Detecting Security Vulnerabilities in Object-Oriented PHP Programs

Mona Nashaat
University of Alberta
nashaata@ualberta.ca

Karim Ali
University of Alberta
karim.ali@ualberta.ca

James Miller
University of Alberta
jimm@ualberta.ca

***Abstract*—PHP is one of the most popular web development tools in use today. A major concern though is the improper and insecure uses of the language by application developers, motivating the development of various static analyses that detect security vulnerabilities in PHP programs. However, many of these approaches do not handle recent, important PHP features such as object orientation, which greatly limits the use of such approaches in practice. In this paper, we present OOPIXY, a security analysis tool that extends the PHP security analyzer PIXY to support reasoning about object-oriented features in PHP applications. Our empirical evaluation shows that OOPIXY detects 88% of security vulnerabilities found in micro benchmarks. When used on real-world PHP applications, OOPIXY detects security vulnerabilities that could not be detected using state-of-the-art tools, retaining a high level of precision. We have contacted the maintainers of those applications, and two applications' development teams verified the correctness of our findings. They are currently working on fixing the bugs that lead to those vulnerabilities.**

I. INTRODUCTION

Static analysis is often used to detect security vulnerabilities in programs, because it enables a security analyst to reason about the program without executing it [1]. Security review tools usually scan the whole program to report the security vulnerabilities found in the code [2]. For PHP, many tools, such as Pixy [3], RIPS [4], PHPSAFE [5] and Weverca [6], have been developed to detect such vulnerabilities. However, certain features of PHP, such as `include` calls, dynamic arrays, and dynamic object-oriented programming (OOP) features, represent major challenges for these approaches. In particular, it is challenging for these tools to reason about the semantics of OOP features in PHP due to its object model. In PHP, object properties do not necessarily have to be declared before accessing them. Therefore, most PHP security tools either do not support (OOP) features in PHP, or partially provide some support at the cost of sacrificing precision and soundness. This limited support cripples the ability of such tools to detect vulnerabilities in a large class of PHP applications, going back to PHP 5 when OOP features were first introduced.

Figure 1 shows a simple PHP script that defines a class `Object` (Line 2) that declares the variable `$value` (Line 4). The class overrides the `toString` method and returns `$value` (Line 7). The script instantiates an object of that class (Line 11), and assigns the URL parameter `'text'` to `$obj->value` (Line 12). Finally, the script echoes `$obj` to the browser (Line 13), which eventually calls `toString` to print the value of `$value`. This last line of the script represents a cross-site scripting (XSS) vulnerability, because it prints out an *unsanitized* value to the browser. Most of the popular PHP analysis tools cannot detect this vulnerability. In particular, the example script uses object references that are not supported by RIPS [4]. Additionally, analyzing the script requires thorough class modelling, which is beyond the support for arrays and object manipulation in Weverca [6]. An analyzer that detects vulnerabilities similar to the one listed in Figure 1 should model the class `Object` properly and track all its methods to capture the propagation of private information throughout the program.

```
1  <?php
2  class Object
3  {
4      var $value;
5      function toString()
6      {
7          return $this->value;
8      }
9  }
10
11 $obj        = new Object();
12 $obj->value = $_GET['text'];
13 echo $obj;
14 ?>
```

Fig. 1. A running example that illustrates OOP features in PHP.

In this paper, we present OOPIXY, a static analysis tool that detects security vulnerabilities in PHP programs that use OOP features. OOPIXY uses interprocedural data-flow analysis [7] to track values of variables and analyze dynamic data structures such as objects and arrays. OOPIXY reports various types of security vulnerabilities, including cross-site scripting (XSS) [8], SQL injection [9], remote code execution, remote command execution, and XPath injection [10]. OOPIXY extends the open-source static code scanner Pixy [3] by providing OOP support. This support enables OOPIXY to analyze more recent PHP programs (PHP 5 and onward), while the original PIXY analyzers only supports PHP 4. Moreover, OOPIXY can detect a wider range of vulnerabilities compared to PIXY that can only detect XSS and SQL injection vulnerabilities. To evaluate OOPIXY, we conducted two experiments, one using micro benchmarks, and the other is a case study of analyzing real-world open-source PHP applications. Our

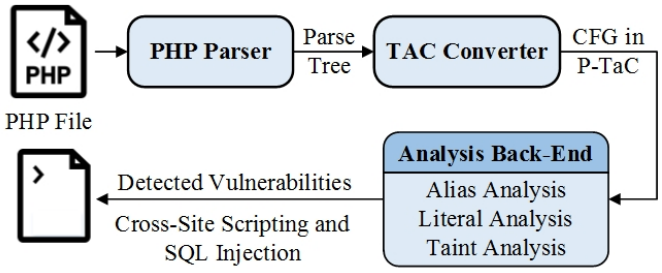

Fig. 2. The original architecture of PIXY.

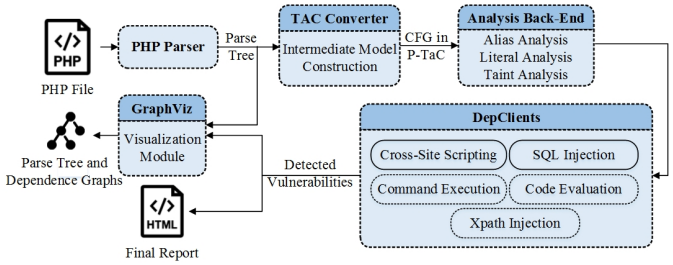

Fig. 3. The architecture of OOPIXY. The components with dashed lines represent modifications or additions to the original architecture of PIXY.

implementation also provides a convenient visualization of the detected vulnerabilities.

II. BACKGROUND

A. Overview of PIXY

PIXY is an open source static analysis tool for PHP programs. The tool is implemented in Java, and scans PHP 4 programs for XSS and SQL injection attacks. Figure 2 shows the original architecture of PIXY. First, the tool parses the input program into an intermediate representation called P-Tac, which is similar to the classical three-address code (TAC) [11]. PIXY then converts the parse tree into a control-flow graph (CFG) using a module called TAC Converter. Afterwards, PIXY runs an alias analysis to collect alias information for variables, followed by a literal analysis that uses the collected alias information to track values of variables throughout the program. Finally, PIXY runs a taint analysis to determine tainted variables (i.e., hold private information). Jovanovic et al. [3] present the first version of PIXY that could only detect XSS vulnerabilities in simple PHP programs. The same authors published a later paper that describes an extension of PIXY that models PHP aliasing [7]. PIXY has also received several other upgrades throughout the years, such as support for detecting SQL injection attacks.

B. Data-Flow Analysis

Data-flow analysis algorithms follow the propagation of data throughout a program by traversing the CFG and marking where data values are generated and where they are used [3]. This information is used in security review tools to determine if private data leaks outside the program without applying proper sanitization. The analysis uses a *lattice* to represent the type of collected information, such that any information associated with a CFG node is an element of the lattice. Each CFG node is also associated with a *transfer function* that takes a lattice element as input and returns a lattice element as output. Each transfer function models the semantics of its corresponding CFG node with respect to the collected information. The analysis applies the transfer functions to propagate the information through the program, and combines information at merge points (e.g., after `if` conditions).

Data-flow analyses have different trade-offs between precision and scalability. A flow-sensitive analysis considers the ordering of program instructions, whereas a flow-insensitive analysis treats a program as a set of unordered instructions. Inter-procedural analyses handle function calls, while intra-procedural analyses operate within a single function. Context-sensitive analyses distinguish between different call sites to a function, unlike context-insensitive analyses that merge the information computed for different calls to the same function. Hence, the highest precision can be achieved by performing an analysis that is flow-sensitive, inter-procedural, and context sensitive [3]. However, inter-procedural analysis sometimes sacrifices precision to achieve scalability (e.g., handling recursive function calls).

III. IMPLEMENTATION OVERVIEW

We have designed OOPIXY to support the dynamic OOP features in PHP 5 applications by adding new modules and modifying existing ones in PIXY. OOPIXY is available on Github[1], we committed the changes to a forked repo of the official PIXY repository. We have issued a pull request to contribute back our changes to the community. Figure 3 shows the architecture of OOPIXY where the components with dashed lines represent modules that we added or modified in the original architecture of PIXY. These changes include adding the ability to scan newer versions of PHP, supporting OOP features in PHP, detecting more types of security vulnerabilities, and visualization for the reporting system.

Similar to PIXY, OOPIXY transforms the input program into a parse tree, then into the intermediate language P-Tac [11]. OOPIXY then performs various static analyses to enrich the intermediate model with information regarding aliases and literal values. Finally, OOPIXY invokes DepClients objects to detect the security vulnerabilities in the generated CFG and generate the final report.

A. PHP Parser

To analyze newer versions of PHP, we replaced the PHP parser module in PIXY with a new parser that supports PHP 5. We generated the new parser in OOPIXY using JFlex, a lexical analyzer generator for Java [12] and a modified version of the Constructor of Useful Parser (CUP) v0.10 tool [13]. To generate the new parser, we also modified the JFlex lexical specification file to include the keywords and features introduced in PHP 5 and later such as the new OOP keywords

[1] https://github.com/uasys/oopixy

that specify access modifiers for methods and properties, interfaces and namespaces, and exception handling features. We use the generated lexer class and the parser specification file to generate the new PHP parser using CUP. OOPIXY uses the new PHP parser to generate the parse tree for the input program.

B. TAC Conversion

In addition to generating a new PHP parser for OOPIXY, we have also modified the intermediate model construction in PIXY to accommodate OOP features. This module performs two main tasks. First, it examines the constructed parse tree and transforms it into the corresponding CFG. Since OOPIXY has a new parser, we changed this module such that it can process the new parse trees with PHP 5 support. Second, the module tracks definitions of custom classes, object references, user defined methods and variables, namespaces and interfaces. This feature enables OOPIXY to resolve each custom object to its class definition during the analysis phases. Therefore, OOPIXY can detect vulnerabilities hidden in user defined objects and custom methods, such as the vulnerability introduced in Figure 1. This also helps OOPIXY detect coding bugs like duplicated variables or methods definitions.

C. New Vulnerability Detectors

PIXY can only detect SQL injection and XSS vulnerabilities. OOPIXY extends the original analysis phases by implementing new `DepClient` classes that detect command execution, code evaluation, and XPath injection vulnerabilities. Each new `DepClient` defines the related sinks, sources, and sanitization routines, and overrides the appropriate `detectVulns` method. This method traverses the constructed CFG to determine, at each program point, whether it may hold a tainted value or not, and report any detected vulnerabilities.

D. Results Visualization

OOPIXY forwards the detected vulnerabilities to its report-generation module, which creates a vulnerability record for each sensitive sink that may receive tainted data at runtime. The record includes the file name that contains the vulnerability, the line number, and the type of the detected vulnerability. All these records are exported in the final report. We have also enhanced the reporting module with a new visualization module that can create dependence graphs [14] for the tainted variables. Dependence graphs include all the files and functions names by which a tainted variable passed until it reached the sink, which helps the user understand the source of a set of related vulnerabilities. Our new visualization module can also visualize the dependence graphs using Graphviz [15]. This new visualization helps the user (e.g., a security analyst or a developer) better understand the reported results by presenting the complex analysis relationships in a graphical form.

Figure 4 shows the dependence graph produced by OOPIXY for the XSS vulnerability reported in Figure 1. The dependence graph illustrates that a tainted variable named `$obj` reaches the sink at Line 13. The graph traces back the origin of the taint to Line 12 where the URL parameter `text` is assigned to the variable `$obj->value`.

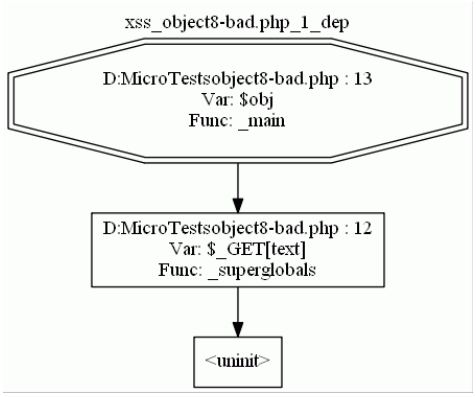

Fig. 4. The dependence graph that OOPIXY generates for the XSS Vulnerability depicted in Figure 1.

IV. EVALUATION

A. Setup

To evaluate OOPIXY, we compare it to three state-of-the-art PHP static analysis tools PIXY [3], RIPS [4] and Weverca [6] with respect to precision and recall. We have conducted two sets of experiments using (1) micro benchmarks and (2) a case study on real-world open-source PHP applications. We compare OOPIXY to PIXY to evaluate the improved capabilities that our modules add to PIXY. Since RIPS is one of the few PHP static analysis tools that support PHP 5, comparing OOPIXY to it provides important insights regarding support for recent PHP 5 features. However, the most recent generation of RIPS is commercial. We have requested an academic license to conduct our experiments, but our request was denied. Therefore, our evaluation uses the last open-source version 0.55. Finally, comparing to Weverca evaluates the ability of OOPIXY to resolve some of the dynamic features of PHP such as OOP.

Throughout our evaluation, we will use the following definitions of precision (P), recall (R), and $F_{measure}$ (F) [16]. Precision is the ratio of the number of true positives (TP) over the number of reported errors, which includes the reported true positives and false positives (FP).

$$P = \frac{TP}{TP + FP} \tag{1}$$

Recall is the ratio of the number of true positives (TP) over the number of actual errors, which is the sum of reported true positives, and false negatives that were not detected (FN).

$$R = \frac{TP}{TP + FN} \tag{2}$$

$F_{measure}$ provides an aggregate measure for precision and recall. For a given tool, the $F_{measure}$ ranges between 0 and 1. We will use P, R, and $F_{measure}$ to provide a ranking of the accuracy of the tools under study.

$$F_{measure} = \frac{(2 * P * R)}{P + R} \tag{3}$$

B. Micro Benchmarks

Our micro benchmarks consist of 110 PHP files that define 55 different test cases that were developed by de Poel et al. [17] to evaluate a set of open source and commercialized PHP analyzers. The test cases are divided into three categories: Vulnerability Detection, Language Support, and Sanitization Routine Support. Table I illustrates the categories of micro benchmarks, the subjects in each category, the number of tests in each subject, and the results of our experiments. Each test consists of a program that exhibits a security vulnerability and a fixed program that resolves the vulnerability. Since we evaluate both true positives and false positives, for each program, we run two tests, true positive test (TPT) and false positive test (FPT). A given tool passes a TPT if it detects the vulnerability in the vulnerable file, and it passes FPT when it does not issue a warning for the file that has the fix.

The results in Table I show that OOPIXY detects all types of vulnerabilities in the micro benchmarks except for Argument Injection, which is currently not supported in the implementation of OOPIXY. The original tool, PIXY, can only detect Server-Side Include and Cross-Site Scripting, and a subset of the SQL Injection vulnerabilities. Moreover, RIPS does not detect one test case for each of Argument Injection, SQL Injection, XPath Injection, and Cross-Site Scripting. Unlike OOPIXY, Weverca can only detect Cross-Site Scripting, Server-Side Include and SQL Injection vulnerabilities. For the OOP features, OOPIXY detects vulnerabilities in Object Model files, including vulnerabilities in custom objects and classes. For that subject, OOPIXY passes 7 out of 8 tests, including the test case that Figure 1 illustrates. On the other hand, both RIPS and PIXY do not pass any of these tests, while Weverca only passes 5 tests. As for sanitization support, OOPIXY detects *weak sanitization* routines when they are used instead of *strong sanitization* routines. OOPIXY fails in only one test in this category. For this test, the `htmlspecialchars` sanitization routine is used to sanitize a parameter of an SQL query. To pass the test, an analysis tool should report a vulnerability, because `mysql_real_escape_string` should be called in this case to sanitize the parameter instead of `htmlspecialchars`. However, the current implementation of OOPIXY mistakenly considers `htmlspecialchars` as a strong sanitization method, which causes OOPIXY to skip reporting a vulnerability for this test. On the other hand, RIPS passes this test successfully, while both Weverca and PIXY do not report the vulnerability at all.

Across all the micro benchmarks, OOPIXY has better precision (0.75) than PIXY (0.64) and Weverca (0.63), but falls slightly short when compared to RIPS (0.80). However, OOPIXY has better recall (0.88) than any of the state-of-the-art tools (PIXY: 0.82, RIPS: 0.77, and Weverca: 0.76). In general, OOPIXY has a better combined precision and recall (F: 0.80) than the state-of-the-art tools (PIXY: 0.70, RIPS: 0.76, and Weverca: 0.66).

C. Case Studies

To evaluate the viability of OOPIXY in real-world settings, we have conducted case studies that involve a set of five open-source large PHP web applications that are available on GitHub. Table II shows the complete list of the applications in our study. For each application, the table shows its name, the application version that is used in the experiments, and the lines of code calculated using the CLOC package [18]. The first four applications in the case studies tests are the content-management systems `Xoops` [20], `phpnuke` [21], `b2evolution` [22], and `concrete5` [19]. The fifth application is the open-source community-edition for the ecommerce platform `Magento` [23]. For each application, we have manually reviewed the security report that each tool generates to identify false positives. Table II shows the running time, the number of vulnerabilities detected by each tool, and the corresponding precision value.

PIXY throws parsing errors for all applications, because they all use syntax of PHP 5 or higher, which the tool does not support. Additionally, Weverca crashes when it analyzes `Xoops` and does not report any warning for `phpnuke` and `Magento`. Across all the programs in our case study, OOPIXY is twice as precise as RIPS on average (RIPS: 0.36, OOPIXY: 0.632). Further investigation shows that, for `phpnuke`, most of the false positives that RIPS report are within the installation files, because RIPS marks most of the installation parameters, such as database name, database user name, and database password, as tainted variables. However, these parameters are sanitized using the user-defined function `mosGetParam()`. For `Xoops`, RIPS reports false positives about variables that are sanitized using the user-defined functions `xformfield()` and `xoFormFieldCollation()`. On the other hand, for `concrete5`, OOPIXY is less precise than RIPS, because it mistakenly considers the `h()` function as a bad sanitization routine.

Table II also shows the overall running time for each tool, which is the time it took the tool to completely analyze the application and generate the final report. On average, OOPIXY is 1.76× faster than RIPS (min: 0.75×, max: 4.9×, geometric mean: 1.76×). However, we observed that, for `Magento` and `phpnuke`, OOPIXY is slower than RIPS, where it spends most of the running time in parsing and resolving file inclusions. At first glance, Weverca seems to have the best performance, because it has, by far, the lowest running times for all applications in our case study. Further investigation shows that Weverca takes such a short time to finish the analysis because it does not properly analyze the given programs and, therefore, does not detect any of the security vulnerabilities that the may exhibit.

To further validate the findings of our case studies, we have sent a set of the reported true positives to the development team of each application. For `Xoops`, we reported a XSS vulnerability in `register.php` that allows unsanitized variables to be echoed to the browser. The development team has responded to our report and they are currently working on a

TABLE I
VULNERABILITY DETECTION IN MICRO BENCHMARKS.

			OOPIXY					PIXY					RIPS					Weverca				
Category	**Subject**	**# Tests**	TP	FP	P	R	F	TP	FP	P	R	F	TP	FP	P	R	F	TP	FP	P	R	F
Vulnerability Detection	Argument Injection	1	0	1	-	0.00	-	0	1	-	0.00	-	0	1	-	0.00	-	0	1	-	0.00	-
	Command Injection	2	2	2	1.00	1.00	1.00	0	2	-	0.00	-	2	2	1.00	1.00	1.00	0	0	0.00	0.00	-
	Code Injection	2	2	2	1.00	1.00	1.00	0	2	-	0.00	-	2	1	0.67	1.00	0.80	0	2	-	0.00	-
	SQL Injection	6	6	4	0.75	1.00	0.86	2	6	1.00	0.33	0.50	5	6	1.00	0.83	0.91	5	4	0.71	0.83	0.77
	Server-Side Include	2	2	2	1.00	1.00	1.00	2	1	0.67	1.00	0.80	2	1	0.67	1.00	0.80	2	0	0.50	1.00	0.67
	XPath Injection	2	2	2	1.00	1.00	1.00	0	2	-	0.00	-	1	2	1.00	0.50	0.67	0	0	0.00	0.00	-
	Cross-Site Scripting	3	3	3	1.00	1.00	1.00	3	3	1.00	1.00	1.00	2	3	1.00	0.67	0.80	3	3	1.00	1.00	1.00
Language Support	Aliasing	4	4	4	1.00	1.00	1.00	4	0	0.50	1.00	0.67	3	1	0.50	0.75	0.60	4	1	0.57	1.00	0.73
	Arrays	2	2	0	0.50	1.00	0.67	2	0	0.50	1.00	0.67	2	2	1.00	1.00	1.00	1	0	0.33	0.50	0.40
	Constants	2	1	1	0.50	0.50	0.50	2	1	0.67	1.00	0.80	1	2	1.00	0.50	0.67	0	2	-	0.00	-
	Functions	5	5	1	0.56	1.00	0.71	5	3	0.71	1.00	0.83	2	5	1.00	0.40	0.57	5	3	0.71	1.00	0.83
	Dynamic Inclusion	3	1	1	0.33	0.33	0.33	1	1	0.33	0.33	0.33	0	3	-	0.00	-	1	3	1.00	0.33	0.50
	Object Model	8	7	5	0.70	0.88	0.78	0	7	0.0	0.0	-	0	8	-	0.00	-	5	5	0.63	0.63	0.63
	Strings	3	3	3	1.00	1.00	1.00	3	3	1.00	1.00	1.00	3	3	1.00	1.00	1.00	3	1	0.60	1.00	0.75
	Variables	3	2	1	0.50	0.67	0.57	2	1	0.50	0.67	0.57	1	2	0.50	0.33	0.40	2	1	0.50	0.67	0.57
Sanitization Support	Regular Expressions	2	2	1	0.67	1.00	0.80	2	0	0.50	1.00	0.67	2	0	0.50	1.00	0.67	2	0	0.50	1.00	0.67
	SQL Injection	1	1	1	1.00	1.00	1.00	0	1	-	0.00	-	1	1	1.00	1.00	1.00	0	1	-	0.00	-
	Strings	2	2	1	0.67	1.00	0.80	2	0	0.50	1.00	0.67	2	0	0.50	1.00	0.67	2	0	0.50	1.00	0.67
	Cross-Site Scripting	2	2	2	1.00	1.00	1.00	2	2	1.00	1.00	1.00	2	2	1.00	1.00	1.00	1	2	1.00	0.50	0.67
	Geometric Mean		-	-	0.75	0.88	0.80	-	-	0.64	0.82	0.70	-	-	0.80	0.77	0.76	-	-	0.63	0.76	0.66

TABLE II
VULNERABILITY DETECTION IN CASE STUDIES.

			OOPIXY				PIXY				RIPS				Weverca			
Application	**Version**	**LOC**	Time (s)	TP	FP	P	Time (s)	TP	FP	P	Time (s)	TP	FP	P	Time (s)	TP	FP	P
`Xoops`	2.5.8.1	111,456	31.00	24	14	0.63	Parsing Error				152.61	8	39	0.17	Crash			
`phpnuke`	8.3.1	195,120	312.00	33	12	0.73	Parsing Error				278.24	33	49	0.40	0.05	0	0	-
`b2evolution`	6.8.8	294,992	287.00	12	8	0.60	Parsing Error				329.53	5	8	0.38	4.01	0	0	-
`concrete5`	8.1.0	437,042	142.00	31	27	0.53	Parsing Error				151.99	17	9	0.65	0.07	0	3	0.00
`Magento`	2.1.5	1,728,396	685.00	2	1	0.67	Parsing Error				511.75	2	8	0.20	0.01	0	0	-

fix. For `b2evolution`, OOPIXY detects a XSS vulnerability in the file `inc\tools\mtimport.ctrl.php` that prints posts to the page. The maintainers of `b2evolution` have validated the vulnerability that we reported to them. However, the developers believe that the issue does not represent a huge problem, because it is on the administrative side of the application, which is typically not exposed to the regular user.

V. RELATED WORK

There exist a considerable number of security assessment tools for PHP. For example, PhpSAFE [5] is a static analysis tool that analyzes plugins developed for PHP-based content management systems (CMS). However, PhpSAFE can only detect Cross-Site Scripting (XSS) and SQL Injection vulnerabilities. The tool is also highly customized to detect vulnerabilities in CMS plugins, focusing on OOP features while ignoring other challenges such as dynamic inclusions.

PIXY [3] is the first open source tool for statically detecting taint-style vulnerabilities in PHP programs. However, PIXY only supports up to PHP 4, and cannot reason about OOP features that were introduced in later versions of the language. Since PIXY does not provide any support for OOP features, it generates many false negatives, rending its analysis results unsound. Moreover, PIXY has high false positive rate, because it cannot resolve various dynamic features of PHP such as dynamic file inclusion.

RIPS [4] is a static source code analyzer written in PHP using the built-in tokenizer functions. RIPS can detect more than twenty types of taint-style vulnerabilities including XSS, SQL injection, code execution, and file inclusion. However, the last open source release of RIPS does not support OOP features in PHP and ignores alias relations between variables.

Weverca [6] is a static analysis framework for PHP. Weverca first parses PHP programs to construct abstract syntax trees that include the intermediate representation, then it performs the analysis. However, Weverca does not scale to large programs, e.g., `mutllidae` [24] and `Xoops` from our case studies. Additionally, Weverca does not categorize the reported vulnerabilities which makes it challenging for code reviewers to understand the reported results.

VI. CONCLUSION

Web applications present a major role in almost all principal services in our daily life. However, vulnerabilities that threaten the personal data of users are discovered frequently. This paper introduces OOPIXY, a tool that detects security vulnerabilities in PHP applications. Unlike the state of the art, OOPIXY can reason about OOP features of PHP, track object references, user-defined methods, and variables. Our empirical evaluation shows that OOPIXY has a better combination of precision and recall compared to other available open-source tools. When analyzing real-world PHP applications, OOPIXY detects vulnerabilities that other tools cannot detect, including some vulnerabilities that have been verified by the development teams behind these applications.

REFERENCES

[1] Dimastrogiovanni, Carlo, and Nuno Laranjeiro. "Towards Understanding the Value of False Positives in Static Code Analysis." Dependable Computing (LADC), 2016 Seventh Latin-American Symposium on. IEEE, 2016.

[2] K. Goseva-Popstojanova and A. Perhinschi, "On the capability of static code analysis to detect security vulnerabilities," *Information and Software Technology*, vol. 68, pp. 18-33, 2015.

[3] N. Jovanovic, C. Kruegel, and E. Kirda, "Pixy: A static analysis tool for detecting web application vulnerabilities," in *IEEE Symposium on Security and Privacy*, 2006, pp. 6 pp.-263.

[4] J. Dahse and T. Holz, "Simulation of Built-in PHP Features for Precise Static Code Analysis," in *NDSS*, 2014.

[5] P. J. C. Nunes, J. Fonseca, and M. Vieira, "phpSAFE: A security analysis tool for OOP web application plugins," in *IEEE/IFIP*, 2015, pp. 299-306.

[6] D. Hauzar and J. Kofro, "WeVerca: Web Applications Verification for PHP," in *International Conference on Software Engineering and Formal Methods*, 2014, pp. 296-301.

[7] N. Jovanovic, C. Kruegel, and E. Kirda. "Precise alias analysis for static detection of web application vulnerabilities," in Proceedings of the 2006 workshop on Programming languages and analysis for security. ACM, 2006.

[8] S. Gupta and B. Gupta, "Cross-Site Scripting (XSS) attacks and defense mechanisms: classification and state-of-the-art," *International Journal of System Assurance Engineering and Management*, pp. 1-19, 2015.

[9] M. K. Gupta, M. Govil, and G. Singh, "Static analysis approaches to detect SQL injection and cross site scripting vulnerabilities in web applications: A survey," in *ICRAIE*, 2014, pp. 1-5.

[10] M. I. P. Salas, P. L. De Geus, and E. Martins, "Security Testing Methodology for Evaluation of Web Services Robustness-Case: XML Injection," in *IEEE World Congress on Services*, 2015, pp. 303-310.

[11] N. Jovanovic, C. Kruegel, and E. Kirda, "Pixy: A Static Analysis Tool for Detecting Web Application Vulnerabilities (Technical Report)," *Secure Systems Lab, Vienna University of Technology*, 2006.

[12] G. Klein. *JFlex - The Fast Scanner Generator for Java*. 2009.

[13] S. E. Hudson, F. Flannery, C. S. Ananian, D. Wang, and A. W. Appel, "Cup parser generator for java," *Princeton University*, 1999.

[14] D. Balzarotti, M. Cova, V. Felmetsger, N. Jovanovic, E. Kirda, C. Kruegel, et al., "Saner: Composing static and dynamic analysis to validate sanitization in web applications," in *IEEE Symposium on Security and Privacy*, 2008, pp. 387-401.

[15] J. Ellson, E. R. Gansner, E. Koutsofios, S. C. North, and G. Woodhull, "Graphviz and dynagraphstatic and dynamic graph drawing tools," in *Graph drawing software*, Springer, 2004, pp. 127-148.

[16] G. Chatzieleftheriou and P. Katsaros, "Test-driving static analysis tools in search of C code vulnerabilities," in *COMPSACW*, 2011, pp. 96-103.

[17] N. L. de Poel, F. B. Brokken, and G. R. R. de Lavalette, "Automated security review of PHP web applications with static code analysis," *Master's thesis*, 2010.

[18] A. Danial, *Cloc count lines of code*. 2009.

[19] 'concrete5', 2017. [Online]. Available: https://www.concrete5.org/download. [Accessed: 23- March- 2017].

[20] 'XOOPS Web Application System', 2016. [Online]. Available: https://sourceforge.net/projects/xoops/. [Accessed: 23- March- 2017].

[21] 'PhpNuke The first PHP CMS'. [Online]. Available: https://www.phpnuke.org/modules.php?name=Release. [Accessed: 23- March - 2017].

[22] b2evolution, 2016. [Online]. Available: http://b2evolution.net/downloads/. [Accessed: 03- April- 2017].

[23] Magento 2017. [Online]. Available: https://magento.com/products/community-edition#interstitial/. [Accessed: 03- April- 2017].

[24] OWASP Mutillidae, Web Pen-Test Practice Application, 2016. [Online]. Available: https://sourceforge.net/ projects/mutillidae/. [Accessed: 23- Oct- 2016].