Prediction of Future Vulnerability Discovery in Software Applications using Vulnerability Syntax Tree (PFVD-VST)

Kola Periyasamy[1] and Saranya Arirangan[2]
[1]Department of Information Technology, Madras Institute of Technology, India
[2]Department of Information Technology, SRM Institute of Engineering and Technology, India

Abstract: *Software applications are the origin to spread vulnerabilities in systems, networks and other software applications. Vulnerability Discovery Model (VDM) helps to encounter the susceptibilities in the problem domain. But preventing the software applications from known and unknown vulnerabilities is quite difficult and also need large database to store the history of attack information. We proposed a vulnerability prediction scheme named as Prediction of Future Vulnerability Discovery in Software Applications using Vulnerability Syntax Tree (PFVD-VST) which consists of five steps to address the problem of new vulnerability discovery and prediction. First, Classification and Clustering are performed based on the software application name, status, phase, category and attack types. Second, Code Quality is analyzed with the help of code quality measures such as, Cyclomatic Complexity, Functional Point Analysis, Coupling, Cloning between the objects, etc,. Third, Genetic based Binary Code Analyzer (GABCA) is used to convert the source code to binary code and evaluates each bit of the binary code. Fourth, Vulnerability Syntax Tree (VST) is trained with the help of vulnerabilities collected from National Vulnerability Database (NVD). Finally, a combined Naive Bayesian and Decision Tree based prediction algorithm is implemented to predict future vulnerabilities in new software applications. The experimental results of this system depicts that the prediction rate, recall, precision has improved significantly.*

Keywords: *Vulnerability discovery, prediction, classification and clustering, binary code analyzer, code quality metrics, vulnerability syntax tree.*

Received October 30, 2014; accepted April 21, 2016

1. Introduction

Software vulnerabilities degrade the performance of the software's, systems and integrated applications. The security of software system is the considerable scrutiny of these technological years. Evaluation of security systems and vulnerability detection and prevention techniques were developed quantitatively. Vulnerable software‘s lead to crack the system or destroy the user's goal. Discovering vulnerability in software components is not a simple one, it has many measurements, analysis, conceptualization and techniques are utilized. The inspected data [9] is tested in five mostly used operating systems and it includes three consecutive versions of Windows and two versions of Red Hat Linux. For vulnerability data discovery, we need standard database. The National Vulnerability Database (NVD) is a public data source that sustains homogeneous information about reported software vulnerabilities. In this database more than 43,000 software vulnerabilities are affecting 17,000 software applications [14].

This information is possibly valuable in understanding trends and patterns in software vulnerabilities. NVD data sets are used to examine the pattern of new vulnerability and new software application flaws. Currently, much security software is available to detect the vulnerable actions present in the system, but it needs history information of that particular action. NVD data's are useful for preventing software's from the vulnerable achievements.

Vulnerability prediction is a vital provision for preventing applications from security harms. There is an attempt to build a prediction model for the attribute [7] Time to Next Vulnerability (TTNV), the time that it will take before the next vulnerability about a particular application will be found. The predicted [12] TTNV metrics could be translated into the likelihood that a zero - day vulnerability exists in the software.

The widely used networking applications like [16] Mozilla, Apache httpd and Apache Tomcat are largely introduced the vulnerabilities. By seeing at 292 vulnerability reports for Mozilla, 66 for Apache, and 21 for Tomcat, and discover the number of people committing vulnerability attack changes proportionally to the number of vulnerability attacks for Mozilla and Tomcat, but not for Apache httpd.

Dynamic taint analysis is used to automatic attack detection [15] and overwrites of attacks. This system does not work with source code of the application; it will convert the input to binary at run time. In this paper first, I described about related work then, proposed system with all modules, experimental analysis with results. Finally conclusion will express the performance of the system.

2. Related Work

To compare the prediction capability of different VDMs, fit the model to the first half of the vulnerabilities, and use it to predict the second half of the vulnerabilities. The prediction will be performed without using the Vulnerability database or history information.

Alabsi and Naoum [2] proposed vulnerability Discovery Models (VDMs) to model the vulnerability discovery and has been fitted to vulnerability data against calendar time. The models have been shown to fit very well with the prediction capabilities that these models offer by evaluating the accuracy of predictions made with partial data. This [4, 5] model examined both the recently proposed logistic model and a new linear model. In addition to VDMs, They regard as static approaches to estimating some of the key attributes of the vulnerability detection process, presenting an inert approach to calculating the initial values of some of the VDM's parameters. A relation is an ordered pair [6] of members, in which data is owned from the source (first member) to the target (second member). There are four types of relation as follow: parameter passing, function return, data read, and data write. But this system is not suitable for nesting level metrics because it does not focus on the entire code base of several versions of Firefox.

Alhazmi and Malaiya [3] proposed the prediction of vulnerabilities in the Hyper Text Transfer Protocol (HTTP) server can permit us to assess the security risk related to its exercise. Vulnerability discovery models have newly been proposed which can be used to estimate the future vulnerabilities expected to be discovered. Both long time predictions relating several years and short time predictions for the subsequent year are considered. Kim *et al*. [10] proposed the vulnerability detection method for a plan that described a rate at which the security vulnerabilities were discovered. Thus, there is a need to extend a model of the discovery process that can predict the total of vulnerabilities that are likely to be discovered in a particular time frame. Recent studies have produced vulnerability discovery models that are fitting for an exact version of the software. However, these [11] models may not accurately estimate the vulnerability discovery rates for software when we believe consecutive versions.

Ozment [17] proposed a usual set of definitions significant for measuring characteristics of vulnerabilities and their discovery procedure and it illustrates the speculative requirements of VDMs. It particularly the assumption that vulnerability discovery is an independent process. A Genetic algorithm is used for tracing the misbehaviours in intrusion [8] detection systems; it has a fitness function to measure two types of attack detection. One detects the attack patterns with the known patterns, by matching the existing records and new records. Second, measure [13] the deviation from the normal patterns. For this type of problem fitness function is used. [1] Machine Learning (ML) algorithm helps to do better prediction in network vulnerabilities. ML can be applicable in many fields like web search engine, mail servers, weather sites etc.

3. Proposed System

The vulnerability prediction scheme is essential to protect software applications from failures. Users always prefer the best and reliable software to develop their products. Some of the software invaders inject vulnerabilities into the applications to corrupt the progression of software. This type of deeds is trimming down the reliability of the software. We need a solution to overcome this problem of developers and users. Without database or history information about the vulnerabilities, vulnerability discovery and prediction is performed.

We predict future vulnerabilities from previously released software applications, based on time and number of vulnerabilities present in the application. Each type of applications is considered to discover the vulnerability in different versions. Predicting Vulnerabilities in software application are plastered into five key concepts such as Classification and clustering, code quality analyzer, binary code analyzer, vulnerability syntax tree construction, vulnerability prediction. Figure 1 illustrates the system architecture of vulnerability prediction.

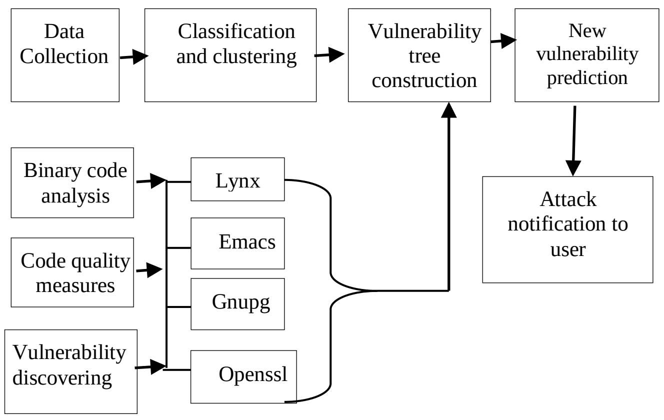
Data Collection
Classification and clustering
Vulnerability tree construction
New vulnerability prediction
Attack notification to user
Binary code analysis
Code quality measures
Vulnerability discovering
Lynx
Emacs
Gnupg
Openssl

Figure 1. System architecture for vulnerability prediction

3.1. Classification and Clustering

Different vulnerabilities are discovered from various software applications. National vulnerability database has a variety of software vulnerabilities and it has been discovered. There is a need to classify the vulnerabilities present in the software's based on the similarities. Initially need to apply the pre-processing (filter) techniques to NVD data samples to select the clean the best attribute for classification.

Naive Bayesian (NB) classification algorithm is one of the efficient and traditional approaches to deal with complex queries. The Naive Bayesian algorithm is programmed into weka data mining tool which makes it easy to handle large number of data at the minimal time. Weka tool classifies the vulnerabilities based on the software application name, status, phase, category and attacks present. We have given 250 records to classify the attacks based on the attack behaviour and it increases recall and precision. Four main attacks such as buffer overflow, remote attack, denial of service, SQL injection attack and DNS poisoning are considered based on the frequency of occurrence in software's.

3.2. Code Quality Analyzer

Complex code has the high probability of vulnerability and it is hard to fix the bugs in it. Many code quality measures are used to evaluate the software reliability. Here, we have used four main complexity metrics such as, Cyclomatic complexity, coupling, cloning between the objects, function point analysis.

We have used the following formula for Cyclomatic complexity analysis of software engineering concepts,

$$M = E - N + 2P \tag{1}$$

In Equation (1), E refers to Number of Edges, N denotes Number of Nodes and P relates to Number of Connected components. Function point analysis is calculated using Equation (2), this will help user to understand the nature of code quality.

$$FP = UFP * VAF \tag{2}$$

Where UFP is Unadjusted Functional Point count, VAF is Value Adjustment Factor given by Equation (3).

$$VAF = 0.65 + (0.01 * TDI) \tag{3}$$

Where, represents the sum of 14 characteristics Total Degree of Influence (TDI). These equations were adapted from software engineering for source code analysis. It incorporates many risk factor evaluations to significantly point the false bug codes.

Based on the code's quality, it is categorized into four types such as info, trace (normal), high and critical. With this categorization process, possibility of vulnerability is measured and intimation to the users is provided. The critical and high risk codes have high probability of attack injection. In addition, PMD plugin is used to cover all types of code quality metrics in order to identify the risk and complexity of the code. It is one of the bugs (code rules) fixing plugin, which is enabled in netbeans and eclipse platform IDs. PMD plugin is totally wrapped into 64 coding rule set to protect the software's. We have used this plugin to improve code quality to obtain highly secured as well as reliable software's. PMD plugin is an easy way to capture the bad quality of code and it supports new types of rule set (user defined rules). So software developers can create a protected environment for the users.

3.3. Binary Code Analyzer

Instead of source code, binary code analysis is done with the proposed Genetic Algorithm Based Binary Code Analysis (GABCA) as Algorithm1. Software source code is converted into binary form to identify the modification or error bit in the core parts. Each input is deeply tested for match with the correct format. Here we have injected the attack code to the source code of the application. Thus attack code is generated based on the significant features of the attacks present in the software's. Genetic algorithm is the best way to identify the binary code behaviour by incorporating crossover and mutation techniques. Attack injected binary code is randomly generated to form 50 individuals and are given to the fittest function. Fittest value is calculated based on the number of bits matching to the original individual. Range is defined on the fittest value to take the best or the fittest individual for further processing. The individuals are processed up to (N) number of generations. In each generation, one best fittest value (F) is selected. Then, the selected best individual is made to perform crossover c with the randomly selected population. Mutation is carried for this individual and again fitness value is calculated. Based on the fitness value, the possibility of vulnerability can be identified. The flow of binary code analysis is explained in the following Algorithm 1 Genetic based Binary Code Analyzer (GABCA). This algorithm depicts the attack type which is presented in software codes.

Algorithm 1: Genetic based Binary Code Analyzer (GABCA)

```
Input: Source code
Output: Attack Type (defined)
Step 1: Process: Fitness value F= max (Fittest fit)
Step 2: Fittest=new pop ∩ original pop
Step 3: Indiv 1= Tournament selection € original
            population
Step 4: Indiv 2= Tournment selection € Fitness
Step 5: Crossover c= indiv 1 U indiv 2
Step 6: C= attack code U c
Step 7: Mutation= crossover (C) € Random bit
Step 8: Again Fitness ()
Step 9: Range R
```

3.4. Vulnerability Syntax Tree Construction

The extracted features of classification such as name, status, phase, category and attack types are considered as key attributes for Vulnerability Syntax Tree (VST). The root node of the tree is defined as base attribute, which contains a unique value of application name. The root node is divided into a number of sub-trees based on application status, which are of two types such as candidate and entry. Then the sub-tree is recursively divided into many sub-trees based on phase and category of the application. Finally the leaf node indicates the attack type which is present in that particular application. Here the phase attribute includes 8 bit numeric value, the first four bits indicate the year of the application published and next four bits indicate the application phase.

Application category is divided into two ways, one is Standard Format (SF) and another is Extended Format (XF). Decision tree is the classification technique used for prediction. All level of leaf nodes are filled by splitting the decisions. Each level of the tree is compared with the known or unknown samples to predict new vulnerabilities. Predicting the vulnerability in a software application is significant for ensuring security in the system. The predicted vulnerabilities are taken and stored into the vulnerability database for further reference.

3.5. Vulnerability Prediction

Predicting vulnerability in software is quite difficult, because some of the malicious software appear to be reliable. Most of the software are published through the internet for the sake of availability, and is prone to attack injection. Each application is nearly similar to the previous version of the software product. So analyzing the attacks in the current version of the software leads to predict the future vulnerabilities present in the next generation software applications. Figure 2 shows the internal structure of the Vulnerability Syntax Tree. Red node represents the base node (root), and it can only decide the node that needs to be split. Green node represents the derived node and green node indicates the leaf nodes.

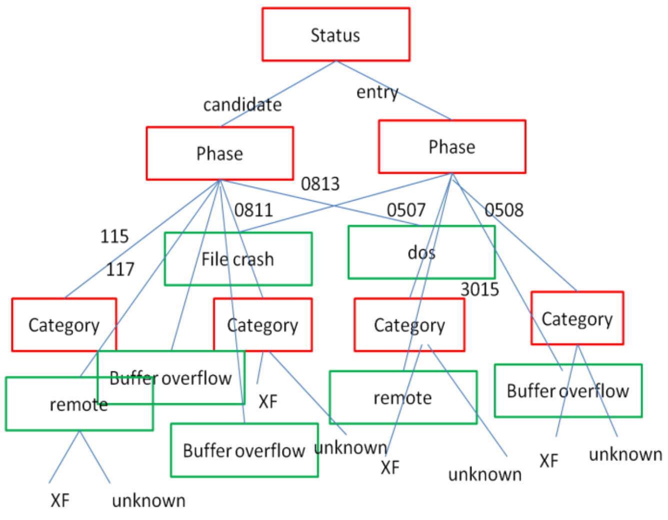

Figure 2. Vulnerability syntax tree.

Prediction is performed by discovering attack vulnerabilities. First one is training phase, it can be done by means of learning the vulnerable data behaviour based on the past history. Second, the testing phase is performed based on the learned data, it can accurately predict the attack which is present or may be present in the application. For this prediction, we have used Decision Tree Learning Applet prediction tool. This tool will improve the prediction rate by reducing the sum of mean square error rate and sum of absolute error rate.

4. Experimental Analysis

Vulnerability data are collected from the National Vulnerability Database, in which some portion of data are pre-processed and it has been propelled to the Naive Bayesian Classifier to perform Classification and Clustering. The obtained results of classification and clustering accuracy are shown in Table 1. The remaining analysis and results of binary code analysis, code quality analysis and vulnerability tree predictions are explained as follows,

Table 1.Classification of vulnerability records.

Vulnerability	**Number of Records (50)**		**Number of Records (250)**	
	Training (30)	**Testing (20)**	**Training (150)**	**Testing (100)**
Buffer Overflow	0.56	0.39	0.3	0.12
Remote Attack	0.11	0.13	0.3	0.57
Denial of Service	0	0.3	0.11	0.06
SQL Injection	0	0	0.09	0.12
File Crash	0.14	0.09	0.08	0.05

Based on the correctly classified records, the True Positive (TP) and False Positive (FP) rates are calculated by following Equations (4) and (5).

True Positive Rate

$$(TPR) = TP/(TP + FN) \tag{4}$$

False Positive Rate

$$(FPR) = FP/(FP + TN) \tag{5}$$

The recall or true positive rate is defined as the amount of correctly predicted attacks to the actual size of the attack class.

The Precision (P) is the amount of attack cases that were correctly predicted relative to the predicted amount of the attack class and is calculated using the Equation (6).

$$\text{Precision } (P) = TP / (TP+FP) \tag{6}$$

In data samples, the number of records that are correctly classified based on the attributes and constraints are taken as true positives. The remaining and imperfectly classified records are considers as false records and are taken as false positives. Here, the data are clustered into five forms of attacks. Table 2 explains the classification accuracy for both the training and testing records.

Table 2. Accuracy of vulnerability data classification.

Accuracy Metric	TP Rate	FP Rate
Training Phase (100)	85 %	15 %
Testing Phase (100)	92 %	8 %

Data set is divided into multiple samples to test the classification system. Each sample is tested to evaluate Classification Accuracy Parameters (CAP) such as true positive, false positive, precision, recall and confusion matrix and is shown in Table 3.

Table 3. Classification accuracy parameters.

Number of Records	True Positive	False Positive	Precision	Recall
50	0.5	0.303	0.43	0.46
100	0.516	0.198	0.462	0.51
150	0.537	0.192	0.49	0.537
200	0.579	0.179	0.56	0.579
250	0.662	0.159	0.665	0.692

Vulnerability data records classified with the attributes of vulnerability software name, status, phase, category and attack types. This result has been obtained from the weka 3.6 data mining tool which has totally 76 classification algorithms. Comparing to other data mining tool, weka is efficient due to its high portability and ease of use. Further, weka tool is implemented with Java code and so users can easily understand the modelling techniques.

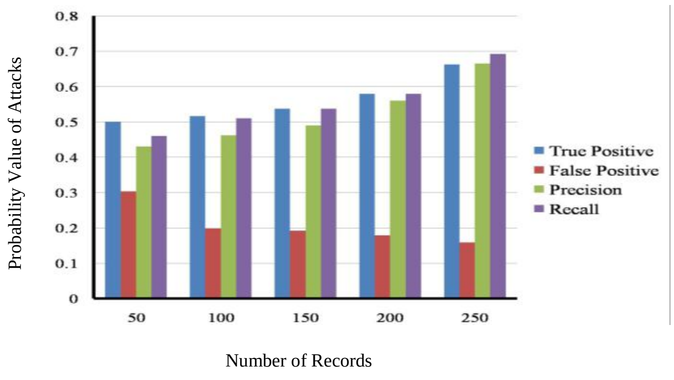

Figure 3. Accuracy measures for naïve bayesian classification.

Result of Naive Bayesian classification shown in Figure 3. For each set of samples, the rate of true positive is increased and false positive is decreased. It can be seen that the value of precision and recall increases with increased number of records. It explains that the data samples have highly related vulnerability records that match with the attack properties. Complexity of conditional codes is used to measure the complexity of software. High complexity of code leads to hijack a software user (victims) session to the other end and also crab the sensitive information. The PMD plugin supports many rule sets to reduce the complexity of code and increase the code quality. Based on the number of lines, complexity of code is increased and is shown in Figure 4.

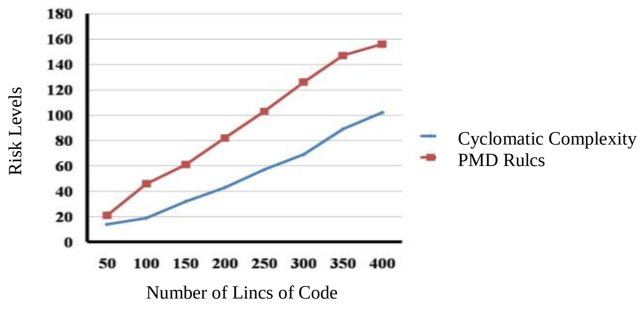

Figure 4. Code complexity measures.

Complexity of code is determined by labels such as if, while, for, case, etc. Based on the severity of code, it is classified into four levels such as info (low level), trace (moderately), high (high risk), critical (most complex and highly unstable) shown in Table 4.

Table 4. Code complexity level.

Number of Lines Code	Complexity of Code	Risk Level
20	5	Low Risk
60	14	Moderate Risk
71	16	Moderate Risk
93	19	Moderate Risk
168	52	High Risk

Table 5. Binary code analysis.

Population	Generation	Possibility of attack
50	14	Buffer Overflow (8), Remote Attack (6)
50	12	Buffer Overflow (7), Remote Attack (5)
50	19	Buffer Overflow (11), Remote Attack (6), Denial of Service (2)
50	17	Buffer Overflow (10), Remote Attack (6), Denial of Service (1)
50	26	Buffer Overflow (17), Remote Attack (7), Denial of Service (2)

Dynamic Program Analysis is used to analyze the computer software while in execution. In the form of binary code, vulnerability analysis is performed based on genetic algorithm. Single source gene namely individual generates number of population strings, and attack code is made to perform crossover and mutation with the populated strings.

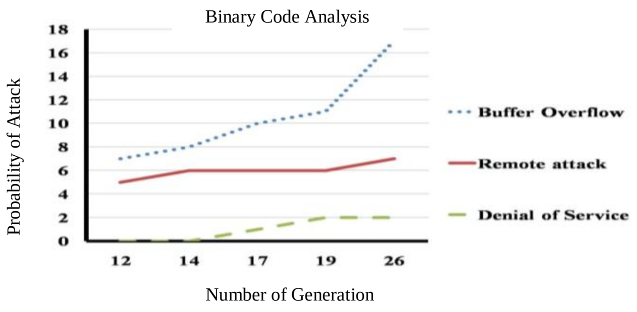

Figure 5. Vulnerability impact on the binary code analysis.

The possibility of vulnerability present in that software application is detected and denoted with name of attack and number of occurrence as shown in Table 5.Thus the attack injection in the binary code reflects the possibility of attack which may appear in the future. This system is experimented in several types of software applications such as web browser, text editors, security protocols, communication control protocols, etc. Figure 5 illustrates the possibility of attacks in common software application with varying generation count. Total vulnerabilities are denoted as N and are divided for training and testing process. Training Samples are taken from the range of 1 to N/2 and remaining N/2 to N vulnerabilities is given to the testing phase, which predicts new vulnerabilities. To fit the Vulnerability prediction model, decision tree parameters like entropy, Gini index are used which helps in tree construction.

Entropy of each data record is derived from the Equation (7).

$$\text{Entropy}(E) = \sum_{i=1}^{e} -P_i \log_2 p_i \qquad (7)$$

Pi is defined as the probability of occurrence; *c* is the number of occurrences (positive & negative). The Information Gain (IG) is calculated using Equation (8)

$$IG = Entropy(bd) - Entropy(ad) \qquad (8)$$

Where, *ad*-after decision *bd*-before decision Gini Index (I_G) is calculated by using Equation (9),

$$\text{Gini Index } (I_G) = 1 - \sum_{i=1}^{n} f_i^2 \qquad (9)$$

From this Equation (9), *i* value ranges from 1 to n and f_i represents the number of items present in this set. These equations are adopted from decision tree parameters for accurate split. The node to split during tree construction is decided by the metrics such as Entropy, Information Gain and Gini Index.

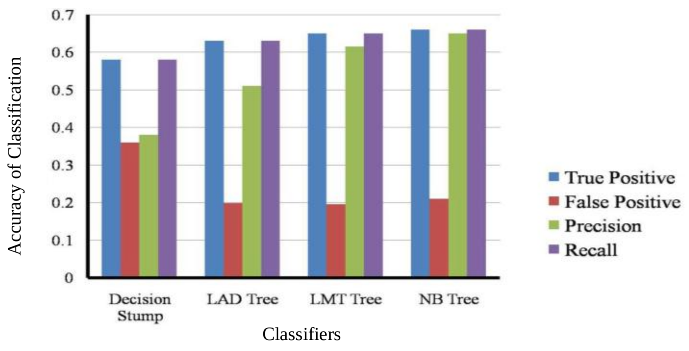

Figure 6. Tree based classification.

The vulnerability data set has been tested with many classifiers. Tree based classification obtains high classification rate as shown in figure 6. But this proposed work has considered probability model. When data set samples are increased, it adapts the training and testing rate to obtain better classification result. Probability models are highly sophisticated and it performs better prediction in large data. The remaining classification algorithms are tested on the pre-processed data set with the same samples like software application name, status, and category, phase and attack types.

5. Conclusions

Vulnerability prediction scheme is performed by the five proposed techniques and is ordered as Classification and Clustering, Code Quality, Binary Code Analysis, Vulnerability Tree Construction, Vulnerability Prediction. Binary code analysis accurately shows the error or changes in the code. Our proposed system improves the vulnerability prediction rate. Without vulnerability database or history information, it is able to predict the future vulnerabilities in software application and to recommend the best quality software to the users. This system further helps to protect the software applications from future vulnerabilities or attacks. The results obtained from data mining techniques shows that the proposed system produces significant improvement in performance compared to other methods.

References

[1] Abdulla S., Ramadass S., Altaher A., and Al-Nassiri A., "Employing Machine Learning Algorithms to Detect Unknown Scanning and Email Worms," *The International Arab Journal of Information Technology*, vol. 11, no. 2, pp. 140-148, 2014.

[2] Alabsi F. and Naoum R., "Fitness Function for Genetic Algorithm used in Intrusion Detection System," *International Journal of Applied Science and Technology*, vol. 2, no. 4, pp. 129-134, 2012.

[3] Alhazmi O. and Malaiya Y., "Measuring and Enhancing Prediction Capabilities of Vulnerability Discovery Models for Apache and IIS HTTP Servers," *in Proceedings of 17th International Symposium on Software Reliability Engineering*, Raleigh, pp. 343-352, 2006.

[4] Alhazmi O. and Malaiya Y., "Prediction Capabilities of Vulnerability Discovery Models," *in Proceedings of Annual Reliability and Maintainability Symposium*, Newport Beach, pp. 86-91, 2006.

[5] Alhazmi O., Malaiya Y., and Ray I., "Measuring, Analyzing and Predicting Security Vulnerabilities in Software Systems," *Computers and Security*, pp. 1-10, 2006.

[6] Basili V., Briand L., and Melo W., "A Validation of Object-Oriented Design Metrics as Quality Indicators," *IEEE Transactions on Software Engineering*, vol. 22, no. 10, pp. 751-761, 1996.

[7] Cavusoglu H., Cavusoglu H., and Raghunathan S., "Efficiency of Vulnerability Disclosure

Mechanisms to Disseminate Vulnerability Knowledge," *IEEE Transaction Software Engineering*, vol. 33, no. 3, pp. 171-185, 2007.

[8] Ingols K., Chu M., Lippmann R., Webster S., and Boyer S., "Modeling Modern Network Attacks and Countermeasures using Attack Graphs," *in Proceedings of Annual Computer Security Applications Conference*, Honolulu, pp. 117-126, 2009.

[9] Joh H., Kim J., and Malaiya Y., "Vulnerability Discovery Modeling using Weibull Distribution," *in Proceedings of 19th International Software Reliability Engineering*, Seattle, pp. 299-300, 2008.

[10] Kim J., Malaiya Y., and Ray I., "Vulnerability Discovery in Multi-Version Software Systems," *in Proceedings of 10th IEEE High Assurance Systems Engineering Symposium*, Plano, pp. 141-148, 2007.

[11] Kishore K., Samarjeet B., "Use of Genetic Algorithms in Intrusion Detection Systems: An Analysis," *International Journal of Applied Research and Studies*, vol. 2, no. 8, 2013.

[12] Nagappan N., Ball T., and Zeller A., "Mining Metrics to Predict Component Failures," *in Proceedings of the 28th International Conference on Software Engineering*, Shanghai, pp. 452-461, 2006.

[13] National Institute of Standards and Technology 2011 [online]. Available: http://www.nist.gov/, Last Visited, 2014.

[14] National vulnerability database: http://www.cvedetails.com/, Last Visited, 2014.

[15] Newsome J. and Song D., "Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software," *School of Computer Science*, Pittsburgh, 2004.

[16] Nguyen V. and Tran L., "Predicting Vulnerable Software Components with Dependency Graphs," *in Proceeding of 6th International Workshop Security Measures Metrics*, Bolzano, pp. 1-8, 2010.

[17] Ozment A., "Improving Vulnerability Discovery Models," *in Proceedings of ACM Workshop on Quality of Protection*, Alexandria, pp. 6-11, 2007.

Kola Periyasamy received the M.C.A., M.E. degree from Anna University, Chennai, India and she completed her research Ph.D. in Anna University, India. She is currently working as an Assistant Professor (Senior Grade) at Madras Institute of Technology, Anna University, India. Her research is focusing on data mining and soft computing.

Saranya Arirangan received the M. Tech. Information Technology degree from Madras Institute of Technology, Anna University, Chennai, India in 2014. She is currently working as an Assistant Professor at SRM Institute of Technology and Engineering, India. Her research is focusing on predicting vulnerabilities in software applications, data mining analytics techniques and Block chain applications.