Detecting And Removing Vulnerabilities in Web Applications Using Data Mining and Static Analysis

ASHA[1], AMANDEEP KAUR[2], ABHISHEK[3], AISHWARYA PATIL[4], KAILASH[5]
[1, 2, 3, 4, 5] *Department of Computer Science and Engineering, Guru Nanak Dev Engineering College, Bidar*

Abstract- ***With the advent of new technologies and applications, the web today is expanding faster than ever. Web application security has been an important subject of research in the last few years, yet it still remains a challenging problem. The issues arise due to vulnerable source codes that are written in unsafe languages like PHP. With the use of static analysis over the source code, we can detect the input vulnerabilitiesin the web application. However, the static analysis of the source code often create false positives, and it takes alot of effort to fix the code. Through our paper, we delve into the approach of detecting vulnerabilities of the web application, but with lesser false positives. With the help of data mining, we remove the false positives generated. Here we will do programmed code amendment by embedding fixes in the source code. Afterwards diverse testing techniques like regression testing will be used to ensure if the code after rectification runs correctly and the points of vulnerability are removed. We materialize our research and this approach with the help of a WAP instrument. Consequently, we perform a trial assessment on numerous web applications with PHP source code to guarantee the accuracy of our software.***

Indexed Terms- ***Vulnerabilities, Static Analysis, Data mining, False Positives.***

I. INTRODUCTION

The World Wide Web is the single largest networkin existence. The inculcation of the internet in our lives have made us hugely dependent on it. Immense amounts of data and information is being exchanged over the internet every moment; thus, it attracts hackers and attackers from every corner looking to exploit and gain the data. Through this paper, we work on making the detection of vulnerabilities simpler and enhancing the ability of the programmers to build better and more secure web applications.

We examine the source code to detect the input validation vulnerabilities and insert fixes in the same code to remove these vulnerabilities. After fixing the code, we test the code through a testing module to ensure whetherthe code has been fixed or not. We use taint analysis to detect the vulnerabilities. However, a major issue in using taint analysis is that it generates false positives, i.e., it shows the presence of vulnerabilities even when they are not present. This is because the application is built using PHP, which is an unreliable and unsafe language. Therefore, we use taint analysis to detect vulnerabilities and data mining to predict the existence of false positives. This actually follows two disjoint approaches, to manually code about the vulnerabilities (for taint analysis), and to automatically obtain the knowledge of vulnerabilities through datamining. Algorithms such as ID3, SVM, Naïve Bayes, Random Forest, K-NN, are some of the classifiers that are used to flag the vulnerabilities as false positives or not. We explore the various induction rules like PART, JRip, Prism and Ridor to present the attributes associated with false positives. Through data mining, the machine only tests the codes which have false positive in them. We also explore Web Application Protection (WAP) which analyses andremoves input vulnerabilities from PHP, which is used in a large number of web applications.

Thus, through this paper, we contribute towards building a safe and secure web application with the combination of taint analysis and data mining.

II. EXISTING SYSTEM

The WAP tool is based on the analyses of the source code in the Intrusion Detection Systems. The IDS are segregated into two categories, the knowledge-based intrusion detection systems hold the database of vulnerabilities or attacks created manually, the behaviour-based systems learn about attacks using labelled datasets automatically. WAP is used to detect

different classes of input vulnerabilities in the source code: XSS (Cross Site Scripting), SQL Injections, Path Traversal, OS command injection and a few more. The tool contains knowledge about these vulnerabilities through manual coding. Using this taint analysis, we check if the inputs can reach a sensitive function without validation or sanitization. Hereby is an example XSS, its input points, sensitive sinks, and validation function in table 1.

Entry Points	Sensitive Sinks	Sanitisation Functions
$_POST $_GET $_REQUEST $_COOKIE HTTP_POST_VARS HTTP_GET_VA RS HTTP_COOKIE_VARS $_FILES $_SERVERS	Printechodie printf exiterror file_put_content sfprintffile_get_contents fgetsfgetc fscanf	Htmlentitieshtmlspecialcharsstrip_tagsurlencode

Table 1. Entry points, Sensitive sinks and sanitization functions for XSS vulnerabilities.

The three main components of WAP are: 1) Taint Analyzer, 2) Data Mining Component,3) Code Corrector.

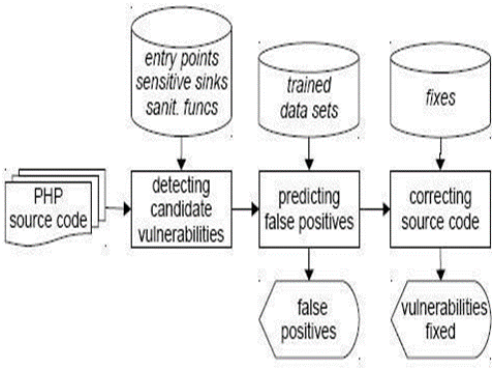

Fig 1:- Overview of the WAP tool

The primary work of the taint analyser is to parse the source code and build an abstract syntax tree (AST). After building the AST, the taint analyzer generates the tree describing candidate vulnerable control flow paths. For eachvulnerability class, the taint analysis holds different attributes related to them, i.e. Entry points, Sensitive sinks, and sanitation/validation functions. However, it tends to create a lot of false positives. False positives are vulnerabilities detected by the taint analyzer which are not true. The application has to be trained to predict newer false positives with the help of data mining as the code gets complex.

The false positive predictor uses the attributes from these vulnerable control flow path and different classifiers to predict if the given candidate vulnerability is a false positive or not. It uses a combination of three different classifiers, i.e. Logistic regression, random tree, support vector machine. After the vulnerabilities have been predicted, the Code Corrector is used to add the fixes to the faulty code. It looks after the fixes to be added and where they should be added. The code corrector also needs the information about the sanitation function which should be used embed the fix. This can only happen when the new classes are understood through data mining. Tainted modelis being used for performing the static and dynamic analysis. The WAP tool does not hold any specification of PHP, this is one of the drawbacks of this tool. Another drawback is that as we tested the WAP on various open source platforms, it could not parse source codes without a constructed grammar. However, with more research, this drawback has been resolved and new rules have been added.Pixy uses taint analysis as well as alias analysis while testing. The alias analysis are used to verify the existence of aliases, i.e. two or more variables which are used to denominate the same variable. The WAP tool performs global level analysis but Pixy performs only module level analysis.To detect the false positives, we use WEKA tool and perform data mining. In some application penetration testing approach has been used. The response are limited only to HTTP as this approach is implemented as black box testing. We use Static Analysis for vulnerability scanning. Static analysis is efficient, fast and reliable method of detecting vulnerabilities. To detect the flaws in the code, techniques used are lexical analysis, constraint analysis etc. Many of the applications are

implementing this technique soas to detect the flaws along with the additional modules.

III. LITERATURE SURVEY

A. *Sonam Panda, Ramani S* put forward a static examination algorithm to detect SQLCIVs. It determines the arrangements of conceivable database queries that aweb application may create utilizing situation free grammars and tracks flow of data through untrusted sources into those grammars. By making use of a general meaning of SQLCIVs based on the background of unreliable substrings, we can dodge the requirement for manually inscribed policies. It was precise, notable unknown liabilities in the web applications of the real world with multiple negatives, showing the feasibility of our method.

B. *Mounika B, A. Krishna Chaitanya* projected an inevitable system that provided an option to produce an output sanitization which is robotized input approval (IPAAS) and it is displayed for avoiding XSS and SQL assaults. This method advances the safe improvement of web applications by the accomplishment of parameter extraction and different type learning techniques by applying commanding information validators at runtime.

C. *N. L. de Poel,* propounded SAFERPHP, a static investigation system used to detect the liabilities in PHP source code. This agenda employs various algorithms including; 1) To implement new type of symbolic performance to check denial-of-service liabilities. 2) A new type of infers procedural analysis to verify the application sanction policy and find misplaced checks before any complex database operations.

D. *Y.-W. Huang* anticipated an approach which gave quick protection at a much lower cost than others since approval is confined to potentially weak segments of code. If Web SSARI classifies the use of untrusted data taking after right treatment, the code is left as it is. As per several experiments, Web SSARI carried only 0.02 percent of allthe statements to be inspected with inappropriate sanitization agendas. Interestingly, Sharp and Scott implements global justification for each data submitted by the user without any complaint and without even concerning that the same validation process may be implemented by web application as well. These finally outcomes in pointless upstairs.

IV. OUTLINE OF THE PAPER

Through our research we outline the following points:

- Improvement in security of web applications by detecting and removing input vulnerabilities in source code of web applications.
- Usage of taint analysis and data mining techniques for detection of vulnerabilities with fewer false positives.
- Automatic correction in the source code and informing programmer about it. 4) Finally experimenting the tool with various web applications to see the correctnessof tool.

V. PROPOSED SYSTEM

Through this paper, we propose a tool that will scan applications based on PHP, to detect and remove input validation and other vulnerabilities. We use a combination of two techniques i.e. Static source code analysis and data mining in our approach. For detection of false positives, Data mining coupled with different machine learning classifiers is used. The presence of false positives is confirmed by Induction rule classifier. The single detection technique fails to provide correct results so different detection techniques are combined. But they also fail to provide entirely correct results. The generation of false positives happens after the detection of candidate vulnerabilities. The proposed system comprises of the tool which will replace the vulnerable code with fixes. Fixes are nothing but the correct code. After replacing the code by fixes, testing will be performed to check for the correct working of the system. It will check the behaviour of the application after replacing vulnerable code with fixes. The proposed system is designed for PHP applications. The system has been experimented with a number of synthetic codes in which vulnerabilities have been induced purposely.

VI. IMPLEMENTATION

The approach can be implemented as a sequence of steps.

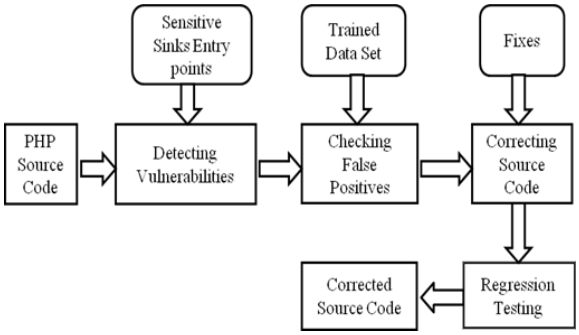

Fig 2:- Architecture of Proposed System.

A. Taint analysis

In taint analysis, the module is used to parse and analyse the source code. After parsing, the module builds an abstract syntax tree which gives us information about candidate vulnerable control flow path. To prevent the development of vulnerabilities, all the variables are checked properly. Thus, before reaching the sensitive sinks, the module checks the variables. The PHP source code which is used in the application is the input given in the module, and the candidate vulnerabilities are the output. AST Lexer and Parser in the module are used to create the tree. 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. The module generates a symbol table containing all the tainted variables. All the symbols dependent on the tainted variables are checked after marking them. Fig 4. shows 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.

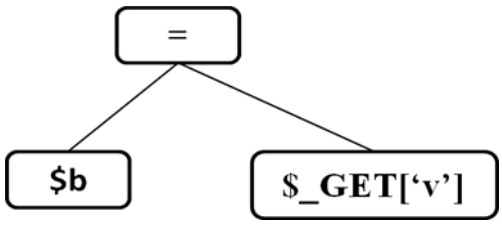

Fig 3: - Abstract Syntax Tree

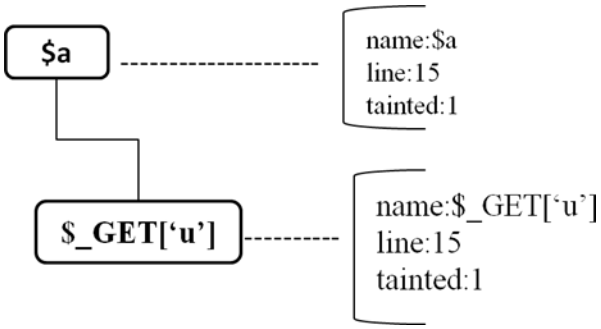

Fig 4: - Tainted Symbol Table and Taint Analysis on variables

B. Data mining

Data mining is used to obtain the attributes from candidate vulnerable control flow-paths. The presence of data mining is actually confirmed by data mining coupled with classifiers. If the presence of false positives is confirmed, further processing will be done with the help of induction rules.

C. Code correction

After detecting vulnerabilities and checking it for false positives each real vulnerability is removed by correction of its source code. Then, the code is corrected with the insertion of the fixes and new files are created. Fixes are small pieces of the code (small PHP functions developed to the effect) that perform sanitization or validation of the user inputs, depending on the vulnerability type. Our approach involves automatic code correction using K-NN algorithm after the detection of the vulnerabilities by taint analyser and the data mining component.

D. Feedback

After testing the module, the programmer provides feedback and observations based on the experience of the client. Experience will be based on data collected from vulnerable paths, vulnerabilities, fixes, false positive probability and the attributes that classified it as a false positive. Feedback will help programmer to avoid the same mistakes.

E. Testing

In this module, we perform testing on the fixed code and scan for more bugs. The testing is done without any automation tool, i.e. it is done manually. Different manual testing tools are Selenium, QTP, Jmeter, Loadrunner.

F. Algorithms Used

The algorithms which will be used in implementing the proposed system are Logistic Regression, Naïve Bayes (NB) and K-Nearest Neighbour (KNN). The other algorithms that will be used are random tree and random forest classifier.

CONCLUSION

Through this paper, we propose a method of scanning and amending vulnerabilities in web applications, we put forward a technique of detecting the input validation vulnerabilities. Two inverse methodologies have been used, i.e., taint analysis of source code and data mining to predict false positives. Data mining is coupled with three different classifiers and an induction rule classifier to predict and fix the false positives. The classifier were chosen after an extensive comparison and testing of several alternatives. Even though through data mining, we can't completely evade static examination, we do get a probabilistic consequence of the issues. This tool is used to embed fixes in the code and contains sanitization and validation functions. Testing is utilized to check if the fixes surely evacuate the vulnerabilities and do not compromise the (correct) conduct of the applications. The tool became explored by way of the usage of synthetic code with vulnerabilities embedded on motive, and with a wide varietyof open-source PHP applications. It turned into additionally contrasted with source code analysis tools: Pixy, and PHP MinerII. This evaluation proposes that the device can discover and correct the vulnerabilities of the classes it is programmed to deal with. It may find out 388 vulnerabilitiesin 1.4 million strains of code. Its exactness and accuracy were round 5% advanced to PHP MinerII's, and 45% Superior to Pixy's.

REFERENCES

[1] L. K. Shar: Predicting common web application vulnerabilities from input validation and sanitization code patterns. Automated Software Engineering (ASE), 27th IEEE/ACM International Conference, 2012.

[2] N. L. de Poel: Automated security review of PHP web applications with static code analysis. ACM 23rd international conference on World Wide Web, 2014.

[3] Y.W. Huang: Securing web application code by static analysis and runtime protection. ACM 1-58113-844- X/04/0005, WWW 2004.

[4] L. K. Shar, H. B. K. Tan: Mining SQL injection and cross site scripting vulnerabilities. International Conference on Software Engineering, 2012.

[5] Sonam Panda, Ramani S: Protection of Web Application against SQL Injection Attacks. IJMER Vol3, Issue.1, Jan-Feb 2013

[6] Iberia Medeiros, Numo Neves: Detection of web application vulnerabilities using static analysis. IEEE transaction on reliability.

[7] Symantec, Internet threat report. 2012 trends, vol. 18, Apr. 2013.

[8] Ashwani Garg, Shekhar Singh: A Review on Web Application Security Vulnerabilities. IJARSCE, Volume 3, Issue 1, January 2013.