The Indonesian Journal of Computer Science
www.ijcs.net
Volume 14, Issue 4, August 2025
https://doi.org/10.33022/ijcs.v14i4.4972

A Review of Vulnerability Detection Algorithms in Software Code

Zelda P. Ramahlo[1], Topside E. Mathonsi[2], Tshimangadzo M. Tshilongamulenzhe[3]
puleyramahlo@gmail.com[1], mathonsite@tut.ac.za[2]; tshilongamulenzhetm@tut.ac.za[3]
[1,2,3] Department of Information Technology, Tshwane University of Technology, South Africa

Article Information

Received : 11 Aug 2025
Revised : 25 Aug 2025
Accepted : 30 Aug 2025

Keywords

Software Vulnerability Detection,
Machine Learning (ML),
Random Forest (RF),
Convolutional Neural Networks (CNN)

Abstract

Detecting software vulnerabilities is essential to keeping modern systems safe in the face of increasingly sophisticated cyber threats. This paper offers a clear and accessible overview of how vulnerabilities are currently identified, reviewing traditional techniques, machine learning (ML), and hybrid approaches. Traditional techniques such as static and dynamic analysis are still widely used but often suffer from high false positive rates and struggle to adapt to new and evolving threats. In contrast, recent ML developments, especially those involving Random Forest (RF) and Convolutional Neural Networks (CNN), have shown significant promise in improving detection accuracy, feature extraction, and classification. Decision Tree (DT) methods remain valued for their transparency, while CNNs and other deep learning tools excel at recognizing structural and spatial patterns in code. Combining these strengths in hybrid models integrating effective feature selection with powerful pattern recognition has the potential to deliver more accurate results and reduce false alarms. However, persistent challenges remain, including limited dataset diversity, weak resilience against adversarial attacks, and the need for real-time adaptability. By bringing together the latest research and practical insights, this review paper aims to guide developers, security analysts, and organizations in creating more robust, automated, and adaptive security tools capable of meeting the fast-changing demands of software vulnerability management.

A. Introduction

Software vulnerability detection plays a crucial role in securing modern digital infrastructures, where a single undetected flaw can lead to severe breaches, financial losses, and reputational damage [1], [2]. The growing complexity of software systems, combined with the rapid evolution of cyber threats, makes early and accurate detection of vulnerabilities a top priority in cybersecurity. High-profile incidents, such as the exploitation of zero-day vulnerabilities in widely used software, demonstrate how undetected weaknesses can have global repercussions [3].

Traditional vulnerability detection methods, including static and dynamic analysis, have been widely used to identify flaws in software code. Static analysis examines code without execution, while dynamic analysis monitors program behaviour during runtime. However, both approaches face limitations, including high false positive rates, limited adaptability to emerging threats, and scalability issues when applied to large or complex codebases [4].

In recent years, ML has emerged as a promising solution to address these challenges, leveraging large datasets to identify complex patterns in source code and detect vulnerabilities more effectively. RF has proven effective for feature selection and classification tasks [5], while excelling in capturing structural and spatial code patterns [6]. Despite these strengths, most studies focus on single-technique ML models, which struggle to balance high accuracy with low false positives across diverse vulnerability types.

While existing reviews have explored ML-based vulnerability detection [7]–[9], many have focused on individual algorithms or general approaches without emphasizing hybrid models that integrate complementary strengths. This paper builds on prior work, such as the survey by [10], extending the scope to address gaps including adaptability to multiple vulnerability types, real-time detection capabilities, and criticality-based classification. To the best of the researcher's knowledge, this is the first comprehensive review linking the integration of RF and CNN to both theoretical advancements and practical deployment in diverse software environments.

The paper adopted a systematic literature review approach, collecting studies from IEEE Xplore, SpringerLink, ACM Digital Library, ScienceDirect, and MDPI. Keywords such as "Software Vulnerability Detection", "RF and "CNN" were used to filter relevant research. In total, 28 articles were reviewed to meet the following objectives:

1. To identify and analyse existing state-of-the-art software vulnerability detection techniques, including traditional and ML-based methods.
2. To investigate the benefits and limitations of using RF and CNN individually in vulnerability detection.

As a result, the main contributions of this paper are:

1. A structured comparison of traditional, ML-based, and hybrid techniques for software vulnerability detection.
2. Insights into practical considerations for deploying hybrid ML models in real-world software security environments.

The remainder of this paper is organized as follows: Section B provides an overview of software vulnerability detection. Section C discusses the concept of software vulnerabilities. Section D describes both traditional and machine learning-based detection techniques. Section E focuses on various ML approaches. Section F reviews related work in the field. Section G highlights open research issues, and Section H concludes the paper with future research directions.

B. Overview of Software Vulnerability Detection

Software vulnerability detection is the process of identifying security weaknesses in software applications that could potentially be exploited by attackers [11]. It plays a crucial role in cybersecurity, as undetected vulnerabilities can be exploited for malicious purposes.

Traditional methods of vulnerability detection mainly rely on Static Application Security Testing (SAST) and Dynamic Application Security Testing (DAST), with conventional tools such as Checkmarx, Fortify Static Code Analyzer, OWASP ZAP, and Burp Suite. However, these tools often suffer from high false positive rates, limited adaptability to new threats, and resource-intensive processing. These limitations have motivated the exploration of machine learning–based approaches, such as RF and CNNs, to improve detection efficiency and accuracy [12].

As cyber threats continue to evolve, detecting vulnerabilities in software applications has become more complex. Therefore, the need for more advanced systems capable of detecting vulnerabilities while minimizing false positives and negatives has never been greater. This paper aims to explore how integrating RF and CNNs models can improve vulnerability detection by addressing these limitations [13].

C. Software Vulnerability

Software vulnerabilities, often referred to as security defects or weaknesses, represent flaws in software that can be exploited to compromise security. According to [14], a software vulnerability is defined as "software faults with security consequences," highlighting the critical impact these errors can have on system integrity. More specifically, vulnerabilities arise from errors in software definition, creation, or configuration that violate established security policies. For instance, a vulnerability may occur due to mistakes in software specification, development, or configuration, resulting in a breach of an asserted or implicit security policy. Understanding the judgment process behind identifying vulnerable code is crucial, as it involves various factors beyond just the code itself. Figure 1 illustrates the key elements influencing the assessment of whether code is vulnerable. These factors include the developer's security knowledge, experience, and understanding of the program, as well as intuitive and perceptual judgments that collectively inform vulnerability detection.

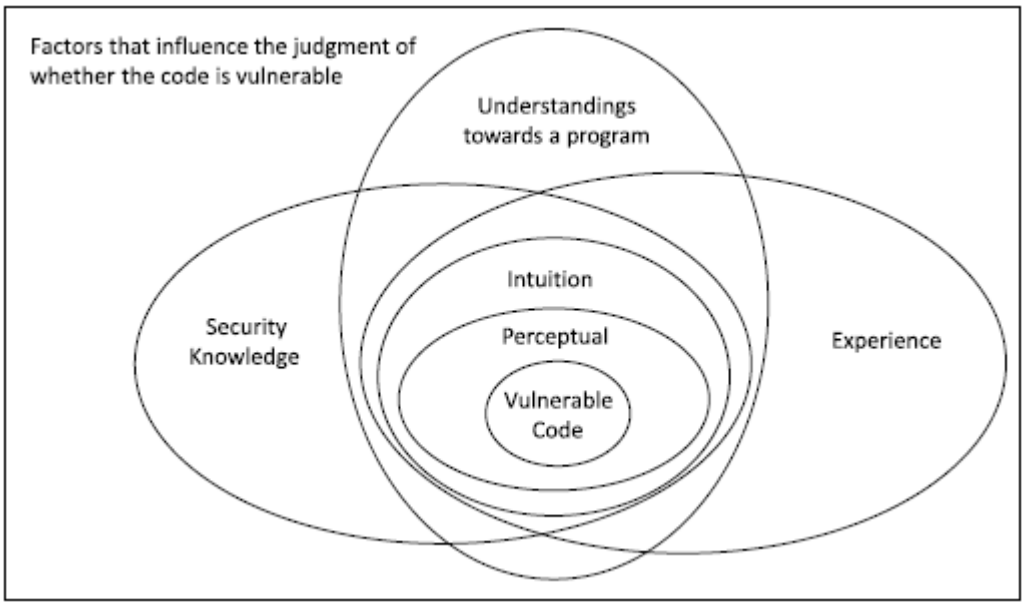

Figure 1. Factors influencing vulnerability detection in code inspection [4]

As illustrated in Figure 1, the practitioner's ability to identify potential flaws in a code snippet depends on multiple dimensions, including both semantic and syntactic understanding of the code. Semantic understanding involves comprehending the meaning and purpose of code statements, such as recognizing the logic behind variable assignments or function calls. In contrast, syntactic understanding refers to the correct form and structure of the code, ensuring that it follows language rules and conventions. According to [14], "the practitioner can examine the code to determine whether there is a missing bound check for a variable (i.e., the cause of buffer errors)."

Beyond this technical understanding, a practitioner's security knowledge and experience play a critical role in vulnerability detection. This expertise allows them to think like a software tester or attacker, guiding their intuition to focus on suspicious code segments more closely. For example, a practitioner might assess whether a source parameter (such as a parameter value) could be influenced or controlled by an external variable, potentially leading to security risks. Additionally, perceptual and cognitive factors such as pattern recognition and prior exposure to similar vulnerabilities further influence judgment. These dimensions combine to form a holistic evaluation process, as depicted in Figure 1, where semantic and syntactic analysis, expert knowledge, and intuitive judgment converge to inform vulnerability detection decisions.

D. Traditional and Machine Learning-Based Detection Techniques

Traditional software vulnerability detection methods can be broadly categorized into the following [15]:

- **SAST**: This technique examines source code without execution to identify potential vulnerabilities [16]. Tools such as Fortify, Coverity, and SonarQube are commonly used. However, SAST techniques struggle with detecting runtime vulnerabilities and often produce many false positives

- **DAST**: DAST involves running the software in a controlled environment to observe its behaviour during execution [17]. Tools like OWASP ZAP and Burp Suite are used for runtime vulnerability detection While dynamic analysis provides insights into runtime vulnerabilities, it may miss issues in unexecuted code paths.
- **Signature-Based Detection**: This technique relies on known vulnerability signatures to identify flaws within the software. Signature-based methods are effective for known vulnerabilities but fail to detect zero-day vulnerabilities.[18]
- **Behaviour-Based Detection**: Behaviour-based approaches focus on analysing the behaviour of software to detect anomalies or deviations from expected patterns [19]. These methods are useful for detecting previously unknown vulnerabilities but are computationally expensive.

E. Machine Learning-Based Approaches

ML has significantly transformed the landscape of software vulnerability detection by enabling systems to automatically learn from data and improve over time. These approaches can detect both known and previously unknown vulnerabilities by identifying anomalies or deviations from expected patterns. ML techniques are generally classified into supervised, unsupervised, and reinforcement learning categories [20]. Each category offers distinct advantages and is suited to specific aspects of vulnerability detection.

i. Supervised Learning

Supervised learning algorithms rely on labeled datasets to train models that can classify or predict outcomes based on input features [20]. These methods are highly effective when historical data with known vulnerabilities is available. Prominent supervised models include:

- Support Vector Machine (SVM): SVM works by finding a hyperplane that best separates different classes of data. In vulnerability detection, SVM is used to classify software components as vulnerable or non-vulnerable based on various features such as code snippets, function calls, and other behavioral attributes [21]. SVM is particularly effective in high-dimensional spaces.
- RF: RF is a supervised learning algorithm that builds an ensemble of DT to make predictions [21]. It is particularly effective in handling large datasets and ranking important features, making it useful for vulnerability classification.
- CNN: CNNs are DL models that excel in recognizing patterns in spatial data, such as images or sequences. When applied to vulnerability detection, CNNs can identify structural and spatial features in source code representations [22].
- DT: DT partition data based on feature values, making them easy to interpret [23]. In vulnerability detection, they are effective at identifying which features (e.g., function calls or code patterns) are most indicative of vulnerabilities. Enhanced variants such as GBDT and XGBoost improve performance by combining weak learners into a strong classifier.

- K-Nearest Neighbors (KNN): KNN classifies a sample by looking at the majority class among its nearest neighbors [24]. It's useful in vulnerability detection when classifying new software or systems by comparing their similarity to existing known vulnerabilities. KNN is simple and effective when the data is not too large.
- Artificial Neural Network (ANN): ANNs, including MLP, are capable of modeling complex, non-linear relationships. These networks learn from large datasets of software vulnerabilities, making them ideal for predicting vulnerabilities in complex code or systems. They have been applied in many cybersecurity contexts, where traditional algorithms fall short [25].
- Logistic Regression: Logistic regression is a straightforward classifier used in binary classification tasks, such as determining whether a given software vulnerability exists [26]. It models the probability of a sample belonging to a certain class based on input features.
- Naive Bayes: Naive Bayes is a probabilistic classifier based on Bayes' theorem. It assumes that features are independent, making it computationally efficient and suitable for tasks where features are many but labeled data is scarce. In vulnerability detection, it's used for screening software based on its feature set, especially when the dataset is large and sparse [27].
- Deep Belief Network (DBN): DBNs are a class of deep learning models composed of multiple layers of restricted Boltzmann machines. They are useful for detecting complex patterns in high-dimensional vulnerability data and large-scale databases [28]. DBNs are effective in modeling intricate relationships among features that may indicate.

ii. Unsupervised Learning

Unsupervised learning is used when labeled data is unavailable. These algorithms uncover hidden structures and patterns, making them particularly suitable for detecting zero-day vulnerabilities and other unknown threats [29]

- K-Means Clustering: K-Means is an unsupervised clustering algorithm that groups data points into clusters based on similarity [29]. In vulnerability detection, it helps identify groups of similar vulnerabilities, and the outliers or anomalies within these clusters can indicate potential new vulnerabilities.
- Principal Component Analysis (PCA): PCA is used for dimensionality reduction, extracting the most important features from large datasets. By reducing the dimensionality, PCA can highlight anomalous behaviors or patterns in software or system data that could be indicative of vulnerabilities [30].
- Autoencoders: Autoencoders are neural networks used for anomaly detection by learning to reconstruct input data [31]. High reconstruction errors can indicate anomalies or outliers, which in vulnerability detection could signify new or unknown vulnerabilities.
- Hierarchical Clustering: This algorithm builds a hierarchy of clusters by merging or splitting them based on distance metrics [32]. In vulnerability detection, hierarchical clustering helps uncover relationships between

software components or vulnerabilities, which can be crucial for detecting novel attack vectors.

- Isolation Forests: Isolation Forest is an anomaly detection method that isolates anomalies instead of profiling normal data. This is particularly useful for identifying rare and suspicious patterns that could indicate vulnerabilities not seen in the training data [33].
- Density-Based Spatial Clustering of Applications with Noise (DBSCAN): DBSCAN is another clustering algorithm that identifies clusters of varying shapes and sizes based on density [34]. It is effective for detecting vulnerabilities in imbalanced or noisy data, where traditional clustering methods might fail.

iii. Reinforcement Learning

RL involves an agent that learns by interacting with an environment and receiving feedback in the form of rewards or penalties [35]. In vulnerability detection, RL can be used to optimize scanning processes and adapt to dynamic threats. Different RL techniques are discussed below:

- Deep Q-Network (DQN): These RL methods help develop adaptive vulnerability scanning strategies by learning the most effective actions over time. By interacting with a security environment and receiving rewards, Q-Learning models optimize the scanning process to detect vulnerabilities more effectively [35].
- Policy Gradient Methods: Policy Gradient methods enable the learning of policies that directly map states to actions. These methods can dynamically adjust security configurations or scanning frequencies based on real-time vulnerability detection performance, ensuring optimal security decisions [36].
- Proximal Policy Optimization (PPO): PPO is an RL algorithm that strikes a balance between exploration and exploitation [37]. It's used in adaptive systems for vulnerability detection, where new vulnerabilities may require frequent adjustments to security measures or scanning protocols.

F. Related Works

Software vulnerability detection has been a critical area of research due to the increasing number of security threats in modern software applications. Various approaches have been explored to enhance the accuracy and efficiency of vulnerability detection mechanisms using ML and DL techniques.

A vulnerability detection algorithm, such as DT, RF, linear SVM, radial SVM, and extreme boosting, was compared by [38] to assess the efficacy of software metrics in discerning vulnerable code units. They aimed to determine the applicability of software metrics in various scenarios and the contribution of ML algorithms to this process. While their study found that ML algorithms effectively detect vulnerable code in security-critical systems, it revealed limitations, including false positives in non-critical systems, rendering the approach less meaningful. Furthermore, their method demonstrated a lack of accuracy in identifying vulnerabilities. A notable gap in their study is the

insufficient consideration of security concerns, especially in non-critical systems. In this paper, this gap is addressed by integrating RF with CNN algorithms to enhance accuracy and obtain meaningful results across criticality levels.

A vulnerability detection algorithm for vulnerability detection in the context of Software Composition Analysis was applied by [39] aiming to predict the relevance of each data item within a software component. While their study emphasizes improvements in precision and recall, it lacks specific insights into false positives and false negatives. In parallel, this paper shares the goal of distinguishing vulnerable code units with ML but introduces unique features such as adversarial robustness and dynamic anomaly detection. Explicitly addressing adversarial attacks and continuous threat monitoring during runtime, these aspects enhance the resilience of our M-Systems, providing additional layers of security is not discussed.

A vulnerability detection algorithm based on the J48 DT was applied by [40] to identify vulnerabilities in software code and assess the effectiveness of ML-based detection methods. While their study successfully adapted J48 for vulnerability detection, it overlooked specific dataset characteristics that could impact generalizability. Like our methodology, their approach used supervised learning by training the model on labeled data. However, their study lacked a mechanism for classifying vulnerabilities based on criticality, limiting its usefulness for prioritized remediation. In contrast, this paper addresses this limitation by integrating the RF algorithm to categorize vulnerabilities by severity. This enhancement enables a more strategic and targeted remediation approach, leveraging RF's strength in capturing complex patterns associated with vulnerability criticality.

A novel vulnerability detection approach called code gadgets was proposed by [41] to reduce the manual effort required by human experts in feature engineering and to minimize false negatives in existing vulnerability detection systems. While their method demonstrated potential for improving detection accuracy, its applicability across diverse real-world software environments remains limited due to variations in software structures and development practices. Like their approach, this study employs advanced techniques to automate and enhance the vulnerability detection process. However, this paper addresses a broader range of challenges by implementing a multi-layered strategy that integrates traditional security practices with cutting-edge ML methodologies, enabling more comprehensive detection and mitigation of software vulnerabilities.

A novel vulnerability detection algorithm named Software Vulnerability Classification using Code Gadgets (SVC-CG), which combines CNN and Gated Recurrent Unit (GRU) neural networks, was proposed by [42]. Their approach aimed to improve classification performance by leveraging the complementary strengths of CNNs for feature extraction and GRUs for sequence modeling. The SVC-CG algorithm achieved strong results in macro recall, macro accuracy, and macro F1-score based on experiments using data from the National Vulnerability Database (NVD). While this study shares the objective of enhancing classification through hybrid models, its reliance on data from the

NVD may limit the model's applicability to more diverse or real-world datasets. In contrast, this paper extends the hybrid model concept by integrating the RF algorithm with CNN and incorporating multiple datasets to improve generalizability and robustness across varying software environments.

A comprehensive vulnerability detection algorithm integrating deep learning, complex network analysis, and subgraph partitioning was proposed by [43]. Their method achieved notable improvements in accuracy and reduced false positive rates. However, a key limitation of their approach was the absence of explicit analysis on detection sensitivity and false negatives, which are critical for assessing real-world applicability. In contrast, this paper not only aims to improve overall detection performance but also addresses this gap by examining the trade-offs between detection sensitivity and false negatives. This approach results in a more balanced and robust detection framework that is better suited for practical deployment.

A hybrid vulnerability detection algorithm was proposed by [44] to address complex problems where individual ML approaches may be insufficient. By combining advanced methodologies from multiple domains, their hybrid model improves overall detection performance and specifically mitigates the susceptibility of ML systems to adversarial attacks. This targeted strategy is intended to fortify systems against manipulative threats. In a similar manner, this paper adopts a multi-layered approach that integrates traditional security practices with advanced ML techniques to further strengthen ML based vulnerability detection systems against evolving threats.

Table 1 summarizes key advantages and disadvantages of various vulnerability detection algorithms reported in recent literature, highlighting their strengths and limitations in practical applications.

Table 1. Advantages and Disadvantages of Vulnerability Detection Algorithms

Article	Advantages	Disadvantages
[38]	1. Effective detection of vulnerable code in security-critical systems. 2. Integration of multiple ML algorithms (e.g., DT, RF, SVM) to assess the efficacy of software metrics.	1. High false positives in non-critical systems. 2. Lack of accuracy in identifying vulnerabilities. 3. Insufficient consideration of security concerns in non-critical systems. 4. No focus on adversarial robustness or dynamic anomaly detection.
[39]	1. Improved precision and recall for vulnerability detection. 2. Application of DT algorithm for software composition analysis. 3. Focus on relevance prediction of data items.	1. No insights into false positives or false negatives. 2. Lack of emphasis on adversarial attacks and continuous runtime threat monitoring. 3. Limited attention to security concerns and resilience of ML systems.
[40]	1. Effective use of J48 for	1. No consideration of dataset

	software vulnerability detection. 2. Successful adaptation of ML algorithms for vulnerability detection. 3. Application of supervised learning with labelled data.	characteristics. 2. Lack of vulnerability classification based on criticality. 3. Limited prioritization for remediation of vulnerabilities.
[41]	1. Automates feature creation and reduces false negatives. 2. Enhanced vulnerability detection system. 3. Helps relieve human experts from manual tasks.	1. Limited applicability to real-world software vulnerability fixes. 2. No comprehensive approach to address security concerns in software code. 3. Lacks integration of traditional security practices.
[42]	1. Fusion of CNN and GRU neural networks for enhanced vulnerability classification. 2. Strong performance metrics: high macro recall rate, accuracy rate, and F1-score. 3. Demonstrates improved vulnerability classification.	1. Reliance on NVD data may limit adaptability. 2. Limited flexibility for diverse real-world contexts. 3. No focus on adversarial robustness or real-time adaptation.
[43]	1. Integration of deep learning, complex network analysis, and subgraph partitioning. 2. Demonstrates improvements in accuracy and reduction in false positives. 3. Strong enhancement in detection capability.	1. No explicit consideration of detection sensitivity and false negatives. 2. Lacks trade-off analysis between sensitivity and effectiveness. 3. Absence of continuous threat monitoring or adversarial resilience.
[44]	1. Hybrid approach that integrates multiple advanced methodologies. 2. Targets adversarial attacks, enhancing ML system resilience. 3. Improves vulnerability detection in complex systems.	1. Complex and may require significant computational resources. 2. May not be suitable for smaller, simpler systems. 3. Limited real-time adaptability to unforeseen vulnerabilities.

Figure 2 below visually represents the comparative performance metrics of the various vulnerability detection algorithms discussed above, highlighting key differences in accuracy, false positive rates, and computational efficiency.

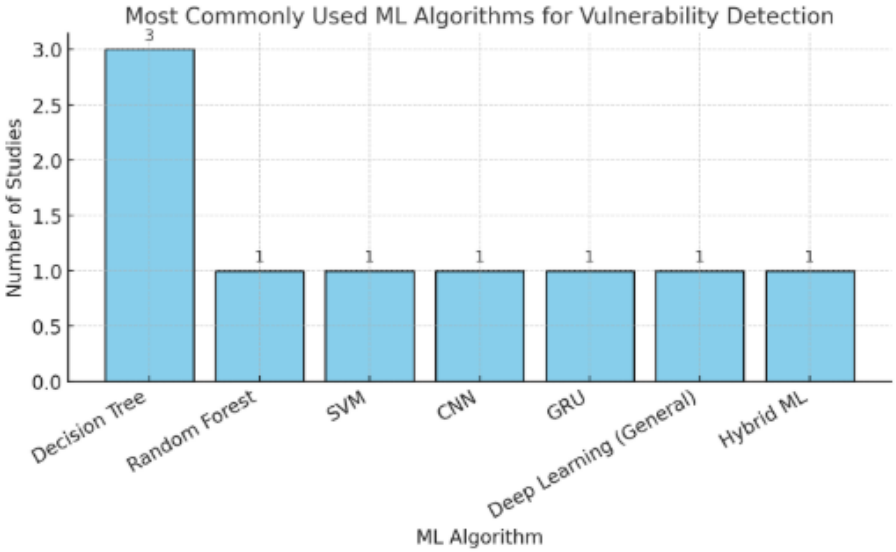

Figure 2. Analysis of the Most Common ML Algorithms Used for Vulnerability Detection in Recent Studies

Figure 2 illustrates the frequency of use of various ML algorithms for vulnerability detection based on a review of recent studies. The x-axis lists the ML algorithms, while the y-axis shows the number of studies that applied each algorithm.

From the data:

- DT emerges as the most popular algorithm, used in 3 different studies. This indicates that DT is favoured for its simplicity, interpretability, and ability to handle both classification and regression tasks effectively in vulnerability detection scenarios.
- Several other algorithms each appear in exactly one study, showing diverse approaches researchers have taken:
    1. RF: An ensemble method based on DT, which improves accuracy and reduces overfitting, reflecting its usefulness in complex vulnerability detection tasks.
    2. SVM: Known for strong performance in high-dimensional spaces and with limited samples.
    3. CNN: A deep learning approach typically used in image processing but increasingly applied in software vulnerability detection for feature extraction and classification.
    4. GRU: A type of recurrent neural network useful for sequential data, which can model temporal relationships in software data.
    5. DT (General): Refers to other deep learning techniques not specifically identified but still employed for vulnerability detection.
    6. Hybrid ML: Combines multiple ML approaches to leverage their respective strengths for improved vulnerability identification.

The distribution of algorithms suggests that while DT is currently dominant, the field explores a broad range of ML techniques, from traditional classifiers

like SVM to advanced deep learning architectures. This diversity reflects ongoing efforts to find optimal models for accurate and efficient vulnerability detection.

G. Open Research Issues

Despite the advancements in ML-based vulnerability detection, existing methods still exhibit limitations in terms of false positives, adaptability to new vulnerabilities, and computational efficiency. Many approaches rely on either static or dynamic analysis techniques, but not both, leading to suboptimal detection performance. Additionally, models such as CNNs struggle with capturing logical dependencies, while graph-based approaches demand extensive preprocessing.

This paper addresses these challenges by integrating RF with CNNs to enhance software vulnerability detection. RF is used for feature selection and classification, while CNNs capture structural and spatial features in code representations. This hybrid model aims to improve detection accuracy, reduce false positives, and ensure adaptability across various vulnerability classes.

H. Conclusion and Future Work

This review paper examined traditional, ML-based, and hybrid approaches to software vulnerability detection, highlighting the strengths and limitations of each. DT-based methods remain popular for their interpretability, while CNNs and other deep learning techniques are gaining traction for their ability to capture structural and spatial code patterns. Hybrid approaches, such as combining feature selection models with deep learning architectures, show promise in improving detection accuracy and reducing false positives. However, persistent challenges remain, including limited dataset diversity, insufficient resilience against adversarial attacks, lack of real-time adaptability, and minimal integration of traditional security practices into ML-based frameworks.

Future research should focus on creating more diverse and representative datasets to improve model generalizability, developing ML models resilient to adversarial attacks, enhancing real-time adaptability of detection systems, and integrating traditional security practices with ML-based approaches to build more robust frameworks. Additionally, exploring hybrid architectures that combine deep learning with interpretable models may further reduce false positives while maintaining high detection accuracy.

I. Acknowledgment

The authors gratefully acknowledge the support provided by the Tshwane University of Technology in facilitating this research. Additionally, they confirm that there are no conflicts of interest related to the publication of this paper.

J. References

[1] J. Lee and H. Kim, “Attention-based convolutional neural networks for software vulnerability identification,” Neurocomputing, vol. 421, pp. 354–364, 2021.

[2] X. Zhou, Y. Liu, and J. Wu, "Deep learning-based vulnerability detection: A survey," IEEE Trans. Reliab., vol. 69, no. 3, pp. 1180–1200, 2020.

[3] D. Ivan, Security Engineering for Software Development, Springer, 2020.
[4] Y. Huang, L. Liu, and T. Zhao, "An overview of vulnerability detection techniques in software," J. Cybersecurity Res., vol. 15, no. 2, pp. 99–111, 2020.
[5] H. Sun, F. Liu, and Q. Yang, “Limitations of SAST tools: An empirical study,” Proc. Int. Conf. Cyber Secur., pp. 21–29, 2019.
[6] Y. Zhang, S. Shauy, and F. Resty, "Dynamic analysis of web applications using DAST tools," Int. J. Inf. Secur., vol. 22, no. 1, pp. 34–45, 2021.
[7] Q. Gao, X. Chen, and Y. Xu, “Signature-based methods for software flaw detection,” J. Syst. Softw., vol. 168, pp. 110657, 2020.
[8] Y. Lin, M. Wang, and L. Zhou, “Behavior-based software vulnerability detection,” Comput. Secur., vol. 77, pp. 556–572, 2018.
[9] C. Cortes and V. Vapnik, "Support-vector networks," Mach. Learn., vol. 20, no. 3, pp. 273–297, 2021.
[10] J. Friedman, “Greedy function approximation: A gradient boosting machine,” Ann. Stat., vol. 29, no. 5, pp. 1189–1232, 2021.
[11] T. Cover and P. Hart, "Nearest neighbor pattern classification," IEEE Trans. Inf. Theory, vol. 13, no. 1, pp. 21–27, 2022.
[12] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," Nature, vol. 521, no. 7553, pp. 436–444, 2020.
[13] D. Hosmer and S. Lemeshow, Applied Logistic Regression, Wiley, 2022.
[14] G. John and P. Langley, “Estimating continuous distributions in Bayesian classifiers,” Proc. Conf. Uncertainty Artif. Intell., pp. 338–345, 2022.
[15] G. Hinton, S. Osindero, and Y. Teh, "A fast learning algorithm for deep belief nets," Neural Comput., vol. 18, no. 7, pp. 1527–1554, 2023.
[16] J. MacQueen, "Some methods for classification and analysis of multivariate observations," Proc. 5th Berkeley Symp. Math. Stat. Probab., pp. 281–297, 2020.
[17] I. Jolliffe, Principal Component Analysis, Springer, 2022.
[18] G. Hinton and R. Salahudin, “Autoencoders in security applications,” Neural Process. Lett., vol. 51, no. 3, pp. 1565–1580, 2021.
[19] F. Murtagh, “Hierarchical clustering for large datasets,” Pattern Recognit., vol. 125, pp. 108506, 2023.
[20] F. Liu, K. Ting, and Z. Zhou, "Isolation forest," Proc. IEEE Int. Conf. Data Mining, pp. 413–422, 2020.
[21] M. Ester, “Density-based clustering for discovering clusters in large spatial databases,” Proc. KDD, pp. 226–231, 2021.
[22] C. Watkins and P. Dayan, "Q-learning," Mach. Learn., vol. 8, pp. 279–292, 2020.
[23] R. Williams, “Simple statistical gradient-following algorithms for connectionist reinforcement learning,” Mach. Learn., vol. 8, no. 3–4, pp. 229–256, 2020.
[24] J. Schulman et al., “Proximal policy optimization algorithms,” arXiv preprint arXiv:1707.06347, 2022.
[25] K. Patel and R. Sharma, “A comprehensive survey on deep learning techniques for vulnerability detection,” Computers & Security, vol. 106, 102309, 2021.
[26] Z. Chen, Y. Zhang, and F. Wang, “ML models for software composition vulnerability detection,” SoftwareX, vol. 12, 2020.

[27] M. Zhang, X. Huang, and L. Wang, “An ensemble learning framework for automated vulnerability detection in IoT software,” IEEE Internet of Things Journal, vol. 8, no. 12, pp. 9568–9577, 2021.
[28] Y. Sun and B. Li, “Explainable AI for software vulnerability detection: A survey,” ACM Computing Surveys, vol. 55, no. 9, 2023.
[29] R. Kumar and P. Singh, “Machine learning-based anomaly detection for zero-day software vulnerabilities,” Journal of Network and Computer Applications, vol. 197, 103266, 2021.
[30] S. Wang, T. Xie, and W. Zou, “Code representation learning for vulnerability detection with graph neural networks,” IEEE Transactions on Software Engineering, vol. 47, no. 6, pp. 1351–1364, 2021.
[31] F. Luo, D. Zhao, and L. Xu, “Adversarial training for resilient software vulnerability detection,” Information Sciences, vol. 578, pp. 1–14, 2021.
[32] Jain, A., & Jha, S., 2018. A deep learning approach for cybersecurity threat detection. Journal of Computer Science & Technology, 33(2), pp. 245-258.
[33] Mukherjee, A., & Saha, S., 2020. Enhancing cybersecurity through machine learning: Applications, challenges, and solutions. Journal of Cybersecurity Research, 8(1), pp. 75-85.
[34] Tariq, M., & Khan, A., 2019. Machine learning techniques for intrusion detection in cloud environments: A survey. Computers, Materials & Continua, 59(1), pp. 35-50.
[35] Tiwari, V., & Bhagat, N., 2021. Survey on the use of machine learning in detecting network anomalies and security attacks. Journal of Cloud Computing: Advances, Systems and Applications, 10(1), pp. 21-35.
[36] S. Ahmed and M. Abdullah, “Hybrid deep learning models for software vulnerability detection: A systematic review,” IEEE Transactions on Software Engineering, vol. 48, no. 5, pp. 1705–1720, 2022.
[37] L. Chen, Y. Hu, and Q. Liu, “Combining Random Forest and CNN for enhanced vulnerability detection in source code,” Journal of Systems and Software, vol. 192, 111496, 2023.
[38] M. Medeiros, L. Silva, and R. Almeida, “Evaluation of ML models in vulnerability detection,” J. Softw. Eng., vol. 35, pp. 99–114, 2020.
[39] H. Li, Y. Wang, Y. Chen, and Q. Zhang, "An enhanced machine learning approach for software vulnerability detection," IEEE Access, vol. 9, pp. 108345–108355, 2021.
[40] R. Gupta, P. Singh, and A. Sharma, “Software vulnerability detection using J48,” Cybersecurity J., vol. 6, no. 2, pp. 122–130, 2021.
[41] A. Ahsan, M. Ali, and A. Rahman, "Code gadgets for software vulnerability detection," J. Comput. Secur., vol. 28, no. 4, pp. 499–520, 2022.
[42] T. Aota, M. Kobayashi, and S. Kato, “SVC-CG: CNN-GRU-based vulnerability classifier,” IEEE Access, vol. 10, pp. 32465–32479, 2022.
[43] Y. Cai, J. Zhang, and L. Wu, “Subgraph-based vulnerability detection,” Inf. Sci., vol. 630, pp. 193–211, 2023.
[44] M. Hasib, N. Ahmed, and F. Noor, “Hybrid ML models for adversarial-resilient vulnerability detection,” J. Cyber Eng., vol. 19, pp. 44–56, 2023.

[45] A. Banerjee and S. Ghosh, “A deep reinforcement learning approach to vulnerability detection,” Proceedings of the AAAI Conference on Artificial Intelligence, vol. 35, no. 3, pp. 2421–2429, 2021.