SSRG International Journal of Electrical and Electronics Engineering
ISSN: 2348-8379/ https://doi.org/10.14445/23488379/IJEEE-V11I1P106

Volume 11 Issue 1, 58-67, January 2024
© 2024 Seventh Sense Research Group®

Original Article

Automatic Software Vulnerability Classification Based on Improved Whale Optimization Algorithm and Attention Guided Deep Neural Network

Shazia Ali[1], Arshia Arjumand Banu[2]

[1]Department of Information Technology & Security, College of Computer Science and Information Technology, Jazan University, Jazan, Kingdom of Saudi Arabia.
[2]Department of Computer Science, College of Computer Science and Information Technology, Jazan University, Jazan, Kingdom of Saudi Arabia.

[2]Corresponding Autohr : arshiabanu27@gmail.com

Received: 17 October 2023 Revised: 19 November 2023 Accepted: 15 December 2023 Published: 20 January 2024

Abstract *- The use of computers and the Internet has had two distinct effects on sectors, given the fast-paced growth of information technology. In addition to ease, they pose significant hazards and covert threats. The primary sources of several safety problems are software flaws. The safety of the system will be severely compromised after hostile assaults have exposed a weakness, and it may even result in catastrophic damage. Automated categorization techniques are thus preferred to manage software vulnerabilities efficiently, enhance system safety, and lower the possibility of system assault and harm. This work proposes a new automatic vulnerability classification model, the Improved Whale Optimisation Algorithm (IWO), and an Attention-guided Deep Neural Network (ADNN). To optimize ADNN hyperparameters, IWO was developed based on the humpback whales' swarm foraging behaviour. The model uses Information Gain (IG), Term Frequency-Inverse Document Frequency (TF-IDF), and ADNN. TF-IDF is employed to find the frequency and weight of every chat from the vulnerability report. IG is employed for feature selection to get the best feature word set. The ADNN is used to build an automatic weakness classifier to classify security issues accurately. The efficiency of the suggested model has been verified using data from the National Vulnerability Database (NVD) of the United States. The ADNN model outperformed SVM, Naive Bayes, and KNN regarding recall rate, precision, accuracy, and F1-score, among other multi-dimensional assessment measures.*

Keywords *- Information technology, Software vulnerabilities, Security, Automatic vulnerability classification model, Attention-guided Deep Neural Network, Improved Whale Optimization algorithm (IWO), Term Frequency-Inverse Document Frequency, Gathering information.*

1. Introduction

Software is becoming more significant in many spheres of life throughout the globe due to the fast growth of information technology, including the military, society, and business. Possible software security flaws are simultaneously rising as a global problem. One of the main factors contributing to security issues is software. High-skilled hackers may leverage software flaws to their advantage to carry out a variety of destructive actions as they see fit, including stealing users' sensitive information and shutting down vital equipment [1].

Once a potential attack exploits a weakness in the platform, the information system's security is deeply compromised. It could have priceless effects. Attackers are using Windows system flaws in 2017 to expose organizations all over the globe to Bitcoin recovery tools. Again, Microsoft issued a total of 372 Office security updates at the same time. Hackers use workplace security flaws to launch Advanced Persistent Threat (APT) assaults, disseminate botnets, and spread other malicious software. The number and diversity of weaknesses are now gradually increasing, making it crucial to analyze and manage software vulnerabilities.

According to Balasubramanian, Indian verticals related to education, Government, and Banking, Financial Services, and Insurance (BFSI) create the maximum risk. The COVID-19 pandemic has also altered the way attacks are deployed. Since 2020, the percentage of email-based attacks has gone up.

Before 2020, around 36% of threats came via email, with the remaining threats originating from the Internet. The percentage of email attacks increased to 89 percent in the first half of 2022 [2]. In the last several decades, many strategies have been put forward to lessen the harm caused by software defects due to their potentially extreme and severe effects. Data mining and machine learning techniques are several approaches to solving this issue.

In [3], a comprehensive analysis of the numerous papers in software security research and discoveries that employ machine learning and data mining techniques is provided. To anticipate software vulnerabilities in various databases, this work has undertaken an operational investigation into using specific well-known ML approaches and statistical methods. Cascade-forward back propagation neural networks, feed-forward back propagation neural networks, multi-layer perceptrons, adaptive-neuro fuzzy inference systems, bagging, support vector machines, M5Rrule, and M5P, along with reduced error pruning trees, are among the ML approaches that have been investigated.

The Alhazmi-Malaiya, linear, and logistic regression models have all been examined statistically. Two distinct methods are used to assess the techniques' applicability: prediction capabilities using various criteria and good-to-fit to check how fine the model matches the data. It has been shown that machine-learning approaches significantly outperform quantitative security prediction algorithms for identifying software issues [4].

In [5], examine how to use neural approaches for learning and comprehending code semantics to simplify security identification and evaluate and critique existing research using Machine Learning and Deep Learning (DL) methods to identify software flaws. Researchers have gradually put forward several automation techniques due to the advancement of DL technology, which has opened up new prospects for examining possible software security vulnerabilities. The purpose of [6] is to investigate how the latest research employs neural techniques to learn and interpret code semantics to enhance susceptibility identification. A review is conducted of the work that uses DL and neural network techniques to uncover security flaws in software.

Despite several approaches in existing work, security is still seen as an issue. So, combining DL technology with program examination technology is essential to support software security research and further encourage the growth of automated detection technology. Traditional detection methods require significant time and effort from domain experts in creating feature engineering. This article provides a detailed overview of the most recent developments in DL research for software security identification. This paper suggests a TFI-DNN system that automatically classifies security risks based on IG, TF-IDF, and ADNN. It can handle high and sparse word vector spaces better and get more out of DL's feature extraction. Following are the work's significant contributions:

- In the model, the pre-processing is done with three steps: stop word filtering, lemmatization, and word segmentation.
- Second, use TFIDFIG, a DL-based ADNN neural network model, after using an algorithm to identify the descriptive text's features and shrink the produced high-dimensional word vector space.
- The NVD's vulnerability data was employed to train and evaluate the ADNN model using an IWO-based hyperparameter selection approach.
- The optimization algorithm tunes the hyperparameters until better accuracy is achieved.
- The results of the tests demonstrate how well the automated susceptibility detection model in this study enhances the efficiency of susceptibility assessment.

There are four primary components to this study. The context and driving forces for this study are discussed in Section 2, along with the conclusion. The materials and procedures are fully described in Section 3. The experimental findings on two benchmark datasets are shown in Section 4. Section 5 closes this study by outlining the following work.

2. Related Work

Wartschinski et al. [7] presented Vudenc, a vulnerability detection tool grounded in DL that automatically picks up characteristics of susceptible code from an extensive and real-world Python codebase. To find code tokens with comparable meaning properties and to produce a vector representation, Vudenc uses a word2vec model.

The next step is to use a Long-Short-Term Memory cells (LSTM) network to classify vulnerable code token sequences at a fine-grained level. This will show exactly which source code parts are most susceptible and give ratings for their predictions' reliability. However, this detection methodology has to be supplemented by automated methods since it takes a lot of time and requires specialist knowledge.

Yan et al. [8] proposed a Hierarchical Attention Network for Binary Software Vulnerability Discovery (HAN-BSVD). The training embedding network, which consists of the word-attention module and Bi-GRU, adopts HAN-BSVD to maintain the pertinent information after the pre-processor has initially improved it with a unified jump address and normalizing training.

Local features are also recorded, and the Text-CNN and spatial-attention module feature extraction networks highlight the most significant regions. The recommended method outperforms the other examined approaches in terms of detection when evaluated on the ICLR19 and Juliet Test Suite datasets. When extracting characteristics from the susceptible code, this detection method does not highlight critical portions relevant to the vulnerability.

Niu et al., [9] A static taint analysis strategy based on DL has been developed to automatically find Internet of Things (IoT) software vulnerabilities. This method may reduce the need for time-consuming human research and increase identification reliability. The two-stage Bidirectional LSTM (BLSTM) is used to locate and identify security vulnerabilities. The suggested method is evaluated using two vulnerabilities in C/C++ programs: buffer error susceptibility (CWE-119), and resource management issues (CWE-399). However, this approach is time-consuming.

Liu et al. [10] proposed a two-level DDoS assault discovery technique founded on DL and information entropy. First, suspect elements and ports are found using the information entropy detection technique at a coarser resolution. The Convolutional Neural Network (CNN) model then uses a sophisticated packet-rooted discovery method to distinguish between legitimate, and suspicious communications. The controller then executes the defense plan to block the assault.

Liu et al. [11] developed a deep balance system that integrates the cutting-edge concepts of deep code representation learning with fuzzy-based class rebalance. Create a deep neural network with BLSTM at this stage to learn exclusionary and invariant code models from labeled vulnerable and non-sensible codes.

Then, by creating fake samples for the class of susceptible code, a novel fuzzy oversampling technique is used to rebalance the training data. At this stage, run several tests using a real-world ground-truth code dataset from the FFmpeg, LibTIFF, and LibPNG projects to assess how well the new system performs. However, the recognition efficiency of this method still has to be improved.

Zagane et al. [12] proposed the first VulDeePecker-based DL solution for multiclass vulnerability identification. The notion of code attention, which may gather data that can aid in identifying specific vulnerabilities when the samples are minimal, is the core insight of VulDeePecker. For this reason, create a dataset from scratch and then use it to assess how successful VulDeePecker is. This susceptibility detection approach cannot solve multiclass categorization since it can only identify problems but not the specific kinds of vulnerability.

Li et al. [13] proposed a structured framework for DL to find flaws in source-coded C/C++ applications. The system, called Syntactic-based, Semantics-based, and Vector Representations (SySeVR), aims to create program models that may consider semantic and syntax data relevant to vulnerability. Effective DL methods are required to minimize the danger of system attack and destruction, increase the administration effectiveness of software susceptibility categorization, and lower the cost of susceptibility repair.

Wang et al. [14] proposed a CNN along with a Gate Recurrent Unit Neural Network (GRU)-based automated approach for classifying software vulnerabilities known as SVC-CG. To begin with, the words in every susceptibility text are mapped into the space with constrained dimensions to reflect the semantic data using the Word2Vec-based Skip-gram language model that was trained to create the word vector. The local text features are then extracted using CNN, while the global text context features are extracted using GRU.

Guo et al. [15] present VulExplore, a model for finding vulnerabilities that uses both a CNN for feature extraction and an LSTM network for deep representation. It is based on the local perception power of CNN, and the time-series forecast power of LSTM. This vulnerability identification approach does not adequately consider the implicit links between the various metric characteristics. Batur Şahin & Abualigah [16] theoretically processed memories of sequential features as well as mapped from whole past of prior inputs to target vectors using deep-learned, long-lived team-hacker features. Using an immune-based feature choice model, the suggested method tried to advance the discovery abilities of static analyses, but there are many false positives in this case.

According to the research above, it is clear that classic artificial vulnerability categorization approaches have their limits when vulnerability complexity rises. As a result, researchers are increasingly focusing on DNN-based automated vulnerability classification. As discussed, many DL-based approaches have been proposed, but still, every work has some disadvantages, mainly in achieving better accuracy and automatic detection. Also, no previous approaches used optimization techniques to optimize the hyperparameters of the proposed model.

3. Proposed Methodology

This study incorporates data from the well-respected NVD [19] as input. The safety of the system will be severely compromised after hostile assaults have exploited a vulnerability, and it may even result in catastrophic losses. Therefore, it is preferable to use automated categorization techniques to manage software vulnerabilities efficiently, enhance system security, and lower the likelihood that the system will be attacked and compromised.

A brand-new automated vulnerability categorization methodology (IWO-ADNN) has been put forward. The model relies on three key components: IG, TF-IDF, and ADNN. TF-IDF determines the frequency along with the weight of every word in the vulnerability description; IG is utilized to choose the best set of feature words; as well as ADNN, a Neural Network Model, is used to create an ad-hoc vulnerability classifier to attain effective vulnerability classification, as illustrated in Figure 1.

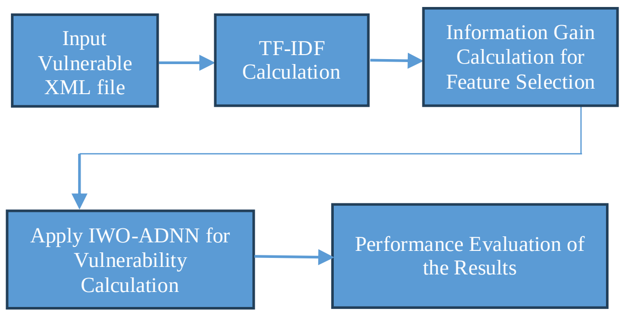
Input Vulnerable XML file
TF-IDF Calculation
Information Gain Calculation for Feature Selection
Apply IWO-ADNN for Vulnerability Calculation
Performance Evaluation of the Results

Fig. 1 The proposed methodology framework

3.1. Data Pre-Processing

First, word segmentation is done so that all coherent susceptibility text information is reduced to a single word, making the complete susceptibility textual data the lowest logical unit that can be tallied statistically. The susceptibility text preparation procedure begins with this, the most crucial phase. The term segmentation for the susceptibility mentioned in English is pretty straightforward.

Second, lemmatization is when a non-root form in a word set becomes a root form, or, depending on the individual, an English verb changes from a descriptive to a verb prototype. Transform a noun's plural form into its single form, a gerund form into a verb prototype, etc. These terms should fall under the exact conceptually comparable words from a data mining standpoint.

Finally, end word filtering, which encompasses both common and professional end words, is a term used to describe words that commonly occur in text but add little to the content. One may obtain a public stop-word list on the Internet [17], including conjunctions, modal verbs, pronouns, and popular rewords.

3.2. TF-IDF Calculation

A DNN neural network model founded on DL is then built into the model after using the TF-IDF-IG (TFIDFIG) method [18] to extract the description text feature and minimize the resulting high-dimensional word vector space dimension. The National Security Database's susceptibility data was utilized to train and test the TFI-DNN model (NVD) [19].

A widely used weighted technology based on statistical techniques is TF-IDF. There are numerous files, and everyone has a certain number of words. The significance of the word I in file "J" should be defined as follows:

$$tk_{ij} = \frac{n_{i,j}}{\sum_k n_{k,j}} \tag{1}$$

If i as well as j both contain positive integers, $n_{i,j}$ It frequently shows that a phrase appears in the j $File$. This is how the IDF formula appears.

$$idf_i = log\ log\ \frac{|File|}{|\{j: w_i\}|} \tag{2}$$

Where, $|File|$ is the number of documents in the collection as a whole, f_j is the jth file, and $|\{j: w_i\}|$ is the amount of documents that include the term $w_i \in f_j$. The following is the TF-IDF equation.

$$TFIDF = tk_{ij} * idf_i \tag{3}$$

The TF-IDF assesses a word's significance to a file inside a corpus or content collection. A word's value rises proportionately to how often it occurs in the file but also falls inversely with its occurrence in the corpus.

If a characteristic is used, it is known as IG X in class Y. If understood, class Y's informational ambiguity will diminish, and the decreased level of ambiguity shows the significance of the characteristic X to the class Y. An instructional data set is DS, where |DS| indicates the number of items in DS. Allowing for K courses $\mathrm{CL_k}$, $\mathrm{k} = 1, 2, \ldots, \mathrm{K}$, |DS| is the proportion of examples in a certain class $\mathrm{C_k}$, $\sum_{k=1}^{K} |\mathrm{CL_k}| = |\mathrm{DS}|$.

If a feature f has n different values $\{f_1, f_2, \ldots, f_n\}$, DS will be separated into n subgroups based on characteristic ratings f, denoted as $\mathrm{DS} = (\mathrm{DS_1}, \mathrm{DS_2}, \ldots, \mathrm{DS_n})$, where $|\mathrm{DS_i}|$ is how many examples there are in $\mathrm{DS_i}$. The group of examples that make up a class $\mathrm{CL_k}$ in $\mathrm{DS_i}$ is $\mathrm{DS_{ik}}$ and whose calculation looks like this:

$$DS_{ik} = |DS_i| \cap |DS_{ik}| \tag{4}$$

Where, $|DS_{ik}|$ the quantity of examples of DS_{ik}. Entropy measured empirically $E(DS)$ of the data set DS is determined as follows:

$$E(DS) = -\sum_{k=1}^{K} \frac{|CL_k|}{|DS|} \tag{5}$$

The entropy that is a conditionally scientific $E(DS|f)$ of feature f for the dataset DS is determined as follows:

$$E(f) = -\sum_{i=1}^{n} \sum_{k=1}^{K} \frac{|DS_i|}{|DS|} \cdot \frac{|DS_{ik}|}{|DS_i|} \cdot \frac{|DS_{ik}|}{|DS_i|} \tag{6}$$

The knowledge gained IG method for every concept's computation f is as follows:

$$IG(DS, f) = E(DS) - E(f) \tag{7}$$

Each feature's informational gain is computed following the feature selection technique of the informative gain criteria, and the features with higher informational gain values are chosen. The step-by-step algorithm for TFIDFIG is given as follows:

Algorithm 1. The steps of the TFIDFIG algorithm

```
Input   : File $File$ and frequency of words $n_{i,j}$.
Output : feature word set ($f$).
1.  Determine each word's TF value in accordance with (1).
2.  Determine each word's IDF value in accordance with (2).
3.  Determine each word's TF-IDF rating in accordance with (3).
4.  TF-IDF value-based descending sorting of the word set.
5.  Choose the first n words as a crucial characteristic group.
6.  Include those n terms in the feature set.
7.  Calculate the empirical conditional entropy $E(DS)$ of the data set $DS$ based on (5 and 6).
8.  Estimate the $IG$ value of every word according to (7).
9.  Keep a dictionary copy with each term and its matching IG value. 15) The set of words is arranged by IG value in decreasing order.
10. The feature words should include the paramount m words you choose as features.
11. Return feature words $m$ as final, essential features.
```

4. Detection of Software Vulnerabilities Using ADNN

Since there are m feature words in the feature word collection, each vulnerability description in the implementation of this study is stated as an m-dimensional vector. The vector description of the susceptibility is created by vectorizing each susceptibility text sample via m-dimensional space. Every susceptibility sample may be considered an opinion in a high-dimensional space representing a single susceptibility sample. v_i is $v_i = w_i\{w_1, w_2, \ldots, w_m\}$.

The number m of feature words in this formula represents the vector space dimension of vulnerability. After the vectorization representations of the susceptibility descriptions, the natural language-presented susceptibility text data is altered into a data structure that can be recognized by a computer and represented via statistical learning.

The words in various terms in a human phrase are associated in this study, which uses the DL framework ADNN. For example, the time adverbial controls whether the predicate adds "ed," and the subject may influence the predicate's forms (single or plural).

Therefore, it is suggested that self-attention considers various words in a single sequence while computing representations of the sequence [19]. As shown in Figure 2, the structure of the ADNN is currently being discussed in depth.

4.1. Embedding Block

The two sub-components of embedding E are word encoding and learning embedding. The research considers the word information in ADNN since the characteristics of various words interact. Consequently, the term "encoding" is established [19]. The word encoding converts a feature's word w from a specific input sequence to a d-dimensional word vector. E_W utilizing the following equations:

$$E_{w,2i} = sin\left(\frac{w}{10000^{\frac{2i}{d}}}\right) \tag{8}$$

$$E_{w,2i+1} = cos\left(\frac{w}{10000^{\frac{2i}{d}}}\right) \tag{9}$$

Where, $2i$, $2i + 1 \in [0,\ d - 1]$ is the channel for the incoming vector. Choose sine and cosine functions with a constant of 10000 for this section. As a result of collecting all text data related to vulnerabilities from 2000 to 2016 for statistical purposes, ten thousand were chosen as the training set, along with 1,000 serving as the test set.

As a result, each output dimension is a sinusoid. Moreover, a geometric progression of wavelengths is $[2\pi,\ 10000 \times 2\pi]$. Thus, for any given offset, these processes enable the model to learn the related words k, E_{w+k} maybe seen as a linear function of E_w. Each characteristic in the vector is given a more profound significance thanks to the learnt integration f [4]. Specifically, let $f_i \in [0, 255]$ be a component of f, have:

$$y_i = C_l onehot(f_i) \tag{10}$$

Where, the one hot encoding is indicated by the term "one hot." w_i ; C_l throughout learning, the learned coefficient matrix C is revised in an adaptable manner and y_i is a d-dimensional embedding vector that has been transformed. Therefore, for each $f_i \in f$, have a PEpos and a y_i at this stage, accordingly. For f, at this moment, afterwards, $C_f = File_w(f)$ and $C_l = File_E(f)$ where $File_w$ and $File_E$ symbolise the words "learned embedding" and "encoding," accordingly. C_f and C_l are the matching outcomes presented as a matrix.

4.2. Encoder Block

Each encoder incorporates self-attention, residual connection, layer normalization, and a straightforward one-dimensional DNN, as illustrated in Figure 1. A function that describes self-attention may map three matrices: File $File$, frequency of words $n_{i,j}$, feature word set (f) to a weighted result of $W_f V$. "Self" indicates that $File = n_{i,j} = f$ and W_f is given as:

$$W_f = softmax \sum_{i=1}^{n}\left(\frac{File \cdot n_{i,j}}{\sqrt{d'}}\right) \tag{11}$$

Where d′ stands for K's dimensions. Here, the softmax function is described as:

$$softmax(y)_j = \frac{expexp\, y_j}{\Sigma_i expexp\, y_i} \quad (12)$$

Now, each element should be subjected to an exponential process y_j and divided. A complete number of linked indices by the length of the input vector y is used to normalize these values. Additionally, the encoder's input ER is a matrix in $ER^{l\times d}$ something so huge that focusing on it directly might cause a loss of local knowledge. Decide to split the d channels into h groups now. (so $d' = d/h$).

Practice has shown that deeper networks may sometimes perform better, thus cascade n encoders to ADNN at this stage. Deep networks, however, exhibit the characteristics of sluggish training. So, to speed up the training at this stage, employ the residual connection [10] and layer normalization [3]. The remaining link may be defined as,

$$y = Op(x) + x \quad (13)$$

Where, the function Op designates a few operations carried out on input x using an attention-guided method. The two "+" symbols in the encoder element of Figure 1 show the residue connections. The left arrow of the "+" sign denotes F(x), while the top arrow denotes x. It's important to note that the remaining connectivity necessitates that the research always ensures that the input dimension dx and the output dimension dy are the same.

Hence, at this point set, $d_x = d_y = d$, where d is the before-specified embed vector's length. In addition, the layer normalization converts the output y's elements to floats with values between 0 and 1 to aid in convergence. The encoder's DNN structure is straightforward. ReLU and Maxpool are placed after the first convolution layer, respectively. The DNN module's convolution and pooling layers parameters are kernel = 3, stride = 1, and pad = 1.

4.3. ADNN Classifier

A feature matrix is the outcome of the final encoder $' \in ER^{l\times d}$. Now, add together the rows to obtain a feature vector. $v \in ER^{l\times d}$. The vector at this position is obtained by performing a linear projection on $v.v_N \in ER_N$, N is the quantity of documents. v_N does the softmax method's input consist of:

$$\Sigma = softmax(v_N) \quad (14)$$

The definition of softmax is in Equation (13). The values of each component $\Sigma_j \in \Sigma$ reflect the likelihood that the packet falls within the relevant classification j. If Σ_j the label j is chosen as the last projection since it has the highest values.

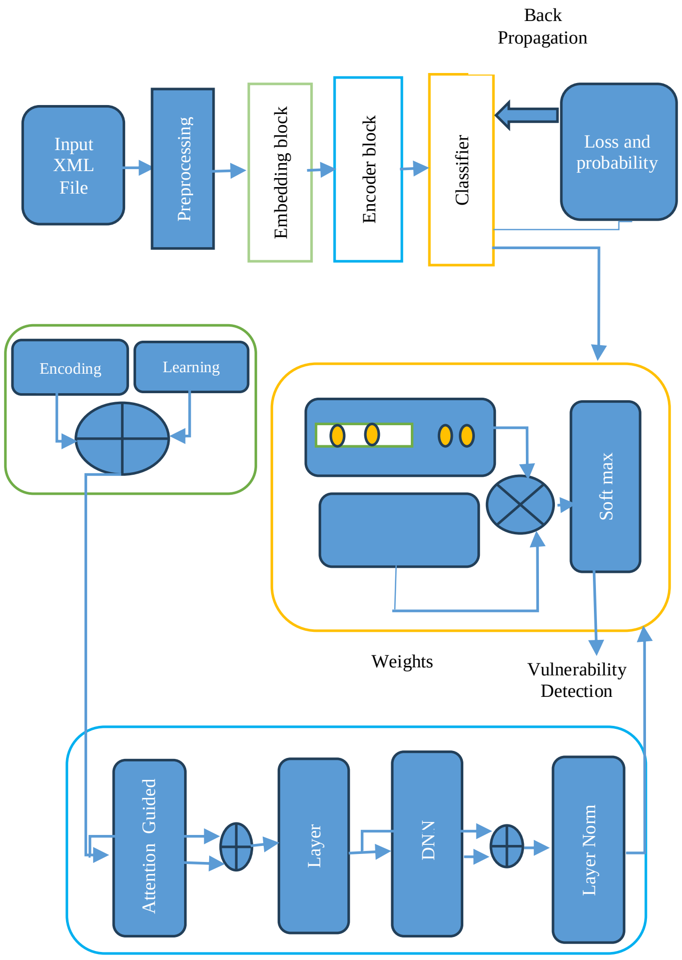

Fig. 2 The overall structure of the ADNN method

The cross-entropy loss is created in the training phase, depending on,

$$Loss = \frac{-loglog\, \Sigma_j}{File} \quad (15)$$

Where, v the actual class index is denoted by the letter j and is the class probability vector from Equation (14). In the present checking batch, the file is the total file count. The backpropagation with loss algorithm will be used to modify all parameters in the ADNN. The NVD dataset training sample is then employed to train the TFI-DNN susceptibility automated categorization model. The susceptibility test set is then used to test how well the system works.

5. Hyperparameter Optimization Using IWO

This part uses IWO to optimize a variety of configuration hyperparameters for the ADNN model, such as the set of hidden layers, the neurons count in every layer, the neural network's learning rate, and the number of iterations. The WOA is a revolutionary population-based optimization approach that was recently created and is inspired by nature. The WOA gathers search agents to identify the ideal response to an optimization issue. The WOA employs a technique recognized as "bubble-net hunting" to mimic the actions used by humpback whales as they pursue prey. The three foremost phases of the WOA are surrounding the prey, bubble net assaulting, and watching for the finest prey [20].

Once its target is located, the whale uses the spiral method to create a bubble net and ascends to reach the prey. This is the fundamental concept of whale bubble-net feeding. This invasive behaviour consists of three stages: chasing the victim, bubble-net assault, and enclosing the prey.

To identify the best solution for ADNN, the whales surround their prey, such as fish, and then attempt to update their locations. Equations provides the primary mathematical representation of the WOA. (16);

$$X(t+1) = \{X^*(t) - A \times |C \times X^*(t) - X(t)| \text{ if } p < 0.5 \ \ |C \times X^*(t) - X(t)| \times e^{bl} \cdot \cos\cos(2\pi t) + X^*(t) \quad \text{if } p \geq 0.5 \qquad (16)$$

Where, X is a vector containing the locations of each whale, and t is a time or repetition index; X^* is now the most excellent option discovered; $A = 2a.(\text{rand} - a)$; $C = 2.\text{rand}$; an is an iteration-dependent coefficient vector that linearly reduces from 2 to 0; and is a random vector with values sandwiched between 0 and with 1; In this paper, b is a constant value that determines the form of the logarithmic spiral depending on the chosen path; l is a random number between -1 as well as 1; and p is a random numeral between 0 along with 1 and is utilized to switch updating the whales' positions.

The probabilities are 50% and 50% in Equation (16), meaning that whales choose either path randomly and with an identical chance during optimization. As the bubble-net phase progresses, the random value for A is [-1, 1]; however, the randomized number of vector A during the finding phase may be more or lower than 1. In Equation (17), the search algorithm is shown.

$$X(t+1) = X_{rand} - A\,.\,|C.X_{rand} - X(t)\,| \qquad (17)$$

This random search method, which stresses the searching process along with causing the WOA algorithm to do a worldwide search, has a value of | A | bigger than one. The WOA searching method begins with the creation of random solutions. The procedure is then used to update these solutions iteratively. The search will continue up to a predetermined maximum number of iterations.

5.1. Improved Whale Optimization Algorithm (IWO)

A good trade-off between investigation and exploiting, two crucial components of an optimizing technique, aids in reaching a specific answer by avoiding local optima. In WOA, a search agent's step size gradually reduces as iterations go. This step size is controlled by the parameter A.

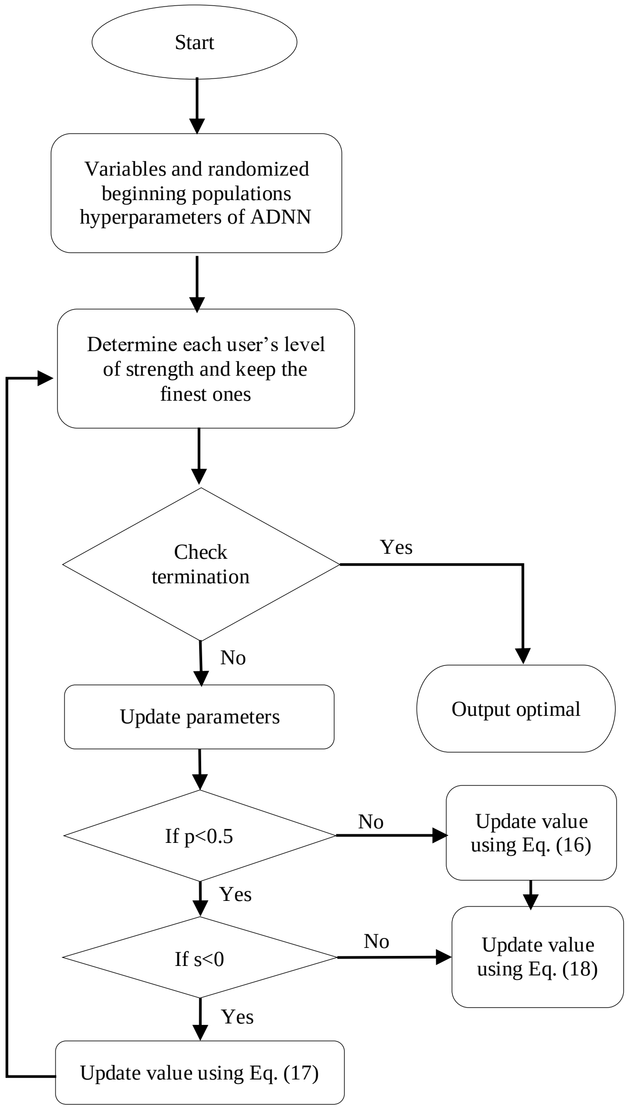
Start
Variables and randomized beginning populations hyperparameters of ADNN
Determine each user's level of strength and keep the finest ones
Check termination
Yes
No
Output optimal
Update parameters
If p<0.5
No
Update value using Eq. (16)
Yes
If s<0
No
Update value using Eq. (18)
Yes
Update value using Eq. (17)

Fig. 3 Flowchart of the WOA

However, it has been shown that WOA's limiting of traps into the local optimum occurs at later phases due to weak convergence. This work employs an updated WOA to get around such problems. This changes the value of A by introducing the levy flying function.

This enhances the capacity to use and explore WOA concurrently. Levy flight calculates the jump size using the Levy probability dispersion function, a power-law function. The Levy distribution's mathematical formula is as follows:

$$L(s,\rho,\mu) = \{\sqrt{\frac{\rho}{2\pi}} \times exp\, exp\left[-\frac{\rho}{2(s-\mu)}\right] \frac{1}{(s-\mu)^{3/2}}\ if\ 0 < \mu < \infty\ 0\ if\ s\ \leq 0 \qquad (18)$$

Where, μ, ρ, The location parameter determines the distribution's scale, whereas the scale parameter determines how many data points are collected for this distribution.

6. Experimental Results and Discussion

In command, unified assessment criteria are required to assess the effectiveness of the automated categorization model IWO-ADNN in this study and to compare the efficacy of various algorithms. The multiclass confusion matrix is the most popular categorization technique for multiclass issue assessment models, as per the standard approach of the data mining model. The identically configured DNN model was restated 20 times to compare the TF-IDF approach without IG with the impact of TFI for feature word selection.

This paper proposes a DL-based vulnerability automated categorization model called TFI-DNN. IWO-performance ADNNs in the categorization of vulnerabilities will now be compared to and evaluated against more established algorithms based on TFI, such as Vudenc [7], deep balance [11], and DNN [19], using correctness, precision, recall, as well as F1-score. The percentage of test examples successfully categorized by vulnerability to the overall test instances count is known as correctness. Following is the computation procedure.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

$$\text{Recall} = \frac{\text{TP}}{\text{FN} + \text{TP}}$$

$$\text{Precision} = \frac{\text{TP}}{\text{FP} + \text{TP}}$$

$$\text{F1} - \text{score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Precision comparison results between the suggested IWO-ADNN, Vudenc, deep balance, and DNN classifiers are shown in Figure 4. The graph indicates that, compared to other methods currently in use, the suggested method has a high precision rate. It is a highly effective technique with an accuracy rate of 0.89% for identifying attacks. Vudenc, deep balance, and DNN show reasonable precision rates of 0.75%, 0.81%, and 0.85%, respectively, when comparing the precision of existing techniques with that of IWO-ADNN. Furthermore, the precision of the IWO-ADNN training functions with the IWO method was used to pick the optimum hyperparameters, resulting in a higher precision rate.

The F-measure comparison of the proposed IWO-ADNN, Vudenc, deep balance, and DNN classifiers is shown in Figure 5. The suggested IWO-ADNN achieves a high F-measure rate of 0.9%, according to the data. When comparing the F-measure rate between the current methodologies, Vudenc, deep balance, and DNN yield lower rates of 0.775%, 0.8%, and 0.87%, respectively. This indicates that the proposed strategy can produce better attack identification results than the earlier methods.

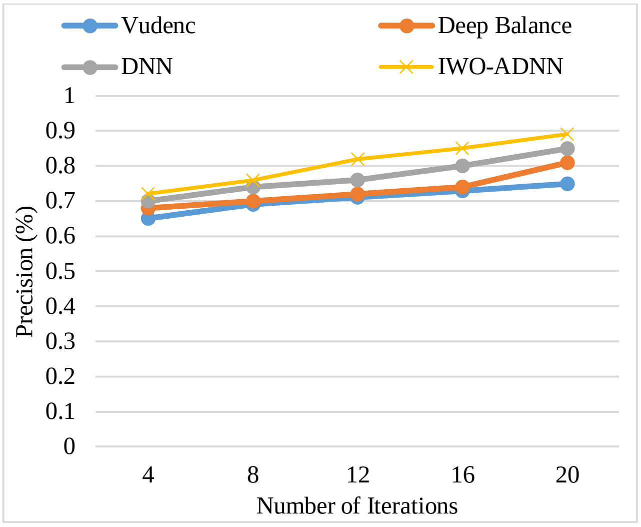

Fig. 4 Precision performance comparison

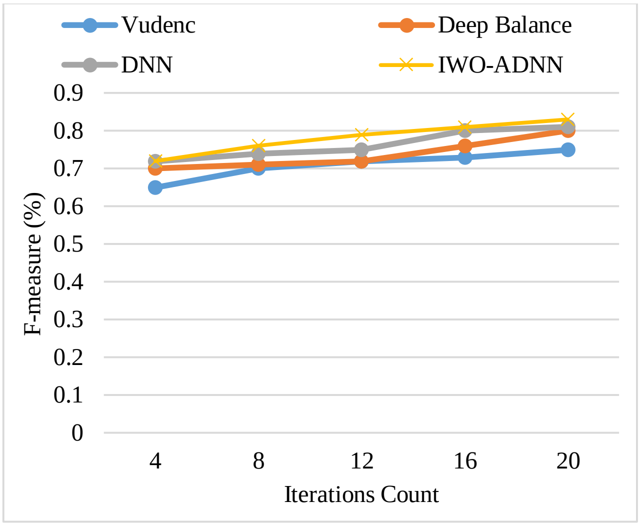

Fig. 5 F-measure comparison

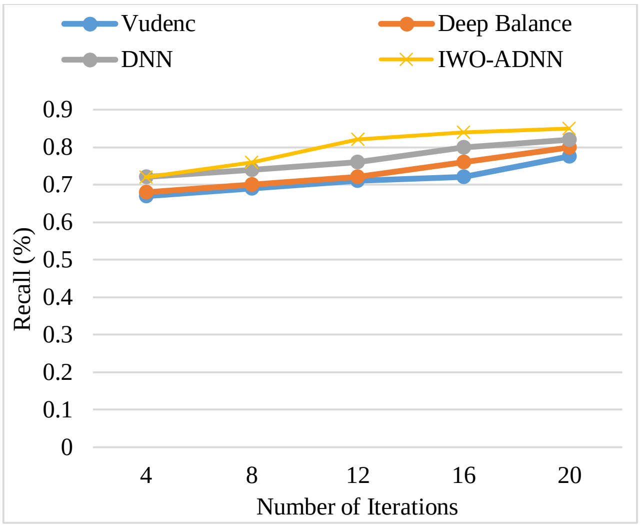

Fig. 6 Recall performance comparison

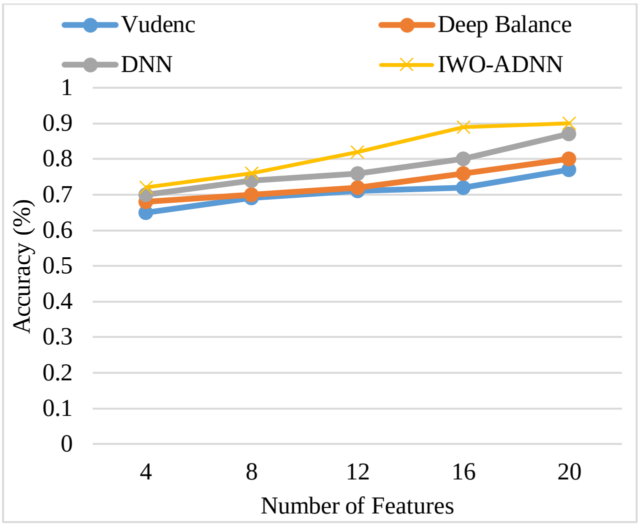

Fig. 7 Accuracy performance comparison

This is explicable because the IWO-ADNN network has practical pre-processing stages that increase the f-measure value and generally trains much more quickly than the Vudenc, deep balance, and DNN networks.

Figure 6 displays the recall comparison results for the proposed DNN, Vudenc, deep balance, and IWO-ADNN classifiers. The suggested method offers an incredibly high recall rate of 0.85%. The results indicate that the proposed IWO-ADNN has vital attack recognition accuracy and a high recall rate value. Vudenc, deep balance, and DNN yield recall rates of 0.775%, 0.8%, and 0.82%, respectively, compared to the other ways' recall rates.

This indicates that the proposed scheme can outperform the earlier methods regarding attack recognition results. Besides the importance of word frequency, TF-IDF also looks at how relevant words are to the susceptibility categories to get a better word set based on the IG value and improve performance in different evaluation indices.

The graph in Figure 7 above compares the accuracy of assault detection. Techniques like Vudenc, deep balance, IWO-ADNN, and DNN multiclass classifiers are applied. IWO-ADNN has a high accuracy rate of 0.9%, making it an excellent tool for getting precise forecasts. The accuracy rates of earlier methods, including Vudenc, deep balance, and DNN, are 0.77%, 0.8%, and 0.87%, respectively. IWO-ADNN learning techniques eliminate the local optima problem and allow for improved accuracy due to their relative resistance to noise in training data. Furthermore, IWO can achieve faster convergence than other methods while removing premature convergence, increasing the pace at which vulnerabilities are discovered.

7. Conclusion and Future Work

This study aims to evaluate IWO-ADNN to detect software vulnerabilities. There is a thorough discussion of the method analysis and the building process of TFI along with DNN. The susceptibility categorization system TFI-DNN on the NVD dataset was now contrasted with TFI-SVM, TFI-Naive Bayes, and TFI-KNN. According to the findings, the suggested TFI-DNN model surpasses the competition regarding correctness clarity and the F1 score and has a high recall rate.

Finding the precise location of the susceptible code may result from this. The research demonstrates that code metrics are reliable information that helps IWO-ADNN understand the qualities of easy code. It is also determined that despite the intriguing findings acquired, IWO-ADNN are excellent but not the best when the acquired findings are matched with the published outcomes of those research.

As a part of the study, a dataset of code metrics has been offered and made accessible to the public. Other researchers may utilize this dataset to test and improve IWO-ADNN. Plan to create an actual vulnerability detection system depending on the suggested methodology and use conventional ML approaches. Addressing the constraints would be fascinating for future work.

One of the urgent future tasks that can be completed is the classification of vulnerabilities in the "exposure leading to access violation" class. The security risk of the software can also be ascertained during the design phase using the classification findings. There are design patterns that act as mitigation strategies after the risk estimation process. This enables program creators to address security flaws in software early in the design process.

Another potential topic of future work is creating an automated vulnerability classification tool. After classifying the data, the first objective will be to identify, assess, and classify design patterns that can be changed to protect the newly developed program against vulnerabilities.

References

[1] Muhammad Shahzad, Muhammad Zubair Shafiq, and Alex X. Liu, "A Large Scale Exploratory Analysis of Software Vulnerability Life Cycles," *2012 34th International Conference on Software Engineering (ICSE)*, Zurich, Switzerland, pp. 771-781, 2012. [CrossRef] [Google Scholar] [Publisher Link]

[2] CNBC TV18, Technology News, Indian Enterprises Highly Vulnerable to Cyber-Attacks, Says Expert. [Online]. Available: https://www.cnbctv18.com/technology/indian-enterprises-highly-vulnerable-to-cyber-attacks-cyber-expert-14684671.html

[3] Seyed Mohammad Ghaffarian, and Hamid Reza Shahriari, "Software Vulnerability Analysis and Discovery Using Machine-Learning and Data-Mining Techniques: A Survey," *ACM Computing Surveys*, vol. 50, no. 4, pp. 1-36, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[4] Gul Jabeen, "Machine Learning Techniques for Software Vulnerability Prediction: A Comparative Study," *Applied Intelligence*, vol. 52, pp. 17614-17635, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[5] Aya El-Rahman Kamal El-Deen Ramadan, Ahmed Bahaa, and Amr Ghoneim, "A Systematic Literature Review on Software Vulnerability Detection Using Machine Learning Approaches," *Informatics Bulletin, Faculty of Computers and Artificial Intelligence*, vol. 4, no. 1, pp. 1-9, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[6] Guanjun Lin et al., "Software Vulnerability Detection Using Deep Neural Networks: A Survey," *Proceedings of the IEEE*, vol. 108, no. 10, pp. 1825-1848, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[7] Laura Wartschinski, "VUDENC: Vulnerability Detection with Deep Learning on a Natural Codebase for Python," *Information and Software Technology*, vol. 144, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[8] Han Yan et al., "HAN-BSVD: A Hierarchical Attention Network for Binary Software Vulnerability Detection," *Computers & Security*, vol. 108, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[9] Weina Niu, "A Deep Learning Based Static Taint Analysis Approach for IoT Software Vulnerability Location," *Measurement*, vol. 152, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[10] Ying Liu, "Software-Defined DDoS Detection with Information Entropy Analysis and Optimized Deep Learning," *Future Generation Computer Systems*, vol. 129, pp. 99-114, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[11] Shigang Liu et al., "Deep Balance: Deep-Learning and Fuzzy Oversampling for Vulnerability Detection," *IEEE Transactions on Fuzzy Systems*, vol. 28, no. 7, pp. 1329-1343, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[12] Deqing Zou et al., "μμVulDeePecker: A Deep Learning-Based System for Multiclass Vulnerability Detection," *IEEE Transactions on Dependable and Secure Computing*, vol. 18, no. 5, pp. 2224-2236, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[13] Zhen Li et al., "Sysevr: A Framework for Using Deep Learning to Detect Software Vulnerabilities," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 4, pp. 2244-2258, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[14] Qian Wang et al., "An Automatic Algorithm for Software Vulnerability Classification Based on CNN and GRU," *Multimedia Tools and Applications*, vol. 81, pp. 103-7124, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[15] Junjun Guo et al., "Detecting Vulnerability in Source Code Using CNN and LSTM Network," *Soft Computing*, vol. 27, pp. 1131-1141, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[16] Canan Batur Şahin, and Laith Abualigah, "A Novel Deep Learning-Based Feature Selection Model for Improving the Static Analysis of Vulnerability Detection," *Neural Computing and Applications*, vol. 33, pp. 14049-14067, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[17] Stop Words, 2018. [Online]. Available: https://pypi.org/project/stop-words/

[18] Guoyan Huang et al., "Automatic Classification Method for Software Vulnerability Based on Deep Neural Network," *IEEE Access*, vol. 7, pp. 28291-28298, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[19] Information Technology Laboratory, National Vulnerability Database. [Online]. Available: https://nvd.nist.gov/

[20] Andrzej Brodzicki, Michał Piekarski, and Joanna Jaworek-Korjakowska, "The Whale Optimization Algorithm Approach for Deep Neural Networks," *Sensors*, vol. 21, no. 23, pp. 1-16, 2021. [CrossRef] [Google Scholar] [Publisher Link]