Aggregating Static and Dynamic Methodologies For PHP Application Security Assessment

Mr. Vishal Vijaykumar Parkar[1], Mr. H. A. Tirmare[2]

[1] *Student, Computer Science & Technology, IDepartment of Technology*
Shivaji University, Kolhapur, Maharashtra, India
[2] *Assistant Professor, Computer Science & Technology, IDepartment of Technology*
Shivaji University, Kolhapur, Maharashtra, India

---***---

Abstract - *In recent years, focus of business world has been moved towards the Internet. Web applications provide a generous interface non-stop thus offering to malicious users a wide spectrum of possible attacks. Consequently, the security of web applications has become a crucial issue.*

The state-of-the-art tools for bug discovery in languages used for web-application development, such as PHP, suffer from a relatively high false-positive rate and low coverage of real errors; this is caused mainly by non-precise modeling of dynamic features of such languages and path-insensitivity of the tools. In this project, we will demonstrate Lacunae of the current tools and implement a novel approach to address these issues. We will show how our technique handles some of the situations where other tools fail and illustrate it on examples.

Key Words: web application security; static security assessment; static security assessment, etc…

1. INTRODUCTION

Recently, as business world has moved its focus towards the Internet, a number of applications have been moved on-line, and this trend is still continuing. Safety and security of the web applications involved in such transactions is therefore of the top priority.

A typical web application is available and operational 24/7, thus not putting any time pressure on malicious users; a generous interface these applications provide further widens the hacker's field. Amongst the 25 most common programming errors, those specific to web applications form a significant part of this group; the examples include improper neutralization of SQL commands, cross-site request forgery, and missing authorization.

The most common programming language used at the server side is PHP. PHP features many special attributes that make it different from common programming languages, especially as far as dynamism is concerned. The examples are inclusion of a file specified by a runtime-computed filename and the eval construct allowing runtime construction of code that is executed afterwards. This makes it hard or sometimes even impossible to apply the same techniques and tools for finding bugs or for correctness verification as in the case of "non-web" programming languages.

2. RELEVANCE / MOTIVATION

Due to increased use of Web applications, their security has become an important issue. The existing tools for bug suffer from a relatively high false-positive rate and low coverage of real errors.

Common server programming language such as PHP have many dynamic features which calls for the combination of static and dynamic bug recovery methods for efficient finding of bugs or for correctness verification for non-web programming languages.

2.2. Problem Statement

In this project, we will develop a software application for the identification of bugs inside web applications caused by data flow of unsanitized inputs from the user to sinks (SQL queries, URL constructions, output in general, etc.) inside web applications written in PHP. We will use the combination of existing techniques which are used for the static and dynamic evaluation of security of Web applications. Particularly, we will compute data flow information using dependence graphs, will identify sources of sensitive data, sinks, and at each program point maintain:

1. the taint and the sanitization status for each variable,
2. the set of possible values of each variable,
3. the set of conditions defined on the program's variables that must hold, and
4. the set of possible types of each variable.

3. LITERATURE REVIEW

[1] Security Analysis of PHP Web Applications: presents a new approach to discovery of bugs inside web applications written in PHP. While being based on known techniques, it is the first which combined them into a single one and improved them to face the most critical issues. It proposes precise modeling of aliasing and taint analysis capable of detecting the most of vulnerabilities.
[2] Finding Bugs in Web Applications: Presents a technique for finding faults in PHP Web applications that is based on combined concrete and symbolic execution. The work is novel in several respects. First, the technique not only detects runtime errors but also uses an HTML validator as an oracle to determine situations where malformed HTML is created. Second, we address a number of PHP-specific issues, such as the simulation of interactive user input that occurs when user-interface elements on generated HTML pages are activated, resulting in the execution of additional PHP scripts. Third, we perform an automated analysis to minimize the size of failure-inducing inputs. The created tool, Apollo, implements the analysis and is evaluated on six open-source PHP web applications.
[3] Slr: Path-sensitive: Presents a technique for detecting semantically infeasible paths in programs using abstract interpretation. This technique uses a sequence of path-insensitive forward and backward runs of an abstract interpreter to infer paths in the control flow graph that cannot be exercised in concrete executions of the program. It then present a syntactic language refinement (SLR) technique that automatically excludes semantically infeasible paths from a program during static analysis. SLR allows to iteratively prove more properties. Specifically, this technique simulates the effect of a path-sensitive analysis by performing syntactic language refinement over an underlying path insensitive static analyzer. Finally, it presents experimental results to quantify the impact of our technique on an abstract interpreter for C programs.
[4] Saner: Composing Static and Dynamic Analysis: presents a novel approach to the analysis of the sanitization process. More precisely, it combines static and dynamic analysis techniques to identify faulty sanitization procedures that can be bypassed by an attacker. The approach has been Implemented in a tool, called Saner, and is applied it to a number of real-world applications. Results demonstrate that it was possible to identify several novel vulnerabilities that stem from erroneous sanitization procedures.
[5] Static analysis of dynamic scripting languages: Has developed a static analysis model for PHP that can deal with dynamic language features such duck-typing, dynamic and weak typing, overloading of simple operations, implicit object and array creation and run-time aliasing. The main focus of our work is alias analysis, but we show how type inference and constant propagation must be used to perform the analysis effectively. We also show how SSA form cannot be used without the presence of a powerful alias analysis.
[6] Common weakness enumeration: The 2011 CWE/SANS Top 25 Most Dangerous Software Errors is a list of the most widespread and critical errors that can lead to serious vulnerabilities in software. The Top 25 list is a tool for education and awareness to help programmers to prevent the kinds of vulnerabilities that plague the software industry, by identifying and avoiding all-too-common mistakes that occur before software is even shipped. Software customers can use the same list to help them to ask for more secure software. Researchers in software security can use the Top 25 to focus on a narrow but important subset of all known security weaknesses. Finally, software managers and CIOs can use the Top 25 list as a measuring stick of progress in their efforts to secure their software.

4. OUTLINE OF PROPOSED WORK

4.1 Scope

The project aims to develop a software application for security assessment of large-scale Web applications. We will use this system for the assessment of vulnerabilities present in large Web applications having many pages for user interactions.

The output will be indicative of any vulnerability present in such applications. Some manual intervention may still be necessary for the complete eradication of the vulnerabilities.

4.1 The Methodology

4.1.1 Methods of data collection:-

The data for the experiments will be collected from Free and Open Source large scale Web applications such as Moodle, Joomla etc.

4.1.2 Probable methods of data analysis:

A. OUTLINE

The main challenge of the analysis is the combination of an arbitrary user input and the dynamism of PHP. To address this problem, our analysis consists of the following steps:

1. Construction of the control-flow graph (CFG).
2. Static analysis of constructed CFG.
3. Detection of vulnerabilities.
4. A path-sensitive validation of vulnerabilities.

B. MODELING OF PHP DATA STRUCTURES

To model variables, array cells, and object fields, we use a points-to graph similar to the one introduced in [5]. The points-to graph contains three types of nodes. A storage node represents a symbol-table, an array, or an object. An index node represents a variable, an array cell, or an object field. Each index node is a child of a single storage node. Finally, a value node represents a scalar value. Index nodes are connected with storage and value nodes that constitute their values using value edges.

C. STATIC ANALYSIS

Our analysis uses a combination of concrete and symbolic execution when propagating literal values through operations. If all inputs of an operation are concrete, the explicit version of the operation is used, otherwise the symbolic version is used. By using concrete operations, we reduce the imprecision; here, we use the reference PHP implementation as in [4] and [1]. As to modeling the symbolic versions, we model arithmetic operations as well as operations with strings. For modeling string operations, we use automata-based approach.

D. DETECTION OF VULNERABILITIES

Table -1 : Potential Vulnerabilities Identified by the Analysis

Data in the sink	**Exploit**
Tainted and match an attack pattern.	SQL injection, XSS.
Tainted and not sanitized.	*Data not escaped:* SQL injection, XSS.
Tainted or can be a null value.	*Data potentially not filtered, can be manipulated by a user:* Sensitive information leakage, semantic URL attack, spoofed form submissions, spoofed HTTP request.
Related to a current session and not guarded.	CSRF attack, session fixation, session hijacking.

E. PATH-SENSITIVE VALIDATION OF VULNERABILITIES

In this phase, for each vulnerability that is uncertain we try to prove the unfeasibility of paths leading to the vulnerability. We identify the program points that contribute to the uncertainty of the vulnerability. These program points correspond to (1) join points of branching statements where some branches do not lead to the vulnerability or causes of the vulnerability are different and to (2) an access to data that cannot be certainly identified and that can lead to the vulnerability

F. EVALUATION

To evaluate our approach, we will analyze a snippet of code from a real web application using the Pixy tool. We will then identify sources of imprecision and show whether they can be handled by our approach by tracking it by hand.

1) Block diagram:

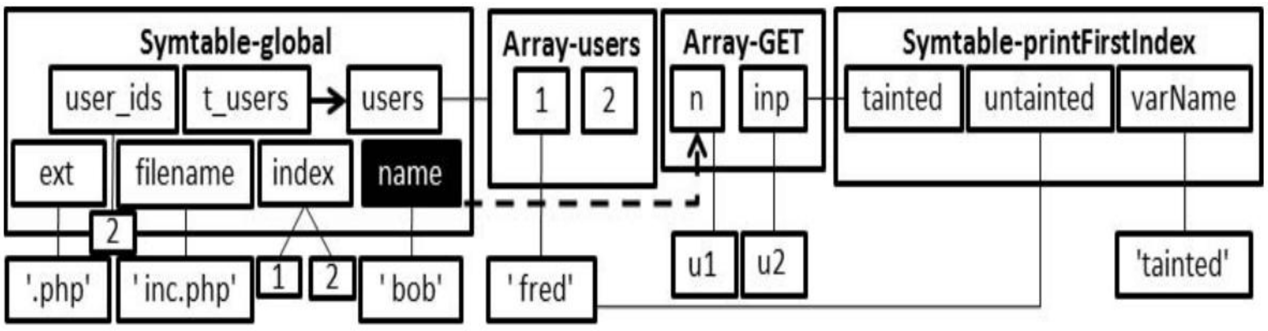

FIG -1: POINTS-TO GRAPH

5. REQUIREMENTS

5.1. Hardware requirements:

Processor	: Pentium or higher
Memory requirements	: 2 GB RAM
Hard disk space	: 80 GB or more

5.2. Software requirements:

Language	: PHP, MySQL
Framework	: Zend
Operating System	: Ubuntu 12.04 (Linux), Windows XP or Above.

5.3. Tools used:

Eclipse
XAMPP

3. CONCLUSIONS

Due to the ubiquitousness and user-friendly approach of Web applications their security has become very important. In this project, we will try to overcome known shortcomings of the current tools and implement a novel approach to address these issues. We will show how our approach better handles some of the situations where other tools perform not as expected or simply fail to work and justify our claim with some proofs of tests done on some real world web applications in PHP.

REFERENCES

[1] Hauzar et al. On Security Analysis of PHP Web Applications. IEEE 36th International Conference, 2012.
[2] Artzi et al. Finding Bugs in Web Applications Using Dynamic Test Generation and Explicit-State Model Checking. IEEE Trans. on Soft. Eng., 36(4), 2010.
[3] G. Balakrishnan, S. Sankaranarayanan, F. Ivancic, O. Wei, and A. Gupta. Slr: Path-sensitive analysis through infeasible-path detection and syntactic language refinement. In Static Analysis, LNCS. Springer, 2008.
[4] D. Balzarotti et al. Saner: Composing Static and Dynamic Analysis to Validate Sanitization in Web Applications. S&P'2008, 2008.
[5] P. Biggar and D. Gregg. Static analysis of dynamic scripting languages, 2009
[6] Common weakness enumeration. http://cwe.mitre.org/top25/

BIOGRAPHIES

Mr. Vishal Vijaykumar Parkar is a student in the master of Computer Science and Technology program at Department of Technology, Shivaji University, Kolhapur. He is interested in domains like web security, web programming, databases.

Mr. H. A. Tirmare is an Assistant Professor in the Computer Science and Technology department, Department of Technology, Shivaji University, Kolhapur.

His domains of interest include but are not limited to computer networks, web security, operating systems, data structures.