International Journal of Innovative Technologies
WWW.IJITECH.ORG
ISSN 2321-8665
Volume.06, Issue.01,
January-June, 2018,
Pages:0117-0120

Detecting and Removing Web Application Vulnerabilities with Static Analysis and Data Mining

D. MAHESH[1], SK. N. REHMATHUNNISA[2]
[1]PG Scholar, Dept of CSE, DJR Institute of Engineering & Technology, Andhrapradesh, India,
E-mail: dmahesh333@gmail.com.
[2]Associate Professor, Dept of CSE, DJR Institute of Engineering & Technology, Andhrapradesh, India,
E-mail: shaikrehmathunnisa@gmail.com.

Abstract: An important part of that problem derives from vulnerable source code, often written in unsafe languages like PHP. Source code static analysis tools are a solution to find vulnerabilities, but they tend to generate false positives, and require considerable effort for programmers to manually fix the code. We explore the use of a combination of methods to discover vulnerabilities in source code with fewer false positives. We combine taint analysis, which finds candidate vulner abilities, with data mining, to predict the existence of false positives. 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). Given this enhanced form of detection, we propose doing automatic code correction by inserting fixes in the source code. Our approach was implemented in the WAP tool, and an experimental evaluation was performed with a large set of PHP applications. Our tool found 388 vulnerabilities in1.4 million lines of code. Its accuracy and precision were approximately 5% better than PhpMinerII's and 45% better than Pixy's.

Keywords: Automatic Protection, Data Mining, False Positives, Input Validation Vulnerabilities, Software Security, Source Code Static Analysis, Web Applications.

I. INTRODUCTION

These applications appear in many forms, from small home-made to large-scale commercial services (e.g., Google Docs, Twitter, Facebook). However, web applications have been plagued with security problems. For example, a recent report indicates an increase of web attacks of around 33% in 2012 [1]. Arguably, a reason for the insecurity of web applications is that many programmers lack appropriate knowledge about secure coding, so they leave applications with flaws. However, the mechanisms for web application security fall in two extremes. 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 vulner abilities, and inserting fixes in the same code to correct these flaws. The programmer is kept in the loop by being allowed to understand where the vulner abilities were found, and how they were corrected. This approach contributes directly to the security of web applications by removing vulnerabilities, and indirectly by letting the programmers learn from their mistakes. This last aspect is enabled by inserting fixes that follow common security coding practices, so programmers can learn these practices by seeing the vulner abilities, and how they were removed. We explore the use of a novel combination of methods to detect this type of vulnerability: static analysis with data mining. Static analysis is an effective mechanism to find vulner abilities in source code, but tends to report many false positives (non-vulnerabilities) due to its un decidability. This problem is particularly difficult with languages such as PHP that are weakly typed, and not formally specified.

Therefore, we complement a form of static analysis, taint analysis, with the use of data mining to predict the existence of false positives. This solution combines two apparently disjoint approaches: humans coding the knowledge about vulner abilities (fortaint analysis), in combination with automatically obtaining that knowledge (with supervised machine learning supporting data mining). To predict the existence of false positives, we introduce the novel idea of assessing if the vulnerabilities detected are false positives using data mining. To do this assessment, we measure attributes of the code that we observed to be associated with the presence of false positives, and use a combination of the three top-ranking classifiers to flag every vulnerability as false positive or not. We explore the use of several classifiers: ID3, C4.5/J48, Random Forest, Random Tree, K-NN, Naive Bayes, BayesNet, MLP, SVM ,and Logistic Regression. Moreover, for every vulner ability classified as falsepositive, we use an induction rule classifier to show which attributes are associated with it. We explore the JRip, PART, Prism, and Ridor induction rule classifiers for this goal .Classifiers are automatically configured using machine learning based on labeled vulner ability data.

II. EXISTING SYSTEM

- There is a large corpus of related work, so we just summarize the main areas by discussing representative papers, while leaving many others unreferenced to conserve space.

- Static analysis tools automate the auditing of code, either source, binary, or intermediate.
- Taint analysis tools like CQUAL and Splint (both for C code) use two qualifiers to annotate source code: the untainted qualifier indicates either that a function or parameter returns trustworthy data (e.g., a sanitization function), or a parameter of a function requires trustworthy data (e.g., mysql_query). The tainted qualifier means that a function or a parameter returns non-trustworthy data (e.g., functions that read user input).

Disadvantages of Existing System:

- These other works did not aim to detect bugs and identify their location, but to assess the quality of the software in terms of the prevalence of defects and vulnerabilities.
- WAP does not use data mining to identify vulnerabilities, but to predict whether the vulnerabilities found by taint analysis are really vulner abilities or false positives.
- AMNESIA does static analysis to discover all SQL queries, vulnerable or not; and in runtime it checks if the call being made satisfies the format defined by the programmer.
- WebSSARI also does static analysis, and inserts runtime guards, but no details are available about what the guards are, or how they are inserted.

III. PROPOSED SYSTEM

- This paper explores an approach for automatically protecting web applications while keeping the programmer in the loop as shown in Fig.1. The approach consists in analyzing the web application source code searching for input validation vulnerabilities, and inserting fixes in the same code to correct these flaws. The programmer is kept in the loop by being allowed to understand where the vulnerabilities were found, and how they were corrected.
- This approach contributes directly to the security of web applications by removing vulnerabilities, and indirectly by letting the programmers learn from their mistakes. This last aspect is enabled by inserting fixes that follow common security coding practices, so programmers can learn these practices by seeing the vulnerabilities, and how they were removed.
- We explore the use of a novel combination of methods to detect this type of vulnerability: static analysis with data mining. Static analysis is an effective mechanism to find vulnerabilities in source code, but tends to report many false positives (non-vulnerabilities) due to its undecidability
- To predict the existence of false positives, we introduce the novel idea of assessing if the vulnerabilities detected are false positives using data mining. To do this assessment, we measure attributes of the code that we observed to be associated with the presence of false positives, and use a combination of the three top-ranking classifiers to flag every vulnerability as false positive or not.

Advantages of Proposed System:

- Ensuring that the code correction is done correctly requires assessing that the vulnerabilities are removed, and that the correct behavior of the application is not modified by the fixes.
- We propose using program mutation and regression testing to confirm, respectively, that the fixes function as they are programmed to (blocking malicious inputs), and that the application remains working as expected (with benign inputs).
- The main contributions of the paper are: 1) an approach for improving the security of web applications by combining detection and automatic correction of vulnerabilities in web applications; 2) a combination of taint analysis and data mining techniques to identify vulnerabilities with low false positives; 3) a tool that implements that approach for web applications written in PHP with several database management systems; and 4) a study of the configuration of the data mining component, and an experimental evaluation of the tool with a considerable number of open source PHP applications.

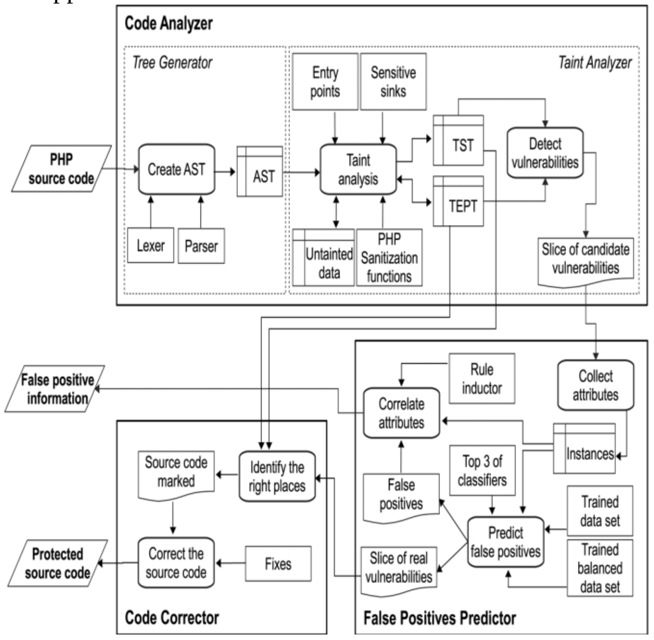


Fig.1.System Architecture.

IV. RELATED WORK

- Taint Analysis
- Predicting False Positives
- Code Correction
- Testing

A. Taint Analysis

The taint analyzer is a static analysis tool that operates over an AST created by a lexer and a parser, for PHP 5in our case. In the beginning of the analysis, all symbols (variables, functions)are untainted unless they are an entry point. 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 subtree of the AST plus some data. For instance, for statement $x = $b + $c; the TST cell contains the subtree 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.

B. Predicting False Positives

The static analysis problem is known to be related to Turing's halting problem, and therefore is undecidable for non-trivial languages. In practice, this difficulty is solved by making only a partial analysis of some language constructs, leading static analysis tools to be unsound. In our approach, this problem can appear, for example, with string manipulation operations. For instance, it is unclear what to do to the state of a tainted string that is processed by operations that return a substring or concatenate it with another string. Both operations can untainted the string, but we cannot decide with complete certainty. We opted to let the string be tainted, which may lead to false positives but not false negatives.

C. Code Correction

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. 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.

D. Testing

Our fixes were designed to avoid modifying the (correct) behavior of the applications. So far, we witnessed no cases in which an application fixed by WAP started to function incorrectly, or that the fixes themselves worked incorrectly. However, to increase the confidence in this observation, we propose using software testing techniques. Testing is probably the most widely adopted approach for ensuring software correctness. The idea is to apply a set of test cases (i.e., inputs) to a program to determine for instance if the program in general contains errors, or if modifications to the program introduced errors. This verification is done by checking if these test cases produce incorrect or unexpected behavior or outputs. We use two software testing techniques for doing these two verifications, respectively: 1) program mutation, and 2) regression testing.

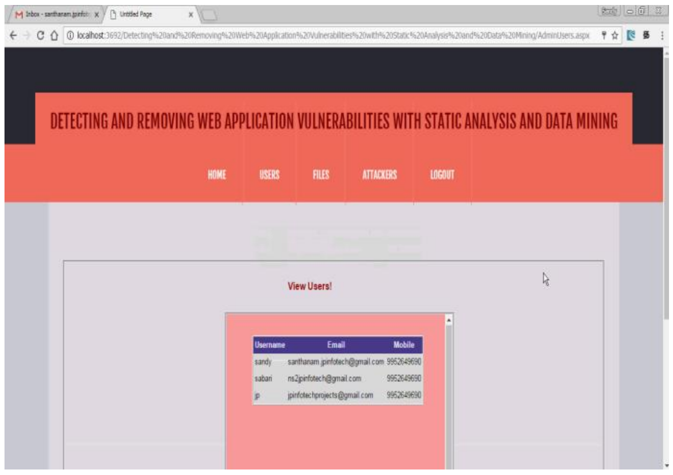
VI. CONCLUSION

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 the top 3 machine learning classifiers, and to justify their presence using an induction rule classifier. 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. The static analysis problem is un-decidable, and resorting to data mining cannot circumvent this un-decidability, but only provide probabilistic results. The tool corrects the code by inserting fixes, i.e., sanitization and validation functions. Testing is used to verify if the fixes actually remove the vulnerabilities and do not compromise the (correct) behavior of the applications.

VII. RESULTS

DETECTING AND REMOVING WEB APPLICATION VULNERABILITIES WITH STATIC ANALYSIS AND DATA MINING
applications
vulnerabilities
web
security
attacks
application
internet
cyber
HOME
USERS
ADMIN
REGISTER

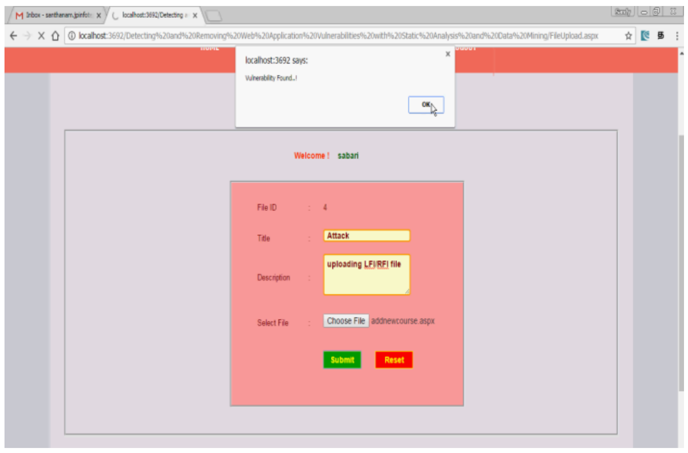
Fig.2.


DETECTING AND REMOVING WEB APPLICATION VULNERABILITIES WITH STATIC ANALYSIS AND DATA MINING
HOME
USERS
FILES
ATTACKERS
LOGOUT

View Users!

Username	Email	Mobile
sandy	santhanam.jpinfotech@gmail.com	9952649690
sabari	ns2jpinfotech@gmail.com	9952649690
jp	jpinfotechprojects@gmail.com	9952649690

Fig.3.



localhost:3692 says:
Vulnerability Found..!

Welcome ! sabari

File ID : 4
Title : Attack
Description : uploading LFI/RFI file
Select File : Choose File addnewcourse.aspx

Fig.4.

VIII. REFERENCES

[1] Symantec, Internet threat report. 2012 trends, vol. 18, Apr. 2013.
[2] W. Halfond, A. Orso, and P. Manolios, “WASP: protecting web applicationsusing positive tainting and syntax aware evaluation,” IEEETrans. Softw. Eng., vol. 34, no. 1, pp. 65–81, 2008.
[3]T.Pietraszek and C. V. Berghe, “Defending against injection attacks through context-sensitive string evaluation,” in Proc. 8th Int. Conf. RecentAdvances in Intrusion Detection, 2005, pp. 124–145.
[4] X. Wang, C. Pan, P. Liu, and S. Zhu, “SigFree: A signature-free bufferoverflow attack blocker,” in Proc. 15th USENIX Security Symp., Aug.2006, pp. 225–240.
[5] J. Antunes, N. F. Neves, M. Correia, P. Verissimo, and R. Neves, “Vulnerabilityremoval with attack injection,” IEEE Trans. Softw.Eng., vol.36, no. 3, pp. 357–370, 2010.
[6] R. Banabic and G. Candea, “Fast black-box testing of system recoverycode,” in Proc. 7th ACM Eur. Conf. Computer Systems, 2012, pp.281–294.
[7] Y.-W. Huang et al., “Web application security assessment by fault injectionand behavior monitoring,” in Proc. 12th Int. Conf. World WideWeb, 2003, pp. 148–159.
[8] Y.-W. Huang et al., “Securing web application code by static analysisand runtime protection,” in Proc. 13th Int. Conf. World Wide Web,2004, pp. 40–52.
[9] N. Jovanovic, C. Kruegel, and E. Kirda, “Precise alias analysis forstatic detection of web application vulnerabilities,” inProc. 2006WorkshopProgramming Languages and Analysis for Security, Jun. 2006,pp. 27–36.
[10] U. Shankar, K. Talwar, J. S. Foster, and D. Wagner, “Detecting formatstring vulnerabilities with type qualifiers,” in Proc. 10th USENIX SecuritySymp., Aug. 2001, vol. 10, pp. 16–16.
[11] W. Landi, “Undecidability of static analysis,” ACM Lett.Program.Lang. Syst., vol. 1, no. 4, pp. 323–337, 1992.
[12] N. L. de Poel, “Automated security review of PHP web applicationswith static code analysis,” M.S. thesis, State Univ. Groningen,Groningen, The Netherlands, May 2010.
[13] WAP tool website [Online]. Available: http://awap. sourceforge.net/
[14] Imperva, Hacker intelligence initiative, monthly trend report #8, Apr.2012.
[15] J. Williams and D. Wichers, OWASP Top 10 - 2013 rcl - the ten mostcritical web application security risks, OWASP Foundation, 2013,Tech. Rep.
[16] R. S. Sandhu, “Lattice-based access control models,” IEEE Comput.,vol. 26, no. 11, pp. 9–19, 1993.

Author’s Profiles:

D. Mahesh, received hisB.Tech degree in computer science and engineering and pursuing M.Tech degree in computer science and engineering from**,** DJR Institute of Engineering & Technology.

Sk.N.RehmathunnisaM.Tech received her M.Tech degree and B.Tech degree in computer science and engineering . She is currently working as an Assoc Professor in, DJR Institute of Engineering& Technology.