International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395 -0056
Volume: 03 Issue: 11 | Nov -2016 www.irjet.net p-ISSN: 2395-0072

Removing Web Application Vulnerabilities with Static Analysis.

Ritesh Phegade , Rishabh Jain , Abhishek Randhir , Pritesh Kadav

Ritesh Phegade , SKNSITS LONAVALA Tal:-Maval Dist:-Pune Pin:-410401
Rishabh Jain, SKNSITS LONAVALA Tal:-Maval Dist:-Pune Pin:-410401
Guided By: Prof.Jacob John , Dept. of Computer Engineering, SKNSITS college Lonavala, Maharashtra,India

---

Abstract - *Web application security is an major issue in today's internet. A major reason of this status is that many developers do not have satisfactory knowledge about secure coding, so they leave applications with vulnerabilities. An approach to solve this issue is to use source code static examination to discover these bugs, but these tools are known to report numerous false positives that make hard the task of correcting the application. This paper analyses the use of a mixture of techniques to detect vulnerabilities with less false positives. After an initial step that uses taint analysis to flag candidate vulnerabilities, our approach uses data mining to predict the existence of false positives. This approach achieves an exchange off between two obviously inverse methodologies, people coding the knowledge about vulnerabilities (for taint analysis) versus naturally obtaining that information (with machine learning, for data mining). Given this more exact type of recognition, we do programmed code correction by inserting fixes in the source code. The approach was executed in the WAP tool and an experimental evaluation was performed with a large set of open source PHP applications.*

Key Words: **Data mining, PHP source code, Software, security, Input validation vulnerabilities ,Web applications** .

1. INTRODUCTION

Removing web application vulnerabilities and static analysis is an 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. The developer can understand where the vulnerabilities were found and how they were redressed. This approach helps for the security of web applications by evacuating vulnerabilities, and giving the software engineers a chance to gain from their errors. This last perspective is enabled by embeddings fixes that follow common security coding practices, so developers can take in these practices by observing the vulnerabilities and how they were evacuated. Static investigation is an effective way to discover vulnerabilities in source code, yet it causes numerous false positives (non-vulnerabilities) because of its undesirability. This issue is especially difficult with languages for example, PHP that are weakly 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. This approach consolidates two evidently inverse methodologies: people coding the information about vulnerabilities (for pollute investigation) versus naturally getting that information (with directed machine learning supporting data mining). Interestingly this division has been available for long in another zone of security, intrusion detection. As its name suggests, signature or knowledge-based intrusion detection relies on knowledge about intrusions coded by humans (signatures), though anomaly-based detection relies on models of normal behavior created using machine learning. Nevertheless, irregularity based detection has been highly criticized and has very limited commercial used today. We demonstrate that the mix of the two broad approaches of human-coded information and learning can be effective for vulnerability detection.

2. Proposed System

Proposed system shows an approach for finding and correcting vulnerabilities in web applications, and a tool that executes the approach for PHP projects and input validation vulnerabilities. The approach and the tool search for vulnerabilities using a combination of two techniques: static source code examination, and data mining. Data mining is utilized to distinguish false positives utilizing the main 3 machine learning classifiers, and to justify their nearness utilizing an acceptance govern classifier. All classifiers were chosen after a careful examination of a few options. mix of detection procedures can't provide entirely correct results. The static investigation issue is not decidable, and depending on data mining can't go around this undesirability, yet just give probabilistic results. The tool corrects the code by inserting fixes, i.e. purification and validation functions. Testing is utilized to check if the fixes really evacuate the vulnerabilities and don't compromise the (correct) behavior of the applications. The tool was experimented with using synthetic code with vulnerabilities inserted on purpose, and with an impressive number of open source PHP applications.

3. System Architecture

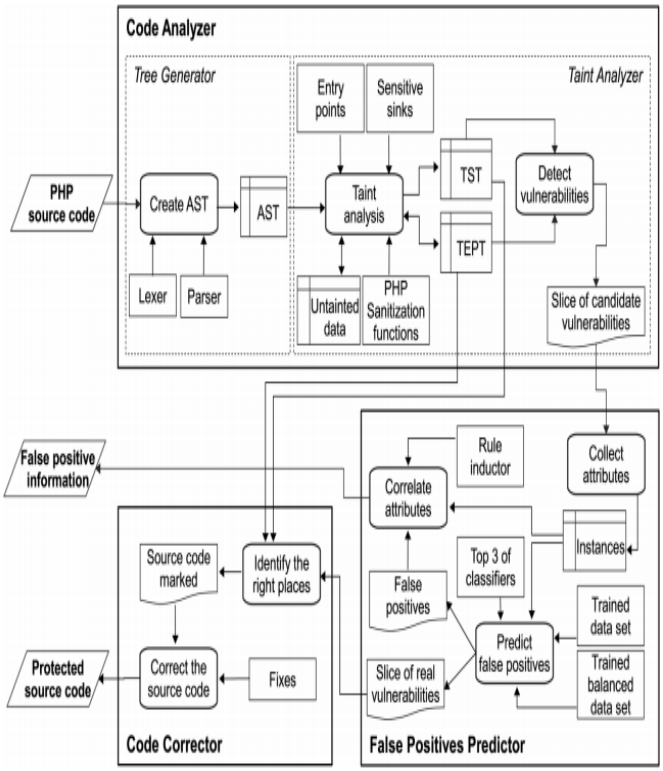

Fig -1: Architecture including main modules, and data structures[1]

4. Algorithm

Graphical and Symbolic 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 relevant an attribute 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 higher information gain. [1]

Probabilistic Algorithms: This category includes Naïve Bayes (NB), K-Nearest Neighbor (KNN), and Logistic Regression (LR). They classify an instance in the class that has the highest probability. 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 neighbors. LR uses regression analysis to classify an instance.[1]

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.

5. Discussion

The WAP tool, similar to whatever other static examination approach, can just recognize vulnerabilities it is modified to. WAP can, however, be extended to handle more classes of input validation vulnerabilities. We talk about it considering WAP's three fundamental segments: corrupt analyzer, data mining component, and code corrector. The corrupt analyzer has three bits of data about each class of vulnerabilities: entry points, sensitive sinks, and sanitization functions. The entry points are always a variant of the same set (functions that read input parameters, e.g., $_GET), while the rest have a tendency to be easy to recognize once the helplessness class is known. The data mining segment must be prepared with new learning about false positives for the new class. his preparation might be skipped at to start with, and enhanced incrementally when more information get to be accessible. For the preparation, we found data about candidate vulnerabilities of that kind found by the corrupt analyzer, which must be marked as genuine or false positives. At that point, the credits related to the false positives must be utilized to arrange the classifier. The code corrector needs data about what sanitization function has to be used to handle that class of vulnerability, and where it should be embedded. Once more, getting this data is feasible once the new class is known and caught on. A limitation of WAP derives from the lack of formal specification of PHP. During the experimentation of the tool with many open source applications (Section VII-A), few times WAP was not able parse the source code for absence of a punctuation govern to manage weird developments. With time, these principles where included, and these issues quit showing up.

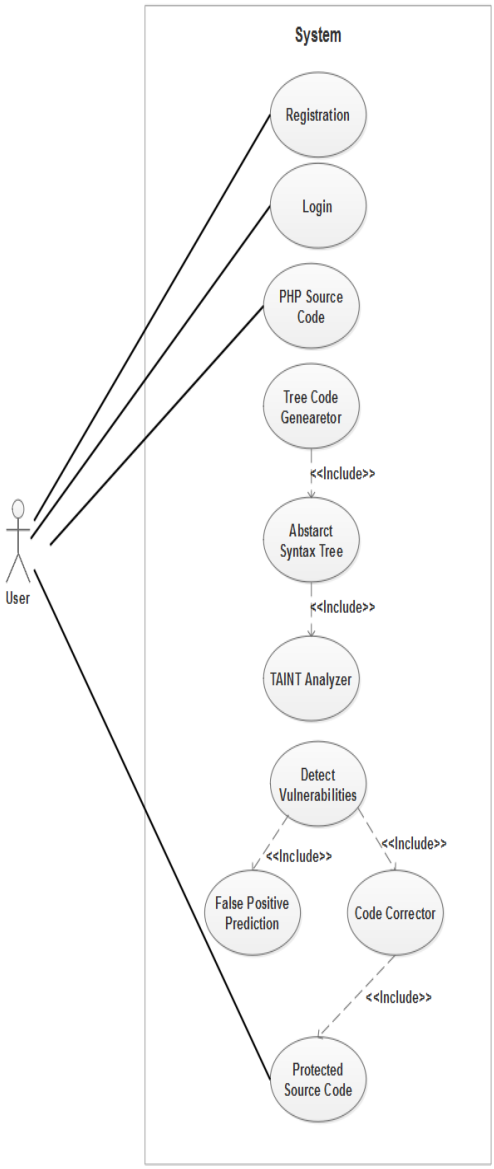

Fig -2: Use Case Diagram

6. CONCLUSION

This report introduces an approach for finding and correcting vulnerabilities in web applications and an instrument that executes the approach for PHP projects and info approval vulnerabilities. The approach and the tool search vulnerabilities utilizing a mix of two systems: static source code investigation, and data mining. Data mining is utilized to recognize false positives utilizing the main 3 machine learning classifiers, and to justify their nearness utilizing an acceptance control classifier. All classifiers were chosen after a careful examination of a few choices. Note that this mix of location procedures can't provide entirely correct out comes. The static investigation issue is undecidable , and turning to information mining can't circumvent this undecidability, yet just give probabilistic results. The tool corrects the code by inserting fixes, i.e., purification and approval capacities. Testing is utilized to check if the fixes really remove the vulnerabilities and do not compromise the (correct) behavior of the applications. The tool was tried different things with utilizing engineered code with vulnerabilities embedded deliberately, and with an extensive number of open source PHP applications. It was likewise contrasted and two source code investigation instruments: Pixy, and PHP Miner-II. This evaluation recommends that the device can identify and revise the vulnerabilities of the classes it is modified to handle. It could discover 388 vulnerabilities in 1.4 million lines of code. Its exactness and accuracy were around 5%better than PHP Miner-II's, and 45% better than Pixy's.

References

[1] Ibéria Medeiros, Nuno Neves,"Detecting and Removing Web Application Vulnerabilities with Static Analysis and Data Mining",vol.65,March 2016.

[2] L. K. Shar, H. B. K. Tan, and L. C. Briand, "Mining SQL injection and cross site scripting vulnerabilities using hybrid program analysis," in 2013

[3] Sonam Panda, Ramani S, "Protection of Web Application against SQL Injection Attacks" In 2013

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

[5] AshwaniGarg, Shekhar Singh, "A Review on Web Application Security Vulnerabilities" in 2013.

[6] L. K. Shar and H. B. K. Tan, "Automated removal of cross site scripting vulnerabilities in web applications," in 2012.

[7] L. K. Shar "Predicting common web application vulnerabilities from input validation and sanitization code patterns," in 2012

[8] R. Banabic and G. Candea, "Fast black-box testing of system recovery code," in 2012.

[9] Y.W. Huang et al., "Web application security assessment by fault injection and behavior monitoring," in 2003

[10] N. L. de Poel, "Automated security review of PHP web applications with static code analysis," in May 2010.

[11] Y.W. Huang et al., "Securing web application code by static analysis and runtime protection," in 2004.