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 6, June 2017

Detection and Removal of Web Application Vulnerabilities Using WAP Tool

Sujata Jadhav, Dr. B.K. Sarkar

M.E. Student, Department of Computer Engineering, PVPIT, Pune, India

HOD, Department of Computer Engineering, PVPIT, Pune, India

ABSTRACT: Despite the fact that a vast research exertion on web application security has been continuing for over 10 years, the security of web applications keeps on being a testing issue. A vital part of that issue gets from defenceless source code, regularly written in perilous dialects like PHP. Source code static examination apparatuses are an answer for discovers vulnerabilities, yet they have a tendency to create false positives, and require impressive exertion for developers to physically settle the code. Here investigate the utilization of a blend of techniques to find vulnerabilities in source code with less false positives. Here join spoil investigation, which discovers hopeful vulnerabilities, with information mining, to foresee the presence of false positives. This approach unites two methodologies that are obviously orthogonal: people coding the information about vulnerabilities (for pollute examination), joined with the apparently orthogonal approach of consequently getting that learning (with machine learning, for information mining). Given this upgraded type of location, we propose doing programmed code remedy by embeddings settles in the source code. This approach was executed in the WAP Tool, and an exploratory assessment was performed with a extensive arrangement of PHP applications. This tool discovered 388 vulnerabilities in 1.4 million lines of code. Its exactness and accuracy were roughly 5% superior to Php MinerII's and 45% superior to Pixy's.

KEYWORDS: Automatic protection, data mining, false positives, input validation vulnerabilities, software security, source code static analysis, web applications

I. INTRODUCTION

Here trying to explore an approach for automatically protecting web applications while keeping the programmer in the loop. The approach consists in analysing 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 contributes directly for 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.

Here explore the use of a novel combination of methods to detect this type of vulnerabilities: static analysis and 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 [18]. This problem is particularly difficult with languages such as PHP that are weakly typed and not formally specified [7]. 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 opposite approaches: humans coding the knowledge about vulnerabilities (for taint analysis) versus 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, Naïve Bayes, Bayes Net, MLP, SVM, and Logistic Regression. Moreover, for every vulnerability classified as false positive, 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 vulnerability data.

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 6, June 2017

Ensuring that the code correction is done correctly requires assessing that the vulnerabilities are removed and that the correct behaviour of the application is not modified by the fixes. Here propose using program mutation and regression testing to confirm, respectively, that the fixes do the function to what they are programmed to (blocking malicious inputs) and that the application remains working as expected (with benign inputs). Notice that we do not claim that our approach is able to correct any vulnerability, or to detect it, only the input validation vulnerabilities it is programmed to deal with.

The paper also describes the design of the Web Application Protection (WAP) tool that implements our approach [1]. WAP analyses and removes input validation vulnerabilities from code1 written in PHP 5, which according to a recent report is used by more than 77% of the web applications. WAP covers a considerable number of classes of vulnerabilities: SQL injection (SQLI), cross-site scripting (XSS), remote file inclusion, local file inclusion, directory traversal/path traversal, source code disclosure, PHP code injection, and OS command injection. The first two continue to be among the highest positions of the OWASP Top 10 in 2013 [16], whereas the rest are also known to be high risk, especially in PHP. Currently WAP assumes that the background database is MySQL, DB2 or PostgreSQL. The tool might be extended with more flaws and databases, but this set is enough to demonstrate the concept. Designing and implementing WAP was a challenging task. The tool does taint analysis of PHP programs, a form of data flow analysis. To do a first reduction of the number of false positives, the tool performs global, inter procedural and context-sensitive analysis, which means that data flows are followed even when they enter new functions and other modules (other files). This involves the management of several data structures, but also to deal with global variables (that in PHP can appear anywhere in the code, simply by preceding the name with global or through the $_GLOBALS array) and resolving module names (which can even contain paths taken from environment variables). Handling object orientation with the associated inheritance and polymorphism was also a considerable challenge.

Here evaluated the tool experimentally by running it with both simple synthetic code and with 45 open PHP web applications available in the internet, adding up to more than 6,700 files and 1,380,000 lines of code. Our results suggest that the tool is capable of finding and correcting the vulnerabilities from the classes it was programmed to handle. 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) combination of taint analysis and data mining techniques to identify vulnerabilities with low false positives;

(3) tool that implements that approach for web applications written in PHP with several database management systems;

(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.

II. REVIEW OF LITERATURE

Web application security[1] 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 program 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.

Web application security is an important problem[21] in today's internet. A major cause of this status is that many programmers do not have adequate knowledge about secure coding, so they leave applications with vulnerabilities. An approach to solve this problem is to use source code static analysis to find these bugs, but these tools are known to report many false positives that make hard the task of correcting the application. This paper explores the use of a hybrid of methods to detect vulnerabilities with less false positives. After an initial step that uses taint analysis to flag candidate vulnerabilities, this approach uses data mining to predict the existence of false positives. This approach reaches a trade-off between two ap parently opposite approaches: humans coding the knowledge about

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 6, June 2017

vulnerabilities (for taint analysis) versus automatically obtaining that knowledge (with machine learning, for data mining). Given this more precise form of detection, we do au- tomatic code correction by inserting fixes in the source code. The approach was implemented in the WAP tool 1 and an experimental evaluation was performed with a large set of open source PHP applications.

Injection vulnerabilities[22] pose a major threat to application level security. Some of the more common types are SQL injection, cross site scripting and shell injection vulnerabilities. Existing methods for defending against injection attacks, that is, attacks exploiting these vulnerabilities, rely heavily on the application developers and are therefore error-prone. This paper introduces CSSE, a method to detect and prevent injection attacks. CSSE works by addressing the root cause why such attacks can succeed, namely the ad-hoc serialization of user-provided input. It provides a platform-enforced separation of channels, using a combination of assignment of metadata to user-provided input, metadata-preserving string operations and context-sensitive string evaluation. CSSE requires neither application developer interaction nor application source code modifications. Since only changes to the underlying platform are needed, it effectively shifts the burden of implementing countermeasures against injection attacks from the many application developers to the small team of security-savvy platform developers. This method is effective against most types of injection attacks, and show that it is also less error-prone than other solutions proposed so far. Here have developed a prototype CSSE implementation for PHP, a platform that is particularly prone to these vulnerabilities. Here the prototype is used with php BB, a well-known bulletin-board application, to validate our method. CSSE detected and prevented all the SQL injection attacks could reproduce and incurred only reasonable run-time overhead.

Fault injection---a key technique[7] for testing the robustness of software systems---ends up rarely being used in practice, because it is labor-intensive and one needs to choose between performing random injections (which leads to poor coverage and low representativeness) or systematic testing (which takes a long time to wade through large fault spaces). As a result, testers of systems with high reliability requirements, such as MySQL, perform fault injection in an ad-hoc manner, using explicitly-coded injection statements in the base source code and manual triggering of failures.
This paper introduces AFEX, a technique and tool for automating the entire fault injection process, from choosing the faults to inject, to setting up the environment, performing the injections, and finally characterizing the results of the tests (e.g., in terms of impact, coverage, and redundancy). The AFEX approach uses a metric-driven search algorithm that aims to maximize the number of bugs discovered in a fixed amount of time. We applied AFEX to real-world systems---MySQL, Apache httpd, UNIX utilities, and Mongo DB---and it uncovered new bugs automatically in considerably less time than other black-box approaches.

The state of web security remains troubling as web applications continue to be favourite targets of hackers. Static analysis tools are important mechanisms [23] for programmers to deal with this problem as they search for vulnerabilities automatically in the application source code, allowing programmers to remove them. However, developing these tools requires explicitly coding knowledge about how to discover each kind of vulnerability. This paper presents a new approach in which static analysis tools learn to detect vulnerabilities automatically using machine learning. The approach uses a sequence model to learn to characterize vulnerabilities based on a set of annotated source code slices. This model takes into consideration the order in which the code elements appear and are executed in the slices. The model created can then be used as a static analysis tool to discover and identify vulnerabilities in source code. The approach was implemented in the DEKANT tool and evaluated experimentally with a set of open source PHP applications and Word Press plugins, finding 16 zero-day vulnerabilities.

III. SYSTEM ARCHITECTURE

A. The Approach

Data flow security with regards to web applications is the proposed approach, generally worried with the server-side, which is regularly composed in PHP, Java, or Perl. Attacks against web vulnerabilities can be communicated as far as infringement of data stream security. The instrument identifies the likelihood of the presence of the data streams spoke to in the figure, and changes the source code to avert them. The approach can be actualized as a series of steps.

1) Taint Analysis: parse the source code, create abstract syntax tree (AST), doing corrupt examination in view of the AST, and producing trees depicting hopeful vulnerable control-flow ways (from a entry point to a touchy sink).

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 6, June 2017

2) Data mining: collecting parameters from the applicant vulnerable control-flow ways, and utilizing the main 3 classifiers to anticipate if every competitor defencelessness is a false positive or not. Within the sight of a false positive, utilize induction rules to display the connection between the characteristics that ordered it.

3) Code Correction: given the control-stream way trees of vulnerabilities (anticipated not to be false positives), distinguishing the vulnerabilities, the fixes to embed, and the spots where they must be embedded; surveying the probabilities of the vulnerabilities being false positives; and adjusting the source code with the fixes.

4) Feedback: giving feedback to the software engineer in view of

the information gathered in the past strides (vulnerable paths, vulnerabilities, fixes, false positive likelihood, and the characteristics that grouped it as a false positive).

5) Testing: Get higher assurance with two types of testing, particularly program transformation to confirm if the fixes do their capacity, and relapse testing to check if the conduct of the application continues as before with generous sources of info.

B. Architecture

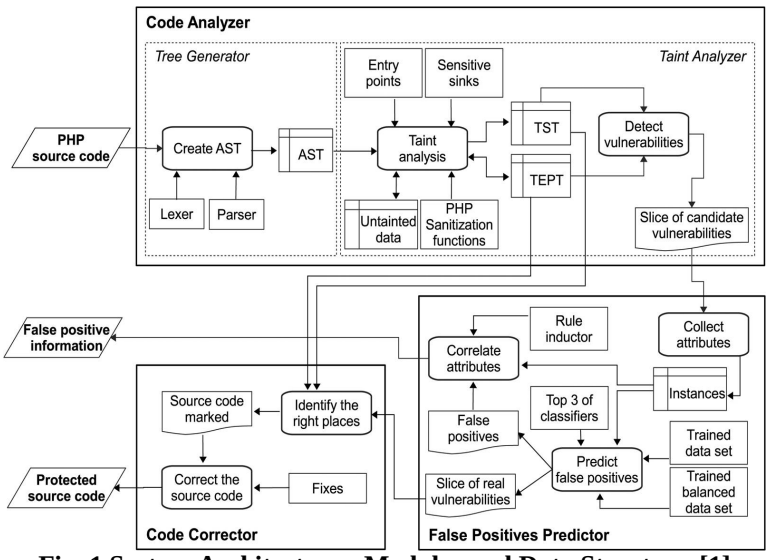

Fig. 1 System Architecture : Modules and Data Structures[1]

Fig. 1 demonstrates the engineering that executes steps 1 to 4 of the approach (testing is not spoke to). It is made out of three modules: code analyser, false positives indicator, and code corrector. The code analyser first parses the PHP source code, and creates an AST. At that point, it utilizes tree walkers to do pollute investigation, i.e., to track if information provided by clients through the section focuses achieves touchy sinks without purification. While doing this examination, the code analyser produces corrupted image tables also, polluted execution way trees for those ways that connection section focuses to touchy sinks without legitimate cleansing. The false positives indicator proceeds where the code analyser stops. For each touchy sink that was observed to be come to by polluted input, it tracks the way from that sink to the section point utilizing the tables and trees just said. Along the track ways (slice candidate vulnerabilities in the figure), the vectors of traits (instances) are gathered and arranged by the data mining calculation as genuine positive (real vulnerability), or false positive (not a real vulnerability). Take note of that we utilize the terms genuine positive what's more, false positive to express that a caution raised by the corrupt analyser is right (real vulnerability) or off base (not a genuine vulnerability). These terms don't mean the genuine and false positive rates coming about because of the information mining calculation, which measure its exactness and precision. The code

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 6, June 2017

corrector picks the ways named genuine positives to flag the spoiled contributions to be sterilized utilizing the tables and trees said above. The source code is revised by embeddings fixes, e.g., calls to disinfection capacities. The engineering depicts the approach, however speaks to likewise the engineering of the WAP tool.

IV. SYSTEM ANALYSIS

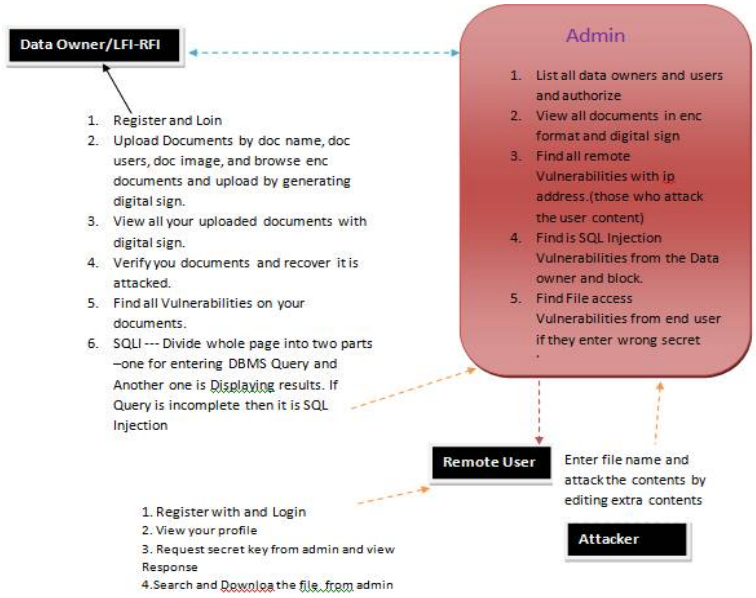

Fig. 2: Modules

SOFTWARE REQUIREMENTS

• Operating system: Windows XP or Windows 7, Windows 8.
• Coding Language: Java / J2EE (Jsp, Servlet)
• Data Base: My Sql Server
• Documentation: MS Office
• IDE: Eclipse Galileo
• Development Kit: JDK 1.6
• Server: Tomcat 6.0

V. MODULES

- **Admin**

In this module, admin has to login with valid username and password. After login successful he can do some operations such as view all user, their details and authorize them , view all owners, their details and authorize them, view all attackers details(like ip address and host name), view all sql injection vulnerabilities and block them(those who are execute wrong query), view all file access vulnerabilities of users(those who are used wrong secret key) , view all blocked data owners , view unblock requests and unblock them, view all secret key requests and generate, view all users download history, view SQL Injection Vulnerabilities in chart, View number of remote vulnerabilities in chart.

- **User**

In this module, there are n numbers of users are present. User should register before doing some operations. After registration successful he can login by using valid user name and password. Login successful he will do some operations like view profile details, request secret key and view response, search documents and download by entering secret key.

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 6, June 2017

- **Data Owner**

In this module, there are n numbers of owners are present. Owner should register before doing some operations. After registration successful he can login by using valid user name and password. Login successful he will do some operations like view profile details, Upload documents and generate digital sign, view uploaded documents, verify his documents and recover if it is attacked, view all on his documents like(downloads and attacked details), Execute SQL queries if query is incomplete the it is SQL Injection Vulnerabilities.

- **Attacker**

Attacker searches the documents and edit document by changing content.

VI. CLASSIFIER EVALUATION MATRICS

false positive - FP
true positive - not FP
to express that an alarm raised by the taint analyzer is incorrect (not a real vulnerability) or correct (a real vulnerability).
The terms, false positive (fp), and true positive (tp)and false negative (fn), and true negative (tn), for the output of the next stage, FP classifier.
tpp - True positive rate of prediction measures quality of
Classifier:

$$tpp = tp/(tp + fn)$$

fpp - False positive rate of prediction measures how the classifier deviates from the correct classification of a candidate vulnerability as FP:

$$fpp = fp/(fp + tn).$$

prfp - Precision of prediction measures the actual FPs
Correctly predicted in terms of the percentage of total number of FPs:

$$prfp = tp/(tp + fp)$$

pd - Probability of detection measures how the classifier is
good at detecting real vulnerabilities:

$$pd = tn/(tn + fp)$$

pfd - Probability of false detection measures how the classifier
Deviates from the correct classification of a candidate vulnerability that was a real vulnerability:

$$pfd = fn/(fn + tp)$$

prd - Precision of detection measures the actual vulnerabilities (not FPs) that are correctly predicted in terms of a percentage of the total number of vulnerabilities:

$$prd = tn/(tn + fn)$$

acc - Accuracy measures the total number of instances well
Classified:

$$acc = (tp + tn)/(tp + tn + fp + fn)$$

pr - Precision measures the actual FPs and vulnerabilities
(not FPs) that are correctly predicted in terms of a percentage
of the total number of cases:

$$pr = average(prfp, prd)$$

kappa - Kappa statistic measures the concordance between
the classes predicted and observed. It can be stratified into six
categories: worst, bad, reasonable, good, very good, excellent.
$kappa = (po - pe)/(1 - pe)$, where $po = acc$, and $pe = (P * P' + N * N')/(P + N)^2$ to $P = (tp + fn)$, $P' = (tp + fp)$, $N = (fp + tn)$, and $N' = (fn + tn)$.

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 6, June 2017

VII. IMPLIMENTATION

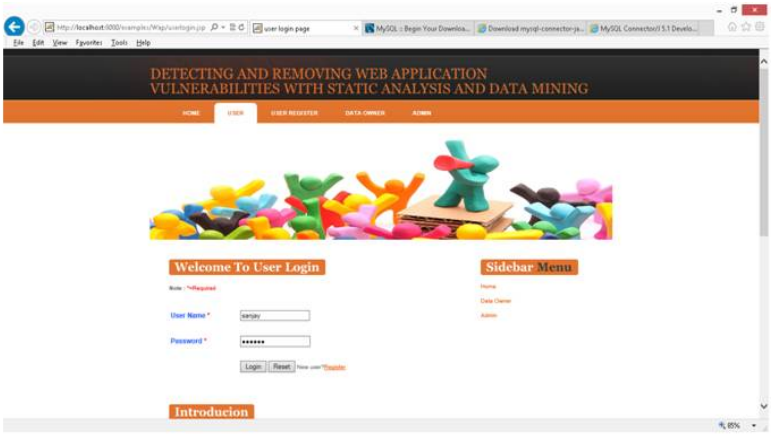

Fig. 3: User Login

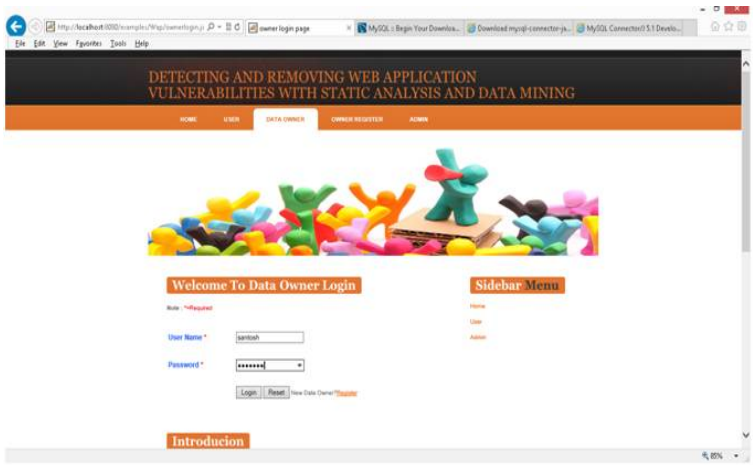

Fig 4: Data Owner Login

Fig. 5: Uploading Documents

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 6, June 2017

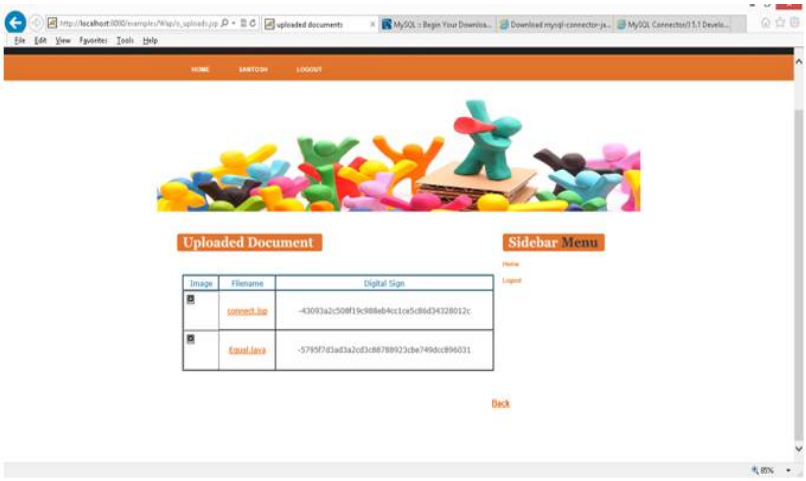

Fig. 6: Viewing Uploaded Documents

VIII. RESULT

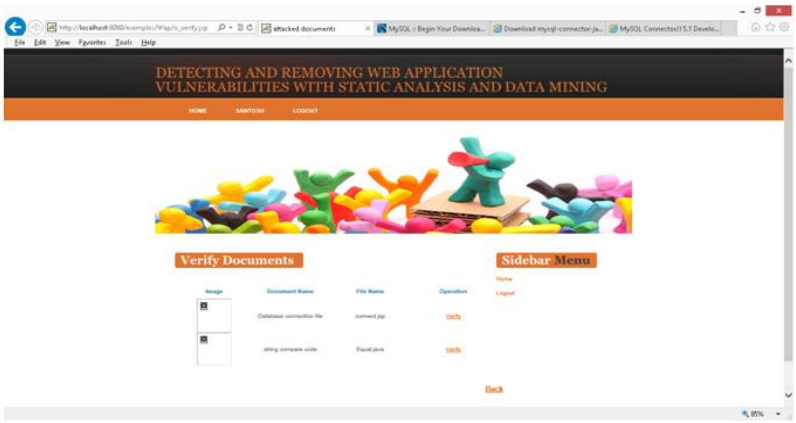

Fig. 7: Verify Documents

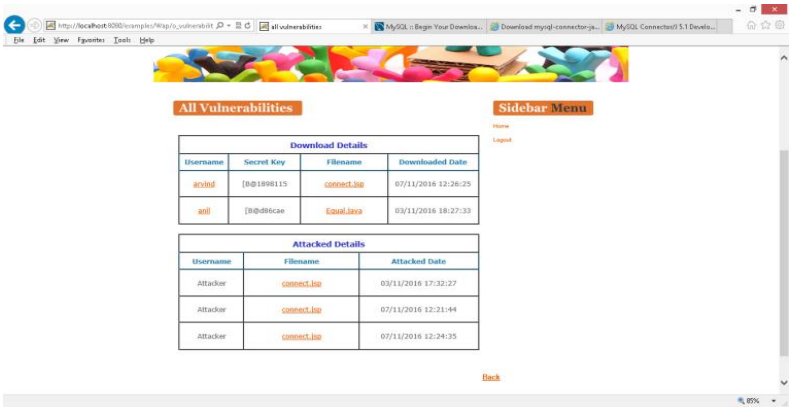

Fig. 8: Viewing Vulnerabilities

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 6, June 2017

IX. CONCLUSION

The paper displays an approach for finding and revising vulnerabilities in web applications and a device that actualizes the approach for PHP projects and information approval vulnerabilities. The approach and the device scan for vulnerabilities utilizing a blend of two methods: static source code examination and information mining. Information mining is utilized to recognize false positives utilizing a main 3 of machine learning classifiers and to legitimize their nearness utilizing an enlistment lead classifier. All classifiers were chosen after an intensive correlation of a few choices. Note that this blend of location systems can't give totally rectify comes about. The static investigation issue is undecidable and the fall back on information mining can't bypass this un decidability, just give probabilistic outcomes. The instrument adjusts the code by embeddings fixes, i.e., disinfection and approval capacities. Testing is utilized to check if the fixes really expel the vulnerabilities and don't trade off the (right) conduct of the applications. The apparatus was explored different avenues regarding engineered code with vulnerabilities embedded deliberately and with a significant number of open source PHP applications. It was additionally contrasted and two source code investigation devices, Pixy and PhpMinerII. This assessment proposes that the apparatus can distinguish and remedy 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% superior to PhpMinerII's and 45% superior to Pixy's.
Future work will be on the intrusion detection systems in the cloud environment, because cloud computing is the trend, and is a major paradigm shift of computer systems.
The future work can be extended by using the combination of two or more tools of vulnerability detection. Due to which the disadvantages of one will be overcome by another. For e. g: The supervised learning technique is suitable where the past data is completely available whereas the semi-supervised learning is not adequate for this. It works efficiently on partially available data. This hybrid approach will lead to decrease in false positive and false negative rates.

X. ACKNOWLEDGMENT

It is our privileges to acknowledge with deep sense of gratitude to the people who directly or indirectly help us from their valuable suggestions and guidance throughout our course of study and timely help given to us in completion of our work.
We are highly obliged to the entire staff of the Computer Engg. Department of PVPIT College of Engg. for their kind co operation and help.
I also take this opportunity to thanks my colleague, who backed our interest by giving useful suggestions and all possible help.

REFERENCES

1. Ibéria Medeiros, Nuno Neves and Miguel Correia, “Detecting and Removing Web Application Vulnerabilities with Static Analysis and Data Mining” vol. 65,pp/ 54 – 69, IEEE Transactions 2016.
2. Hong Zhu and Yufeng Zhang ,”A Test Automation Framework for Collaborative Testing of Web Service Dynamic Compositions”, Vol. 5, no. 1, pp: 116 - 130, 2012.
3. T. Pietraszek and C. V. Berghe, “Defending against injection attacks through context-sensitive string evaluation,” in Proc. 8th Int. Conf. Recent Advances in Intrusion Detection, 2005, pp. 124–145.
4. Y. Shin, A. Meneely, L. Williams, and J. A. Osborne, “Evaluating complexity, code churn, developer activity metrics as indicators of software vulnerabilities,” IEEE Trans. Software Eng., vol. 37, no. 6, pp. 772–787, 2011.
5. 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.
6. W. Halfond and A. Orso, “AMNESIA: analysis and monitoring for neutralizing SQL-injection attacks,” in Proc. 20th IEEE/ACM Int. Conf. Automated Software Engineering, Nov. 2005, pp. 174–183.
7. R. Banabic and G. Candea, “Fast black-box testing of system recovery code,” in Proc. 7th ACM Eur. Conf. Computer Systems, 2012, pp. 281–294.
8. Y.-W. Huang et al., “Web application security assessment by fault injection and behavior monitoring,” in Proc. 12th Int. Conf. World Wide Web, 2003, pp. 148–159.
9. Y.-W. Huang et al., “Securing web application code by static analysis and runtime protection,” in Proc. 13th Int. Conf. World Wide Web, 2004, pp. 40–52.

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 6, June 2017

10. N. Jovanovic, C. Kruegel, and E. Kirda, "Precise alias analysis for static detection of web application vulnerabilities," in Proc. 2006 Workshop Programming Languages and Analysis for Security, Jun. 2006, pp. 27–36.
11. U. 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.
12. W. Landi, "Undecidability of static analysis," ACM Lett. Program. Lang. Syst., vol. 1, no. 4, pp. 323–337, 1992.
13. 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.
14. WAP tool website [Online]. Available: http://awap.sourceforge.net/
15. Imperva, Hacker intelligence initiative, monthly trend report #8, Apr. 2012.
16. E. Merlo, D. Letarte, and G. Antoniol, "Automated Protection of PHP Applications Against SQL Injection Attacks," in Proc. 11th Eur. Conf. Software Maintenance and Reengineering, Mar. 2007, pp. 191–202.
17. R. S. Sandhu, "Lattice-based access control models," IEEE Comput., vol. 26, no. 11, pp. 9–19, 1993.
18. Sabelfeld and A. C. Myers, "Language-based information-flow security," IEEE J. Sel. Areas Commun., vol. 21, no. 1, pp. 5–19, 2003.
19. J. C. Huang, Software Error Detection through Testing and Analysis. New York, NY, USA: Wiley, 2009.
20. 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.
21. Ibéria Medeiros, Nuno F. Neves, Miguel Correia "Automatic Detection and Correction of Web Application Vulnerabilities using Data Mining to Predict False Positives" Proceedings of the 23rd international conference on world wide web Pages 63-74 ACM 2014
22. Tadeusz Pietraszek, Chris Vanden Berghe, "Defending against injection attacks through context-sensitive string evaluation", ACM 2005, RAID'05 Proceedings of the 8th international conference on Recent Advances in Intrusion Detection, Pages 124-145
23. Ibéria Medeiros, Nuno Neves, "DEKANT: a static analysis tool that learns to detect web application vulnerabilities", ISSTA 2016 Proceedings of the 25th International Symposium on Software Testing and Analysis, Pages 1-11

BIOGRAPHY

Sujata Jadhav Student of Master of Engineering in Computer Engineering from the PVPIT college of engineering pune.

Prof. B.K. Sarkar HOD at the PVPIT college of engineering, Pune for the Department of Computer Engineering