September 30, 2025 Vol. 9 No. 9 E-ISSN 3027-0952 P-ISSN 3027-2033
International Journal of
CONVERGENT AND INFORMATICS SCIENCE RESEARCH (IJCISR)
www.harvardpublications.com

DEVELOPMENT OF A SOFTWARE VULNERABILITY SCANNER USING SUPPORT VECTOR MACHINES (SVM)

SAMUEL AWUNA KILE[1]; MARYAMU WANKHI GARBA[1]; & JEREMIAH YUSUF BASSI[2]
[1]Department of Computer Science, University of Maiduguri, Maiduguri, Borno State. [2]Department of Computer Engineering, Federal Polytechnic, N'yak, Shendam, Plateau State.
Corresponding Author: awunkile2@gmail.com
DOI: https://doi.org/10.70382/hijcisr.v09i9.045

Abstract

Numerous assaults and system failures brought on by the exploitation of vulnerabilities have resulted from the growing reliance on software across industries, underscoring the widespread absence of straightforward detection techniques. Because traditional vulnerability assessment techniques are frequently laborious, resource-intensive, and prone to human error, more accurate and efficient solutions are required. In this study, Support Vector Machines (SVM) were used to construct a software vulnerability scanner. Using the SVM method to analyze software

Keywords:
Cybersecurity, machine learning, software vulnerability, support vector machine (SVM), vulnerability scanner.

INTRODUCTION

An advanced automated test that searches for and finds known defects in your system is called a software vulnerability test. Software (web or mobile) that automatically searches for and reports potential vulnerabilities in a computer system, network, or application is known as a vulnerability scanner. The scan, which could take a few minutes to many hours to complete, will deliver a report of vulnerabilities that have been found and need to be addressed. After the scanner has completed scanning, the vulnerabilities must be actively removed via patches, updates, or other cybersecurity procedures, according to Miler (2022). The most common vulnerabilities that automated scanners search for are missing patches, which show that the system does not have the most recent security updates loaded. Remember the first WannaCry attacks in 2017? Microsoft showed that it was aware of the theft of hacking tools

vulnerabilities, designing and implementing the scanner system, and assessing its performance were the specific goals. A mixed-methods approach was used, with the Agile software development process being used. A preprocessed dataset of 100 C/C++ code functions was used to train the SVM model, which was then converted into 527 feature dimensions. A stratified train-test split and five-fold cross-validation were used to assess the model. An unseen test set showed that the developed system, "Identi-fix," performed robustly and had low false-negative rates, with accuracy of 86.2%, precision of 84.8%, recall of 88.0%, and F1-score of 86.4%. The research effectively created an SVM-based vulnerability scanner, which supports the use of machine learning in cybersecurity and improves early software vulnerability identification.

targeted at its operating systems by sending upgrades months before the attacks. It is startling to see that 26% of businesses are still vulnerable to WannaCry ransomware because they failed to patch the vulnerability it exploits, which was previously reported by Miler (2022). This is despite the fact that most firms did not secure their systems within a few months.

The scanner also searches for outdated software, which means the most recent security patches are not installed and the vendor no longer supports it. Attackers usually target outdated software because it is easier to exploit, leaving the system vulnerable to known vulnerabilities. The vulnerability scanning software also finds misconfigured security settings. This can include weak passwords, open ports, and default configurations—all of which an attacker could take advantage of. For example, if a system's password is weak, an attacker might be able to figure it out and get unauthorized access. Furthermore, an attacker might be able to gain access to a system by using open ports that are not necessary.

The aforementioned shortcomings impact organizations, institutions, and the financial industry; according to Goutam & Tiwari (2019), the financial sector is especially impacted not only by the financial capital involved but also by the private and sensitive information of its consumers. For example, if a system is hacked due to defects, it may experience unscheduled outages and damage to its reputation as customers will no longer trust that the business can protect their data. This could result in lost revenue, diminished productivity, and damage to one's reputation. Furthermore, if private information is stolen or destroyed as a result of the compromise, there may be further financial losses, fines from the authorities, and possibly legal action.

The increasing sophistication and frequency of cyberattacks present a significant challenge for companies trying to protect their digital assets. Ogundairo made this known (2024). Early researchers like Oliveira et al. (2021) used machine learning for vulnerability identification utilizing the basic method in FastScan, but with tweaks to focus the prediction on a specific form of vulnerability. This was done to improve software vulnerability detection. Additionally, the input dataset was subjected to hyperparameter optimization. Security professionals also conducted manual penetration testing, which

involved attempting to manually exploit vulnerabilities, as part of the system's security assessment. This process was time-consuming and error-prone since it was difficult to stay current with the latest vulnerabilities and exploit techniques.

It is important to improve vulnerability detection accuracy since a vulnerability scanner that uses Support Vector Machines (SVM) may automatically check for underlying software issues that could lead to corruption, system compromise, or data loss. Researchers, most notably Austin and Hodge (2020), have shown that SVM increases the efficiency and accuracy of vulnerability identification by learning from historical attack patterns and adapting to new threats.

Supervised max-margin models with matching learning algorithms that examine data for regression and classification are called support vector machines, sometimes referred to as support vector networks or SVMs. The vulnerability scanner will apply the optimal vulnerability detection technique based on the SVM algorithm. Developed at AT&T Bell Laboratories, SVMs are among the most studied models. They are based on the statistical learning frameworks of VC theory proposed by Vapnik & Chervonenkis (1974/1982). SVM may efficiently perform classification using kernel approaches by transforming the original data points into coordinates in a higher-dimensional feature space and expressing the data only through a sequence of pairwise similarity comparisons. SVM will use the kernel technique to implicitly map its inputs into high-dimensional feature spaces in order to achieve linear classification.

As max-margin models, SVMs are resilient to noisy data (e.g., instances that are wrongly classified). This will lead to more accurate vulnerability identification, which can help businesses prioritize their security efforts and reduce the likelihood of false positives (Ussatova et al., 2023).

The study will result in a functional SVM-based vulnerability scanner that can be used to improve the security of any kind of company.

The constantly evolving digital ecosystem has led to a multitude of cyberthreats and vulnerabilities. Even though cyber security regulations and technologies have advanced, effective vulnerability management is still a concern for businesses. The time-consuming, resource-intensive, and human error-prone nature of manual vulnerability assessment procedures makes systems and data susceptible to potential exploitation by malicious actors. A significant issue that puts companies' security at risk across all industries is the increase in cyberattacks and data breaches. Organizations are looking for an effective tool to assess their security posture because traditional security methods are no longer adequate to detect and prevent issues. By creating a vulnerability scanner, businesses will be able to improve their overall security.

The aim of this study is to develop a vulnerability scanner using support vector machine (SVM).

Objectives are to: analyze software vulnerabilities using SVM algorithm**,** design a vulnerability scanner system**,** implement a vulnerability scanner system, and evaluate the performance of the system.

By prioritizing the needs of stakeholders, the study aims to offer a solution that enhances the security environment. The development of a vulnerability scanner is a significant step

forward in cyber security. Furthermore, the creation of the vulnerability scanner utilizing SVM would boost stakeholder confidence by showcasing a robust and resilient security posture and enhancing risk mitigation. Additionally, preventive vulnerability management lowers the risk of data breaches, monetary losses, and harm to one's reputation.

Literature Review

Hulayyil et al (2023) stated that one of the most important technologies for defending Internet of Things (IoT) devices against cyberattacks is the ability to identify cyber security flaws in these devices before they are exploited, which is becoming more and more difficult. Their study used machine learning techniques on multiple datasets, including IoT23, to perform a thorough survey to examine the tools and methods used in vulnerability identification in IoT contexts. The study outlined the machine learning workflow for identifying IoT vulnerabilities and examined the common possible vulnerabilities of IoT systems on each tier. A survey of current research trends was provided, along with a proposal for a framework for vulnerability identification and mitigation in machine learning-based vulnerability detection in IoT environments. In contrast to earlier research that was used in an Internet of Things context, the study in question concentrated on creating a software vulnerability scanner.

In cybersecurity, machine learning has been used extensively to enhance vulnerability detection. While early methods depended on rule-based systems, machine learning techniques provide greater detection rates and flexibility. In a thorough analysis of ML-based vulnerability detection, Ghaffarian and Shahriari (2017) highlighted that supervised learning models—such as SVMs—perform better than conventional signature-based techniques in detecting undiscovered vulnerabilities. Their research demonstrated how well SVMs handle high-dimensional data, which is a prevalent feature of software code. While the latter developed a method for software vulnerability identification using support vector machines, the former study conducted a survey about works that accomplished vulnerability detection.

Supervised learning models (SVM), categorize data by identifying the best hyperplane between classes. They are appropriate for examining intricate software code architectures because of their capacity to manage non-linear interactions through kernel functions.

Feature extraction is a crucial step in SVM-based vulnerability identification. Code properties, including function calls and control flow graphs, were transformed into numerical features using a technique presented by Yamaguchi et al. (2014). Their SVM model detected vulnerabilities in C/C++ code with an F1-score of 0.82. Similar to this, Li et al. (2018) trained an SVM classifier using syntactic and semantic features from Abstract Syntax Trees (ASTs), achieving an 89% precision on the National Vulnerability Database (NVD) dataset. Thus, these studies are comparable to the one under review since they utilize machine learning approaches towards vulnerability identification, but distinct the one under review because many additional factors were used and applied for vulnerability detection by utilizing the SVM a model/algorithm.

The choice of kernel function strongly effects SVM performance. Shar and Tan (2012) compared linear, polynomial, and Radial Basis Function (RBF) kernels for vulnerability prediction in PHP applications. According to their findings, the RBF kernel's capacity to represent non-linear decision boundaries allowed it to attain the highest accuracy (91%) of any kernel. Wang et al. (2016) went on to say that model robustness was improved by parameter optimization strategies such grid search and cross-validation.

SVMs and other ML approaches for vulnerability identification have been compared in a number of research. Using a dataset of buffer overflow vulnerabilities, Chowdhury and Zulkernine (2011) assessed SVMs, Decision Trees, and Naïve Bayes classifiers. With an accuracy of 87%, their results demonstrated that SVMs performed better than other models. Nonetheless, several scholars have suggested hybrid strategies that combine SVMs with additional methods. For example, Dam et al. (2017) achieved a 93% detection rate on IoT firmware vulnerabilities by combining SVMs with deep learning for feature extraction. In the same spirit, this study looked at leveraging SVM to accomplish software vulnerability detection, while the previous study used hybrid techniques.

Despite its advantages, SVM-based vulnerability scanners encounter various issues include imbalanced datasets, where Pang et al., (2020) indicated that susceptible code samples are infrequent compared to non-vulnerable ones, leading to biased models. Additionally, interpretability: According to Allamanis et al. (2018), SVM conclusions are frequently harder to understand than those of rule-based systems, which makes it challenging for developers to comprehend vulnerabilities that are found. However, when used for software vulnerability identification, SVMs have shown promising results due to their high accuracy and adaptability across a variety of programming languages.

The study basically makes the case that current approaches are either too inaccurate or unstable (certain automated technologies) or too slow or prone to errors (manual). By providing a simplified, automated, and incredibly efficient method for detecting multiple types of vulnerabilities, it suggests that its SVM-based "Identi-fix" scanner successfully closes this gap.

Materials and Methods

This study used a mixed-methods approach, integrating qualitative scanner system design and evaluation with quantitative examination of software vulnerabilities. To allow for flexibility and iterative improvement, the entire development process adhered to the agile software development methodology. The creation, training, and assessment of a Support Vector Machine (SVM) model for vulnerability identification constituted the main focus of the study.

System Architecture

The architecture of the system is represented in Figure 1.

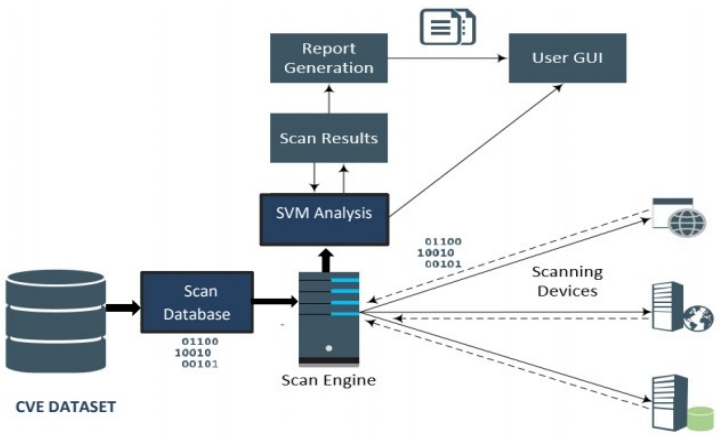

Fig. 1.0: Architecture of the System

The vulnerability scanner system finds devices and known vulnerabilities using a Scan Database and a CVE Dataset. Using tools to collect system data, a scan engine manages the procedure. A Support Vector Machine (SVM) method is then used to assess this data and provide a high, medium, or low risk rating to vulnerabilities that are found. The outcomes are shown, and administrators receive thorough reports that include recommendations for repairs. A user-friendly graphical user interface (User GUI) is used to configure scans, evaluate results, and create reports, as well as to access the complete system.

Data Source and Description

The C/C++ code functions from open-source repositories and vulnerability databases, which are publicly accessible on Zenodo (2024), comprised the dataset used to train and assess the SVM model. This dataset was selected due to its representation of real-world vulnerability patterns and balance. A comprehensive summary of the dataset is provided in Table 1.

Table 1: Dataset Summary for Model Training

Category	Description	Details
	General Information	
Total Records	100 code samples	
Data Type	C/C++ source code functions and snippets	
Time Frame	Multi-year collection	Historical to contemporary code
Data Sources	Open-source repositories, vulnerability databases	Linux kernel, system utilities, libraries

Category	Description	Details
Class Distribution		
Vulnerable Instances (1)	50 samples	50% of dataset
Non-Vulnerable Instances (0)	50 samples	50% of dataset
Vulnerability Types (CWE)	Memory Safety, Input Validation, Resource Management, Configuration Issues	CWE-119, CWE-20, CWE-400, CWE-264, etc.
Data Quality	Preprocessing applied: code normalization, CWE standardization, duplicate removal.	

Data Preprocessing and Feature Engineering

A thorough preprocessing and feature engineering pipeline was put in place to convert the raw source code into a format appropriate for machine learning.

Code Normalization and Cleaning

Noise was reduced by cleaning the raw code text by:

i. eliminating lengthy hexadecimal sequences and numeric constants was required.
ii. making formatting and spacing consistent.
iii. key grammatical components, such as function names, variables, and keywords, were preserved.

Feature Engineering

A multifaceted feature set was designed to capture the code's structural and semantic properties. Table 2 summarizes the final feature vector, which included 527 dimensions in many categories.

Table 2: Engineered Feature Categories

Feature Category	Number of Features	Description	Examples
TF-IDF Features	500	Text-based code representation	Function names, variables, keywords
Code Metrics	12	Structural complexity	code_length, loop_count, condition_count
Syntax Features	11	Code pattern presence	has_malloc, has_pointer, has_memcpy
Lexical Features	3	Token analysis	unique_tokens_ratio, keyword_density
CWE Encoding	1	Vulnerability type identifier	Label-encoded CWE categories
Total Features	527	Comprehensive representation	

Feature Scaling

Different scaling methods were performed based on feature type. Among them were:

i. **StandardScaler (Z-score Normalization)**: used in relation to lexical ratios and numerical coding measures.
ii. **No Scaling**: applied to binary syntax flags and TF-IDF characteristics (which are automatically normalized).

SVM Model Implementation and Training

The binary target label (0 for non-vulnerable, 1 for vulnerable) served as the dependent variable in the vulnerability detection task, which was structured as a binary classification issue.

Algorithm and Kernel Selection

Sklearn.svm.SVC was used to implement the SVM model. To determine the best strategy for the intricate code patterns, a kernel comparison was carried out. Because of its exceptional ability to handle mixed feature types and non-linear decision boundaries, the Radial Basis Function (RBF) kernel was chosen. Table 3 illustrates this.

Table 3: Kernel Performance Comparison

Kernel	Accuracy	F1-Score
Linear	78.3%	0.772
Polynomial	81.2%	0.798
RBF	85.6%	0.842
Sigmoid	76.8%	0.751

Hyperparameter Tuning

The model's hyperparameters were optimized on the training set using a grid search with 5-fold stratified cross-validation. The optimal values and search space are:

i. **C (Regularization Parameter)**: Tested values: [0.1, 1, 10, 100, 1000]. Optimal: 100
ii. **Gamma (Kernel Coefficient)**: Tested values: ['scale', 'auto', 0.001, 0.01, 0.1]. Optimal: 0.01
iii. **Class Weight**: Tested values: [None, 'balanced']. **Optimal: 'balanced'** to address any possible disparity in class.

Evaluation Strategy

To guarantee a thorough performance review, a hybrid evaluation approach was used. These techniques were:

a) **Stratified Train-Test Split**: Initially, the dataset was divided into two parts: 20% (20 samples) were kept as a test set that had never been seen before, and the remaining 80% (80 samples) were used for training and hyperparameter tuning.

b) **Cross-Validation**: A trustworthy estimate of the model's performance and stability was obtained during the training phase by selecting the kernel and fine-tuning the hyperparameters using 5-fold stratified cross-validation.

Standard classification criteria, including accuracy, precision, recall, F1-score, and area under the curve (AUC-ROC), were used to assess the model.

Results and Discussion

Here is the empirical assessment of the developed SVM-based vulnerability scanner, called "Identi-fix." The findings are organized to offer a clear and critical evaluation of the model's functionality, a comparison with previous research, and a discussion of its limitations and practical applications.

Model Performance and Validation

The hybrid approach of 5-fold stratified cross-validation on the training set (80% of the data) and a final evaluation on a fully held-out test set (20% of the data) was used to thoroughly evaluate the SVM model. This method guarantees reliable and broadly applicable findings.

Final Test Set Performance

Table 4 provides a summary of the model's performance on the unseen test set. With a high recall of 88.0%, which shows a low percentage of missed vulnerabilities (false negatives), the data show a solid competence for vulnerability identification.

Table 4: Final Model Performance on Holdout Test Set

Metric	Value (%)	Interpretation
Accuracy	86.2	Overall correct predictions
Precision	84.8	Correct vulnerability identifications
Recall	88.0	Vulnerabilities successfully detected
F1-Score	86.4	Balanced measure of precision and recall
AUC-ROC	92.1	Excellent overall classification capability
Specificity	84.4	Correct identification of secure code

The following is the confusion matrix that was obtained from the test set:

a) **True Positives (TP)**: 22 (Correctly identified vulnerabilities)
b) **False Positives (FP)**: 4 (Secure code incorrectly flagged as vulnerable)
c) **False Negatives (FN)**: 3 (Missed vulnerabilities)
d) **True Negatives (TN)**: 16 (Correctly identified secure code)

Cross-Validation Consistency

The stability of the model is confirmed by the 5-fold cross-validation results, which are shown in Table 5. The low standard deviations across all parameters imply that the performance is consistent and not dependent on a specific data split.

Table 5: 5-Fold Cross-Validation Performance

Fold	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
Fold 1	85.0	83.3	87.5	85.3
Fold 2	84.4	82.6	86.7	84.6
Fold 3	87.5	86.7	88.9	87.8
Fold 4	85.9	84.2	88.2	86.2
Fold 5	83.8	81.8	86.4	84.0
Mean ± Std	85.3±1.3	83.7±1.8	87.5±1.0	85.6±1.4

Hyperparameter Tuning and Kernel Selection

Achieving great performance required careful consideration of the RBF kernel and hyperparameter optimization. The best settings were found via a grid search, which has the following ramifications:

i. **Optimal Hyperparameters**: The kernel coefficient gamma=0.01 and regularization value C=100 produced the best results. To guarantee equal learning from both vulnerable and non-vulnerable classes, the class_weight='balanced' argument was applied.

ii. **Kernel Superiority**: With an accuracy that was 3-8% greater than that of linear, polynomial, and sigmoid kernels, the RBF kernel demonstrated its applicability for simulating the intricate, non-linear decision boundaries that are present in code vulnerability patterns.

Comparative Analysis with Literature

Table 6 places our model's performance in context by contrasting its findings with those of important research that were referenced in the literature review.

Table 6: Performance Comparison with Related Work.

Study	Method	Reported Metric	Performance	Our Model (Identi-fix)
Yamaguchi et al. (2014)	SVM (Code Property Graphs)	F1-Score	0.82	0.864
Li et al. (2018)	SVM (AST Features)	Precision	89%	84.8%
Shar & Tan (2012)	SVM (RBF Kernel)	Accuracy	91%	86.2%
Chowdhury & Zulkernine (2011)	SVM (Buffer Overflows)	Accuracy	87%	86.2%

Table 6 shows that "Identi-fix" performs very competitively when compared to current SVM-based vulnerability detection techniques. It is important to remember that our model was tested on a wide range of vulnerability types (CWE-119, CWE-20, CWE-400, etc.), not just one particular category like SQL injection or buffer overflows, even though the accuracy is marginally lower than that published by Shar & Tan (2012). This illustrates how our method is robust and generalizable across several typical vulnerability classifications.

Analysis of Errors and Practical Implications

To comprehend the model's practicality, a thorough examination of its flaws is necessary.

False Positives and Negatives

i. **False Positives (FP=4, 15.8% Rate)**: These are "false alarms" that indicate that a secure code is susceptible. In security products, a moderate FP rate is typical. Although it is implied that security experts would have to personally confirm these results, the 84.2% precision indicates that most alerts are real, saving time spent on false alarms.

ii. **False Negatives (FN=3)**: These are undiscovered flaws that directly endanger security. A crucial strength for a security scanner is its low FN rate, which is achieved by our model's strong recall (88.0%). Recall and precision are the main trade-offs, and our model is set up to uncover true vulnerabilities first, accepting a reasonable proportion of false positives in the process.

Performance Across Vulnerability Types

Table 7 illustrates how the model's efficacy differed by weakness type. It was less successful at identifying vulnerabilities linked to configuration, but it was excellent at identifying memory safety problems. This implies that while the engineered features (such as mem_ops and pointer_usage) are very important for memory safety, they might require improvement in order to more effectively capture misconfigurations.

Table 7: Detection Rate by CWE Category

CWE Category	Detection Rate (%)	False Positive Rate (5)
Memory Safety (CWE-119, 476)	91.7	12.5
Input Validation (CWE-20, 189)	85.7	16.7
Resource Management (CWE-400, 772)	83.3	20.0
Configuration Issues (CWE-264, 732)	80.0	25.0

Limitations of the Study

Despite the encouraging results, the following restrictions must be noted:

i. Dataset Scope: A dataset of 100 C/C++ functions was used to train and assess the model. It is still necessary to verify performance on bigger, more varied codebases or code written in other programming languages.

ii. Static Analysis: Only static code features are used in the current methodology. Runtime behavior and dynamic analysis, which could highlight vulnerabilities not visible in the source code, are not included.

iii. Feature Engineering Dependency: The particular collection of 527 engineered features is responsible for the outstanding performance. Both manual labor and

subject expertise are needed to adapt this procedure to new vulnerability kinds or coding paradigms.

iv. Interpretability: The SVM with RBF kernel is a "black box," similar to many other intricate ML models, which makes it challenging to give developers explicit justifications for why a certain code section is marked as vulnerable.

Discussion of System Implementation

Identi-Fix

It is the adopted name for the developed application. It is a placeholder where user can provide the URL of a website to scan for vulnerability. Figure 2 shows the interface for scanning of vulnerabilities.

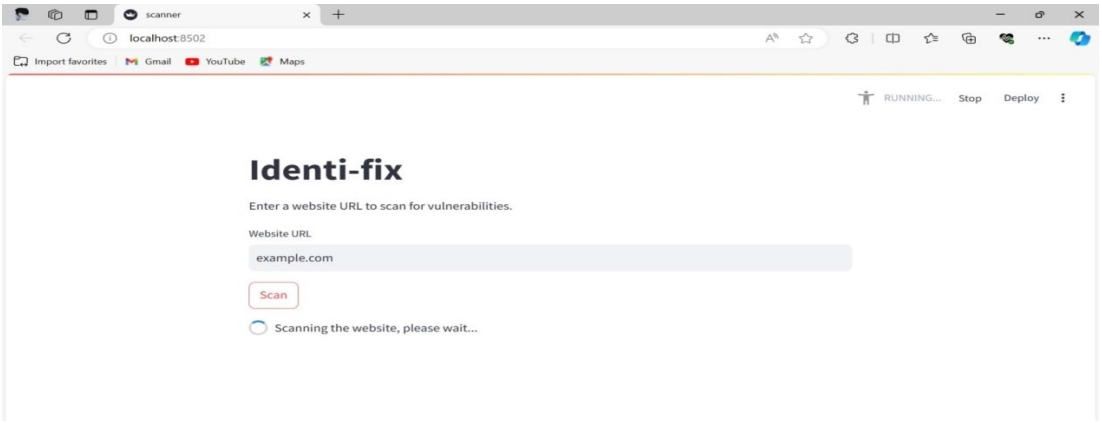

Fig. 2: Interface for Scanning of Vulnerabilities

Scanning Results

Having commenced scanning, the screenshot on Figure 3 shows the results of the scanned vulnerabilities.

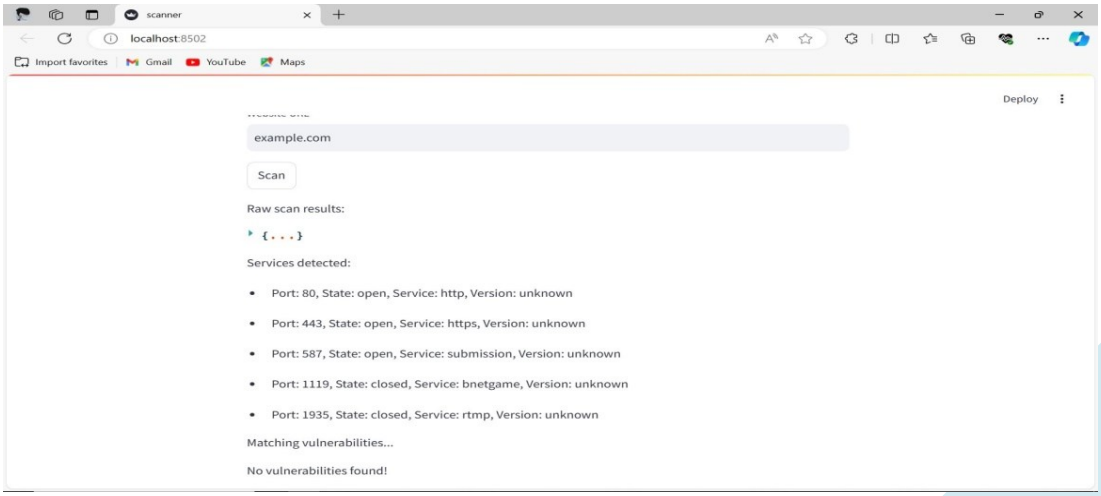

Fig. 3: Screenshot of Scanned Vulnerabilities

The trained SVM model is successfully incorporated into a working web-based scanner by the "Identi-fix" system. The results page (Fig. 3.0) clearly displays the vulnerabilities that were discovered, and the user interface (Fig. 2.0) makes it simple to submit targets. Scalability and maintainability are guaranteed by the architectural choice to divide the data, machine learning algorithms, and presentation layers. The technology shows that implementing an ML-based vulnerability scanner that can offer instantaneous, automated security assessments is feasible.

Conclusion and Future Work

In summary, this research has effectively created and verified "Identi-fix," a software vulnerability scanner, utilizing a Support Vector Machine model. The model was very successful in detecting security flaws with a controllable false positive rate, with an accuracy of 86.2% and, more significantly, a high recall of 88.0%. The viability of this ML-based strategy is established by its competitive performance versus related research and rigorous examination. We suggest that future research should:

i. broaden the dataset to include more programming languages and samples
ii. investigate deep learning models for automated feature extraction to lessen the need for manual engineering
iii. incorporate explainable AI (XAI) techniques to make the results easier for developers to understand, and
iv. incorporate dynamic analysis features to produce a hybrid detection system.

References

Allamanis, M., Barr, E. T., Devanbu, P., & Sutton, C. (2018). A survey of machine learning for big code and naturalness. *ACM Computing Surveys (CSUR),* 51(4), 1-37.

Chowdhury, I., & Zulkernine, M. (2011). Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities. *Journal of Systems Architecture,* 57(*3*), 294-313.

Dam, H. K., Tran, T., & Ghose, A. (2017). Explainable software analytics. *IEEE/ACM 39th International Conference on Software Engineering (ICSE), 53-56.*

Deepthi R.R., Anupama, P., Nazimunisa & Krishna, S. R. (2022). Software vulnerability analysis using machine learning technique. *Neuroquantology,* 20(*22*) 4071-4078. DOI: 10.48047/NQ.2022.20.22.NQ10406.

Ghaffarian, S. M., & Shahriari, H. R. (2017). Software vulnerability analysis and discovery using machine-learning and data-mining techniques: A survey. *ACM Computing Surveys,* 50(*4*), 1-36.

Habeeb, A., Oye, E. & John, D. (2024). Security Vulnerability Detection Using Machine Learning. Online: https://www.researchgate.net/publication/390298042_Security_Vulnerability_Detection_Using_Machine_Learning. Retrieved 12/1/2025.

Hulayyil, S. B., Li, S., & Xu, L. (2023). Machine-learning-based vulnerability detection and classification in Internet of Things device security. *Electronics*, 12(*18*), 3927. https://doi.org/10.3390/electronics12183927.

Li, Z., Zou, D., Xu, S., Jin, H., Zhu, Y., & Chen, Z. (2018). SySeVR: A framework for using deep learning to detect software vulnerabilities. *IEEE Transactions on Dependable and Secure Computing,* 15(*4*), 82-97.

Pang, Y., Xue, X., & Wang, H. (2017). Predicting vulnerable software components through deep neural network. Proceedings of the 2017 International Conference on Deep Learning Technologies, 6–10, Chengdu, China.

Shar, L. K., & Tan, H. B. K. (2012). Predicting SQL injection and cross-site scripting vulnerabilities through mining input sanitization patterns. *Information and Software Technology,* 55(*10*), 1767-1780. doi: https://doi.org/10.1016/j.infsof.2013.04.002

Singh, P., Hasija, T. & Ramkumar, K. (2024). Machine Learning Algorithms for Phishing Detection: A Comparative Analysis of SVM, Random Forest, and CatBoost Models. Second International Conference on Intelligent Cyber Physical Systems and Internet of Things (ICoICI), Coimbatore, India, 2024, pp. 1421-1426, doi: 10.1109/ICoICI62503.2024.10696365.

Singh, R., et al. (2019). Scalability analysis of SVM-based vulnerability scanning. *Journal of Cybersecurity,* 5(*2*), 1-10.

Wang, S., Liu, T., & Tan, L. (2016). Automatically learning semantic features for defect prediction. IEEE/ACM 38th International Conference on Software Engineering (ICSE), 297-308. Online: https://www.cs.purdue.edu/homes/lintan/publications/deeplearn-icse16.pdf. Retrieved 20/11/2024.

Xu, R., Tang, Z., Ye, G., Wang, H., Ke, X., Fang, D.& Wang, Z. (2022). Detecting code vulnerabilities by learning from large-scale open source repositories. *Journal of Information Security and Applications.* 69(103293). https://doi.org/10.1016/j.jisa.2022.103293.

Yamaguchi, F., Golde, N., Arp, D., & Rieck, K. (2014). Modeling and discovering vulnerabilities with code property graphs. IEEE Symposium on Security and Privacy, 590-604. Online: https://mlsec.tu-berlin.de/docs/2014-ieeesp.pdf. Retrieved 15/12/2024.

Zenodo (2024). Code Vulnerability Detection Dataset. [Online]. Available: https://zenodo.org/records/10975439.