ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 3, March 2017

Vulnerability Detection in Web Applications

Nadiya UP [1] , Maya Mathew[2]

M. Tech Computer Science, KMCT College of Engineering, Kozhikode, Kerala, India[1]

Assistant Professor, Department of Computer Science, KMCT College of Engineering, Kozhikode, Kerala, India [2].

ABSTRACT: Most of the web applications are developed using php language like Google, Face book etc. WAP is a recent popular open source tool that detects vulnerabilities in the source code of web applications written in PHP. Security of web applications continues to be a challenging problem. Security is provided by analysing the source code if it contains any input validation vulnerabilities. Open source static analysis tools provide a sensible approach to mitigate this problem. However, these tools are programmed to detect a specific set of vulnerabilities and they are often difficult to extend to detect new ones .Source code static analysis tools are a solution to find vulnerabilities, but they tend to generate more false positives. We combine taint analysis which detect the vulnerability with data mining which predict whether it is false positive or real vulnerability.

KEYWORDS : Data mining, false positives, input validation vulnerabilities, source code analysis;

I. INTRODUCTION

We all familiar with different types of web applications which are the part of our daily life. Web application vulnerabilities have been a problem for several years. Although security starts to be taken into account during the application development, the tendency for source code contains vulnerabilities. Although there are tools to deal with these vulnerabilities, the fact is that good programming practices are still not sufficiently adopted and that the attacks against such vulnerabilities are very common. Both SQL injection and cross-site scripting are what we call input validation vulnerabilities. They are characterized by allowing malicious input to reach certain function calls without appropriate sanitization or validation. To make the problem even more difficult, continuously new technologies are Constantly being introduced in web applications increasing their complexity.

Security of web application is an important thing regarding all applications, which is started during the development stage. If the application is not secure means the hacker can easily attack the system, which is due to poor coding of developers. We propose a method for protecting web application by analysing the source code whether it contains any input validation vulnerability. This method first detects it and removed by inserting fixes in same code. This is done by the combination of static analysis with data mining ,initially detect vulnerability using static analysis then data mining is used for checking whether it is real or not by using classifiers. We have four main categories of vulnerabilities

Query manipulation-> Vulnerabilities related with structures that store data, like databases, and where the malicious code manipulates the queries, changing them. **Client-side injection->** Vulnerabilities associated to malicious code injected by client-side, such as java Script, and processed by server-side. **File and path injection->** Class of vulnerabilities that manipulate relative paths or files to, respectively, redirect to a different location or access the local system and web application files**. Command injection->** Vulnerabilities exploited by injection of file system commands and PHP instructions. The first main focus is on identifying and locating input validation vulnerabilities by analyzing statically the source code of web applications, using taint analysis with data mining. The second main focus is on removing these vulnerabilities by fixing the source code of the applications and on blocking injection attacks against databases inside database management systems.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 3, March 2017

The proposed method do the following activities:
A. Code analyzer: Do taint analysis ,it is a type of data flow analysis which track the sensitive data and verifies which point of the code it reaches.
B. Data mining: obtaining attributes from the candidate vulnerable control-flow paths, with the help of classifier then predict if each candidate vulnerability is a false positive or not.
C. Code correction: After the identification of vulnerability modify the source code with the fixes.

II. RELATED WORK

In [1] author used mechanisms to protect web applications from malicious inputs are web application Fire- walls (WAFs), sanitization/validation functions, and pre-pared statements in the application source code. The first two method inspect web application inputs and block and sanitize those that are considered malicious/dangerous whereas in prepared statement bounds inputs to placeholders in the query. Some of these mechanisms monitor SQL queries and block them if they deviate from certain query models, but the queries are inspected without full knowledge about how the server-side scripting language and the DBMS process them.

[2] In this paper, we combine dynamic analysis and static analysis, we firstly implement the static analysis based on HHVM (HipHop Virtual Machine) , from which we get the static analysis result and special path, variables, parameters and others, we then put them into the dynamic analysis to construct a dynamic test set, which will be used in real dynamic detection. In this way, we effectively make use of their both advantages, overcoming shortcomings. Web applications are part of our daily life. Although security starts to be taken into account during web application development, the tendency for source code to contain vulnerabilities persists. The input validation vulnerability category is arguably the most relevant. This is improved by the fact that code injection vulnerabilities such as SQL injection (SQLI) and cross-site scripting (XSS) remain at the center of this tendency, as reported by OWASP in its top [10] [4] and confirmed by the recent Ashley Madison fiasco [4].

Nevertheless, new technologies are becoming widely deployed as part of web applications. An example are NoSQL databases, particularly convenient to store “big data”. With new technologies come also new attack vectors, with 600 TB of data recently stolen from the most used [7] NoSQL database, MongoDB [23]. Vulnerability detection approaches identify vulnerabilities through tracking the flow of taints (user inputs) to sensitive sinks [3, 14]. Static and dynamic analysis techniques are generally used for taint tracking. Static analysis-based techniques suffer from low precision as these techniques generally overestimate the tainted-ness of inputs. Dynamic analysis-based techniques such as model checking [6] and console execution [4] produce zero false positive in principal. But these techniques are generally complex and expensive. By contrast, our approach only requires lightweight static analysis and data mining methods to report vulnerabilities.

Input Validation Vulnerabilities in Web Applications

1. SQL injection: SQL injection (SQLI) vulnerabilities are caused by the use of string-building functions to create SQL queries. SQLI attacks mix normal characters with meta characters to alter the structure of the query and read or write the database in an unexpected way. This vulnerability can be removed either by sanitizing the inputs (e.g., preceding with a backslash meta characters such as the prime) or by using prepared statements. Sanitization depends on the sensitive sink, For the MySQL engine, PHP provides the mysql_real_escape_string function.

2. Cross-site scripting (XSS) : This attacks execute malicious code (typically JavaScript) in a victim’s browser. Differently from the other attacks we consider, a XSS attack is not against a web application itself, but one of its users. A script vulnerable to reflected XSS can have a single line, echo $_GET[’user’];.The attack involves convincing the user to click on a link that accesses the web application, sending it a script that is reflected by the echo instruction and executed in the browser. These kinds of attacks can be prevented by sanitizing the input (e.g., html entities PHP function) and/or by encoding the output.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 3, March 2017

3. Local File Inclusion: Local file inclusion (LFI) differs from RFI by inserting a file from the file system of the web application. This kind of attack has been motivated by a default configuration introduced in PHP 4.2 that disallows remote file inclusion. LFI includes local files so the attacker needs to insert PHP code in the server beforehand

4. Remote file inclusionPHP allows a script to include files, which can be vulnerability if the file name takes user input. Remote file inclusion (RFI) attacks exploit this kind of vulnerability by forcing the script to include a remote file containing PHP code.

5. PHP Code Injection: PHP injection attack consists of a script that does not properly validate user inputs in the page parameter. An attacker can supply a specially crafted URL to pass arbitrary code to an eval() statement, which results in code execution. The function eval() is used to evaluate php string as php code and its exploitation is termed as Direct Dynamic Code Evaluation ('Eval Injection').

6. OS Command Injection: command An operating system injection (OSCI) attack consists in forcing the application to execute a command defined by the attacker. Consider as example a script that uses the following instruction to count the words of a file: $words = shell exec("/usr/bin/wc $_GET['file']");. The attacker can do command injection by inserting a filename and a command separated by a semicolon, e.g., /usr/bin/wc paper.txt; cat /etc/passwd.

III. PROPOSED ALGORITHM

The architecture composed of three modules: code analyzer, false positives predictor, and code corrector. The code analyzer first parses the PHP source code and generates an AST. Then, it uses tree walkers to do taint analysis, i.e., to track if data supplied by users through the entry points reaches sensitive sinks without sanitization. While doing this analysis, the code analyzer generates tainted symbol tables and tainted execution path trees for those paths that link entry points to sensitive sinks without proper sanitization. The false positives predictor continues where the code analyzer stops. For every sensitive sink that was found to be reached by tainted input, it tracks the path from that sink to the entry point using the tables and trees just mentioned. Along the track paths (slice candidate vulnerabilities in the figure), the vectors of attributes (instances) are collected and classified by the data mining algorithm as true positive (a real vulnerability), or false positive (not a real vulnerability).

A. Detecting Candidate Vulnerabilities By Taint Analysis

The taint analyzer is a static analysis tool that operates over an AST created by a lexer and parser. In the beginning of the analysis, all symbols (variables, functions) are untainted unless they are an entry point (e.g., $a in $a = $_GET['u']). The tree walkers build a tainted symbol table (TST) in which every cell is a program statement from which we want to collect data. Each cell contains a sub tree of the AST plus some data. For instance, for statement $x = $b + $c; the TST cell contains the sub tree of the AST that represents the dependency of $x on $b and $c

For each symbol, several data items are stored, e.g., the symbol name, the line number of the statement, and the taintedness. Taint analysis involves travelling though the TST. If a variable is tainted, this state is propagated to symbols that depend on it, e.g., function parameters or variables that are update using it. While the tree walkers are building the TST, they also build a tainted execution path tree (TEPT). Each branch of the TEPT corresponds to a tainted variable, and contains a sub-branch for each line of code where the variable becomes tainted (a square in the figure). The entries in the sub-branches (curly parentheses in the figure) are the variables that the tainted variable propagated its state into (dependent variables). Taint analysis involves updating the TEPT with the variables that become tainted.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 3, March 2017

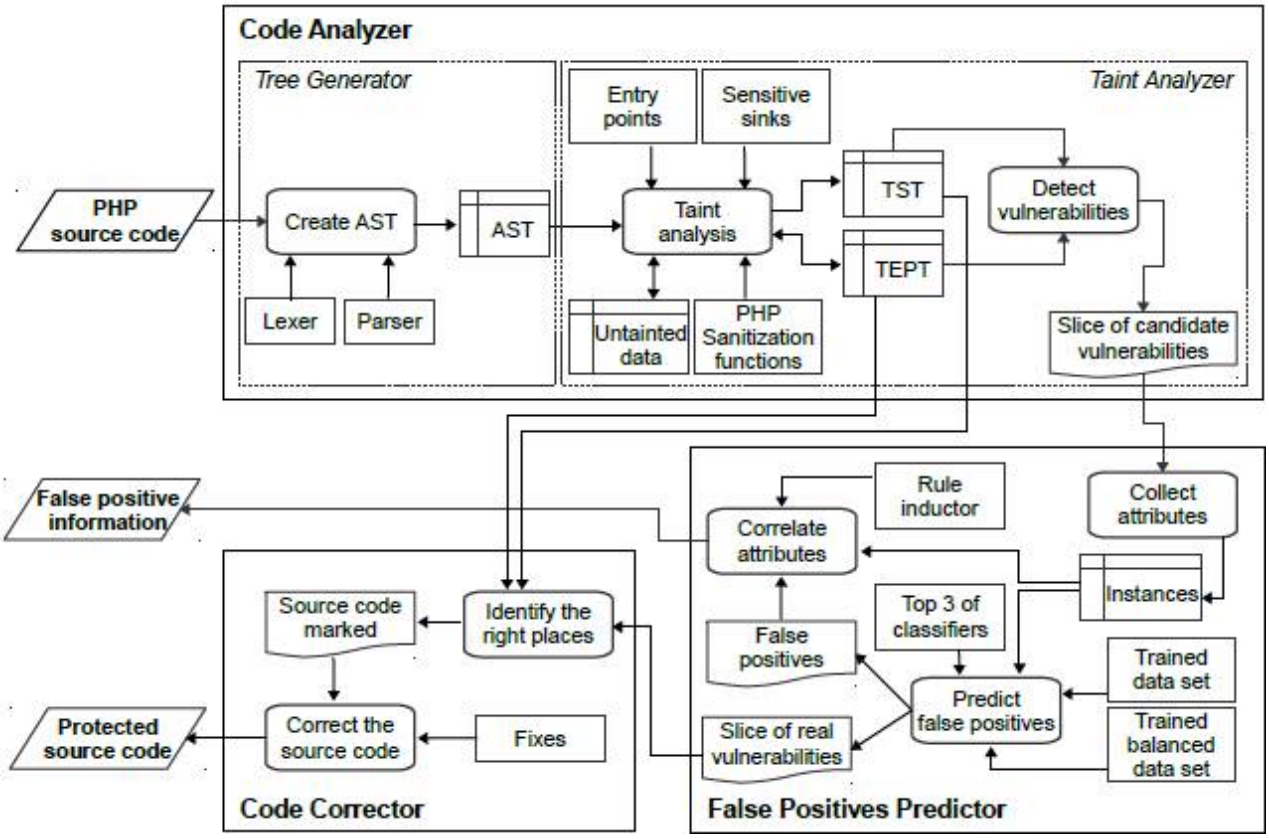

Fig 1.Architecture including main modules

B. Predicting False Positives

The false positive prediction is done by using three classifiers, which use 15 attributes to classify the vulnerabilities found by the taint analyzer as true or false. These attributes represent 24 symptoms that may be present in source code, divided in three categories: validation, string manipulation and SQL query manipulation. The symptoms are PHP functions that manipulate entry points or variables. The attributes represent symptoms of the same kind, e.g., the *type checking* attribute represent the symptoms that check the data type of variables. Therefore, an attribute represents several symptoms. The assessment has mainly two steps, as follows.

1. *Definition of the classifier*: pick a representative set of vulnerabilities identified by the taint analyzer, verify if they are false positives or not, extract a set of attributes, analyze their statistical correlation with the presence of a false positive, evaluate candidate classifiers to pick the best for the case in point, and define the parameters of the classifier.

2. *Classification of vulnerabilities*: given the classifier, for every vulnerability found determine if it is a false positive or not.

Any process of classification involves two aspects: the attributes that allow classifying an instance, and the classes in which these instances are classified. We identified the attributes by analyzing manually a set of vulnerabilities found by WAP's taint analyzer. We found three main sets of attributes that led to false positives.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 3, March 2017

i. **String manipulation**: attributes that represent PHP functions or operators that manipulate strings. These attributes are substring extraction, concatenation, addition of characters, replacement of characters, and removal of white spaces.

ii. **Validation**: a set of attributes related to the validation of user inputs, often involving an if-then-else construct. We define several attributes: data type (calls to is_int(), is_string()), is value set (isset()), control pattern (preg_match()), a test of belonging to a white-list, a test of belonging to a black-list, and error and exit functions that output an error if the user inputs do not pass a test.

iii. **SQL query manipulation**: attributes related to insertion of data in SQL queries (SQL injection only). We define attributes: string inserted in a SQL aggregate function (AVG, SUM, MAX, MIN, etc.), string inserted in a FROM clause, a test if the data are numeric, and data inserted in a complex SQL query.

C. Code Corrector

Our approach involves doing code correction automatically after the detection of the vulnerabilities is performed by the taint analyzer and the data mining component. The taint analyzer returns data about the vulnerability, including its class (e.g., SQLI), and the vulnerable slice of code. The code corrector uses these data to define the fix to insert, and the place to insert it. Inserting a fix involves modifying a PHP file when vulnerability is found, the code corrector inserts a fix that does sanitization or validation of the data flow. A fix is a call to a function that sanitizes or validates the data that reaches the sensitive sink. Sanitization involves modifying the data to neutralize dangerous meta characters or metadata, if they are present. Validation involves checking the data, and executing the sensitive sink or not depending on this verification. Most fixes are inserted in the line of the sensitive sink instead of, for example, the line of the entry point, to avoid interference with other code that sanitizes the variable..

VI. EXPERIMENTS

The new version of the tool was evaluated with 6 new vulnerability classes using web applications .The results show that this extensibility allows WAP to find many new vulnerabilities. WAPe analyzed a total of 8,374 fil corresponding to 2,065,914 lines of code of the 54 packages. It detected 413 real vulnerabilities from several classes in 17applications.Table shows that WAPE detect more vulnerabilities

WEB APPLICATIONS	WAP and WAPE			WAPE		
	Sqli	xss	File	Phpcod	os	Total
Admin Control Panel Lite 2	9	72	-	4	2	87
Clip Bucket	-	1	9	2	8	20
RCR AEsir	9	1	3	7	9	29
Refbase	46	-	4	10	9	69

TABLE 1.vulnerabilities found by two versions of WAP in web applications

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 3, March 2017

The largest packages analyzed were *Play sms v1.3.1* and *phpBB v3.1.6 Es* with 248,875 and 185,201 lines of code. Table V summarizes this analysis presenting the 17 packages where these vulnerabilities were found and some information about the analysis. These 17 packages contain 4,714 files corresponding to 1,196,702 lines of code. The total execution time for the analysis was 123 seconds, with an average of 7.2 seconds per application. This average time indicates that the tool has a good performance as it searches for 15 vulnerability classes in one execution.
The graphical representation shows WAPE performs well compared to wap and it detect new vulnerabilities.

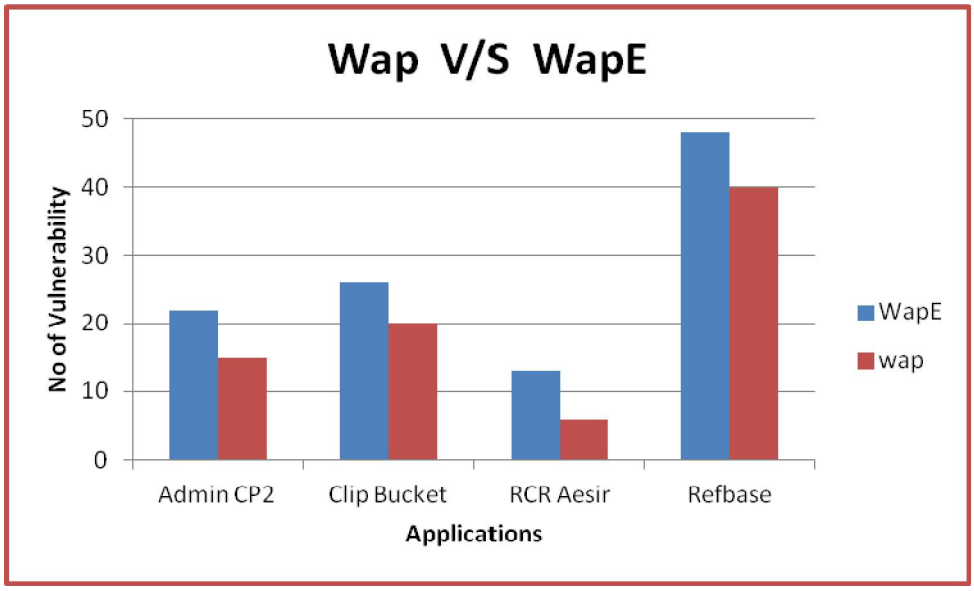

Fig 2:Vulnerabillity found by WAP and WAPE

VII. Conclusion and Future Work

This paper propose an approach for finding and correcting input validation vulnerabilities in web applications, and a tool that implements the approach for PHP programs and input validation vulnerabilities. The approach and the tool search for vulnerabilities using a combination of two techniques: static source code analysis and data mining. Data mining is used to identify false positives using 3 machine learning classifiers. It is important to note that this combination of detection techniques cannot provide entirely correct results. The static analysis problem is undecidable, and resorting to data mining cannot circumvent this undecidability, but only provide probabilistic results.

References

1. Ibéria Medeiros, Nuno Neves, *Member, IEEE*, and Miguel Correia, *Senior Member, IEEE* Detecting and Removing Web Application Vulnerabilities with Static Analysis and Data MiningW.
2. Halfond, A. Orso, and P. Manolios, "WASP: protecting web applications using positive tainting and syntax aware evaluation," *IEEE Trans. Softw. Eng.*, vol. 34, no. 1, pp. 65–81, 2008
3. OWASP WAP – Web Application Protection. https://www.owasp.org/index.php/OWASP WAP-Web Application Protection.
4 J. Antunes, N. F. Neves, M. Correia, P. Verissimo, and R. Neves, "Vulnerability removal with attack injection," *IEEE Trans. Softw. Eng.*, vol.36, no. 3, pp. 357–370, 2010 .
5. N. L. de Poel, "Automated security review of PHP web applications with static code analysis," M.S. thesis, State Univ. Groningen, Groningen, The Netherlands, May 2010.
6. J. Williams and D. Wichers, OWASP Top 10 - 2013 rcl - the ten most critical web application security risks, OWASP Foundation, 2013, Tech. Rep.
7. D. Evans and D. Larochelle, "Improving security using extensible lightweight static analysis," *IEEE Softw.*, pp. 42–51, Jan./Feb. 2002.
8. G. Wassermann and Z. Su, "Sound and precise analysis of web applications for injection vulnerabilities," in *Proc. 28th ACM SIGPLAN Conf. Programming Language Design and Implementation*, 2007, pp. 32–41 .

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 3, March 2017

9. R. A. DeMillo, R. J. Lipton, and F. G. Sayward, "Hints on test data selection: Help for the practicing programmer," *Computer*, vol. 11, no. 4, pp. 34–41, Apr. 1978.
10. L. K. Shar and H. B. K. Tan, "Mining input sanitization patterns for predicting SQL injection and cross site scripting vulnerabilities," in *Proc. 34th Int. Conf. Software Engineering*, 2012, pp. 1293–1296
11. L. K. Shar *et al.*, "Predicting common web application vulnerabilities from input validation and sanitization code patterns," in *Proc.27th IEEE/ACM Int. Conf. Automated Software Engineering*, 2012, pp. 310–313.

BIOGRAPHY

Nadiya U P received her B. Tech. degree in Computer Science and engineering from Mahatma Gandhi University, and currently pursuing M Tech in Computer Science at KMCT College of Engineering, Calicut.. Her research interest include Data mining

Maya Mathew received her B Tech. degree in Computer Science and engineering from Coorg institute of Technology and M Tech degree in Computer Science from New Horizon college of engineering. . She has four years of teaching experience and currently working as Assistant Professor in KMCT College of Engineering, Kozhikode