (contains 2 XSS vulnerabilities),

```php
<?php
function test() {
        global $pass;
        $id = isset($_GET['name']);
        $pass = isset($_GET['password']);
        $b = $id;
        echo $b;
}
test();
echo $pass;
?>
```

Through the first step, the regular expression following will match lines 4 and 5,

```
/\$.*=(.)*isset(.)*\((.)*\$_GET.*;/g
```

Then the transpiler label them, and the resulting intermediate representation is as follows,

```php
<?php
function test() {
    global $pass;
    $id = 'tainted';
    taint($id);
    $pass = 'tainted';
    taint($pass);
    $b = $id;
    echo $b;
}
test();
echo $pass;
?>
```

Give the intermediate representation code to the taint framework for processing, and the output is as follows,

```
Warning: test() [echo]: Attempt to echo a string
that might be tainted on line 9
Warning: main() [echo]: Attempt to echo a string
that might be tainted on line 12
```

Finally, the results can be reconstructed into the source code,

```
<?php
function test() {
        global $pass;
        $id = isset($_GET['name']);
        $pass = isset($_GET['password']);
        $b = $id;
        echo $b;
}
test();
echo $pass;
?>
```

4.3 Graphical User Interface for Taint

In order to make it easier for users to test files, we built a web page using apache2 and php. The web framework is shown in Figure 7.

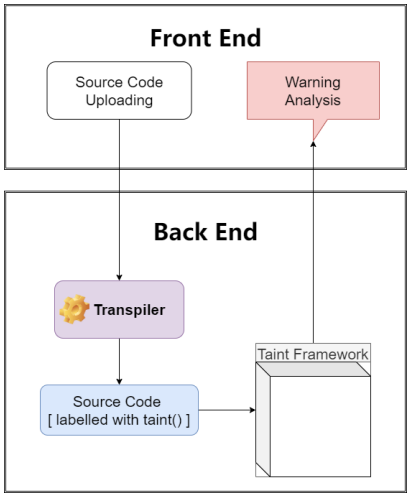

Figure 7: GUI Framework

After setting up the server, users do not need to configure the local environment, nor manually generate taint marked files. Users can upload files at any time by simply visiting the web page. On the server side, when it receives the uploaded file, it uses the transpiler to generate a file marked by taint, and then we use taint to analyze it and return the analysis result to the web page.

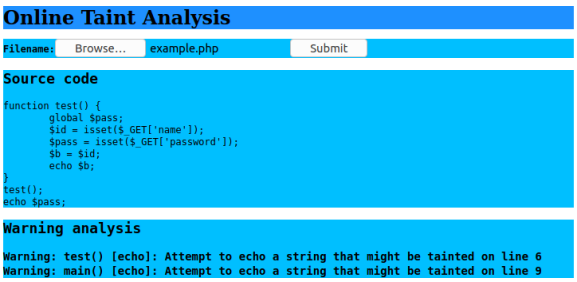

Figure 8: GUI Design

5. ANALYSIS OF TOOL

5.1 Experiment

Types of Vulnerabilities	Level of Difficulty For Detecting	If detect successfully			
		RIPS	phpSafe	Pixy	Fortify
xss_d	low	FALSE	FALSE	*	FALSE
	medium	TRUE	FALSE	*	FALSE
	high	TRUE	FALSE	*	FALSE
xss_r	low	TRUE	FALSE	FALSE	FALSE
	medium	TRUE	FALSE	FALSE	FALSE
	high	TRUE	FALSE	FALSE	FALSE
xss_s	low	TRUE	TRUE	FALSE	FALSE
	medium	TRUE	TRUE	FALSE	FALSE
	high	TRUE	TRUE	FALSE	FALSE
sqli	low	TRUE	TRUE	TRUE	TRUE
	medium	TRUE	TRUE	TRUE	FALSE
	high	TRUE	FALSE	FALSE	FALSE
sqli_blind	low	TRUE	TRUE	TRUE	TRUE
	medium	TRUE	TRUE	TRUE	FALSE
	high	TRUE	TRUE	FALSE	TRUE

Types of Vulnerabilities	Level of Difficulty For Detecting	Number of False Positive		
		RIPS	phpSafe	Pixy
xss_d	low	0	0	0
	medium	1	0	0
	high	1	0	0
xss_r	low	0	0	0
	medium	0	0	0
	high	1	0	0
xss_s	low	1	0	0
	medium	1	0	0
	high	2	0	0
sqli	low	1	0	0
	medium	3	0	0
	high	1	0	0
sqli_blind	low	1	0	0
	medium	0	0	0
	high	2	0	0

In order to compare the performances of all the tools above, we conducted an experiment on them using 15 sample programs from DVWA. DVWA (Damn Vulnerable Web Application) is a PHP/MySQL web application, which contains sample programs containing a wide range of vulnerabilities for security research. [1][2] In our experiment, we used five test sets from DVWA: xss_d, xss_r, xss_s, sqli and sqli_blind, which stands for DOM Based XSS, reflected XSS, stored XSS, SQL injection and Blind SQL injection respectfully. In each test set, there are three levels of vulnerabilities: Low, Medium and High, which refers to the difficulty of detecting the vulnerability contained in the program. In Table 1, we display the results if the tool detected the vulnerability contained in the test program. In Table 2, we display the number of false positive results these tools generated in each test.

5.2 Data Analysis

From the data we can notice that the success rate of RIPS (93.33%) is significantly higher than all the other tools (53.33% for phpSafe, 33.33% for pixy, 20% for Fortify). In contrast, Pixy and phpSafe had very bad performances on XSS related vulnerability tests, which reveals their inner defects on analysis of XSS vulnerabilities. And Fortify only succeeds in three out of fifteen test. However, RIPS also showed some problems during the tests. For 73.33% of the tests, RIPS reported false positive results and the average number of false positive results is 1 per test. While other tools did not report false positive cases. This indicates that RIPS may sacrifice its accuracy for sensitivity, which makes the review of the test results necessary.

5.3 Theoretical Explanation

5.3.1 *phpSafe&Pixy&Fortify*

*1)*As we can see in the table, for SQLi vulnerabilities, the result of phpSafe are good, only have one false on high difficult SQLi vulnerabilities, but for DOM Based XSS and Reflected XSS, phpSafe can not find even one vulnerabilities, that's because phpSafe does not have local files to detect this two types of XSS vulnerabilities, so phpSafe does not support detection of DOM Based XSS and Reflected XSS vulnerabilities.
*2)*Pixy does not support object-oriented features of PHP. Each use of object member variables and methods is treated in an optimistic way, meaning that malicious data can never arise from such constructs. In addition, files included with "include" and similar keywords are not scanned automatically.
*3)*Fortify is a commercial software which charges quite a lot of money to buy related extensions to enhance the performance of the software. In this test, we use the basic version of Fortify without any additional purchase.

5.3.2 *RIPS*

The reason for those false positive results from RIPS is that RIPS is a path-insensitive static code analyzer. [8] This feature results in the ignorance of conditional filters inside the function and the internal protection schemes inside the database (such as whitelist or blacklist). In the implementation, RIPS simply integrates all possible values of into function summary without considering the condition of taking each value. As a result, RIPS cannot take into account the existing protecting scheme inside the code and may therefore cause false positive results.

```
1  <?php
2  function not_harmful($string, $protected = false){
3  if($protected == true) {
4      return htmlspecialchars($string);
5  }else{
6      return "Unprotected";
7  }
8  ?>
```

For example, the code fragment above actually uses the second argument to filter the first one and thus should not be considered harmful. However, RIPS put both return values into the function summary and is then evaluated by the algorithm without considering the dependency of the value of first argument with the second one, which causes a report of XSS vulnerability.

6. ACKNOWLEDGMENTS

We would like to extend our heartfelt gratitude to Professor Hugh Anderson and Teaching Assistant Harish Venkatesan for their patience, encouragement and guidance during the project time. We are also very grateful to the NUS School of Computing for the organization and preparation of this workshop. Finally, we want to thank all of the developers for their pioneering work in RIPS, pixy, phpSafe, which is the founding stone of this paper.

7. REFERENCES

[1] W3Techs. Usage of Server-side Programming Languages for Websites. http://w3techs.com/technologies/overview/programming language/all, as of December 2013.
[2] MITRE. Common Vulnerabilities and Exposures (CVE). http://cve.mitre.org/, as of July 2013.
[3] N. Nostro, A. Ceccarelli, A. Bondavalli, and F. Brancati, "Insider threat assessment: A model-based methodology," *SIGOPS Oper. Syst. Rev.*, vol. 48, no. 2, pp. 3–12, Dec. 2014.
[4] S. Neuhaus, T. Zimmermann, "Security Trend Analysis with CVE Topic Models", *International Symposium on Software Reliability Engineering*, pp. 111-120, 2010
[5] P. Nunes, J. Fonseca, "phpSAFE: A Security Analysis Tool for OOP Web Application Plugins", *45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2015
[6] "RIPS - Wikipedia", En.wikipedia.org, 2021. [Online].*Available: https://en.wikipedia.org/wiki/RIPS. [Accessed: 27- Jul- 2021].*
[7] "SonarSource acquires RIPS Technologies", Blog.sonarsource.com, 2021. [Online].*Available: https://blog.sonarsource.com/sonarsource-acquires-rips-technologies. [Accessed: 27- Jul- 2021].*
[8] Dahse, Johannes, and Thorsten Holz. "Simulation of Built-in PHP Features for Precise Static Code Analysis." *NDSS*. Vol. 14. 2014.
[9] Jovanovic, Nenad, Christopher Kruegel, and Engin Kirda. "Pixy: A Static Analysis Tool for Detecting Web Application Vulnerabilities (Technical Report)." *Secure Systems Lab, Vienna University of Technology* (2006).