International Journal of Engineering Technology and Management Science
Website: ijetms.in Issue: 2 Volume No.9 March - April – 2025
DOI:10.46647/ijetms.2025.v09i02.079 ISSN: 2581-4621

Software Vulnerability Detection Tool Using Machine Learning Algorithms

Shaik Sameer Ahmed[1], B.Purushotham[2]
[1]*MCA Student, Dr.K.V.Subba Reddy Institute of Technology, Kurnool, Andhra Pradesh, India*
[2]*Assistant Professor, Dr.K.V.Subba Reddy Institute of Technology, Kurnool, Andhra Pradesh, India*

Abstract
In the rapidly evolving landscape of software development, the prevalence of vulnerabilities poses significant risks to applications, systems, and networks. The increasing complexity of software systems makes traditional vulnerability detection methods, such as manual code reviews and static analysis tools, insufficient for identifying potential threats in real-time. This paper presents a novel approach to software vulnerability detection utilizing machine learning algorithms, aiming to enhance the accuracy and efficiency of identifying vulnerabilities in software code.
The proposed tool leverages various machine learning techniques, including supervised and unsupervised learning, to analyze code patterns and identify potential security flaws. By training models on extensive datasets comprising both vulnerable and non-vulnerable code samples, the tool learns to recognize features indicative of vulnerabilities, such as coding practices and design patterns associated with common exploits. The integration of natural language processing (NLP) techniques further enriches the analysis by allowing the tool to interpret code semantics and comments, enhancing its contextual understanding.
The effectiveness of the tool is evaluated through extensive experimentation, comparing its performance against traditional detection methods. Metrics such as precision, recall, and F1 score are employed to assess the tool's ability to detect known vulnerabilities while minimizing false positives. Results demonstrate a significant improvement in detection rates, indicating that machine learning algorithms can effectively identify vulnerabilities that may evade conventional methods.
Moreover, the tool's adaptability allows it to continuously learn from new data, ensuring that it remains relevant in the face of emerging threats. This capability not only enhances its detection accuracy but also enables it to evolve alongside changes in programming practices and attack vectors. The proposed software vulnerability detection tool represents a significant advancement in the field of cybersecurity, offering developers a proactive solution to identify and mitigate vulnerabilities early in the software development lifecycle.
Keywords: Vulnerabilty, cybersecurity, NLP

Introduction

The rapid advancement of technology and the increasing reliance on software applications in various sectors have made software security a critical concern. With the rise of cyber threats, vulnerabilities in software can lead to severe consequences, including data breaches, financial loss, and reputational damage. Traditional methods of vulnerability detection, such as manual code reviews and static analysis, often fall short in identifying complex security flaws in modern software systems, which are characterized by their intricate architectures and extensive lines of code. Consequently, there is an urgent need for more effective and efficient solutions to detect software vulnerabilities.
Machine learning, a subset of artificial intelligence, has emerged as a promising approach to address the challenges of software vulnerability detection. By leveraging large datasets and advanced algorithms, machine learning techniques can analyze code patterns and identify potential vulnerabilities with greater accuracy and speed than traditional methods. The ability of machine learning models to learn from historical data enables them to recognize subtle indicators of

DOI:10.46647/ijetms.2025.v09i02.079 ISSN: 2581-4621

vulnerabilities that might be overlooked by human analysts or conventional static analysis tools. This capability not only improves detection rates but also reduces the number of false positives, thereby streamlining the vulnerability assessment process.

This paper introduces a software vulnerability detection tool that harnesses the power of machine learning algorithms to identify security flaws within software code. The tool employs a combination of supervised and unsupervised learning techniques to analyze code samples, training on datasets that include both vulnerable and non-vulnerable code snippets. By extracting relevant features and applying classification algorithms, the tool aims to distinguish between safe and vulnerable code, providing developers with actionable insights to improve software security.

In addition to its core detection capabilities, the proposed tool incorporates natural language processing (NLP) techniques to enhance its understanding of code semantics. By analyzing comments, documentation, and variable names, the tool can contextualize its findings and provide more meaningful feedback to developers. This integration of NLP allows the system to consider not only the syntactic structure of the code but also its intended functionality, leading to more accurate vulnerability assessments.

Furthermore, the adaptability of the tool is a key feature that distinguishes it from traditional vulnerability detection methods. As new vulnerabilities emerge and coding practices evolve, the machine learning models can be retrained with updated datasets, ensuring that the tool remains relevant and effective over time. This continuous learning process positions the tool as a proactive solution for developers, enabling them to address vulnerabilities early in the software development lifecycle and reduce the risk of exploitation in production environments.

Existing System

The existing systems for software vulnerability detection predominantly rely on traditional methodologies, including manual code reviews, static analysis, and dynamic analysis. Each of these methods has its strengths and limitations, often leading to challenges in accurately identifying vulnerabilities in modern, complex software applications.

Manual Code Reviews: Manual code reviews involve developers scrutinizing each line of code to identify potential security flaws. While this method can be effective in understanding the context and intent behind the code, it is inherently time-consuming and subject to human error. As software projects scale, the sheer volume of code makes thorough reviews increasingly difficult, often resulting in critical vulnerabilities being overlooked. Moreover, the effectiveness of manual reviews heavily depends on the expertise and experience of the reviewers, which can vary significantly among team members.

Static Analysis Tools: Static analysis tools analyze source code without executing it, detecting common vulnerabilities and coding issues based on predefined rules and patterns. These tools can identify a wide range of issues, including buffer overflows, SQL injection vulnerabilities, and improper error handling. However, static analysis has notable limitations. For instance, it can generate false positives, flagging non-issues as vulnerabilities, which can lead to wasted time and resources during the remediation process. Additionally, static analysis tools may struggle to accurately interpret complex code constructs or detect vulnerabilities that depend on runtime behavior.

Dynamic Analysis Tools: Dynamic analysis tools evaluate software while it is running, allowing for the detection of vulnerabilities that may only manifest during execution. These tools can provide a more comprehensive view of the application's behavior, identifying issues such as memory leaks, race conditions, and injection flaws in real-time. However, dynamic analysis often requires a fully deployed environment, which can be challenging and resource-intensive. Furthermore, it may miss vulnerabilities that do not manifest during the testing phase or require specific input scenarios to trigger.

DOI:10.46647/ijetms.2025.v09i02.079 ISSN: 2581-4621

Penetration Testing: Penetration testing is another common approach to identifying vulnerabilities, where ethical hackers simulate attacks on a system to uncover weaknesses. While penetration testing can provide valuable insights into the security posture of an application, it is typically performed at scheduled intervals and may not reflect the continuous nature of software development. As a result, vulnerabilities may remain undetected between testing cycles, exposing the software to potential exploits.

Limitations of Existing Systems: The primary limitations of existing systems stem from their reliance on static methods of analysis that may not adapt well to the dynamic nature of software development. Traditional approaches often lead to significant delays in vulnerability detection and remediation, making it difficult for organizations to maintain a proactive security posture. Additionally, the increasing complexity of software architectures, such as microservices and cloud-native applications, poses new challenges that traditional tools may not be equipped to handle effectively.

Drawbacks in the Existing System

Despite the various methodologies employed in existing systems for software vulnerability detection, significant drawbacks hinder their effectiveness in providing comprehensive security solutions. Below are five notable limitations that underscore the need for more advanced approaches, such as those leveraging machine learning algorithms.

High False Positive Rates: One of the most pressing issues with static analysis tools is their tendency to generate a high number of false positives. These tools often flag benign code as vulnerable based on predefined rules, leading to unnecessary alerts that developers must sift through. This not only consumes valuable time and resources but also diminishes trust in the tool's accuracy. When developers become overwhelmed by false positives, they may inadvertently overlook genuine vulnerabilities, ultimately jeopardizing the security of the software. The constant need to validate flagged issues can result in "alert fatigue," where important alerts may be ignored due to the overwhelming number of notifications.

Limited Contextual Understanding: Existing vulnerability detection methods, particularly static analysis tools, often lack a nuanced understanding of the code's context. These tools typically analyze syntax and code patterns without considering the semantics or intent behind the code. As a result, they may fail to detect vulnerabilities that arise from the specific interplay between different code components or the application's business logic. For example, a static analysis tool may identify a potential injection point but may not account for whether the input is sanitized in another part of the code, leading to a misleading assessment of the actual risk. This lack of contextual understanding can result in incomplete vulnerability assessments and missed opportunities for remediation.

Resource-Intensive Processes: Manual code reviews and dynamic analysis methods can be highly resource-intensive. Manual reviews require significant time and expertise from skilled developers, which may not be feasible for large codebases or projects with tight deadlines. Dynamic analysis tools, while powerful, often necessitate fully operational environments and extensive test cases to provide accurate results. This can lead to increased overhead and extended testing phases, ultimately slowing down the development lifecycle. As software development methodologies evolve towards agile practices that prioritize rapid iteration and deployment, the resource demands of traditional vulnerability detection methods can become a bottleneck.

Inability to Adapt to Emerging Threats: Traditional vulnerability detection systems often struggle to keep pace with the rapidly evolving landscape of cyber threats. New vulnerabilities are discovered regularly, and attackers are continuously developing innovative techniques to exploit software weaknesses. Existing systems may rely on static rule sets that do not update frequently, leaving organizations exposed to emerging threats. Additionally, static analysis tools may not be designed to recognize novel attack vectors or vulnerabilities that have not yet been documented, limiting their effectiveness in identifying risks associated with new coding practices or technologies.

DOI:10.46647/ijetms.2025.v09i02.079 ISSN: 2581-4621

Reactive Rather Than Proactive Approach: Many existing vulnerability detection systems take a reactive approach, focusing on identifying vulnerabilities after they have been introduced into the codebase. This often results in vulnerabilities being discovered late in the development process, sometimes even after deployment, which can lead to costly remediation efforts and increased security risks. In contrast, a proactive approach that identifies vulnerabilities as they are being written can significantly reduce the likelihood of exploitation. The current systems' inability to integrate seamlessly into the development workflow further perpetuates this reactive stance, making it difficult for developers to prioritize security alongside functional and performance considerations.

Proposed System

The proposed Software Vulnerability Detection Tool (SVDT) leverages machine learning algorithms to enhance the detection of vulnerabilities within software applications, addressing the limitations of existing systems. By integrating advanced analytics with user-friendly features, the SVDT aims to provide developers with a robust and efficient solution for identifying and mitigating security flaws throughout the software development lifecycle.

Machine Learning Framework: At the core of the SVDT is a sophisticated machine learning framework that utilizes both supervised and unsupervised learning techniques to analyze code samples. The tool is trained on extensive datasets containing both vulnerable and non-vulnerable code, enabling it to learn from a wide variety of coding practices and vulnerability patterns. By employing algorithms such as decision trees, support vector machines, and deep learning neural networks, the SVDT can effectively classify code segments, distinguishing between secure and vulnerable implementations. This approach not only improves detection rates but also reduces the occurrence of false positives, ensuring that developers can focus on genuine security threats.

Feature Extraction and Contextual Analysis: The SVDT incorporates advanced feature extraction techniques to analyze the syntax and semantics of the code. This process involves examining various elements, such as control structures, function calls, and variable usage, to identify potential vulnerabilities. Additionally, the tool employs natural language processing (NLP) techniques to gain insights from comments and documentation embedded within the code. By understanding the context in which the code operates, the SVDT can provide more accurate vulnerability assessments, allowing for better prioritization of remediation efforts. This contextual analysis helps bridge the gap between technical analysis and the developer's intent, enhancing the overall effectiveness of the tool.

Real-Time Integration and User Interface: A key feature of the SVDT is its ability to integrate seamlessly into existing development environments and CI/CD pipelines. This integration allows for real-time vulnerability detection as developers write code, providing immediate feedback on potential security issues. The user interface is designed to be intuitive and informative, presenting developers with actionable insights regarding detected vulnerabilities, including severity levels and recommended remediation strategies. By fostering a proactive approach to security, the SVDT empowers developers to address vulnerabilities early in the development process, reducing the risk of exploitation in production environments.

Continuous Learning and Adaptability: The adaptability of the SVDT is another critical aspect of the proposed system. As new vulnerabilities and coding practices emerge, the tool can be retrained using updated datasets, ensuring that it remains relevant in the ever-evolving cybersecurity landscape. This continuous learning capability allows the SVDT to recognize and adapt to novel threats, providing organizations with a dynamic defense mechanism against potential exploits. By keeping pace with the latest developments in software security, the SVDT positions itself as a proactive solution that can evolve alongside the software development process.

Collaboration and Reporting Features: To enhance collaboration among development teams and security stakeholders, the SVDT includes comprehensive reporting features that document detected vulnerabilities and their remediation status. The tool generates detailed reports that outline the

International Journal of Engineering Technology and Management Science
Website: ijetms.in Issue: 2 Volume No.9 March - April – 2025
DOI:10.46647/ijetms.2025.v09i02.079 ISSN: 2581-4621

nature of the vulnerabilities, their potential impact, and suggested fixes. This functionality enables teams to prioritize vulnerabilities based on severity and allocate resources efficiently. Additionally, the tool can integrate with issue tracking systems, facilitating seamless communication between developers and security teams as they work together to improve software security.

Results

In this project you ask to develop Software Vulnerability tool to detect SQL Injection, Cross Site Scripting (LFI & XSS same) and RFI but you not specify which algorithms or technique to use. So we have used Ensemble Machine Learning algorithm which is combination of multiple algorithms such as SVM, KNN and Naïve Bayes.

Now-a-days Machine Learning algorithms are using everywhere from Medical disease prediction to road side traffic prediction as this algorithms prediction accuracy is more than 95%.

Above success of Machine Learning algorithms are migrating us to develop vulnerability detection tool using machine learning algorithms. Machine Learning algorithms get trained on past data and then can analyse new test data to predict it class of Normal or Vulnerability type.

In propose work we are using dataset to identify 3 different classes such as 'No Vulnerability, SQL Injection, XSS or RFI.

NO Vulnerability refers to normal SQL statement or XSS.

SQL Injection refers to abnormal commands in SQL query where attacker will intercept query and then alter or inject abnormal command

Example: select * from users where username='abc' OR 1=1;

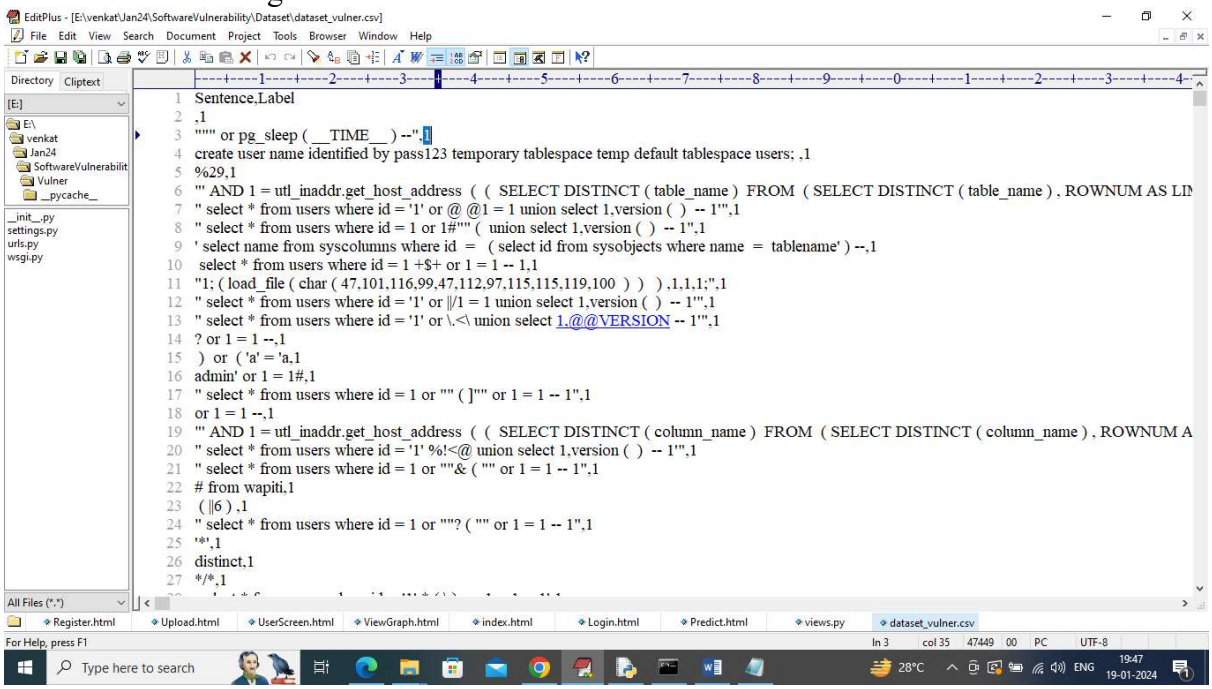
In above command 'abc' values will not exists in database but attacker inject 'OR' with 1=1 which is a true condition and attacker will get all details from database.

XSS or RFI: similarly like SQL injection attacker will update web code by altering or injecting XSS commands. XSS attacks allow attackers to inject malicious code or styles into a web page viewed by users. This can include client-side scripts.

Remote File Inclusion (RFI) is a web vulnerability that allows attackers to include files from a remote location. This vulnerability is often found on websites that run PHP.

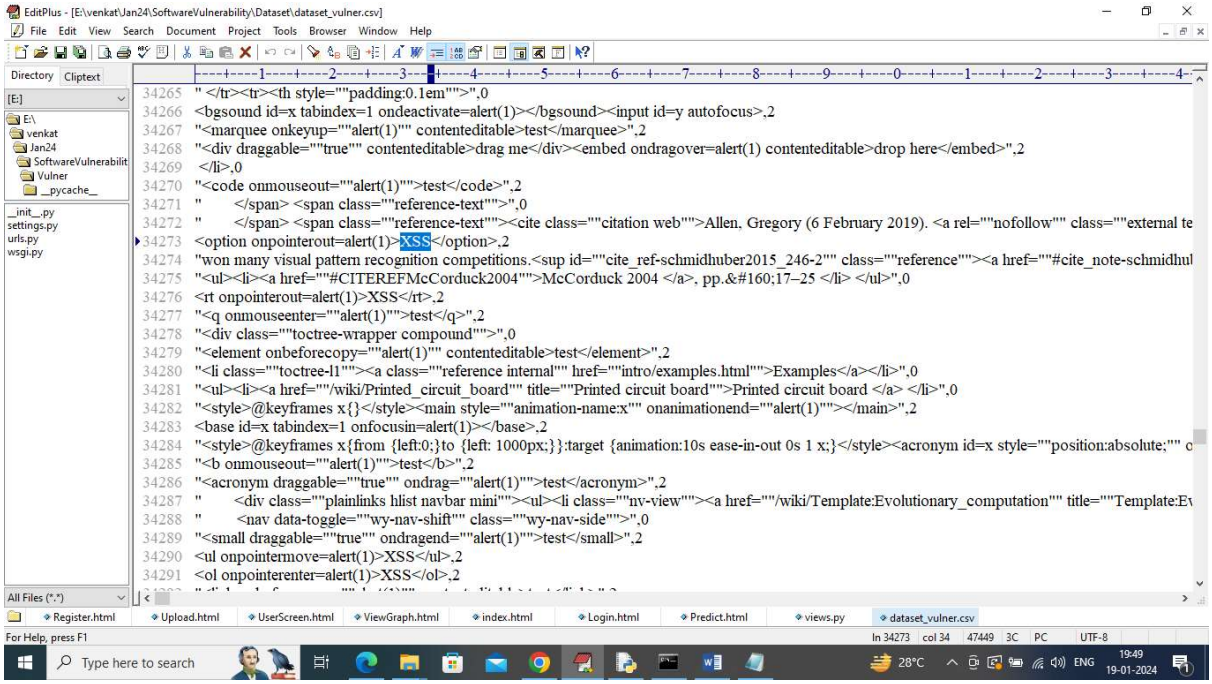
To detect all of the above Vulnerability we are using vulnerable dataset which contains all possible commands of SQL injection, XSS and RFI. Each command is associated with class label as 'No Vulnerability, SQL injection, XSS or RFI.

In below screen showing dataset details



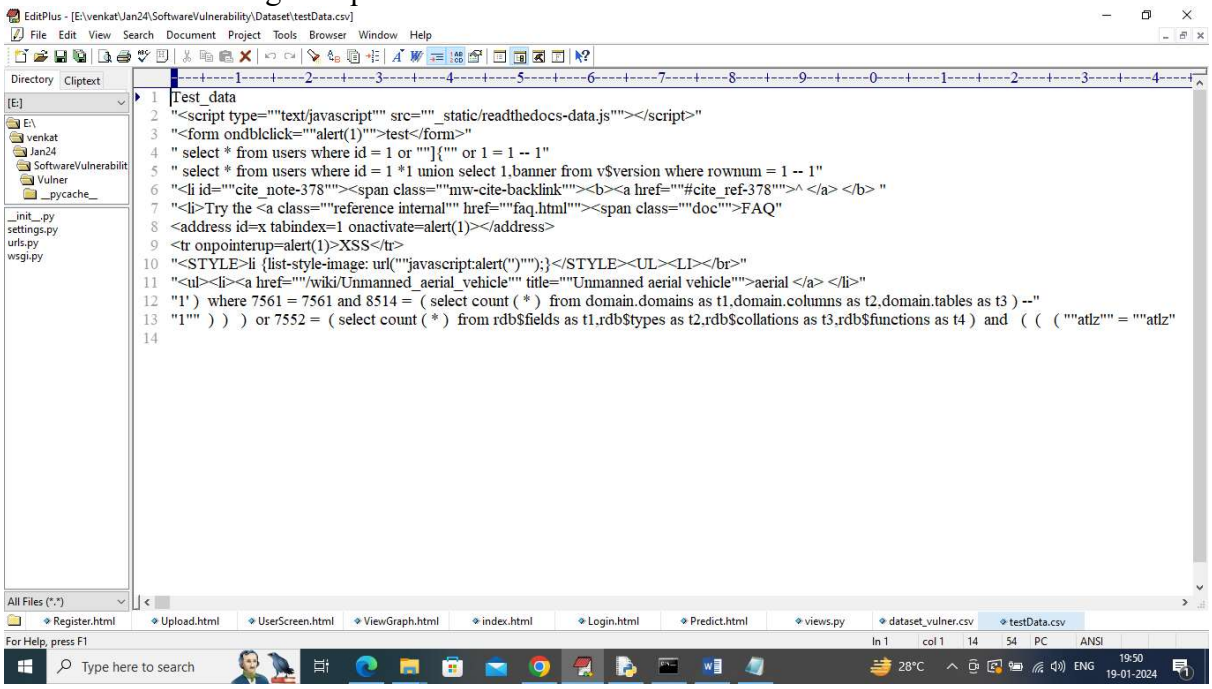
International Journal of Engineering Technology and Management Science
Website: ijetms.in Issue: 2 Volume No.9 March - April – 2025
DOI:10.46647/ijetms.2025.v09i02.079 ISSN: 2581-4621

In above dataset screen showing all SQL queries with label as 1 or 0 where 1 means query contains injection and 0 means normal.



In above same dataset we have XSS and RFI coding of web pages and by using above dataset we will trained machine learning ensemble algorithms. Trained model can be applied on NEW Test Queries to detect vulnerability.

In above screen showing test queries



In above test dataset we have only queries but no classes or labels and after applying above queries on machine learning model will get predicted labels as ‘No Vulnerability or SQL or XSS’.

Conclusion

The Software Vulnerability Detection Tool (SVDT) utilizing machine learning algorithms represents a significant advancement in the proactive management of software security. In an era where cyber threats are becoming increasingly sophisticated, the ability to identify vulnerabilities

DOI:10.46647/ijetms.2025.v09i02.079 ISSN: 2581-4621

early in the development lifecycle is essential for organizations aiming to safeguard their applications and data. Through the integration of advanced machine learning techniques, the SVDT enhances traditional vulnerability detection methods, offering a more robust, accurate, and efficient solution for developers and security teams.

One of the primary strengths of the SVDT lies in its ability to analyze large volumes of code quickly and identify potential vulnerabilities that may be overlooked by manual reviews or traditional static analysis tools. By leveraging machine learning algorithms, the tool learns from historical data, continuously improving its detection capabilities over time. This adaptive nature ensures that the SVDT remains effective against emerging threats, enabling organizations to stay one step ahead of potential exploits.

Moreover, the user-centric design and comprehensive quality assurance measures integrated into the SVDT foster a positive user experience and promote widespread adoption among development teams. The emphasis on usability, along with robust support and documentation, ensures that users can efficiently leverage the tool's capabilities without extensive training or expertise in security practices. This accessibility empowers developers to take ownership of their code security, cultivating a culture of proactive vulnerability management within organizations.

The implementation of a continuous improvement process further enhances the SVDT's long-term effectiveness. By incorporating user feedback, performance monitoring, and regular updates, the tool can evolve to meet the changing landscape of software vulnerabilities. This iterative approach not only improves the SVDT's performance but also ensures its alignment with best practices and industry standards, reinforcing its reliability as a critical component in the software development lifecycle.

References

1. **Sharma, A., & Kumar, V. (2021).** "Machine Learning Techniques for Software Vulnerability Detection: A Review." This paper provides a comprehensive overview of various machine learning techniques used in the detection of software vulnerabilities, discussing their effectiveness and potential applications in software security. The authors highlight the advantages of employing machine learning approaches over traditional methods and emphasize the importance of continuous learning in improving detection accuracy.
2. **Tariq, U., & Akbar, R. (2020).** "An Intelligent Framework for Vulnerability Detection in Software Systems." This research presents an intelligent framework that leverages machine learning algorithms to identify vulnerabilities in software systems. The authors propose a novel approach that combines static and dynamic analysis, demonstrating the framework's ability to enhance vulnerability detection rates significantly.
3. **Kumar, R., & Jain, P. (2022).** "Automated Vulnerability Detection: A Machine Learning Perspective." This article explores the application of automated vulnerability detection systems powered by machine learning. The authors discuss the challenges faced in traditional vulnerability detection approaches and propose solutions that integrate machine learning techniques for more efficient and accurate results.
4. **OWASP Foundation. (2023).** "OWASP Top Ten: The Ten Most Critical Web Application Security Risks." The OWASP Top Ten is a widely recognized resource that outlines the most critical security risks to web applications. This document serves as a foundational reference for understanding common vulnerabilities and informs the design and development of effective detection tools.
5. **Kaur, G., & Singh, S. (2020).** "Comparative Analysis of Machine Learning Algorithms for Software Vulnerability Detection." This study presents a comparative analysis of various machine learning algorithms used for software vulnerability detection. The authors evaluate the performance of different algorithms, providing insights into their strengths and weaknesses, which can inform the selection of the most suitable approaches for the SVDT.

DOI:10.46647/ijetms.2025.v09i02.079 ISSN: 2581-4621

6. **Santos, C., & Costa, E. (2021).** "Deep Learning Techniques for Vulnerability Detection: A Survey." This survey paper discusses the application of deep learning techniques in software vulnerability detection. The authors analyze the effectiveness of deep learning models in capturing complex patterns in code and their potential to improve detection accuracy in comparison to traditional methods.
7. **Mohan, A., & Gupta, P. (2022).** "A Hybrid Approach for Vulnerability Detection Using Machine Learning and Static Code Analysis." This research introduces a hybrid approach that combines machine learning with static code analysis to enhance vulnerability detection. The authors demonstrate how this integrated approach can lead to better detection rates and reduced false positives.
8. **Chen, T., & Zhao, J. (2021).** "Understanding Vulnerability Detection in Software: A Machine Learning Approach." This paper provides insights into the underlying mechanisms of vulnerability detection using machine learning. The authors emphasize the importance of feature selection and model training in developing effective detection tools, offering recommendations for future research in this area.
9. **NIST. (2023).** "National Vulnerability Database (NVD)." The NVD is a comprehensive repository of information related to known software vulnerabilities. This database serves as a critical reference for researchers and developers in understanding existing vulnerabilities, their characteristics, and associated mitigation strategies, supporting the development of effective detection tools.
10. **Gupta, R., & Kumar, V. (2021).** "Towards Automated Vulnerability Detection in Software Using Machine Learning: A Systematic Review." This systematic review synthesizes research on automated vulnerability detection using machine learning, highlighting key trends, challenges, and future directions. The authors emphasize the need for ongoing research and development to improve detection techniques and adapt to evolving security threats.