ISSN 2278-3091
Volume 9, No.4, July – August 2020
International Journal of Advanced Trends in Computer Science and Engineering
Available Online at http://www.warse.org/IJATCSE/static/pdf/file/ijatcse358942020.pdf
https://doi.org/10.30534/ijatcse/2020/358942020

# Software Vulnerability Classification based on Machine Learning Algorithm

**Markad Ashok Vitthalrao[1], Mukesh Kumar Gupta[2]**
[1]Department of Computer Science & Engineering, School of Engineering & Technology, Suresh Gyan Vihar University, Jagatpura, Jaipur, India, ashok.markad@gmail.com
[2]Department of Computer Science & Engineering, School of Engineering & Technology, Suresh Gyan Vihar University, Jagatpura, Jaipur, India, mukeshkr.gupta@gyanvihar.org

## ABSTRACT

Vulnerabilities in security are the main issues in computer security. Throughout recent years, several strategies have been used to minimize the risk of software vulnerabilities due to their high severity impacts. Machine-learning and data-mining techniques are among other solutions to investigate such issues in different environments. In this research, we investigate a comprehensive investigation and analysis of the several approaches which work for vulnerability assessment using machine learning as well as data mining techniques in the field of software vulnerability analysis and discovery. The work proposed software bug classification and vulnerability identification form completed software code using machine learning techniques. Various pre-processing and natural Language Processing (NLP) techniques have been used to extract the features from the heterogeneous dataset and generate normalized feature vectors. Those vector passes to the training module and generates Background Knowledge (BK) respectively. Three different machine learning algorithms like SVM, ANN, Random Forest have used to detect the bugs and evaluated proposed system effectiveness with some existing researches. Finally, we conclude system provides drastic supervision and better detection accuracy which is most effective and better than other machine learning algorithms.

**Key words :** Machine Learning, RNN, Random Forest, Natural Language processing, classification, Bug classification, Vulnerability assessments.

## 1. INTRODUCTION

The Vulnerabilities in software include Program errors. Bugs are coding errors that cause undesirable action on the system. All software has some form or something of bugs. Many bugs cause the device to break, some cause networking to crash, some prevent a person from logging in and some cause scanning to fail. Many bugs establish leakage of information or increase user rights, or otherwise grant unauthorized access. There are vulnerabilities to protection. When all software has bugs and certain bugs naturally become vulnerabilities to security, all technology will have weaknesses in data protection. Each year lots of security issues are detected throughout production software. Weaknesses often manifest in subtle ways that are not evident to either the code reviewers or the developers. With the wealth of open-source software accessible for analysis, there's an experience to study the patterns of bugs that can lead directly from data to security vulnerabilities. In essence software vulnerabilities are defined as system operating systems defects being exploited illegally by unauthorized personnel. In the information engineering domain device vulnerability identification is an important area of study. This is partly because security vulnerabilities are continually disclosed. While previous studies show the utility of using multiple detection methods, models, and frameworks to identify software vulnerabilities, enhancing the efficacy of these detection models and tools remains a major challenge for researchers and practitioners alike. In this work, we present a data-driven security vulnerabilities detection approach using deep learning. Motivated by the effectiveness of the model in these fields of study, we use a theoretical framework to investigate its feasibility within the vulnerability detection domain. Dynamic analytics tools rely on detailed monitoring of runtime properties, including log files and memory, and require a wide range of representative test cases for the application to be used. Therefore, a standard practice still relies heavily on domain knowledge to identify the most vulnerable part for intensive security inspection of a software system. Machine learning techniques are a common approach to building vulnerability prediction models. Several software codes representing functions are selected for use as vulnerability predictors. The most common features used in previous work are software metrics (e.g. code size, number of dependencies, and complexity of code functions), code churn metrics (e.g. number of lines of code changed), and developer activity. However, those features cannot distinguish code regions from different semantics. For certain instances, two pieces of code can have the same complexity metrics but have different behaviour's and therefore have different susceptibility to attack.

## 2. LITERATURE SURVEY

Based on the study, vulnerabilities in software allow around 30% of all successful attacks. This percentage is highly significant, as it implies the loss of billions of dollars due to whatever is mostly an avoidable problem. Below we review some existing systems that various authors have tried to deliver similar approaches to eliminate the vulnerabilities.

Marian Gawron et. Al.[1], has proposed a way to manual sorting, as the number of bugs found every day can no longer be managed manually. They have introduced two separate approaches that can automatically detect potential risks on an overview of the weakness. They assessed our strategies using the methodologies Neural Networks and Naive Bayes, respectively, based on widely known vulnerabilities.

Andrej Queiroz. Al. [2] illustrates a Support Vector Machine (SVM) prediction models, utilizing Twitter messages (tweets) as a guide to sorting weakness-based knowledge applicable to particular applications. Tweets deemed relevant for the paper will be those alerting about new potential vulnerabilities (being exploited or not), and posting alerting users to security updates and patches. Non-relevant information will be known as non-warning information, i.e.: the message of opinion, general discussion, and non-warning subjects. Through utilizing simple features including word frequency (unigram and bigram), the suggested framework reached a precision of 94%. The attractive class values showed adequate standards of recall and precision for the same simple features as, respectively, 68% and 46%. Such research paves the way for future study of the interaction of how the protection department addresses information security warnings and social networking updates..

And Jacob A. Harer. Al. [3] used machine learning to implement a vulnerability detection approach powered by data. They then combined hundreds of open-source functions marked with feedback from a static analyzer. We then compare methods applied directly to source code with methods applied to objects derived from the build phase, seeking a better output of the models based on the source. We also compare the implementation of deep neural network models with more conventional models like random forests and consider the best output is by integrating features learned from deep networks with tree-based networks. Our best-performing model ultimately achieves an area below the 0.49 precision-recall curve and an area below the 0.87 ROC curve.

Jeesoo Jurn, et. Al.[4], introduced a trend in automated vulnerability detection techniques and tools and remediation. We propose an automated vulnerability detection method based on the analysis of binary complexities to prevent a zero-day attack. We also introduce an automated patch generation process by modifying the PLT table to respond to zero-day vulnerabilities.

Detecting software vulnerabilities (or short vulnerabilities) is a significant problem that has yet to be resolved, as reflected in many of the vulnerabilities published daily, says Zhen Li et al[5]. Learning methods must have automatic vulnerability detection. Deep learning is good for this case since it doesn't require humans to manually define features. Despite the tremendous success of deep learning in other domains, its applicability is not systematically understood for vulnerability detection. They propose the first systematic framework of using deep learning to detect vulnerability. The framework, dubbed Syntax-based, Semantics-based, and Vector Representations (SySeVR), focuses on getting program representations that can accommodate syntax and semantic information relevant to vulnerability.

By Sabetta Antonino, et. Al.[6], an approach that uses machine learning to test repositories with source code and to automatically identify commits related to security (i.e. vulnerability-sensitive ones). They consider the improvements made by commits to the source code as documents written in a natural language, classifying them using standard methods for document classification. Our method can deliver high accuracy (80%) while ensuring acceptable recall (43%) by combining independent classifiers using information from various facets of commitments. Use information extracted from source code enhancements provides considerable improvement over the best-known state-of-the-art methodology while offering a significantly reduced amount of training data and utilizing a simpler architecture.

Tamas, Abraham, et. Al. [7] not only explores new methods but also helps SVR practitioners simplify and optimize their processes. Given the variety of applications currently in evidence, we believe that ML will continue to provide support for SVR in the future as new areas of use are explored and updated algorithms become available to improve existing functionality.

Hoa Khanh et. Al. [8], Defined a new approach focused on the powerful Long-Term Short-Term Memory Model for automated learning of both semantic and syntactic language. Our research on 18 Android applications and the Firefox framework shows that the predictive strength of our learned apps is stronger than what is accomplished with state-of-the-art security vulnerabilities computer models for both in-project prediction and bridge-project prediction.

According to Rebecca L. Russell et.al.[9], amounts of software bugs are discovered annually, whether publicly reported or internally identified under proprietary code. This can pose vulnerabilities a major risk of violence, leading to system breaches, information leaks, or service denial. They implemented the enormous amount of open source code accessible from various files to build a large scale vulnerability detection system that uses machine learning. They collected and labeled a massive dataset of millions of open-source functions to complement existing vulnerability databases labeled with carefully selected results from three

separate static analyzer's suggesting possible exploits. Dataset labeling can be found in https:/osf.io / d45bw/. Using these datasets they created a fast and scalable vulnerability detection tool focused on learning the profound representation of features that interprets fixed source code directly. We checked both the individual software packages and the benchmark dataset for NIST SATE IV on our application platform. Our findings indicate that deep feature representation learning on source code is a viable candidate for automatic vulnerability detection software.

Ren jiadong, et. el. [10], Suggested use of software metrics and a decision tree algorithm to assess system buffer overflow susceptibility. First, the software metrics were retrieved from the source code of the program, and data was obtained at the document level through data analysis from the flexible data stream. Second, a model was developed to calculate different forms of directory traversal deficiencies at the functional level, based on a decision tree algorithm. Finally, the experimental results showed that our program was running in less time than the algorithms of the SVM, Bayes, Adaboost, and random forest and achieved 82.53% and 87.51% accuracy in two different time frames. The method explored in this paper predicts exactly system buffer overflow bugs in applications such as C , C++, and Java.

Ashok M and Dr. Mukesh Gupta proposed [11] software vulnerability classification using a deep learning approach, various NLP techniques have used to extract the different features and train modules accordingly. All deep learning algorithm they have evaluated like DNN, RNN, PNN, etc and shows the best classification results.

Hajah T. Sueno, Bobby D. Gerardo, Ruji P. [12] proposed the improved Naïve Bayes vectorization used Laplace smoothing to ensure that posterior probabilities are never zero and logarithmic function to solve the result of the probability calculation that is too small that cannot be represented. The text classification algorithms based on the vector space model, such as the Support Vector Machine (SVM), use this probability distribution as the vectors to represent the document that is used to classify the documents. To validate the improvement of the Naïve Bayes vectorization technique, the results are compared to TF-IDF vectorization.

H. D. Gadade, Dr. D.K.Kirange [13] have modeled a system for automatic feature extraction and classification. We have evaluated the performance of the system using different performance measures to conclude with appropriate features set and classification technique for tomato leaf disease classification. The experimental results validate that Gabor features effectively recognize different types of tomato leaf diseases. Accuracy of SVM is better as compared to other classification techniques but the execution time are more.

## 3. RESEARCH METHODOLOGY

Social The function is compiled with actual data, the Proposed method tests the exact behaviour of run-time Technology. Dynamic analysis can be as quick as the execution of the program, whereas static analysis Typically more computation time is needed to obtain Pretty decent performance. The principal challenge in dynamics Research methods perform whatever execution is necessary System routes, and all vulnerabilities disabled in Those itineraries. In reality, the acquisition of proper test data Set, which will make the curriculum more diverse, is a Problem regarding those methods. The most important of these

The weakness of complex analytical methods is that they cannot guarantee an overview of all the feasible Places to execute. The dynamic analysis, therefore, isn't Visual and often used to demonstrate the presence of Relevant Programming vulnerabilities. The Power Methods are divided into two main groups. Methods and methods using symbolic input values and using the real input values (concrete) to check the Schedule. Cantered on recent complex advances Methods of analysis, we classify those methods into three Classes by type of input values applied: Concrete results, symbolic execution, and the console The execution methodologies (tangible + meaningful). Examples are Subgroups that define each class in many more details.

### 3.1 Concrete Execution

The function is compiled with real statistics in this method, as well as its behaviour is analysed for vulnerability detection. During the analysis, there are four methods of dynamic analysis that use actual data to execute the program: fault infusion, mutation suitable starting, dynamic taint assessment, and dynamic system check.

### 3.2 Fault Injection

In this approach, the software is injected with external faults to analyse its actions. The external faults exploit the internal faults and trigger inappropriate behaviours within the system according to our interpretation. In other words, internal faults are triggered by the external fault and propagated to meet the limits of the system. The inability to control external defects can, therefore, expose a flaw within the system

### 3.3 Mutation Based Analysis

Acquisition of appropriate tests, as mentioned before Data is a subject for complex analysis. When it's the plan that has normal behaviour during the test phase, that means If the software does not pose any vulnerability or the test The data do not disclose software vulnerabilities. In the latter case, the data set is not sufficiently large to turn the vulnerabilities on. The mutation is a method of Concerned with the improvement of data set during the analysing dynamics. Specific vulnerabilities In this method Are intentionally inserted into the software code. The existing collection of data does not detect the inserted one vulnerability; related vulnerabilities will not be contained in the Initial Computer Edition. In this way, the analyser increases the data set to detect the Fragility. A version of a system which contains a specific Vulnerability is established, Mutant is named. For instance, In mutations, the strncpy() (function is replaced

with strcpy) makes it vulnerable to buffer overflows. A strong one Test data collection makes a distinction between the mutants and the original Application version and kills them. When there's no test case Kills the mutants, and raises the data collection.

### 3.4 Dynamic Model Checking

The resultant behaviour of both the program is evaluated by monitoring and reviewing and its atmosphere, e.g. registers and stack, for each random input and schedule choice. Here every state contains the set state of the system. When the execution enters a state in which the configuration is violated, the corresponding input value and the option of schedule are viewed as counterexamples

### 3.5 Vulnerability Forecasting

Fault prediction is commonly used for predicting the nature or quantity of the faults left in the File System and will be enabled in the future. Essentially, it is concerned with the latest reliability figures of the Device and predicted its reliability for the future. That's it. Predictions may be either largely qualitative. The qualitative prognosis describes and Ranks futures modes of failure. The case is also Combinations are established which lead to failures.

### 3.6 Hybrid analysis

In existing, a lot of research groups combined those other methods, to make up for them in different ways Weaknesses on each other. For instance, combine static and dynamic XSS detection analysis with Security flaws of SQL injection in PHP applications. The method proposed initially analyses the code statically and removes the operations in the power flow diagram. Then these graphs are connected to get a Diagram of the inter-procedural control flow graph (iCFG). The iCFG Analysed to extract paths from contaminated Sources in it to sinks. For every laver, backward Slicing is used to detect declarations affecting the Muddled argument. These statements are tracked in the Long run. By using a corrupted attribute in a drain, the Monitoring protocol shall move it to an oracle to check if it is that vulnerability can be exploited. The oracle has a database of Known patterns of attack used to exploit oracle for MySQL query execute a syntactically limited Review and checks on the SQL queries Protagonists in unsafe roles tainted. In the process, various weaknesses are assumed for the disinfecting processes. For instance, the enforced various techniques have been used over the last decades Presented to mitigate vulnerabilities of Software. Full classification of the methods proposed Helps to get a general understanding of this Zone of research. We described Software in this paper Vulnerability, as a fault within. By taking applications into account we listed failure as the type of fault Method of vulnerability mitigation based on the general principle Classification of the strategies for reducing faults. We extended the general list of mitigating faults Methods, described in the Computer sense Vulnerability, and the addition of more detailed subclasses. We have classified risk reduction strategies into four Key classes: Prevention of vulnerability, vulnerability Tolerance, the elimination of vulnerability, and vulnerability Prediction. The methods for preventing vulnerabilities Test to prevent software from happening Fragility. Security of Software and secure coding Examples of those initiatives are best practices. The avoidance of vulnerability still creating efforts, vulnerabilities**.**

## 4. PROPOSED SYSTEM DESIGN

This section describes the actual working of the proposed system. Here the different methods to analyze whether the cloned code can be refactored or not has been described in detail. Moreover, after the analysis, some algorithms are explained in detail which performs the function of refactoring of code. Thus the brief process of vulnerability assessment and bug triage is explained in this section.

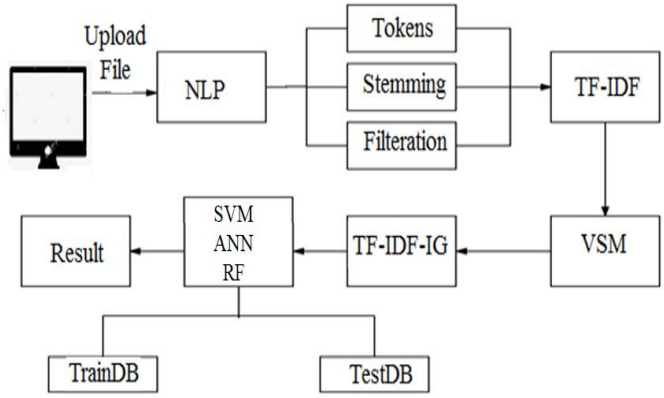


**Figure 1:** Proposed System Architecture Design

The above figure 2 shows a system overview of execution process flow, and how it works with different algorithms. Initially, we have a dataset of various software codes that contain numerous functions as well as procedures. The data set has processed buy Natural Language Processing with some basic algorithms, tokenization husband to splitting the data into separate words. Stop word removal is another algorithm that has used to eliminate stop words that are already available in programming functions or procedures. Porter stemmer algorithm has used to extract features and finally, we use filtration technique for eliminates misclassified instances or null values. The TF-IDF features extracted based on the density of respective tokens; this is the technique for feature extraction used in training as well as testing respectively. The vector space model has generated for feature selection purposes and boosting with information gain to get the best feature from the vector space model. Three different machine learning algorithms have been illustrated all training as well as testing. Once training has done system automatically generates some background knowledge according to a supervised learning strategy. This technique has been applying to test data set and classify detection accuracy for heterogeneous data on different platforms.

## 5 ALGORITHM DESIGN

The below algorithms has used during the features extraction with NLP, entire NLP features not sufficient for classification and generates the train modules. Using below algorithms we extract some features and some NLP features will provide better vector space for selected features.

**Algorithm for find edges from each tree node**
This algorithm is responsible to find the clone opportunities.
**Input:** each class as ***TreeNode***
**Output**: Find all similar edges **Treenodelist** from ***TreeNode***
**Array list=Treenodelist;**
**Step 1:** Read each Link from ***Treenode***
**Step 2:** if ***(Treenode.pattern (Linknodepattern))***
**Step 3:** **Treenodelist** ← ***Treenode***
End for
**Step 4:** execute step 1 on all TreeNode

**Algorithm for finding clones from master class**
This algorithm is responsible for finding out the clones from the possible opportunities of clones detected in previous step.
**Input:** Master class *M* with multiple subclass
**Output:** Classified clones viz ***(Type1, Type2, Type3)***
**Temporary variables:** Reader, ArrayList T1, T2, T3
**Step 1:** read each line data= Reader. Line ()
**Step 2:** check pattern from (data)
if (clone.Node.mapped.single (data))
T1= data. Node
Else if (clone.Node.mapped.double (data))
T2=data. Node
Else if (clone.Node.mapped.tripple (data))
T3=data. Node
**Step 3:** consider as mapped node of (T1, T2, and T3)
**Step 4:** classify all clones as Type(1)…….Type(3)
**Step 5:** Show all clones
End for

**Algorithm for Precondition violation checking**
This algorithm is for checking out the precondition violation. If one precondition is not violated then that clone is considered as refactorable clone opportunity, otherwise it will be not processed further to find a refactorable clone. This will give us possible refactorable clone.

**Input** each class C with nodes
**Output:** Boolean 1 if violet else 0

**Step 1:** Read each line from data=reader.read©
**Step 2:** if(data.contains(break,control) || SubClass.Type || Param name diff || does not have void Type || continue, break || conditional(return) || Equals (break,continue))
Early stop, violation
**Step 3:** end for

**Algorithm A statement mapping process based on nesting structure.**
This algorithm gives us refactorable clones form possible refactorable opportunities.
**Input:** two isomorphic NSTs
**Output:** the final mapping solution
**Step 1:** SetLength = NST*i.Length +NSTJ.Length*
**Step 2:** **while** level SetLength !=0 **do**
**Step 3:** cpNodesi = nodes at level of NSTi
**Step 4:** cpNodesj = nodes at level of NSTj
**Step 5:** **for each** cpi 2 cpNodes i **do**
**Step 6:** SimScore =Mapping (NSTi, NSTj)
**Step 7:** if (SimScore ==0)
Count ++
End for
**Step 8:** if(Count>1)
Display to as refactoring possible.
**Step 9:** end procedure

## 6. RESULTS AND DISCUSSIONS

The implementation of the proposed system has been completed for the training module. As per our first module, we have used standard data from various .java classes set of 1000 files for the dataset. The below figure 2 shows the time required during the processing of whole data

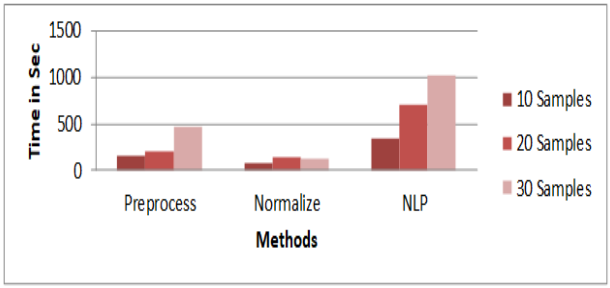


**Figure 2 :** Time required in seconds for data processing using proposed techniques

In the actual classification base, experimental analysis has done with various cross-fold validations. From 1000 heterogeneous class files have distributed in different code packages which contain different vulnerabilities and violations of code permissions. All data have evaluated with given three algorithms, the achieve results shows in below figures.

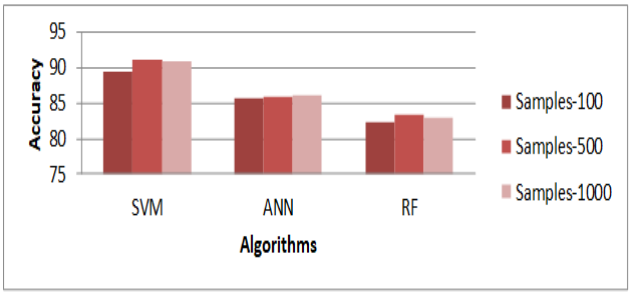


**Figure 3:** Classification accuracy of all machines learning algorithm on different dataset

According to above figure 3 we conclude on machine learning, classification algorithms provide a better classification accuracy on heterogeneous data set Moreover out of three algorithms support vector machine provide the best classification algorithms then artificial neural network and random forest. Three different cross-validation techniques evaluated 0with all the algorithms and SVM provides better around 91.0% classification and detection accuracy in each iteration.

## 7. CONCLUSION

Vulnerabilities in security are defined as system defects that are illegally exploited by unauthorized personnel. Thousands of safety problems are found in production software each year. Vulnerabilities sometimes appear in hidden forms that Software testers can't identify or left. Machine learning algorithms are a popular method for developing predictive models of vulnerability, authors add. They suggest the data-driven method of finding weaknesses utilizing deep learning is a possible solution. In this work, we will develop a cost-effective tool for developing heterogeneous vulnerability assessment and bug triage on windows as well as a web platform. Many tools don't support for a web-based application to detect code vulnerability. The system can work different datasets to extract the features and detect the vulnerability. SVM provides a better classification than the other two machine learning classifiers. In future developers to be needed is that to detect for code triage runtime mobile-based application programs so existing tools do not supports mobile application programs. Now a day's need is that in software engineering code clone management. Good quality of design achieved with the help of bugs free code clone in developing software. With the help of good quality of software improve the productivity of the software. Some several programmers are to be used in the future research management scope are as following.

- Improve the code clone detection system
- Visualization of clones
- Automating Refactoring Systems.

## REFERENCES


1. Gawron, Marian, Feng Cheng, and Christoph Meinel. **Automatic vulnerability classification using machine learning.** *In International Conference on Risks and Security of Internet and Systems*, pp. 3-17. Springer, Cham, 2017S. https://doi.org/10.1007/978-3-319-76687-4_1
2. Queiroz, Andrei, Brian Keegan, and Fredrick Mtenzi. **Predicting Software Vulnerability Using Security Discussion in Social Media,** (2017).
3. Harer, Jacob A., Louis Y. Kim, Rebecca L. Russell, Onur Ozdemir, Leonard R. Kosta, Akshay Rangamani, Lei H. Hamilton et al. **Automated software vulnerability detection with machine learning,** *arXiv preprint arXiv*:1803.04497 (2018).
4. Jurn, Jeesoo, Taeeun Kim, and Hwankuk Kim. **An Automated Vulnerability Detection and Remediation Method for Software Security.** *Sustainability 10, no. 5* (2018): 1652.
5. Li, Zhen, Deqing Zou, Shouhuai Xu, Hai Jin, Yawei Zhu, Zhaoxuan Chen, Sujuan Wang, and Jialai Wang. SySeVR: **A Framework for Using Deep Learning to Detect Software Vulnerabilities.** *arXiv preprint arXiv:*1807.06756 (2018).
6. Sabetta, Antonino, and Michele Bezzi. **A Practical Approach to the Automatic Classification of Security-Relevant Commits**. I*n 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 579-582. IEEE, 2018. https://doi.org/10.1109/ICSME.2018.00058
7. Han, Yi, Benjamin IP Rubinstein, Tamas Abraham, Tansu Alpcan, Olivier De Vel, Sarah Erfani, David Hubczenko, Christopher Leckie, and Paul Montague. **Reinforcement learning for autonomous defence in software-defined networking.** *In International Conference on Decision and Game Theory for Security*, pp. 145-165. Springer, Cham, 2018. https://doi.org/10.1007/978-3-030-01554-1_9
8. Dam, Hoa Khanh, Truyen Tran, Trang Thi Minh Pham, Shien Wee Ng, John Grundy, and Aditya Ghose. **Automatic feature learning for predicting vulnerable software components.** *IEEE Transactions on Software Engineering (2018).* https://doi.org/10.1109/TSE.2018.2881961
9. Russell, Rebecca, Louis Kim, Lei Hamilton, Tomo Lazovich, Jacob Harer, Onur Ozdemir, Paul Ellingwood, and Marc McConley. **Automated vulnerability detection in source code using deep representation learning.** *In 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pp. 757-762. IEEE, 2018. https://doi.org/10.1109/ICMLA.2018.00120
10. Ren, Jiadong, Zhangqi Zheng, Qian Liu, Zhiyao Wei, and Huaizhi Yan. **A Buffer Overflow Prediction Approach Based on Software Metrics and Machine Learning.** *Security and Communication Networks* 2019 (2019). https://doi.org/10.1155/2019/8391425
11. Markad Ashok Vitthalrao and Mukesh Kumar Gupta **Software Vulnerability Classification Based On Deep Neural Network,** *International Journal of Engineering and Advanced Technology (IJEAT) ISSN: 2249 – 8958*, Volume-9 Issue-1, October 2019 https://doi.org/10.35940/ijeat.A9746.109119
12. Hajah T. Sueno, Bobby D. Gerardo, Ruji P. **Multi-class document classification using SVM Based on improved Naive Bayes Vectorisation Techniques**, *International Journal of Advanced Trends in Computer Science and Engineering*, pp. 3937-3943 ,Volume 9, No.3, 2020.

https://doi.org/10.30534/ijatcse/2020/216932020

13. H. D. Gadade, Dr. D.K.Kirange, **Machine Learning Approach towards Tomato Leaf Disease Classification,** *International Journal of Advanced Trends in Computer Science and Engineering*, pp.490-495, Vol.9 , No.1,2020.
https://doi.org/10.30534/ijatcse/2020/67912020