SOFTWARE VULNERABILITY ANALYSIS USING MACHINE LEARNING TECHNIQUES

R.Deepthi Reddy,
Research Scholar, GITAM UNIVERSITY, Rudraram, Hyderabad.
draavi@gitam.in.
P.Anupama,
Assistant Professor, CSE Dept, SreeDattha Institute of Engineering and Science,Hyderabad,anunithyalohith@gmail.com
Dr.Nazimunisa
Associate Professor, CSE Dept, Sree Dattha Institute of Engineering and Science, Hyderabad
pnajma22@gmail.com.
S.Krishna Reddy
Assistant Professor, CSE Dept, Sree Dattha Institute of Engineering and Science, Hyderabad
krishnareddy.seelam99@gmail.com

ABSTRACT:

Software defects, errors, or weaknesses in the software's implementation, design, or architecture can all result in vulnerabilities. There is a potential to discover security vulnerabilities and learn about bug patterns as a result of the expansion of open-source code that is now available for analysis. The explosion has given rise to an opportunity. Recent developments in deep learning for image processing, audio identification, and natural language processing have shown how powerful neural models can be at understanding natural language. Many individuals are currently interested in the research of vulnerability identification, and over the past few decades, specialists have offered a wide range of methodologies. The detection of malware, the identification of software vulnerabilities, and the recognition of functions are just a few of the numerous real-world applications in which machine learning (ML) has been successfully used as a method for learning high-quality feature representations. Other examples include those previously mentioned. This is a comprehensive summary of the research that has been conducted in the field of detecting software vulnerabilities using machine learning techniques. First, we'll go over the fundamental ideas that underpin each of the three primary approaches to discovering vulnerabilities: static analysis, symbolic execution, and fuzzing. These detection methods are extremely common and frequently employ more standard methods. They do not include the most popular machine learning methods, such as supervised learning and deep learning techniques. When we are better prepared, we will be able to anticipate and focus on the research challenges in software vulnerability detection that require immediate attention. There are various possible uses for machine learning techniques in software measurement. Software defect prediction and software work estimation are two of the machine learning techniques most frequently used in software maintenance. The artificial neural network for SM is the most commonly used machine learning technology today. NASA and Promise databases are frequently used in empirical research. Over the last decade, the SM paradigm has gradually shifted away from ensembles of single ML models and toward deep learning models. Several factors contributed to this shift. Machine learning (ML) methodologies have typically produced favourable results when applied to a wide range of different prediction endeavours.

DOINumber:10.48047/nq.2022.20.22.NQ10406 **NeuroQuantology2022;20(22):4071-4078**

eISSN1303-5150

1.INTRODUCTION

These flaws are often created by coding faults and are easily transferable from one system to another due to the widespread use of open-source software and the reuse of existing code. Vulnerabilities still pose a serious problem despite academic and commercial efforts to increase software quality. This is evidenced by the fact that the Common Vulnerabilities and Exposures database receives a sizable number of vulnerability reports each year (CVE). The number of vulnerabilities reported in a single year will not have been higher than in 2020, according to the National Vulnerability Database (NVD) Report and the Common Vulnerabilities and Exposures (CVE) organization. This forecast is based on the expectation that the number of reported vulnerabilities will continue to rise. Figure 1 clearly shows that there has been a discernible increase in the severity of security concerns over the last three years.

Figure 1 depicts the number ofCVEs reported in 2010 versus 2020.

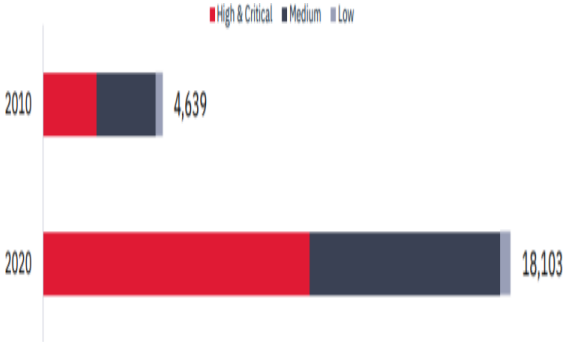

Figure 2 shows a comparison of the number ofCVEs reportedin 2010 and 2020.

Figure 2 depicts the rate of change, as demonstrated by the fact that there were more critical and high-severity vulnerabilities in 2020 (10,342) than in 2010. (4,639 total, including levels of low, medium, high, and critical severity).

The goal of this project is to create a web service that anticipates software vulnerabilities by utilising machine learning techniques. This service will be available to the general public and will be hosted on Microsoft's Azure cloud platform. The Area Under the ROC Curve (AUC) metric was used to assess the performance of a variety of machine learning models within the context of the Azure Machine Learning Studio environment. Following an evaluation of which model would produce the highest quality output, the most efficient model was chosen, In addition to these factors under the researcher's control, the research included a dependent variable titled "is vulnerable." This variable can be used to determine whether or not a vulnerability report for the module exists. As a result, we see the problem as a categorization exercise involving two categories. The investigations made use of three distinct datasets, each from an independent initiative. These datasets contain a total of 223 different types of vulnerabilities. When we compared their average area under the curve (AUC) values across three different datasets, the neural network performed significantly better than the other methods.

There could be a link between the rapid rise in the number of serious security issues and the rise in software vulnerabilities. It is beneficial for both customers and vendors to have an accurate forecast of the frequency of vulnerability disclosure before software developers release a product to their customers. This leads to an improvement in resource management on both sides. These forecasts can provide useful information that can be used to assess the risk associated with software. This is in addition to meeting the outlined functional and technical requirements. Fixes for vulnerabilities that will undoubtedly be discovered after the product has been released must be made readily available in order for developers to address those vulnerabilities.

Following the completion of the statistical investigation, we developed a set of metrics for the class level as well as a set of metrics for the method level. Following that, we used these measures as features in machine

learning techniques to predict which classes and methods were most vulnerable to vulnerabilities. Historically, software security research has been almost entirely focused on a few security models. within the confines of their respective systems. The primary goal of these models is to develop and construct models using mathematical functions and vulnerability density functions. The vast majority of the research examined was published in academic journals and focused on investigating AI methods based on either machine learning or deep learning models. The current research focuses on either a single method for representing features or on selecting the granularity level and dataset. Neither of these methods is ideal. The goal of this paper is to present a qualitative comparative analysis of models based on ML, DL, and GNN to provide a comprehensive picture of VDM. This paper investigates the difficulties that a researcher faces when attempting to choose an appropriate architecture, feature representation, and processing requirements for developing a VDM. These difficulties can be divided into three categories: architectural, feature, and processing. This effort is aimed at gathering all of the necessary components for building an automated software vulnerability detection model using any of the various AI approaches that are currently available. These fundamental components are as follows:
Currently, quantum machine learning has made progress in a number of domains that are related to one another. Natural language processing, recommendation systems, speech recognition, image classification, and the medical field are examples of these fields. The use of QML can speed up and improve the efficiency of programme execution. It is possible to achieve higher levels of performance than with traditional deep learning and other techniques by utilising big data training data that is available at a reasonable cost. This contributes to resolving the issues raised earlier in the conversation. To the best of our knowledge, no research has been conducted into the possibility of using quantum neural networks for vulnerability discovery. The memory bottleneck problem plagues classical machine learning, and it has been demonstrated that quantum neural networks can solve it. As a result, these networks have enormous potential in terms of vulnerability discovery. This is the first paper we've written, as well as the first study to use vulnerability data word embedding with a quantum neural network. As a result of this discovery, it demonstrates that a quantum neural network can be used to identify weak points in a system.

According to the most recent study, when it comes to predicting reported vulnerabilities (such as cross-site scripting, SQL Injection, and so on) based on a description of the flaw in the software, the neural network algorithm outperforms Naive Bayes, Support Vector Machines, and K-Nearest Neighbors. This is the conclusion reached after comparing the efficacy of these four methods. The neural network method's accuracy was compared to that of the Naive Bayes algorithm, the Support Vector Machine algorithm, and the Random Forest algorithm. The outcomes of these assessments led to the formation of this conclusion. After the efforts of a large number of researchers focused on developing methods for fault detection, the source code is eventually labelled as vulnerable in some form. This is the result of the efforts of the researchers. Both the papers and the presentation introduce the term "vulnerability extrapolation." Vulnerability extrapolation is the process of discoveringpreviously unknown vulnerabilities through a series of steps. This method is based on the detection of programming patterns found in existing security issues. This is an important step in the process that must be completed completely. The topic of projecting one's level of vulnerability comes up frequently in both the papers and the presentation. The study describes and covers the concept of PhpMinerI, which can detect whether a specific statement in the source code of PHP-based applications is vulnerable to cross-site scripting or SQL injection attacks. The study was conducted by This information is presented in the previously read document. The research findings are presented in this

section, along with an explanation of the PhpMinerI concept. The study was conducted and provides an explanation of the idea under consideration. Regrettably, none of these approaches is appropriate for use in the types of large-scale, real-world software development and security assurance operations that must be performed.

2.BACKGROUND INFORMATION AND RELATED WORK

Cybersecurity research makes extensive use of machine learning techniques. The body of scholarly research currently contains a number of publications .Identification and categorization of software vulnerabilities are the two major categories that can be used to roughly identify these problems. Below is a summary of the most important works from each of these groupings. Network traffic classification is a third type that needs to be highlighted. It refers to technologies that track network traffic accessing a particular asset to identify cyberattacks using ML algorithms. The incoming network traffic is monitored by these algorithms. Research on various self-defense techniques is also being done extensively in the area of mobile computing. On the other hand, these groups won't be covered in further detail because the study at hand focuses more on software vulnerabilities than network vulnerabilities.

Vulnerabilities in software security are often referred to as "security flaws," "security bugs," and "software weaknesses." The Mitre organisation, which is in charge of updating the Common Flaws and Exposures (CVE) dictionary, defines software security vulnerabilities as follows: "Software and hardware components are found to include a fault in computational logic, such as code, which, when exploited, has a detrimental effect on confidentiality, integrity, or availability. “A weakness in the computational logic, such as code, is detected in software and hardware components." Code-based vulnerabilities, which are merely one kind of vulnerability, are where the great majority of exploits originate. (Programs, data, or instructions that, if used, could violate a security policy.) "An instance of a mistake in software specification, development, or configuration that allows its execution to break a security policy" is the definition of a software vulnerability. In the article "What Is a Software Vulnerability?" this is defined. NIST is an acronym that stands for "National Institute of Standards and Technology. “A threat source could exploit or create a vulnerability in an information system's system security practises, internal controls, or implementation. Finding software flaws is referred to as "vulnerability detection." The use of humans is required for traditional vulnerability identification techniques, which can result in both false positives and false negatives. Domain experts must invest a significant amount of time and energy into creating custom features for spotting vulnerable code. Frequently, these customised qualities fall short of reflecting the semantic and structural details of the VUDDY and VulPecker both have a significant percentage of false-negative findings; VUDDY has 18.2% and VulPecker has 38%. The method for analysing code similarity is divided into three parts. The first step is to disassemble a programme into its component pieces of code and then reassemble it. In the second phase, code fragments can be abstractly represented in a variety of ways, including tokens, trees, and graphs. These are only some examples. The abstract representations created in the second phase will be used in the third phase, which evaluates the degree of similarity between different snippets of code by comparing them to those created in the first phase. approaches based on patterns, which can be further subdivided into methods based on rules and methods based on machine learning. Rule-based approaches to vulnerability detection rely on rules that are typically established manually by human professionals. These rules are used to determine whether a vulnerability exists or not. (Take Flawfinder, RATS, and Checkmarx, for instance.) These methods commonly produce both false positives and false negatives at high rates.

Table 1: current format lays out the informational framework

Field Name	Separation token	Description
Application Name	XXAN	Name of application in which con- text a vulnerability was detected. It could be an IP address when the deficit is affecting network object, URL when the issue was reported by the DAST scanning software and the name of the code repository if the vulnerability is present in the source code.
Application context	XXAC	Context of the application. This field contains information about the source of a vulnerability, authentication type, network zone (internal or external) and the type of end user (employee or customer).
Vulnarability Name	XXVN	Otaken from the scanning softwart It could be the CVE name or a direct phrase like "Cross-Site Script- ing Reflected".
Vulnarability Description	XXVD	This field contains a description of a vulnerability and information about why it is detected in the specific context.

The effectiveness of the machine learning method in identifying vulnerabilities has been proven. It was claimed that many problems had the wrong labels and that labelling bugs as vulnerabilities was ineffective. 223 vulnerabilities were accumulated during the creation of this vulnerability collection. According to reports, security issues and complexity measurements are related. Researchers examined the association between developer activity, complexity, code churn, and the existence of software security vulnerabilities using logistic regression. Decision trees were used to predict vulnerabilities based on assessments of complexity, cohesion, and coupling. It has been found that conventional standards like complexity only loosely correlate with Windows Vista vulnerabilities. Additionally, the researchers looked at SQL hotspots, or regions. It was decided whether including statements had anything to do with vulnerabilities. To predict vulnerabilities, an approach based on dependency graphs was created. The variables linked to data flow and sanitization were used by the researchers to build models. On PHP applications, a study of the relationship between vulnerability density and code metrics was conducted. The creation of vulnerability prediction models utilised static analysis warnings

3. MATERIALS AND METHODS

A "dataset" is a collection of data that is usually associated with a specific field of study or investigation. There are currently 156468, with seven of the most cutting-edge being among them. SARD is an acronym that stands for "known security vulnerabilities." This term is used to describe flaws discovered in test cases and test suites. Both of these datasets have a flaw in that their data may not be representative of real-world software products, which can lead to disparities in performance findings across studies. This is a problem because real-world software is used in the real world. The imbalance of the datasets was a problem in the majority of the studies examined, which led to certain biases

in the findings. In this section, we will begin by introducing the datasets that were used during the evaluation process. In addition, we provide a method for feature extraction. The methodology used throughout the experimental stage is discussed in the final step of this process. This process resulted in a tagged dataset, with each vulnerability labelled as "CRV" or "DNRV."

Table 1 in its current format lays out the informational framework of the compiled vulnerabilities.

A "dataset" is a collection of data that is frequently associated with a specific field of study or investigation. Seven of the most recent and innovative studies we examined used both the National Vulnerability Database (NVD) and the Software Assurance Reference Dataset. The remaining studies used one of those datasets or open-source initiatives.

SARD stands for "security vulnerabilities already known," and it refers to flaws discovered in test cases and test suites. Both of these datasets have the same problem in that their It is possible that the data do not accurately reflect the software products used in the real world; if this is the case, many studies' findings may need to be revised. This presents a challenge due to the prevalence of real-world software in said real world. The imbalance of the datasets was a problem in the majority of the studies examined, which led to certain biases in the findings. In this section, we will begin by introducing the datasets that were used during the evaluation process. In addition, we provide a method for feature extraction. The methodology used throughout the experimental stage is discussed in the final step of this process.

Table 1 in its current format lays out the informational framework of the compiled vulnerabilities.

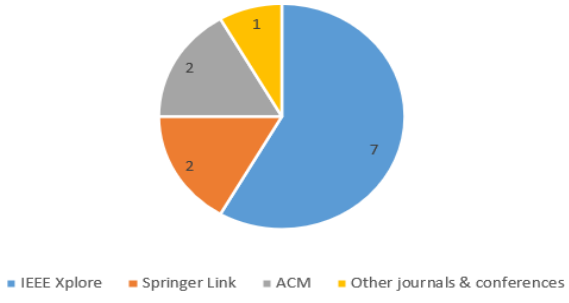

Figure 3: First-tier research classification

The first method is called Multi-Layer Perceptron (MLP), and the other two are called Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN). Deep learning neural networks are an umbrella term for all three approaches. It is necessary to put in place a variety of neural networks, each of which has the potential to be used for vulnerability identification. RQ3's investigation is focused on the level of granularity of the source code used to describe the programme used to extract the characteristics of the vulnerabilities. Some preliminary research on the function level discovered that by applying automatic analysis of deep learning and machine learning approaches to the challenge of vulnerability identification, it is possible to extract features in an automated manner. This was successful in achieving the desired outcome. RQ5 focuses on the assessment metrics used to evaluate vulnerability detection models. These metrics include the false-positive and false-negative rates, as well as precision, recall, and F1 Score.

4. OBSERVATION AND FINAL REMARKS

This experimental evaluation used an application-context-aware dataset to determine whether or not a machine learning algorithm could effectively classify vulnerabilities within that dataset. This result was made possible by the combination of the prepared input (discussed in the previous section) and the algorithms. Table 2, which can be found further down, provides a concise summary of the results obtained. They show the connection between the classifier and the validation dataset.

Table 2 displays the results of experiments conducted to classify software vulnerabilities using a variety of different machine learning methods.

Table 2: Classify software vulnerabilities

ML Algorit	Acc(%)	Pre(%)	Recall(%)	F1 sco	Learning

hm				re	time
NN	0.98	0.89	0.98	0.98	0.89
RN LSTM	0.99	0.87	0.99	0.97	0.96
RNN GRU	0.98	0.78	0.97	0.98	0.97
RF	0.97	0.97	0.99	0.89	0.98
SVM	0.99	0.99	0.94	0.96	0.99

The CNN, plain NN, and RNN with LSTM precision and recall values were all comparable, and the appropriate levels of customer satisfaction were 98% and 99%, respectively, the results for both metrics were significantly worse than if either one had been used alone. Given that the GRU unit is not particularly adept at dealing with lengthy sentences, this type of behaviour was to be expected. As a result, such behaviour was to be expected. The results produced by the RNN equipped with LSTM memory were superior in terms of consistency and recall. Table 2 shows the results of using the RF classifier and the SVM classifier, respectively. The published results are the averages obtained over ten distinct time periods. Individual runs with variances of less than one percent result in extremely consistent metrics produced by these classifiers. This supports the claim that the data was correctly and arbitrarily divided into training and verification sets. We purposefully manipulated the dataset to ensure that the research presented was reliable. We chose cases that shared the same risk and context but had different applicability to demonstrate our concepts. 50% of these samples have had their labels reversed to show the opposite value. As a result, the dataset contained contradictions that should not have existed in the first place. When it was discovered that 10% of all cases were corrupted, metrics for every classifier suffered significantly. The RNN, CNN, and RF models lost 13% of their accuracy and precision as a result of the dataset adjustment, while the SVM model lost 20%. Regrettably, the aforementioned dataset contains information, such as the financial services sector. This issue would eventually become the primary focus of our research.

5. CONCLUSION

Researchers now have the opportunity to investigate potential software security issues made possible by advances in machine learning and deep learning technologies. Various automated approaches have been presented as a result of these advancements. One of the most important research areas recently has been the automatic detection of security flaws. Traditional research methods and machine learning were used to analyse these works. As a result of these advancements, a plethora of automated methods have emerged as viable problem-solving options. Recently, one of the most important research areas to focus on has emerged: the automatic identification of security concerns. This development occurred as a result of recent events. In this paper, we investigate and analyse previous studies on the detection of software vulnerabilities using deep learning and machine learning approaches. These techniques were used both separately and in combination. These investigations' findings have only recently been compiled. The goal of this study was to compare the effectiveness of various machine learning strategies for categorising software vulnerabilities. We determined that certain strategies were more useful than others based on our findings. The research for this study was conducted in the United Kingdom in order to provide more information. These results were obtained by employing the NN and RF algorithms in the analysis process, respectively. To advance to the next level, you needed a score of 98 in each of these phases. When all relevant factors, including measurements and training time, were taken into account, the Random Forest algorithm produced the highest-quality output. The evaluation, on the other hand, used a dataset that only included examples of software bugs from one hundred different applications. The vast majority of these software flaws are caused by the abundance of e-commerce platforms on the market today.

REFERENCE:

1. BahaaFarid, Ahmed & Kamal, Aya & Ghoneim, Amr. (2022). A Systematic Literature Review on Software Vulnerability Detection Using Machine Learning Approaches. 4. 1-9.
2. Jiang, Jian& Yu, Xiangzhan& Sun, Yan & Zeng, Haohua. (2019). A Survey of the Software Vulnerability Discovery Using Machine Learning Techniques. 10.1007/978-3-030-24268-8_29.
3. Catal, Cagatay&Akbulut, Akhan&Ekenoglu, Ecem&Alemdaroglu, Meltem. (2017). Development of a Software Vulnerability Prediction Web Service Based on Artificial Neural Networks. 59-67. 10.1007/978-3-319-67274-8_6.
4. Jabeen, Gul& Rahim, Sabit& Afzal, Wasif& Khan, Dawar& Khan, Aftab&Bibi, Tehmina. (2022). Machine learning techniques for software vulnerability prediction: a comparative study. Applied Intelligence. 52. 10.1007/s10489-022-03350-5.
5. Hanif, Hazim& Nasir, Mohd&Faizal, Mohd&Firdaus, Ahmad &Anuar, Nor. (2021). The rise of software vulnerability: Taxonomy of software vulnerabilities detection and machine learning approaches. Journal of Network and Computer Applications. 179. 103009. 10.1016/j.jnca.2021.103009.
6. Zhou, Xin& Pang, Jianmin&Yue, Feng & Liu, Fudong&Guo, Jiayu& Liu, Wenfu& Song, Zhihui&Shu, Guoqiang& Xia, Bing & Shan, Zheng. (2022). A new method of software vulnerability detection based on a quantum neural network. Scientific Reports. 12. 10.1038/s41598-022-11227-3.
7. Goyal, Somya. (2022). Software Measurements Using Machine Learning Techniques - A Review. Advances in Mathematics of Communications. 16. 10.2174/2666255815666220407101922.
8. Ban, Xinbo& Liu, Shigang& Chen, Chao & Chua, Caslon. (2018). A performance evaluation of deep-learnt features for software vulnerability detection. Concurrency and Computation: Practice and Experience. 31. 10.1002/cpe.5103.
9. Sultana, KaziZakia&Anu, Vaibhav& Chong, Tai-Yin. (2020). Using software metrics for predicting vulnerable classes and methods in Java projects: A machine learning approach. Journal of Software: Evolution and Process. 33. 10.1002/smr.2303.
10. Pooja, S & Chandrakala, C. &Raju, Laiju. (2022). Developer's Roadmap to Design Software Vulnerability Detection Model Using Different AI Approaches. IEEE Access. 10. 1-1. 10.1109/ACCESS.2022.3191115.