# Study By Detecting And Removing Web Application Vulnerabilities With Static Analysis And Data Mining

**Deepika.N[1], Dr.V.Saravanan[2]**
[1] Dept of Information Technology
[2]Associate professor & HOD in PG dept of Information Technology
[1, 2] Hindusthan college of arts and science,coimbatore,india

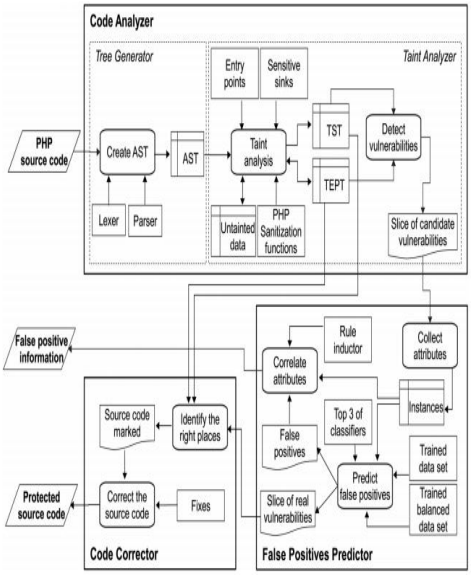
***Abstract-*** *Data mining, the extraction of undetected predictive information from large databases, is a prevailing new technology with great potential to help companies focus on the most important information in their data warehouses. Data mining tools expect future trends in addition behaviours, allowing businesses to make proactive, knowledge-driven decisions. The automated, prospective analyses offered by data mining move beyond the analyses of past events provided by retrospective tools typical of decision sustain systems. Data mining tools can answer business questions that traditionally were too time consuming to resolve. They scour databases for hidden patterns, finding predictive information that experts may miss because it lies outside their expectations . This approach brings together two approaches that are apparently orthogonal: humans coding the knowledge about vulnerabilities (for taint analysis), joined with the seemingly orthogonal approach of automatically obtaining that knowledge (with machine learning, for data mining). A major cause of this status is that many programmers do not have adequate knowledge about secure coding, so they leave applications with vulnerabilities This paper explores the use of a hybrid of methods to detect vulnerabilities with less false positives. Static analysis plays a great role in detecting and removing these attacks, a large examine has been going on that. Static analysis will often result in a great number of false positives. They will negatively affect the precision of the system. The approach was implemented in the WAP tool and an experimental evaluation was performed with a large set of open source PHP applications..*

***Keywords***- Data mining, PHP source code, Software, security, Input validation vulnerabilities, Web applications.

## I. INTRODUCTION

In two decades of existence, the Web evolved from a platform to access hypermedia to a framework for running complex web applications. These applications appear in many forms, from small home-made to large-scale commercial services such as Gmail, and Facebook. Removing web application vulnerabilities and static analysis approach for naturally protecting web applications while keeping the software engineer the up and up. The approach comprises in investigating the web application source code looking for vulnerabilities and remedies the source code. This last perspective is enabled by embeddings fixes that common security coding practices, so developers can take in these practices by observing the vulnerabilities and how they were evacuated. A major reason behind the insecurity of these applications is that most of the programmers have inaccurate and improper knowledge relating to the source code, as a reason of which the programmers leave web application with defects and faults. PHP that are dimly typed and not formally determined. In this manner, we supplement a type of static examination – pollute investigation – with the utilization of data mining to foresee the presence of false positives. We demonstrate that the mix of the two broad approaches of human-coded information and learning can be effective for vulnerability detection. These web applications are easily susceptible to the different vulnerabilities and attacks leading to the various problems such as breaching of data integrity, unauthorized access to the confidential data. This paper explores an approach for automatically protecting web applications while keeping the programmer in the loop. The approach consists in analyzing the web application source code searching for input validation vulnerabilities.The most frequent and significant vulnerabilities.

## II. SYSTEM ARCHITECTURE:




**Architecture including main modules, and data structures.**

## III. ALGORITHM

**3.1 Graphical and representative Algorithms:** This class includes algorithms that represent using a graphical model. Random Tree, and Random Forest classifiers, the graphical model is a decision tree. They use the information gain rate metric to decide how significant an trait is to classify an instance in a class (a leaf of the tree). An attribute with a small information gain has big entropy (degree of impurity of attribute or information quantity that the attribute offers to the obtaining the class), so it is less relevant for a class than another with a elevated information gain.

**3.2 Probabilistic Algorithms:** This category includes Naïve Bayes (NB), K-Nearest Neighbour (KNN), and Logistic Regression (LR). They classify an instance in the class that has the highest prospect. NB is a simple probabilistic classifier based on Bayes theorem, based on the assumption of conditional independence of the probability distributions of the attributes. K-NN classifies an instance in the class of its neighbours. LR uses regression study to classify an instance.

.

**3.3 Neural Network Algorithms:** This category has two algorithms: Multi-Layer Perceptron (MLP), and Support Vector Machine (SVM). These algorithms are inspired on the functioning of the neurons of the human brain. MLP is an artificial neural network classifier that maps sets of input data (values of attributes) onto a set of appropriate outputs (our class attributes, Yes or No). SVM is an evolution of MLP.

## IV. VALIDATION VULNERABILITIES

### 4.1 INPUT

The main problem in web application security lies in the offensive validation of user input, so this is the kind of vulnerabilities we currently consider . A remote file Iinclusion (RFI) vulnerability allows attackers to embed a remote file containing PHP code in the vulnerable program. Local file inclusion (LFI) differs from RFI by inserting in a script also from the file system of the web application, not a remote file. Input validation vulnerability is regarded as by the ability to decide at each step of the execution whether or not the program is in a safe state. Input validation refers to how your application filters, scrubs, or rejects input before additional processing. A directory traversal or path traversal (DT/PT) attack consists in an attacker accessing unpredicted les, possibly outside the web site directory.

## V. PREDICTING FALSE POSITIVES

The static analysis problem is known to be related to Turing's halting problem, so undecidable for non-trivial languages. In practice this difficulty is solved by making only partial analysis of some language constructs, tools to be unsound. This affirmation is finished by checking if these experiments basis incorrect or surprising comportment or yields. We picked to give the string a chance to be corrupted, which may prompt duplicitous positives yet not deceiving negatives. However, coding explicitly more knowledge in a static analysis tool is hard, and on average has to be done for each class of vulnerabilities follows this direction, but considers a single class of vulnerabilities, SQLI.

## VI. TAINT ANALYSIS

Taint analysis is nothing but parsing the source code, generating an abstract syntax tree (AST), doing taint analysis based on the AST, and generating trees describing candidate vulnerable control-flow paths. The taint analyzer is a static analysis tool that operates over an AST created by a lexer and a parser, for PHP 5 in our case (in WAP we implemented it using ANTLR. In the beginning of the analysis, all *symbols* (variables, functions) are *untainted* unless they are an entry point. The corrupt analyzer is a static investigation execute that works over an AST induced by a lexer and a parser, for JAVA 5 for our situation. Positive tainting mainly goes for trusted data .Trusted data can be easily and accurately identified compare to the untrusted data sources results in better automation.

## VII. CODE CORRECTION

WAP does code correction automatically after the detection of the vulnerabilities is performed by the taint analyzer and the data mining component. After detecting vulnerabilities and checking it for false positives each real Vulnerability is removed by correction of its source code. This module for the type of vulnerability selects the fix that removes the vulnerability and signalizes the places in the source code where the fix will be inserted. This is done by accepting user inputs as security tactful functions such as code execution function, directory creating functions and so on.

## VIII. IMPLEMENTATION

The approach can be implemented as a sequence of steps.

**8.1 Ruin analysis:** The job of this module is to parse the source code. This also gives the trees which describe more about candidate vulnerable control flow paths. If the variables are not checked properly they May ledto development of vulnerabilities. So the variables need to check before reaching the sensitive sinks. Input to the module of taint analysis is a PHP source code and the output of this module will be the candidate vulnerabilities. The first step will be parsing of the source code which will generate an Abstract Syntax Tree i.e. AST. Lexer and Parser will do the job of creating AST. In the Fig 3 shows the Abstract Syntax Tree for $b=$_GET[‘v’]. All the variables that act as entry points are marked tainted in the beginning. A symbol table will be generated which will have tainted variables. Taint analysis travels through this Tainted Symbol Table. If a variable is marked as tainted then the symbols depending on this variable are checked.the Tainted Symbol Table for specific symbols having name, line number, and tainted flag as its variables. In this way, all the candidate vulnerabilities are marked which will be checked by a false positive predictor to make sure about the real vulnerability.

***Testing*:** In this module testing will be performed on the corrected code to check for more bugs. We will perform manual testing on the source code .In manual testing. testing is done without using automation tools. Test cases are executed manually. Different manual testing tools are Selenium ,QTP, Jmeter, Loadrunner.

**8.1 Sorting of vulnerabilities**:

Some process of sorting involves two aspects: the attributes that allow classifying a sample, and the classes in which these samples are classified. We identified the attributes by analyzing manually a set of vulnerabilities found by WAP’s taint analyzer. We studied these vulnerabilities to recognize if they were false positives. This involved both reading the source code and executing attacks against each vulnerability found to understand if it was attackable (true positive) or not (false positive). This data set is extra discussed in Section V-C. From this analysis we found three main sets of attributes that led to false positives: String manipulation: attributes that represent PHP functions or operators that control strings. These are: substring extraction, concatenation, addition of characters, replacement of characters, and deletion of white spaces. Recall that a dataflow starts at an entry point, where it is marked tainted, and ends at a sensitive sink. The taint analyzer flags a vulnerability if the data flow is not untainted by a sanitization function before reaching the sensitive sink. These string manipulation functions may result in the sanitization of a data flow, but the taint analyzer does not have enough knowledge to change the status from tainted to untainted, so if a vulnerability is flagged it may be a false positive. The combinations of functions/operators that untainted a data flow are hard to establish, so this knowledge is not simple to retrofit into the taint analyzer.

**Validation**: set of attributes related to the validation of user Inputs, often involving an if-then-else construct. We define the following attributes: data type (calls to is_int(), is_string()), is value set (isset()), control pattern (preg_match()), test of belong to a white-list, test of belong to a black-list, error and exit functions that output an error if the user inputs do not pass a test. Similarly to what happens with string manipulations any of these attributes can sanitize a data flow and lead to a false positive.

## IX. CONCLUSION

The approach and the tool search for vulnerabilities using a combination of two techniques: static source code analysis, and data mining. It was able to find 388 vulnerabilitiesin 1.4 million lines of code. Its accuracy and precision were approximately 5% better than PHP MinerII's, and 45% better than Pixy's. A single tool consumes a lot of amount and resources which is not worthy as the amount of resources and time available is limited to an extent.

Due to which the disadvantages of one will be overcome by another. Data mining approach for false positive prediction is selected after considering alternative algorithms using the metrices. The tool corrects the code by inserting fixes, i.e., purification and approval capacities. This evaluation suggests that the tool can detect and correct the vulnerabilities of the classes it is programmed to handle.. All classifiers were selected after a thorough comparison of several alternatives. It is important to note that this combination of detection

techniques cannot provide entirely correct results. Data mining approach for false positive prediction is selected after considering alternative algorithms using the metrices, which will ensure the choice of our algorithm. Rather than a Web application Protection Tool. This evaluation suggests that the tool can detect and correct the vulnerabilities of the classes it is programmed to handle.

## REFRENCES


[1] Wang, B. Li, and H. Li, "Oruta: Privacy-preserving public auditing for shared data in the cloud," IEEE Trans. Cloud Computing, vol. 2, no. 1, pp. 43–56, 2014

[2] Wang, K. Ren, W. Lou, and J. Li, "Toward publicly auditable secure cloud data storage services," Network, IEEE, vol. 24, no. 4, pp. 19–24, 2010.

[3] 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.

[4] Shankar, K. Talwar, J. S. Foster, and D. Wagner, "Detecting format string vulnerabilities with type qualifiers," in Proc. 10th USENIX Security Symp., Aug. 2001, vol. 10, pp. 16–16

[5] Felmetsger, Viktoria, Ludovico Cavedon, Christopher Kruegel, and Giovanni Vigna. "Toward automated detection of logic vulnerabilities in web applications." In USENIX Security Symposium, vol. 58.

[6] Keir, Robin M., and Stephen A. Ecker. "System and method for network vulnerability detection and reporting." U.S. Patent 7,673,043, issued March 2, 2010.

[7] Salas, Palma, Marcelo Invert, and Eliane Martins. "A Black-Box Approach to Detect Vulnerabilities in Web Services Using Penetration Testing." Latin America Transactions, IEEE ( Revista IEEE America Latina) 13, no. 3 (2015): 707-712.

[8] Q. Zheng and S. Xu, "Fair and dynamic proofs of retrievability," in Proc. 1st ACM Conf. Data and Application Security and Privacy (CODASPY 11), 2011, pp. 237–248.

[9] Sommer, R., Paxson, V.: Outside the closed world: On using machine learning for network intrusion detection. in Proceedings of the 30th IEEE Symposium on Security and Privacy. pp. 305 316. IEEE (2010).

[10] Shar, L.K., Tan, H.B. K.: 10 predicting common web application vulnerabilities from input validation and sanitization code patterns. In: Proceedings of the 27th IEEE/ACM International Conference on Automated 13.Software Engineering. pp. 310 313 (2011) .