Survey on Vulnerability Prediction from Source Code by using Machine Learning Algorithm

B. DEEPTHI[1], DR.K. KUMAR[2]

[1]ME Student, Department of Computer Science & Engineering, Government College of Technology, Coimbatore.
[2]Associate Professor, Department of Computer Science & Engineering, Government College of Technology, Coimbatore.

---***---

ABSTRACT - *Now Web applications have been gaining increased popularity around the globe, in such a way that a growing number of users are attracted to make use of the functionality and information provided by these applications. While providing solutions to complicated problems in a fast and reliable way, it focusing on building a prediction model for detecting vulnerabilities of web applications. Based on the static analysis Machine Learning methods used to predict the vulnerabilities. Making use of data on any open-source web application to test the vulnerability filles. By applying machine learning techniques of Support vector machines (SVM) and Naïve Bayes (NB) techniques are used to prevent the vulnerability. Moreover, according to results of various classifiers, and methods offer possible causes of vulnerabilities and reasonable suggestions for avoiding vulnerabilities in the future. To conclude the main contributions are valuable feature engineers find the vulnerability localization, and machine learning model to predicting vulnerabilities effectively.*

Keywords: Machine Learning, Software Metrics, Software Security, Vulnerabilities.

1.INTRODUCTION

Web applications play a crucial role in many of our daily activities like social networking, email, banking, shopping, registrations, and so on. As web software is additionally highly accessible, web application vulnerabilities arguably have greater impact. It's is highly critical to detect and eliminate potential vulnerabilities as early as possible. A vulnerability is defined as a weakness in a data system, internal controls, system security procedures, or implementation which may be exploited by a threat source [3], whereas a flaw or bug may be a defect during a system which will (or may not) cause a vulnerability [4]. Thus, vulnerabilities are literally the subclass of software bugs which will be exploited for malicious purposes [5], [6] Vulnerabilities require quite different identification process than defects because they're often not realized by users or developers during the traditional operation of the system while defects are more easily and naturally noticed [6]. These make the fighting against vulnerabilities far more challenging than typical defects.

The two traditional approaches used for vulnerability detection: (1) static analysis (2) dynamic analysis. In static analysis, the code is examined for weaknesses without executing it. Therefore, the potential impact of the executable environment, such as the operating system and hardware, is not taken into consideration during analysis [7]. On the other hand, in dynamic analysis, the code is executed to check how the software will perform in a run-time environment, but this can only reason about the observed execution paths and not all possible program paths [7]. Hence, both static and dynamic code analyses have some problems on their own. Software defect prediction techniques have been proposed to detect defects and reduce software development costs. Defect prediction techniques is used to build a model from source codes, and use the models to predict whether new instances of code regions, e.g., files changes, possible attack files and methods contain defects.

Vulnerability analysis is a process that defines, detects and classifies security vulnerabilities in a system, network or communication infrastructure. It also suggests the countermeasures and the effectiveness of the implementation techniques. The vulnerability exists within a web application if it does not provide a proper validation process for the data entered by the user as input. The Machine Learning (ML) for software security analysis not only reduces the feature extraction, but also helps to simplify and automate processes for the current security analysis techniques. Abstract Syntax Tree (AST) for performing automated intelligent analysis directly on ASCII text file requires to unravel some challenges like representing ASCII text file during a proper form to enable further analysis in ML algorithms and localizing detected vulnerabilities on ASCII text file. Vulnerability prediction task as a binary classification problem for each targeted vulnerability class such our ML model takes a ASCII document fragment as input and decides whether it's vulnerable (i.e. containing the targeted vulnerability) or non-vulnerable.

2.LITERATURE SURVEY

The Literature Survey will have a review of papers about detecting software failure or vulnerabilities and how machine learning techniques can be used in the security area. I will first introduce some papers Quite a number of researchers have already made efforts on studying how to detect injection attack risk hole in web applications from

TABLE 1: LITERATURE SURVEY

s/no	Title	Input application	Technique/ algorithm	Performance	Limitation
1	Automated removal of cross site scripting vulnerabilities in web applications (2011)	PHP web application	Taint-based Static Analysis (Java)	Detection of stored and reflected XSS Vulnerability	Results by using taint-based static analysis, might miss some vulnerabilities since the method do not track information flow across web pages.
2	Evaluating complexity, code churn, and developer activity metrics as indicators of Software vulnerabilities. (2011)	Mozilla Firefox Web Browser, Red Hat Enterprise Linux kernel	Logistic regression, J48, Random forest, NB, Bayesian network	Code complexity, code churn, and developer activity metrics C++ / General vulnerabilities	-
3	Using complexity, Coupling and cohesion metrics as early indicators of vulnerabilities (2011).	Mozilla Firefox Web Browser	Logistic regression, C4.5, Random forest, NB	code complexity, coupling and cohesion metrics C++ / General vulnerabilities	Confidently lessen the effects of algorithmic bias, not attempting to identify the most effective, technique. Limit oneself.
4	Software vulnerability prediction using text analysis techniques. (2012).	K9 mail client application	SVM	Unique word Java/any vulnerabilities	The learning may fail to create any meaningful features. In the following section, we present the proposed approach
5	Mining sql injection and cross site scripting using hybrid program analysis (2013).	PHP Web Applications	Hybrid Analysis (PHP) + cluster	Detection of SQL injection and XSS Vulnerability	Not accurate as full static or dynamic analysis.
6	Predicting vulnerable software components via text mining (2014).	Java Application &Drupal CMS	Decision Trees, k-Nearest Neighbour, NB, Random Forest and SVM	Unique-words & Uni_tokens Java & PHP / General & XSS vulnerabilities	-
7	Web application vulnerability prediction using hybrid program analysis and machine learning (2015).	Any open-source application.	Hybrid Analysis (PHP) + semi-supervised	Detection of SQL injection and XSS vulnerability	static and dynamic analysis results are achieved by using Pixy.
8	Experimenting Machine Learning Techniques to Predict Vulnerabilities (2016).	Glibc, Xen HV, httpd, Mozilla	Random Under sampling (RU), Decision Tree algorithm. Logistic Regression	Detection vulnerability	Not all the configuration of the approaches are available, it is not guaranteed that the experiments were reproduced in ideal conditions.
9	Automatic feature learning for vulnerability Prediction (2017).	Quicksearchbox, Email, Mustard, Crosswords,	Deep Belief Network, Long Short-Term Memory (LSTM)	Detection vulnerability	The original dataset did not unfortunately contain the source files. Data set may not be representative of all kinds of Android applications.
10	Vulnerability Prediction From Source Code Using Machine Learning (2020).	GitHub, NIST's Samate project	Abstract Syntax Tree (AST), leverage machine learning (ML)	Detection vulnerability	imbalance problem, Can't find localization and interpretation aspects of the vulnerability prediction.

different aspects. In this Literature survey many methods are discussed. These methods are used to detect the software attacks. The table 1 shows the papers performance some limitation. To overcome these limitations by using SVM and neural networking.

3. MACHINE LEARNING

Machine learning technique is widely used for data analysis to build prediction models. Machine learning techniques, which are widely used these days, can be divided into three categories: supervised learning [10], unsupervised learning and reinforcement learning. To conclude, a supervised learning method can only learn from labelled training data, and on the contrary, unsupervised learning does not require the access to the label of data. Especially, reinforcement learning does not have a restriction on using labelled and unlabelled data. This method is designed to learn from feedback that is retrieved from its interaction with the environment. After considering the advantages and disadvantages of different types of machine learning methods, decided to use supervised learning for this research. Supervised learning algorithms can be used to train a model of class labels distribution, and this model is able to predict class labels for testing instances. An example of supervised learning algorithms process flowchart is shown in Figure 1, this whole process is also called classification. This is the foundation of designed prediction model as well. It is essential to select which classification method to use for a certain problem.

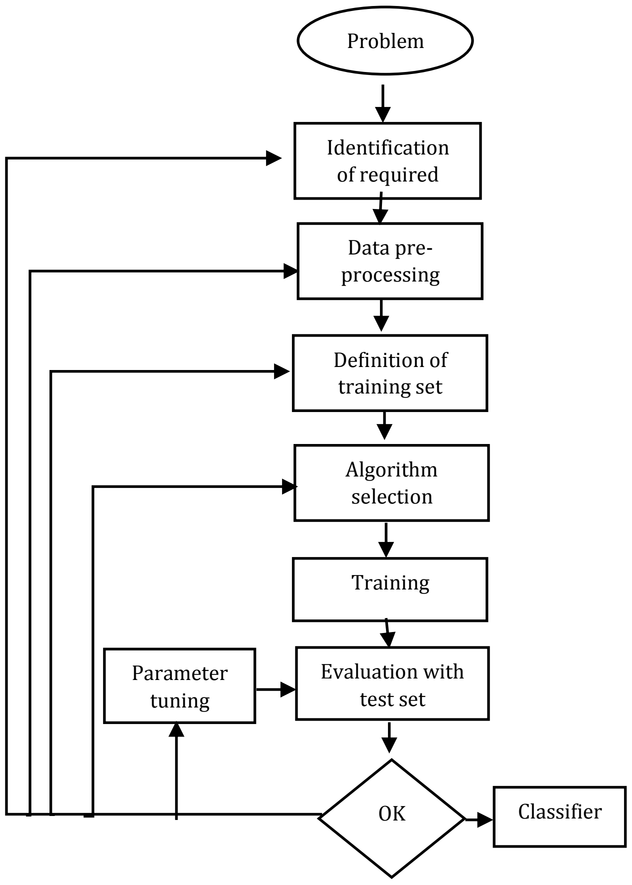

Figure 1. supervised learning algorithms process flowchart

There is a review on several widely used supervised learning algorithms in [2]. To decide which classifiers are more suitable for this research, first look into their pros and cons. In paper [2], the author pointed out that comprehensibility of Decision Tree makes this classifier helpful for understanding why an instance is assigned to a certain class, and Decision Tree is a suitable choice when dealing with discrete features. Linear Discriminant Analysis (LDA) and Naive Bayes are both statistical learning algorithms, which can provide a probability about labelling an instance. Moreover, in order to meet the requirement of this research, accuracy, tolerance to noise, the risk of being overfitting [20] and explanation ability are some vital aspects to consider when selecting classifiers. These models are considered in this research.

4.SYSTEM MODEL

In this paper, the study of vulnerability identification from web applications. By using different machine learning algorithms to prevent the attacks. The system model process flow shown on figure 2 and the following sections are descripting the model processes.

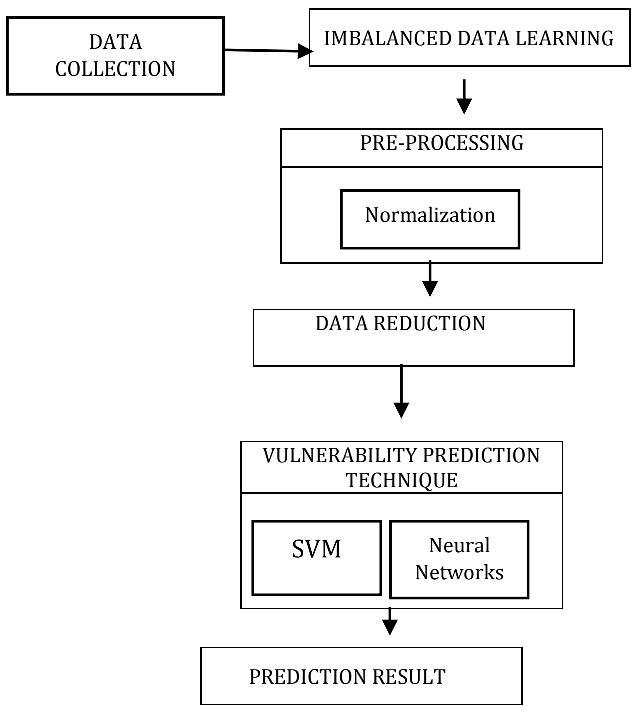

Figure 2. System Model

DATA COLLECTION

The web application is facing with various types of vulnerabilities, but in general, they can be categorized into two main kinds based on the causes of them, including design flaw and implementation bug. Obviously, most design flaws are hard to detect by only analysing individual source code files.

INPUT APPLICATION: Any Open-Source Web Applications like, www.github.com and http://google-gruyere.appspot.com/

IMBALANCED DATA LEARNING

DATASET: The Datasets downloaded from NIST SATE IV [21]. The Datasets distribution: Training (80%), Validation (10%), Testing (10%). This dataset consists of 1.27 million of source code functions mined from open-source software, labelled by static analysis for potential vulnerabilities.

Because of the class distribution skew problem, it is a crucial issue to deal with imbalanced data learning in this research [14] [15] [16].

Sampling Techniques

Overall, there are two methods suitable for sampling imbalanced data, including random under-sampling and random oversampling. The idea of these two sampling methods is randomly adding(removing) a randomly selected dataset from minority(majority) class to make the whole set becoming balanced. However, these random sampling methods have some shortages. The under-sampling method would cause information loss to majority class, and the oversampling could bring about the over-fitting issue on minority class. Due to a limited number of attack files, this research only uses random oversampling on the dataset.

min-max method

It transforms all values to values in the interval [0, 1]. Given a feature f, denote the maximum and minimum value for f as max(f) and min(f) respectively. For each value of the feature f, the normalized value zi is given in equation (1).

$$(z_i) = \frac{xi - \min(f)}{\max(f) - \min(f)}$$

(1)

F1-score

The F1-score is a commonly-used measure to evaluate classification performance. The F1 score formula given in equation (4). F1 combines Precision (2) and Recall (3) and can be derived from a confusion matrix. It lists all four possible prediction results. If an instance is correctly classified as “buggy”,

i) Real Positive (TP)-if an instance is misclassified as “buggy”

ii) False Positive (FP) - “Similarly”

iii) False Negatives (FN)

iv) True Negatives (TN)

Based on these four numbers, Precision, Recall and F1-score are calculated. Precision is that the ratio of correctly predicted “buggy” instances to all or any instances predicted as “buggy”.

$$\text{precision} = \frac{TP}{TP+FP}$$

(2)

$$Recall = \frac{\text{TP}}{TP+FN}$$

(3)

$$\text{F1} - \text{measure} = \frac{2*\text{Recall}*\text{P recision}}{\text{Recall}+\text{P recision}}$$

(4)

Recall is that the ratio of the amount of correctly predicted “buggy” instances to the particular number of “buggy” instances. Finally, F1-score may be a mean of Precision and Recall.

F1-score is usually used as a summary measure to evaluate if a rise in precision outweighs a discount in recall (and vice versa).

DATA PRE-PROCESSING AND DATA REDUCTION

More specifically, as depicted in Figure 3, initially split source code into smaller parts to allow more granular analysis. Then, we generate and extract AST for each departed code component, which also includes a tokenization process via a laxer. Later on, convert the extracted AST into the complete binary tree that has a deterministic shape where it is specified how many nodes are located at each level of the tree.

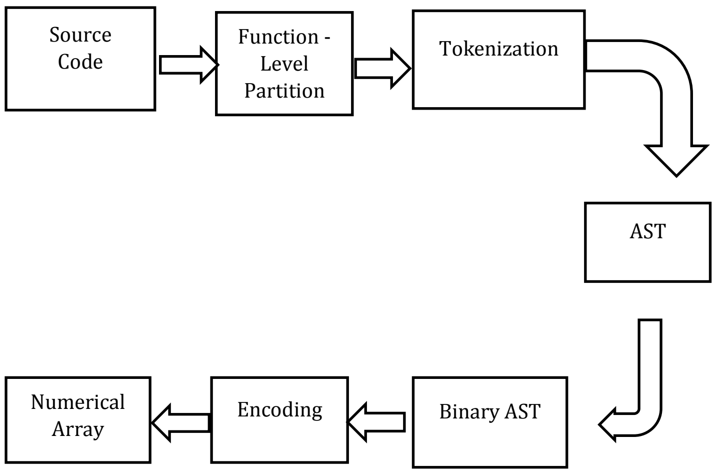

Figure 3. data pre-processing and data reduction

Afterward, each token is encoded in the complete binary AST to pre-defined numerical tuples and finally represent a one-dimensional numerical array of the corresponding function-level source code. This source code concatenating the assigned numerical tuples from the root node to leaves in order. It justifies each step-in detail in the following parts, along with examples.

TOKENIZATION

The source code is cleaned by removing its unnecessary elements such as comments, whitespaces, tabs, newlines, etc. Then, the remaining part is converted into a series of tokens, where a token is a sequence of characters that can be treated as a unit in the grammar of the corresponding programming language. This can be achieved by using a laxer developed explicitly for the language of the source code.

Example of SQL Injection:

Query 1: Original Query SELECT * FROM usertable WHERE username= 'greg';

Query 2 Original Query SELECT * FROM usertable WHERE username= 'greg' OR '1'= '1';

The query 2 is converted into series of tokens,

[Token 1: Select*from

Token 2: usertable

Token 3: where

Token 4: username='greg'

Token 5: password='secret' or '1'='1']

AST (Abstract Syntax Tree)

AST contains syntax and semantic information about the source code, and therefore it is highly useful for further analysis. Figure 4 shows AST of the main function given in Step 2. The structural relations (e.g., parent-child) are important in the AST and could be useful for vulnerability identification [15].

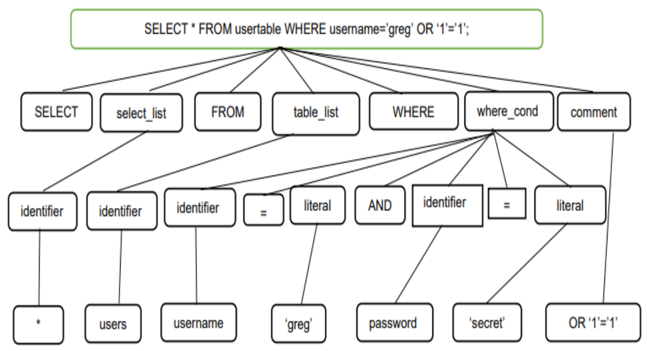

Figure 4. AST construction of SQL Injection tokens

Therefore, the relational pieces of code information should not be lost or fail during the transformations in source code representation stages. The small challenging given that a regular AST is a kind of m-array tree where there is no restriction on the values that m can take, which means each node may have an arbitrary number of child nodes that makes the structural shape of the AST unpredictable. To overcome this issue, we apply the next step to the binary AST.

Binary AST

AST is a tree type data structure and need to convert it into an array format to feed an ML algorithm [18]. After converting an AST into a binary array by placing nodes starting from root level to deeper levels, from leftmost node to rightmost node at each level. Several problems would occur in such a conversion. First of all, structural relations among AST nodes such as parent-child relations would be lost in the resulting array because a parent node may have arbitrary number of children. Second, the resulting arrays would be in different lengths. However, it is highly important to preserve structural relations among AST nodes while mapping them into a one-dimensional array because both it contains some semantic information about the code and neural network-based models use such spatial information to extract hidden patterns. As a solution, convert a regular AST to the corresponding complete binary AST, where all leaves have the same depth [3]. These values are used to process the vulnerability prediction machine learning algorithms.

VULNERABILITY PREDICTION

1. Support vector machines (SVM)
2. Neural networks

1.Support vector machines (SVM)

SVM (support vector machine) is a typical algorithm in machine learning. Its core idea is to seek out the foremost suitable separation hypersurface within the sample space, which may distinguish the samples significantly. The SVM include linear separable, linear support, and nonlinear support vector machines. Among them, the linear regression of SVM is expressed as follows.

Set the sample set as $(y_1,x_1),....(y_l, x_l)$, x £ R^n, y £ R and use a linear equation to represent the regression function.

$$f(x)=w^T \phi(x) + b \tag{5}$$

The essence of formula (5) are often considered a constrained optimization problem, and its expression is as follows In formula (6),

$$\Phi(w, £, b) = \frac{1}{2} |w|^2 + C(\sum_{i=1}^{l} £_i + \sum_{i=1}^{l} £_i^*) \tag{6}$$

C refers to the penalty factor and £ and £* represent the upper and lower limits of the relief variable, respectively. The Formssula (6) is used to solved the Lagrangian constraint equation, which is shown as follows.

$$\bar{\alpha},\ \bar{\alpha}^* = \arg\min \{\frac{1}{2} \sum_{i=1}^{l} \sum_{i=1}^{l} (\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)(\phi(x_j)\,\phi(x_j)) - \sum_{i}^{l} (\alpha_i - \alpha_i^*)\, y_i + \sum_{i}^{l} (\alpha_i - \alpha_i^*) £\} \tag{7}$$

sin formula (7), $\phi(x)$ is a kernel function. If $\phi(x_i)\ \phi(x_j) = x_i\ x_j$, then it represents a linear support vector machine; otherwise, it is a nonlinear support vector machine. The solution expressions of the sum of the coefficients to be determined, the regression coefficients, and the constant terms are as follows. The Ŵ separate the vulnerable and non-vulnerable codes given in equation (8).

$$\hat{W} = \sum_{i=1}^{l} (\alpha_i - \alpha_i^*)\ x_{i,} ƀ = -\frac{1}{2}\ \hat{W}\ [x_r + x_s] \tag{8}$$

These formulas are used to separate the vulnerable and non-vulnerable codes.

2. Neural networks

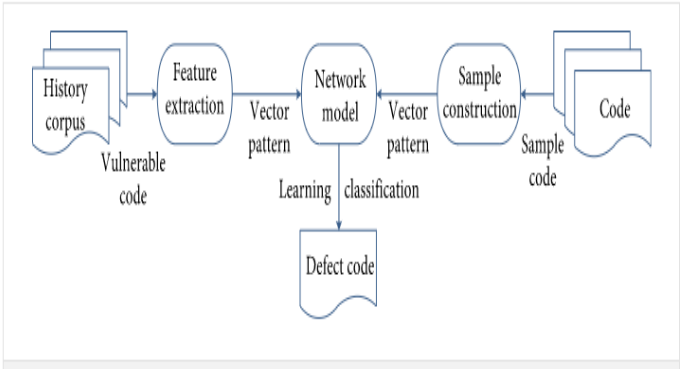

Figure 5. Code static analysis and neural network training Principle

The parameters of the Neural Network model are set as follows.

1) Initialize the weight of DNN model with normal distribution function with standard deviation of 0.1.

2) The offset values of the input layer and hidden layer are set to 0.1.

3) The forward propagation of the input layer and hidden layer uses tanh as the activation function, while the output layer uses SoftMax as the activation function.

4) Use the dropout function in TensorFlow to prevent overfitting.

5) The cross-entropy function is used as the loss function to measure the loss between the calculated output of the training sample and the actual output of the training sample.

6) Optimize the DNN model using a batch gradient descent algorithm with a batch size set to 100 and a learning rate set to 0.2.

Next, the training sample of the NVD data set is used to train the TFI-DNN vulnerability automatic classification model, and then the vulnerability test set is used to evaluate the model performance.

The whole process includes the subsequent steps: sample code construction, feature extraction, word vector generation, and neural network model training and classification. Among them, vulnerability feature extraction mainly involves the way to select appropriate granularity to represent software programs and vulnerability detection shown on figure 5. Since deep learning or neural networks take vectors as input; it like to represent programs as vectors that are semantically meaningful for vulnerability detection. Use "bridge" act as intermediate representation between a program and vector representation, which is that the actual input to deep learning. Vulnerability feature extraction is to rework programs into some intermediate representation which will preserve (some of) the semantic relationships between the programs' elements (e.g., data dependency and control dependency). Word vector generation is predicated on feature extraction, applying the foremost mainstream word vector generation technology in order that intermediate representation is often transformed into a vector representation, that is, the actual input to neural networks. Neural network training classification involves two stages of coaching and detection. The training phase takes the source code extracted from the historical code base as input, whose output is neural network of fine-tuned model parameters. In the detection phase, the code vector representation extracted from the new software program is taken as input, and therefore the output is that the classification result.

CONCLUSION -In this context, first proposed a source code representation method that is capable of characterizing source code into a proper format for further processes in ML algorithms. The presented method extracts and then converts AST of a given source code fragment into a numerical array representation while preserving structural and semantic information contained in the source code. Thus, it enables us to perform ML-based analysis on source code through resulting numeric array representation. To examine the presented source code representation technique for different objectives rather than vulnerability prediction, such as similarity analysis and code completion. and improve localization and interpretation aspects of the vulnerability prediction by using Support Vector Machine Learning (SVM) and Neural Networks.

FUTURE ENHANCEMENTS-

Future investigation involves building a fully end-to-end prediction system from raw input data (code tokens) to vulnerability outcomes. It would be interesting to examine the presented source code representation technique for different objectives rather than vulnerability prediction, such as similarity analysis and code completion. To improve localization and interpretation aspects of the vulnerability prediction. The presented method to apply a model trained on a certain language to other languages.

REFERENCES

1. ZEKI BILGIN "Vulnerability Prediction From Source Code Using Machine Learning", Received July 28, 2020, accepted August 9, 2020, date of publication August 14, 2020, date of current version August 26, 2020.
2. S. B. Kotsiantis, I. Zaharakis, and P. Pintelas, "Supervised machine learning: A review of classification techniques," Emerging artificial intelligence applications in computer engineering, vol. 160, pp. 3–24, 2007.
3. R. S. Ross, ''Information security,'' Joint Task Force Transformation Initiative, Guide Conducting Risk Assessments, NIST Special Publication, Gaithersburg, MD, USA, Tech. Rep. 800-30 Revision 1, 2012.
4. A. M. Delaitre, B. C. Stivalet, P. E. Black, V. Okun, T. S. Cohen, and A. Ribeiro, ''Sate V report: Ten years of static analysis tool expositions,'' NIST, Gaithersburg, MD, USA, Tech. Rep. SP-500-326, 2018.
5. M. Dowd, J. McDonald, and J. Schuh, The Art of Software Security Assessment: Identifying and Preventing Software Vulnerabilities. Reading, MA, USA: Addison-Wesley, 2006.
6. M. Jimenez, ''Evaluating vulnerability prediction models,'' Ph.D. dissertation, Dept. Sci., Technol. Commun., Univ. Luxembourg, Rue Mercier, Luxembourg, Oct. 2018.
7. T. Abraham and O. de Vel, ''A review of machine learning in software vulnerability research,'' Cyber Electron. Warfare Division, Dept. Defense, Austral. Government, Edinburgh, SA, Australia, Tech. Rep. DST-GroupGD-0979, 2017.
8. B. McCorkendale, X. F. Tian, S. Gong, X. Zhu, J. Mao, Q. Meng, G. H. Huang, and W. G. E. Hu, ''Systems and methods for combining static and dynamic code analysis,'' U.S. Patent 8,726,392, May 13, 2014.
9. E. Ustundag Soykan, Z. Bilgin, M. A. Ersoy, and E. Tomur, ''Differentially private deep learning for load forecasting on smart grid,'' in Proc. IEEE

Globecom Workshops (GC Wkshps), Dec. 2019, pp. 1–6

10. R.Scandariato, J. Walden, A. Hovsepyan, and W. Joosen, "Android study dataset, "https://sites.google.com/site/textminingandroid, Accessed on 15 Jan 2017, 2014.
11. Weiwei Li a, b, Zhiqiu Huang a, and Qing Li, "Software Vulnerability Prediction using Feature Subset Selection and Support Vector Machine, 2016.
12. Hoa Khanh Dam, Truyen Tran, Trang Pham, Shien Wee Ng , John Grundy and Aditya Ghose University of Wollongong, Australia, "Automatic feature learning for vulnerability prediction "arXiv:1708.02ss368v1 [cs.SE] 8 Aug 2017.
13. M. Jimenez, ''Literature survey on security vulnerabilities,'' Ph.D. dissertation, Dept. Sci., Technol. Commun., Univ. Luxembourg, Rue Mercier, Luxembourg, Oct. 2018.
14. H. He, Y. Bai, E. A. Garcia, and S. Li, "Adasyn: Adaptive synthetic sampling approach for imbalanced learning," in Neural Networks, 2008. IJCNN 2008. (IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on, pp. 1322–1328, IEEE, 2008.
15. S. Wang and X. Yao, "Using class imbalance learning for software defect prediction," IEEE Transactions on Reliability, vol. 62, no. 2, pp. 434–443, 2013.
16. H. He and E. A. Garcia, "Learning from imbalanced data," IEEE Transactions on Knowledge & Data Engineering, no. 9, pp. 1263–1284, 2008.
17. Zhidong Shen and Si Chen "A Survey of Automatic Software Vulnerability Detection, Program Repair, and Defect Prediction Techniques" 2020. https://doi.org/10.1155/2020/8858010 P. Vogt, F. Nentwich, N. Jovanovic, E. Kirda, C. Kruegel, and G. Vigna, "Cross site scripting prevention with dynamic data tainting and static analysis.," in NDSS, vol. 2007, p. 12, 2007.
18. Dima Bekerman, Sarit Yerushalmi "The State of Vulnerabilities in 2019" https://www.imperva.com/blog/the-state-of-vulnerabilities-2019/.
19. R. Scandariato, J. Walden, A. Hovsepyan, and W. Joosen, "Predicting vulnerable software components via text mining," IEEE Transactions on Software Engineering, vol. 40, no. 10, pp. 993–1006, 2014.
20. S. B. Kotsiantis, I. Zaharakis, and P. Pintelas, "Supervised machine learning: A review of classification techniques," Emerging artificial intelligence applications in computer engineering, vol. 160, pp. 3–24, 2007.
21. R. Russell, L. Kim, L. Hamilton, T. Lazovich, J. Harer, O. Ozdemir, P. Ellingwood, and M. McConley, ''Automated vulnerability detection in source code using deep representation learning,'' in Proc. 17th IEEE Int. Conf. Mach. Learn. Appl. (ICMLA), Dec. 2018, pp. 757–762.