

Enhancing Security Audit for Web Applications with Dynamic Modeling Approach

Mr. Vishal Vijaykumar Parkar¹, Mr. H. A. Tirmare²

¹ Student, Computer Science & Technology, Department of Technology
Shivaji University, Kolhapur, Maharashtra, India

² Assistant Professor, Computer Science & Technology, Department of Technology
Shivaji University, Kolhapur, Maharashtra, India

Abstract – *The use of the Internet is dramatically increasing in business day by day. The user-friendly user interfaces offered by web applications also provide a many possibilities of attacks to crackers. Thus, web applications have become a crucial candidate for security analysis.*

Tools currently used for analysis of security of languages used for development of web applications such as PHP, suffer from a relatively high false-positive rate and low true positive rate; which is due to incorrect modeling of dynamic features of such languages and lack of path-sensitivity in the tools. In this project, we compare one of these tools with its version modified to have the dynamic modeling ability. We show how the modified tool handles some of the situations better than the original tool and illustrate it with examples.

Keywords: *web-application security; static and dynamic security-assessment*

1. INTRODUCTION

Today, a large number of web applications have been deployed to implement various business functions and the number is growing rapidly. As the businesses rely more and more on these applications, assessing their security has therefore become really very essential and urgent.

Typically, web applications offer a user interface which is inherently made easy to understand and use and is additionally available and operational ubiquitously round the clock. This has got twofold benefit to malicious users; they can easily understand the interface of the web applications and accordingly plan their attack. Further, the ubiquitous nature of the applications provides the attackers enough scope and time to try various attacks. Programming errors pertaining to web applications constitute a significant part of the 25 most common programming errors e.g. cross-site request forgery, incorrect SQL neutralization, and lack of authorization [6].

Nowadays, PHP happens to be the most popular server-scripting language. PHP has many special features which make it too different from other similar programming languages, mostly due to its dynamic nature. The examples

are inclusion of a file specified by a runtime-computed filename and the eval construct which allows runtime generation of code which is executed afterwards. This not only makes it hard but sometimes even impossible also to apply the same techniques and tools for finding errors or for assessment of correctness not in line with the case of programming languages not related to the web.

2. RELEVANCE / MOTIVATION

Security of web applications has become an important issue because of their increased use. Current tools for bug discovery have a relatively high false-positive rate and low true-positive rate.

Due to many dynamic features, common server programming language like PHP require the static and dynamic bug recovery methods be combined for efficient finding of bugs or for correctness check for non-web programming languages.

2.2. Problem Statement

To develop a PHP web application for the discovery of bugs inside web applications caused by data flow of nonsanitized inputs from the user to sinks like SQL queries, URL building, output, etc. in those applications. We will combine existing static techniques used for evaluation of security of web applications with some dynamic techniques.

3. LITERATURE REVIEW

[1] Security Analysis of PHP web Applications: presents a new approach to finding bugs present in web applications written in PHP. Though it is based on known techniques, it is the first which aggregated them into a single one and improved them to face the most critical issues. It proposes precise alias-modeling and taint analysis which can detect most of vulnerabilities.

[2] Finding Bugs in web Applications: Presents a technique for tracing bugs in web applications written in PHP. The technique is based on combined concrete and symbolic execution. It is unique in several aspects. First, the technique not only finds runtime errors but also uses an HTML validator to determine scenarios where malformed HTML is created. Second, it addresses a number of issues specific to PHP, such as the simulation of interactive user input that occurs when

user-interface elements on generated HTML pages are activated, which results in the execution of additional PHP scripts. Third, it does an automated analysis to minimize the size of inputs which may induce failure. The tool created, Apollo, implements the analysis and is tested on six PHP web applications.

[3] Slr: Path-sensitive: This paper presents an approach for finding paths which are semantically infeasible in programs using abstract interpretation. It uses a series of path-insensitive forward and backward scans of an abstract interpreter to detect paths in the CFG that cannot be implemented in concrete executions of the program. It then presents a technique called syntactic language refinement (SLR) that automatically excludes such paths from a program in the process of static analysis. SLR allows to prove more properties iteratively. Specifically, it simulates a path-sensitive analyzer by performing syntactic language refinement over an underlying path insensitive static analyzer. Finally, it judges the impact of the technique by giving experimental results to quantify on an abstract interpreter for programs in C.

[4] Saner: Composing Static and Dynamic Analysis: This presents a novel technique for analysis of the sanitization. Actually, it combines static and dynamic analysis techniques to identify problematic sanitization procedures that can be escaped by an attacker. This technique is implemented in Saner, a tool developed, and is evaluated on many real-world applications. It can identify several novel vulnerabilities that stem from erroneous sanitization procedures.

[5] Static analysis of dynamic scripting languages: Presents a static analysis model for PHP that can handle dynamic language features e.g. duck-typing, dynamic and weak typing, simple operation overloading, run-time aliasing and implicit object and array creation. The emphasis is on alias analysis, but it shows how constant propagation and type inference must be used to perform the effective analysis. It also presents how SSA form requires the presence of a powerful analysis of alias.

[6] Common weakness enumeration: The 2011 CWE/SANS Top 25 Most Dangerous Software Errors gives a list of the widespread and crucial errors that generate serious vulnerabilities in any software. The list helps programmers to prevent the kinds of vulnerabilities by identifying and avoiding common mistakes that occur in software. Researchers in software security can use this list to focus on a precise but important subset of all known security weaknesses. Finally, software managers and CIOs can use this list to measure the progress in their efforts to secure their software.

[7] Simulation of Built-in PHP Features for Precise Static Code Analysis:

[8] Aggregating Static And Dynamic Approaches For web Application Security Assessment: Surveys contemporary tools having static approaches for web application security assessment. It also gives a basic idea about how the dynamic aspects can be combined with the static techniques to improve the discovery of real bugs in the PHP web applications.

4. OUTLINE OF THE WORK

4.1 Scope

As a part of this project, we have enhanced a previously built web application in PHP for security assessment of large-scale web applications which are also written in PHP. Using this application one can assess the vulnerabilities that may be present in large web applications. The output indicates the vulnerabilities present in such applications, if any. Further manual intervention is necessary for the removal of the vulnerabilities [8].

4.1 The Methodology

4.1.1 Methods of data collection:-

The data for the experiments are collected from large-scale web applications (free and open source) such as HotCRP, a CMS built by us, single web pages etc.

4.1.2 Methods of data analysis:

The main challenge for the analysis is the combination of a random user input and the dynamic nature of PHP. To address this, our analysis consists of the following steps:

1. Control-flow graph (CFG) construction
2. Static and dynamic analysis of constructed CFG
3. Detection of vulnerabilities
4. A context-sensitive validation of vulnerabilities

4.1.2.1 Construction of the control-flow graph (CFG)

The following steps are taken to build CFG:

- 1) We first build an Abstract Syntax Tree (AST) based on PHP's open source internals for each PHP file in the web application to be tested. Next, relevant information like the name and parameters of all user-defined functions are extracted and are stored. Function's body is saved as separate AST and is removed from the main AST of the parsed file.
- 2) We start converting each main AST into a Control Flow Graph (CFG) as follows. A new basic block is built and is connected to the previous basic block with a block edge whenever a node of the AST takes a conditional jump. The jump condition is added to the block edge and the AST nodes following the node are added to the new basic block.
- 3) The data flow of each basic block is simulated as soon as a new basic block is created. The main merit is that the analysis of a basic block is only dependent on previous basic blocks when performing backwards-directed data flow analysis. Next, the analysis results are combined into the so called block summary that is generated during simulation. It summarizes the data flow within a basic block.
- 4) If a call to a previously unknown user-defined function is encountered during simulation, the CFG is built from the function AST and a function summary is created once with intra-procedural analysis. Then, the pre- and post-conditions

for this function can be extracted from the summary and then inter-procedural analysis is performed. At the end, the construction of the main CFG is continued.

4.1.2.2 Static and dynamic analysis of constructed CFG

A taint analysis is then performed beginning from the currently simulated basic block for each vulnerable parameter of a user-defined function or configured sensitive sink.

Taint Analysis: While simulating a basic block, each function call is checked for potential vulnerabilities. Sensitive sinks in the PHP language have been identified which we configured by function name, sensitive parameter, and vulnerability type. For each sensitive sink called, a new taint analysis is done for the corresponding vulnerability type. RIPS is aware of 20 different vulnerability types which are refined to 45 different scopes.

Data-flow analysis: In order to find all possible values of a sensitive sink's argument, the argument is traced backwards through all basic blocks which are linked to the current basic block as entry edge.

We loop through all entry edges of the current basic block that do not sanitize this argument and look-up its name in the property of each block summary which gives the flow of data. The argument is replaced with the mapped symbol if a match is found, and all sanitization tags and encoding types are copied. Then, the trace is continued through all entry edges linked to the basic block. At the end, the unique sum of the return values for each path in the CFG is returned.

Dynamic Vulnerability inspection: Through this, the original tool is enhanced by adding new dynamic checks in the form of some additional functions of the following types in a new configuration file – dynamic file inclusion, dynamic code execution and their corresponding securing functions. This causes the code to be evaluated in dynamic way to check for the presence of vulnerabilities incorporated in the code due to the dynamism of these function and which could have been found with only static approach.

4.1.2.3 Detection of vulnerabilities

At first, all possible strings which are going into the sensitive argument are recomputed by data flow analysis directed backwards. Next, each string is checked in a context-sensitive way for user input. If the user input was unsanitized and the markup context is found to be exploitable, a new vulnerability is reported.

Data in the sink	Exploit
Tainted and match an attack pattern.	SQL injection, XSS.
Tainted and not sanitized.	<i>Data not escaped:</i> SQL injection, XSS.
Tainted or can be a null value.	<i>Data potentially not filtered, can be manipulated by a user:</i> 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.

Table -1: Probable vulnerabilities found out by the analysis

4.1.2.4 A context-sensitive validation of vulnerabilities

The strings obtained from the data-flow analysis, which is done while detecting the vulnerabilities, are checked for user input tags. A different analyzer is invoked for each vulnerability type. This analyzer identifies the context within the markup. Based on the context, specific vulnerability tags are determined. The taint symbols are marked as a tainted symbol only if they are not sanitized against the current vulnerability tag, and a vulnerability is issued.

If user input was not found, but the analyzed sensitive sink is called within a user-defined function, the strings are checked for parameter and global tags. When these are obtained in one of the strings, the symbols corresponding to them are added as vulnerable parameters or as vulnerable global variables to the user-defined function summary. These symbols are analyzed during inter-procedural analysis starting from the basic block of the function call [8].

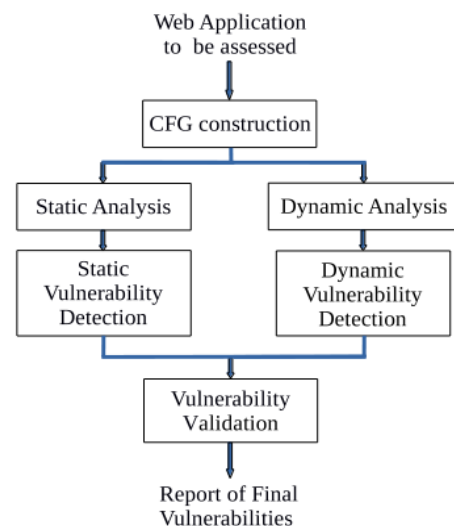


Fig – 1: Block diagram of the combined approach

5. ANALYSIS AND EVALUATION

To evaluate our combined approach, we analyze the code snippet from real web applications like HotCRP, a CMS developed by us, single web forms etc. using this tool. We then compare the bugs reported by the original tool with that of ours for the same web applications and show how the true positive rate is more and in our approach.

Tested Appl. →	HotCRP		Custom CMS		Single PHP Script	
	Org tool	Imp tool	Org tool	Imp tool	Org tool	Imp tool
File Inclusion	0	38	0	126	0	0
Code Execution	0	87	0	0	0	2
Total Found	0	125	0	126	0	2

Table -2: Comparative analysis of vulnerability detection by both the tools

The above analysis gives clear comparative remarks on the difference in the capabilities of the two tools – the original tool and the improved tool.

It clearly shows that the improved tool goes ahead and detects additional dynamic vulnerabilities such as file inclusion and code execution from the two big PHP applications and a single PHP script.

6. CONCLUSION



Due to the user-friendliness as well as ubiquitousness of web applications their security has become very important. In this project, we have tried to overcome known drawbacks of the current tools and have implemented tool that combines the static techniques with some dynamic ones to address these issues. We show how our technique handles some of the situations better where other tools either under-perform or simply fail to work and justify our efforts with some valid 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/>

[7] Johannes Dahse, Throsten Holz, Simulation of Built-in PHP Features for Precise Static Code Analysis.
 [8] Parkar V. V., Tirmare H. A., Aggregating Static and Dynamic Approaches For web Application Security Assessment, IRJET, 2015.
 [9] Tirmare H. A., A Novel Technique for mapping user queries to categories in personalized web search, IJMTER, 2015.

BIOGRAPHIES

	<p>Mr. Vishal Vijaykumar Parkar is a student in the master of Computer Science and Technology program at the Department of Technology, Shivaji University, Kolhapur. He is interested in domains like web security, web programming, and databases.</p>
	<p>Mr. H. A. Tirmare is an Assistant Professor in the Computer Science and Technology department, Department of Technology, Shivaji University, Kolhapur.</p> <p>His fields of interest include but are not limited to computer networks, web security, operating systems, and data structures.</p>