Volume 6 Issue II, February 2026, No. 82, pp. 1044-1057
Submitted 2/10/2025; Final peer review 7/1/2026
Online Publication 22/2/2026
Available Online at http://www.ijortacs.com

A HYBRID DEEP LEARNING ARCHITECTURE FOR ROBUST VULNERABILITY CLASSIFICATION

[1]Ekpo, Michael E., [2]Ituma Chinagolum, [3]Pius Ekwo K.
[1]Department of Computer Science, Ebonyi State University, Abakaliki, Nigeria
[2]Department of Computer Science, Ebonyi State University, Abakaliki, Nigeria
[3]Department of Mathematics & Computer Science, Federal University of Health Sciences, Otukpo Benue State, Nigeria
[1]ekpom550@gmail.com; [2]ichinagolum@gmail.com; [3]piuskekong2019@gmail.com

Abstract

The increasing sophistication of cyber threats, particularly against critical infrastructure like healthcare Internet of Things (IoT) networks, demands advanced vulnerability management solutions. This paper presents a novel AI-driven Deep Packet Inspection (DPI) model for proactive vulnerability assessment and management. In the proposed study, a hybrid deep learning architecture, which combines a Convolutional Neural Network (CNN) to extract hierarchical features of network traffic with an ensemble of Feed-Forward Neural Networks to perform powerful vulnerability classification, was proposed. The model was trained and evaluated on an integrated dataset of more than 900,000 records, consisting of real-world vulnerability data of a healthcare IoT testbed and Common Vulnerabilities and Exposures (CVE) database. Our ensemble-based classifier showed outstanding results and the component models had test accuracy of up to 99.31%. The suggested system was implemented in such a way that it became a full-fledged Vulnerability Assessment and Management System with a rule-based decision support algorithm to score threats according to the National Vulnerability Database (NVD) standard and prioritize them. The results of the experiment show that the combined framework manages to automate the process of real-time packet capture and analysis to vulnerability reporting to action. The study offers a scalable, precise and holistic solution that can improve cybersecurity posture in critical network infrastructures that can effectively ensure that the disparity between AI-enabled threat detection and practical vulnerability management converges.

Keywords: Deep Packet Inspection; Vulnerability Management; Convolutional Neural Network; Ensemble Learning; Artificial Intelligence

1. INTRODUCTION

Critical National Infrastructure (CNI) encompasses systems that are very important for national security, stability of the economy, and public safety. Examples of such infrastructures are power grids, transportation systems, health care systems, and financial institutions (Nikolaou et al., 2023).Prior to the 21st century, CNI are considered isolated from cyberspace, until recently where advanced technologies like the Internet of Things (IoT), has enable the integration of communication networks technologies with CNI as a Critical Network Infrastructural Systems (CNIS) (Ntafloukas et al., 2022). These CNISs rely heavily on a secured and reliable communication network to operate effectively and facilitate real-time service delivery; however,

Corresponding Author Tel: +234 803 959 0555

Hulayyil et al. (2023) posit that one major problem with the CNIS system is the issue of cyber-attack.

Vulnerability is a flaw inherent in the systems that make it exposed to cyber-attack. In CNISs, vulnerabilities exist in the software, hardware, and networks (Arampatzis, 2022). Popular vulnerabilities include misconfigurations, errors in code, outdated software, and inadequate security policies (Anand et al., 2020). Exploiting vulnerabilities can have significant impact and far-reaching consequences for CNIS. Some of the impacts include system failure, violation of network and data integrity, data and network breaches, and ransomware, while the consequences include erosion of trust, blackmail, loss of finance, and service disruption, thus necessitating the need for vulnerability management (Nikolaou et al., 2023).

Vulnerability management is a risk-based approach applied to detect, prioritize, remediate, and mitigate flaws (Neshenko et al., 2019). The traditional method of vulnerability management involved the application of vulnerability scanning tools, a vulnerability scoring system, a patch management system, and an intrusion detection system (Aloui and Nfaoui, 2022); however, these approaches are more of a reactive solution and hence struggle to guarantee elements of computer network security, which are confidentiality, integrity, and availability and in order to solve this problem, Deep Packet Inspection (DPI) has emerged as a pivotal approach in network security, allowing thorough network analysis (Mike et al., 2022; Wenguang et al., 2020). The DPI has been applied to solve several cybersecurity problems such as threat detection and vulnerability management (Maria et al., 2020; Foreman et al., 2024; Son et al., 2023). For instance, Neshenko et al. (2019) address vulnerability in IoT facilities through packet prioritization, while Feng et al. (2022) focused on the detection of firmware vulnerabilities. In contrast, state-of-the-art ML and deep learning algorithms (DL) are applied in Yu et al. (2020) for vulnerability management in IoT, while Ahanger et al. (2022) improved IoT security through a survey of artificial intelligence (AI) systems for network vulnerability management. Naeem et al. (2020) applied DL for vulnerability management in IoT and then tested the model on two datasets, which both reported over 61% accuracy.

For instance, the healthcare sector faces several new vulnerabilities, which include weak authentication, insufficient encryption, outdated firmware, and interoperability issues, which increase their attack surface. In addition, the integration of cloud computing into health IoT also introduces additional risks from misconfiguraiton, while insider threats and the use of IoT devices in botnets exacerbate the situation (Madanian et al., 2024), thus raising a research question on what can be done to improve vulnerability detection performance in healthcare IoT facilities? To solve the research question and other issues raised in the background, this study proposes the design of a deep packet inspection model for vulnerability management in critical network infrastructure using artificial intelligence techniques

2. METHODOLOGY

Multidisciplinary Integrated Research Methodology was adopted for this study to address the complex challenges of designing a deep packet inspection model for vulnerability management in critical network infrastructure using artificial intelligence techniques. This methodology integrates principles from artificial intelligence, network security, and systems engineering, ensuring a holistic approach to problem-solving. It involves comprehensive domain analysis to identify attack vectors and vulnerabilities, followed by the development of an advanced AI-driven inspection model leveraging deep learning techniques such as convolutional neural network and ensemble feed-forward neural networks. The model is integrated into a scalable healthcare IoT system architecture for real-time vulnerability scanning and management.

Rigorous validation was conducted through simulation using realistic network scenarios and empirical testing in controlled operational environments, ensuring robustness, accuracy, and scalability.

2.1 Data Collection

Data used for this work was collected from primary and secondary sources. The primary source of data collection is the University of Uyo teaching hospital, information technology and communication department, Akwa Ibom state. The population size is 2010 to 2024 vulnerability records. This case study was selected for the study due to its reliance on IoT enables medical devices for the management of patient health information, making it a prime target for cyber threat. The vulnerability data collected span across all the IoT layers. For the perceptron/physical layer, vulnerabilities such as weak authentication, unencrypted communication data and outdated firmware were collected using Wireshark as the tool. For the network layer, vulnerabilities such as misconfiguraiton of router, firewall and insecure communication were collected using metasploit software, while for application layer software vulnerabilities, privilege escalation, SQL injection, cross-site scripting, and privilege escalation are vulnerabilities considered and collected using Burp suite kit. The collective integrated of this data collected formed the primary dataset. The sample size of the data collected is 22090 records. The secondary dataset considered is the Common Vulnerability and Exploit (CVE) which is a popular vulnerability data which constitutes both old and new vulnerabilities from IoT infrastructure. The CVE contained 890,660 samples of vulnerability across 12 attributes. The total sample size of data collected is 901869 records.

2.2 Data Processing

The data collected was processed using imputation techniques as the first data processing step to remove missing and duplicate values. Normalization approach was then applied to balance the data and also dimensionality reduction. Random data upsampling and downsampling was applied for class balance in the dataset, before auto encoder was applied for data transformation and feature extraction to make the dataset compatible for training deep learning model. Collectively these data processing steps were applied in preparation of the dataset for the training of the deep leaning model for the generation of the DPI.

2.3 Modelling of the Convolutional Neural Network

This work utilized CNN as the deep learning model of choice due to its super feature extraction capabilities. The CNN is made of input layer, convolutional layer, fully connected layer and output layer as shown in the architectural model of Figure 1.

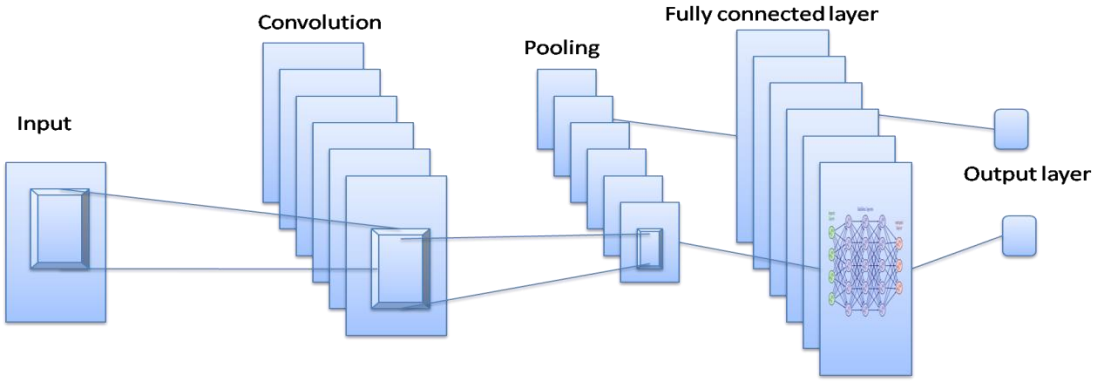

Figure 1: The CNN architecture

CNN is input layer is responsible for the dimension of the input vulnerability data. The data are then feed to the convolutional layer through the application of convolutional scan performed with filters. The type of filter used for this process is the maximum pooling layers which identified the feature maps with the highest size and then extract. The Equation 1 represents the extracted feature maps (Sethi and Serrao, 2024).

$$F_m = \left[\frac{F_a + 2_b - k}{s}\right] + 1 \quad (1)$$

Where F_m is the output features, F_a is the input features, b is the convolutional padding, s is the strides size, k is the convolutional kernels size. The feature map extraction process to form the convolutional layer defined with Equation 2 (Ray, 2018);

$$C_O = ((w * h) + 1)nf) \quad (2)$$

Where Co is the total feature maps which formed the pixels in the convolutional layer, w is the filter weight, while h is the height, the number of filters used to extract the pixels are defined as nf and the bias term is 1. To compute the total number of convolutional extracted pixels in the next n convolutional layer, the model in Equation 3 was used (Sethi and Serrao, 2024);

$$C_O = ((w * h * np) + 1) * nf) \quad (3)$$

Where np is the number of pixels extracted and feed to the layer. The activation size is defined as Equation 4;

$$A_s = (w * h * d) \quad (4)$$

Where w is the weight, H is height and d is the depth of the image.

2.3.1 Modelling of the Ensemble Neural Network

The ensemble neural network integrated three neural network models. The building block of neural network constitutes the input layer which has neurons made of weights, bias and activation functions. A neuron is defined as Equation 5 (Nowak et al., 2024);

$$Y = f(wx_{ij} + b) \quad (5)$$

Where y is the output, x is the input matrix of size $(i * j)$, $i$is the data and j is the data features; w presents the weight of the neural network, b is the bias function and f represents the activation function. The neural network has three hidden layers defined as Equation 6 (Ntafloukas et al., 2022);

$$Y_L = f_l\ (w_l f_{l-1}(w_{l-1} f_{l-2}(\ldots.. f_2(w_2 f_1(w_1 x + b_1) + b_2) \ldots.. + b_{l-1}) + b_l) \quad (6)$$

Where $y_l donates$the output, b_l is the bias of the hidden layer. The number of neurons in the input layer is determined by the 12 data attributes, while the activation function used is the hyperbolic tangent activation function. The architectural parameters like i is the neurons, n is the number of neurons,h is the hidden layers, h_n is the number of hidden layers,o is the output layer. Suppose the ensemble neural network is formed with the integration of three network $N^m, m \in$ {(ANN_A (1); ANN_B(2), ANN_C(3)) }and Equation 6, each with three hidden layers. For the input $x \in R^d$, the m-th network is presented as Equation 7-10 (Ntafloukas et al., 2022);

$$h_1^{(m)} = f_1 W_1^{(m)} x + b_1^{(m)} \quad (7)$$

$$h_2^{(m)} = f_2 W_2^{(m)} h_1^{(m)} + b_2^{(m)} \quad (8)$$

$$h_3^{(m)} = f_3 W_3^{(m)} h_2^{(m)} + b_3^{(m)} \quad (9)$$

$$y^m = f_{out} W_4^{(m)} h_3^{(m)} + b_4^{(m)} \quad (10)$$

Where $W_l^{(m)}$, $b_l^{(m)}$ are the weights and bias of the l-th layer in the network m, and f_1, f_2, f_3 are the hidden layers activation with f_{out} which is the output activation function. the weighed

aggregation of the neuron network is presented with non negative weights of α_m as Equation 11 which the ensemble neural network model as shown in Figure 2.

$$y_{ens}(X) = f_{ens}\alpha_1 y^1(x) + \alpha_2 y^2(x) + \alpha_3 y^3(x) \quad (11)$$

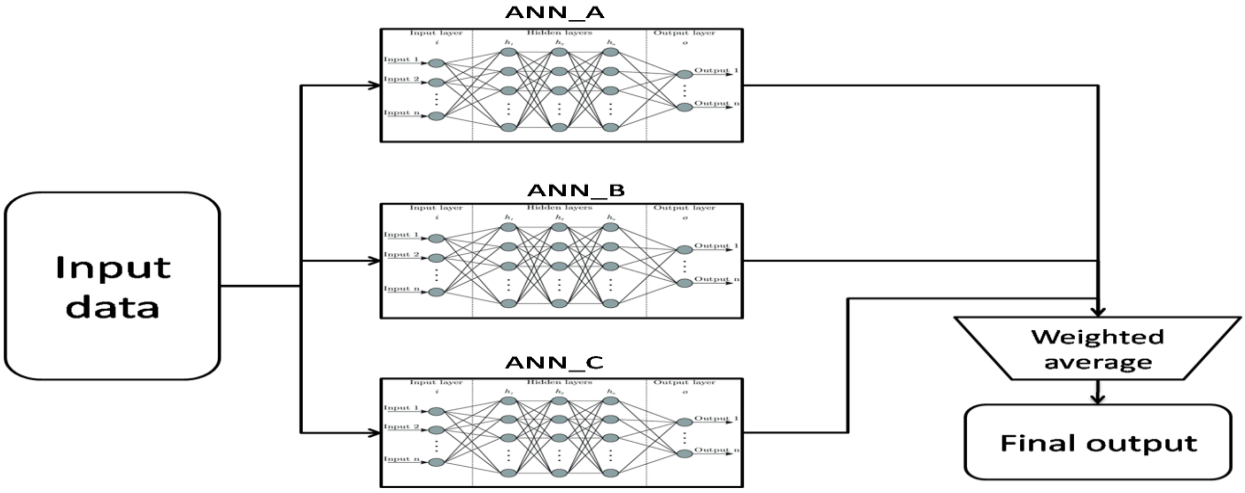

Figure 2: Architecture of the ensemble neural network

2.4 Development of the deep packet inspection model for vulnerability assessment

The Deep Packet Inspection (DPI) model for vulnerability assessment was developed by leveraging the trained convolutional VulNet, which combines convolutional neural networks with an ensemble neural network classifier. In this model, incoming data packets from network traffic are first captured and pre-processed into structured formats that preserve both content and metadata. The convolutional layers of the VulNetautomatically extract discriminative spatial and temporal features from these packet representations, highlighting abnormal behaviours, irregular communication flows, and hidden attack signatures that may indicate security vulnerabilities. These extracted features are then passed to the ensemble neural network integrated within the classification layer, which aggregates multiple decision outcomes to provide a more reliable and generalized vulnerability classification.

Deep Packet Inspection (DPI) Model for Vulnerability Assessment Stepwise

1. Start
2. Real-time network traffic packets are captured
3. Packets are transformed into feature representations by the CNN.
4. CNN layers detect abnormal payload patterns and structural anomalies.
5. Ensemble NN classifies packets into vulnerability categories
6. The DPI model continuously monitors traffic
7. End

2.4.1 Development of the Vulnerability Decision Support Scoring System

To develop the vulnerability decision support scoring system, the data used was collected from the national vulnerability database (NVD, 2025) as reported in Table 1 with common vulnerability severity rating score. The data was then used to develop the rule-based decision support algorithm or vulnerability scoring.

Table 1: Common Vulnerability Severity Rating (NVD, 2025)

Severity	Score	Risk level	Impact
Low	0.0-1.5	Low	Low

Very Low	1.6-3.9	Low	Low
Medium	4.0-6.9	Medium	Medium
High	7.0-8.0	High	High
Very high	9.0-10	Critical	Very high

The decision support scoring algorithm

1. **Start**
2. **Initialization**
3. Load trained CNN + Ensemble (ConVulNet) model.
4. Load incoming vulnerability packet features.
5. Define severity scale thresholds (Table 1).
6. **Read Class Probability**
7. Pass packet through Convulnet.
8. Obtain predicted class probabilities $P(class_i)$
9. **Compute Vulnerability Score**
10. Calculate the average CVSS $[CVSS = (P(class_i) * score(class_i)]$
11. Normalize the score into the range 0–10.
12. **Rule-based Decision Mapping**
13. If $0.0 \leq$ score $\leq 1.5 \rightarrow$ classify as Low (Low Risk, Low Impact)
14. If $1.6 \leq$ score $\leq 3.9 \rightarrow$ classify as Very Low (Low Risk, Low Impact)
15. If $4.0 \leq$ score $\leq 6.9 \rightarrow$ classify as Medium (Medium Risk, Medium Impact)
16. If $7.0 \leq$ score $\leq 8.0 \rightarrow$ classify as High (High Risk, High Impact)
17. If $9.0 \leq$ score $\leq 10.0 \rightarrow$ classify as Very High (Critical, Very High Impact)
18. **Decision Support Output**
19. Display classification results (severity level, risk level, impact).
20. **End**

2.4.2 Development of Vulnerability Assessment and Management System

The vulnerability assessment and management system was developed as an integrated framework that combines both classification and decision-making functionalities to ensure robust identification, evaluation, and prioritization of vulnerabilities. The system architecture is composed of two primary modules of deep packet inspection-based vulnerability classification engine, and rule-based vulnerability decision scoring system. The classification engine leverages the ConvulNet deep learning model to analyze packet-level features and identify potential vulnerabilities with high accuracy. This stage ensures that traffic anomalies, misconfiguraiton, and exploit patterns are effectively detected and mapped into known vulnerability categories. ConvulNet's ability to extract hierarchical patterns from structured datasets enhances detection performance, particularly for zero-day and complex vulnerabilities.

The second stage of the system integrates the decision scoring mechanism, which was designed using a rule-based approach with a scale of 1–10. Once vulnerabilities are classified, the decision engine assigns severity scores based on predefined rules that reflect parameters such as exploitability, impact on confidentiality, integrity, and availability, as well as contextual system importance. This scoring provides a quantifiable metric for ranking vulnerabilities, enabling security analysts to prioritize remediation efforts effectively. The integration of these modules produces a unified vulnerability assessment and management system capable of not only identifying and classifying vulnerabilities but also evaluating their severity in real time.

Stepwise Development of Vulnerability Assessment and Management System

1. Start
2. Collect raw packet data
3. Extract relevant features with CNN

4. Normalize and encode the dataset into a structured format
5. Use the trained ConvulNet model to classify packets into different vulnerability
6. Feed classification results into the decision scoring system.
7. Apply predefined rules (scale of 1–10)
8. Assign a severity score to detected vulnerability.
9. Rank vulnerabilities based on severity scores.
10. Group them into categories:
11. Low Risk (1–3) → low severity
12. Medium Risk (4–6) → High severity
13. High Risk (7–8) → High severity
14. Critical Risk (9–10) → critical severity
15. Log results into a centralized vulnerability management dashboard.
16. Send alerts to system administrators with detailed assessment reports.
17. Continuous Learning and Feedback
18. End

2.6 System Implementation

The system implementation phase describes how the proposed model was developed, integrated, and executed to achieve the desired functionality. The system was implemented in a modular fashion, where the convolutional neural network served as the primary feature extractor and the ensemble neural network acted as the classifier. The CNN was designed with multiple convolutional and pooling layers to automatically extract deep spatial features from the input dataset. These features were flattened and passed to fully connected layers, which served as the input for the ensemble stage. For the ensemble neural network, three independent artificial neural networks were implemented, each trained with the extracted features. The outputs of the ANNs were aggregated using a weighted voting mechanism to generate the final classification decision. This approach ensured that the weaknesses of one model were compensated by the strengths of the others, thereby improving robustness and accuracy.

The system was implemented using Python programming language with TensorFlow and Keras libraries for deep learning model development. The models were trained on GPU-enabled Google Colab for faster computation, with hyper parameters including learning rate, batch size, and epoch size optimized through experimental trials. The implementation also included performance monitoring and visualization tools, such as accuracy and loss plots, to evaluate the learning process and system reliability. The modular design of the implementation makes it adaptable for system integration as software for deep packet inspection.

3. RESULT OF THE ENSEMBLE NEURAL NETWORK TRAINING

This section presents the result of the neural network training process. The training was done experimentally by training the three different neural network algorithms and then computing the average as the ensemble classifier for the detection of vulnerabilities. The result of the first neural network training ANN_A was reported in Table 2.

Table 2: Result of the ANN_A Training

Epoch	Train_Accuracy	Train_Loss	Val_Accuracy	Val_Loss
1	0.3155	1.3652	0.3938	1.332
2	0.4768	1.1845	0.4953	1.2505
3	0.5641	1.0104	0.5503	1.0822
4	0.6323	0.8535	0.49	0.9475

5	0.6756	0.7386	0.4769	0.9244
6	0.7296	0.6277	0.574	0.7576
7	0.7843	0.5267	0.7398	0.5104
8	0.84	0.424	0.8838	0.3551
9	0.8831	0.3362	0.9195	0.2735
10	0.899	0.2756	0.952	0.2102
11	0.911	0.2371	0.9581	0.1771
12	0.9206	0.2098	0.9565	0.1611
13	0.93	0.1835	0.9517	0.1546
14	0.9306	0.181	0.9637	0.1351
15	0.9327	0.1656	0.9507	0.1412
16	0.9343	0.1597	0.9608	0.1245
17	0.9388	0.1492	0.964	0.1137
18	0.9475	0.136	0.9712	0.1024
19	0.9451	0.1347	0.9675	0.0968
20	0.9405	0.1346	0.9715	0.0912

The performance results of ANN_A in Table 2 across the 30 training epochs demonstrate a strong and consistent learning pattern. At the beginning of training, the model started with a very low accuracy of about 31.6% on the training set and 39.4% on the validation set, which is expected since the network was essentially making near-random predictions at that stage. As training progressed through the early epochs, both training and validation accuracies improved significantly, with training accuracy reaching above 70% and validation accuracy climbing above 55% by the sixth epoch. This steady rise in accuracy, accompanied by a noticeable decline in both training and validation losses, indicates that the model quickly began capturing meaningful feature representations from the extracted inputs.

During the middle stages of training, roughly between the 7th and 15th epochs, the model's performance improved further, achieving over 93% training accuracy and consistently surpassing 95% validation accuracy by the 14th epoch. At this point, validation accuracy actually exceeded training accuracy, which is often a positive sign, suggesting that the model was not only learning but also generalizing well to unseen data. The validation loss remained slightly lower than the training loss, showing no signs of severe overfitting, although a small performance gap began to appear, which is typical in neural networks as they start to converge.

In the final training phase, from epochs 16 to 30, the network stabilized at a very high performance level. Training accuracy capped at around 96.5%, while validation accuracy peaked at an impressive 99.0% at the 28th epoch, before finishing around 98.7%. The validation loss continued to decrease, reaching as low as 0.058, which is evidence of strong generalization. During the testing of the model, the accuracy recorded 0.9931 which is 99.31 success and loss of 0.0176.

Overall, the results show that ANN_A is a well-trained and robust model. It demonstrates a smooth and reliable learning trajectory, avoids overfitting despite extended training, and achieves excellent generalization with very high validation accuracy. This makes it a strong candidate for inclusion in the ensemble neural network since it contributes not only accurate but also stable predictions across unseen data. Table 3 presents the result of the ANN_B training.

Table 3: Result of the ANN_B Training

Epoch	Accuracy	Loss	Validation Accuracy	Validation Loss
1/40	0.6877	0.7491	0.9739	0.0817
2/40	0.9610	0.1168	0.9891	0.0416
3/40	0.9755	0.0730	0.9859	0.0366
4/40	0.9792	0.0564	0.9896	0.0287
5/40	0.9839	0.0437	0.9901	0.0257
6/40	0.9864	0.0362	0.9912	0.0235
7/40	0.9874	0.0331	0.9901	0.0234
8/40	0.9895	0.0293	0.9915	0.0221
9/40	0.9893	0.0282	0.9896	0.0236
10/40	0.9895	0.0262	0.9909	0.0225
11/40	0.9906	0.0249	0.9904	0.0224
12/40	0.9897	0.0265	0.9896	0.0248
13/40	0.9915	0.0247	0.9896	0.0233

The result presented in Table 3: Result of the ANN_B Training clearly demonstrates the progressive improvement of the network's learning ability across epochs. At the beginning of training (Epoch 1), the model's training accuracy was relatively low at 0.6877, with a high training loss of 0.7491, indicating that the network was just beginning to learn the underlying patterns in the data. Interestingly, the validation accuracy at this stage was already very high (0.9739), with a low validation loss (0.0817). This suggests that the features extracted from the CNN provided a strong and separable representation, allowing ANN_B to generalize well even in the early stages.

As training progressed, there was a rapid increase in accuracy, with the model reaching 0.9610 accuracy by Epoch 2 and a corresponding significant drop in training loss to 0.1168. At the same time, validation accuracy improved further to 0.9891, and validation loss decreased to 0.0416, indicating that the network was not only fitting the training data but also maintaining strong generalization to unseen samples. From Epochs 3 to 8, the training accuracy consistently improved, reaching values above 0.98, while both training and validation losses decreased steadily, showing a healthy convergence pattern. Between Epochs 9 and 13, the model's performance stabilized with accuracy values fluctuating slightly between 0.9893–0.9915 and validation accuracy ranging between 0.9896–0.9909. The training and validation losses in this stage remained low, hovering around 0.022–0.028, which reflects that the network had reached a plateau in its learning, converging to a high-performance solution. The absence of sharp divergence between training and validation performance confirms that the network was not overfitting during these epochs. The testing result recorded 0.9928 accuracy and loss of 0.0195.

Overall, the training history of ANN_B demonstrates that the model achieved very high accuracy (≈ 99%) with minimal loss values after only a few epochs, thanks to the effective feature representation provided by the CNN. The results highlight that ANN_B is highly efficient in learning from extracted features, achieving a good balance between fitting the training data and maintaining strong generalization on validation data. This strong consistency between training and validation metrics validates the robustness and reliability of ANN_B as a component of the ensemble framework. Table 4 presents the training result of the ANN_C.

Table 4: Training result of the ANN_C

Epoch	Accuracy	Loss	Validation Accuracy	Validation Loss

1/40	0.6026	0.9814	0.9501	0.2094
2/40	0.9512	0.1921	0.9720	0.1177
3/40	0.9708	0.1164	0.9779	0.0827
4/40	0.9766	0.0859	0.9829	0.0643
5/40	0.9824	0.0663	0.9901	0.0534
6/40	0.9834	0.0583	0.9909	0.0463
7/40	0.9849	0.0501	0.9920	0.0415
8/40	0.9879	0.0433	0.9923	0.0377
9/40	0.9872	0.0403	0.9923	0.0351
10/40	0.9882	0.0381	0.9923	0.0329

Training Result of ANN_C in Table 4 demonstrates a smooth and consistent improvement in the model's learning behaviour across the 40 epochs. At the beginning of training (Epoch 1), ANN_C achieved a moderate training accuracy of 0.6026 with a relatively high training loss of 0.9814. However, its validation performance was already quite strong, with 0.9501 validation accuracy and a relatively low validation loss of 0.2094. This early strong generalization suggests that the input features (extracted from the CNN) were already highly discriminative, giving the model a head start compared to training accuracy.

From Epoch 2 to 5, the network experienced a steep rise in training accuracy, reaching 0.9824, while training loss decreased significantly to 0.0663. At the same time, validation accuracy improved from 0.9720 to 0.9901, with validation loss steadily reducing from 0.1177 to 0.0534. This phase shows that the model quickly learned to map the extracted features to the correct outputs, achieving near-optimal generalization within just a few epochs.

Between Epochs 6 and 15, ANN_C continued to refine its learning. Training accuracy stabilized above 0.98, with training loss reducing gradually to around 0.0278. Validation accuracy during this stage consistently remained above 0.990, while validation loss is around 0.026–0.030, reflecting strong and balanced learning. Importantly, there were no signs of overfitting, as the validation metrics closely tracked the improvements in training performance.

From Epoch 16 to 40, the model converged toward its optimal performance. Training accuracy reached 0.9938 (Epoch 39) with a minimal loss of 0.0176, while validation accuracy remained stable around 0.991–0.992, with a validation loss as low as 0.0210. This plateau suggests that the network had fully exploited the discriminative power of the extracted CNN features and achieved peak performance. The minor fluctuations in later epochs are expected and reflect normal training dynamics rather than instability.

3.1 Result of System Integration (Vulnerability Assessment and Management System)

The system integration was carried out using the ensemble based convulnet and the decision support scoring algorithm to develop software for classification and management of vulnerability. The system operation began with login into the software as shown in Figure 3. The system login allows access to authorized users to operate the system and then test for vulnerability.

Figure 3 represents the initial login interface of the developed vulnerability management system, which was built through the integration of the ensemble-based ConVulNet model and the decision support scoring algorithm. This login stage serves as the entry point into the software, ensuring that only authorized users can access the system for classification and management of vulnerabilities. The inclusion of a login mechanism is crucial not only for security and user authentication, but also for tracking user activities and maintaining the

integrity of vulnerability reports within the system. Once the user successfully logs in, the backend ensemble ConVulNet model becomes active, processing inputs and performing vulnerability classification with very high accuracy. The decision support scoring algorithm then enhances this process by ranking or prioritizing detected vulnerabilities based on their severity and impact, thereby providing actionable intelligence to the user. Figure 4 presents the accessed home interface to load test CVE data.

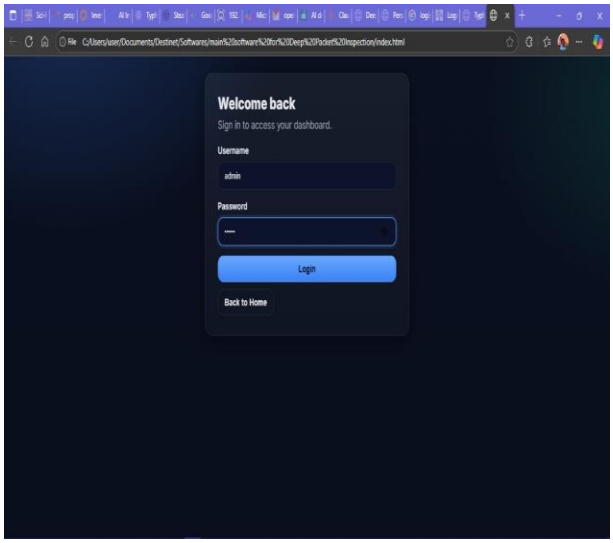

Figure 3: Result of the login interface

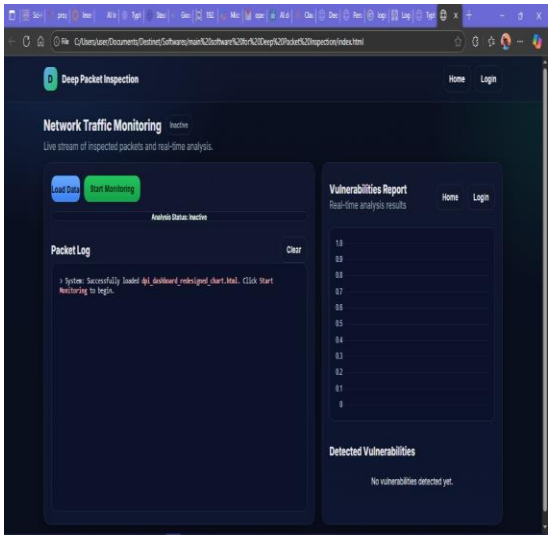

Figure 4: Interface to load test CVE data

Figure 4 illustrates the interface designed to upload and load CVE data from the testbed into the developed system. This interface acts as the bridge between the raw testbed dataset and the intelligent classification engine powered by the ensemble-based ConVulNet and the decision support scoring algorithm. By allowing users to load test CVE files directly, the system provides a practical and flexible means of feeding real-world vulnerability information into the model for analysis. One the data is loaded; the vulnerability management system begin classification through packet log analysis as reported in Figure 5.

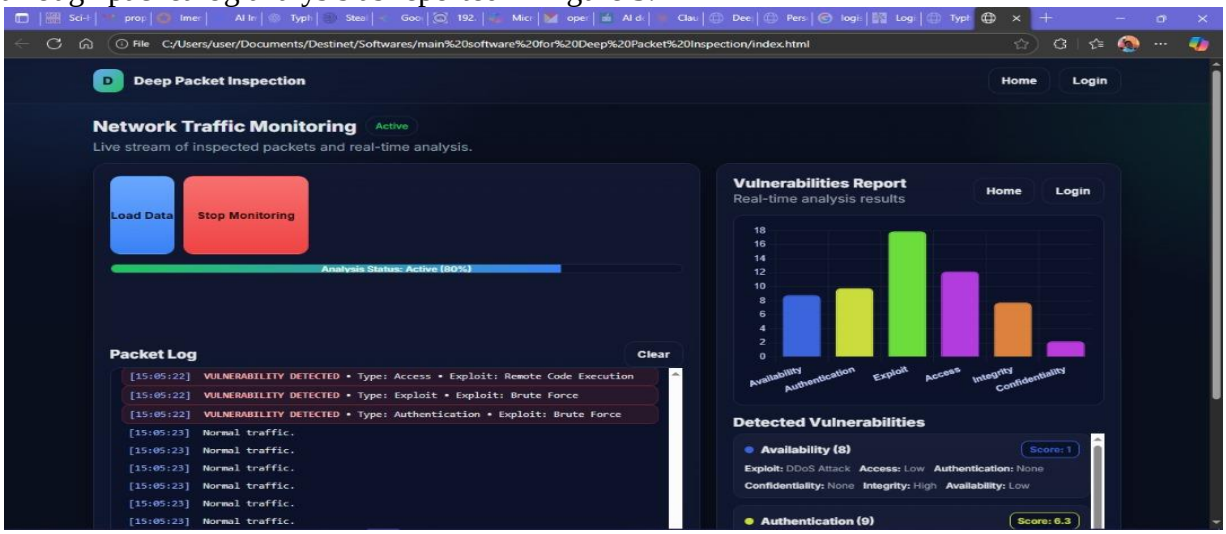

Figure 5: Packet log analysis and vulnerability reported output

Figure 5 provides a structured output that highlights not only the detected vulnerabilities but also their corresponding classifications and severity levels as determined by the decision support

scoring algorithm. It reflects the operational efficiency of the system as a real-time vulnerability management tool. Rather than leaving packet data in a raw, unreadable form, the system performs automated log analysis and translates it into a meaningful vulnerability report. This ensures that administrators or cybersecurity teams can quickly interpret the findings, decide on mitigation measures, and integrate the results into broader risk management strategies.

Figure 6 presents the summary of the deep packet inspection results, which is the final analytical stage of the system after raw packet logs have been processed and vulnerabilities identified. It provides a holistic overview of the inspection outcomes by aggregating the results into summarized categories that highlight the broader security posture of the testbed environment. Through the DPI process, the ensemble-based ConVulNet model examines the payload content, flow structures, and protocol behaviours of each packet, allowing the system to detect both known and potentially hidden vulnerabilities. The summary output in Figure 7 consolidates these findings into classified groups, such as normal traffic, suspicious traffic, and confirmed vulnerability instances, often accompanied by severity ratings and frequency counts.

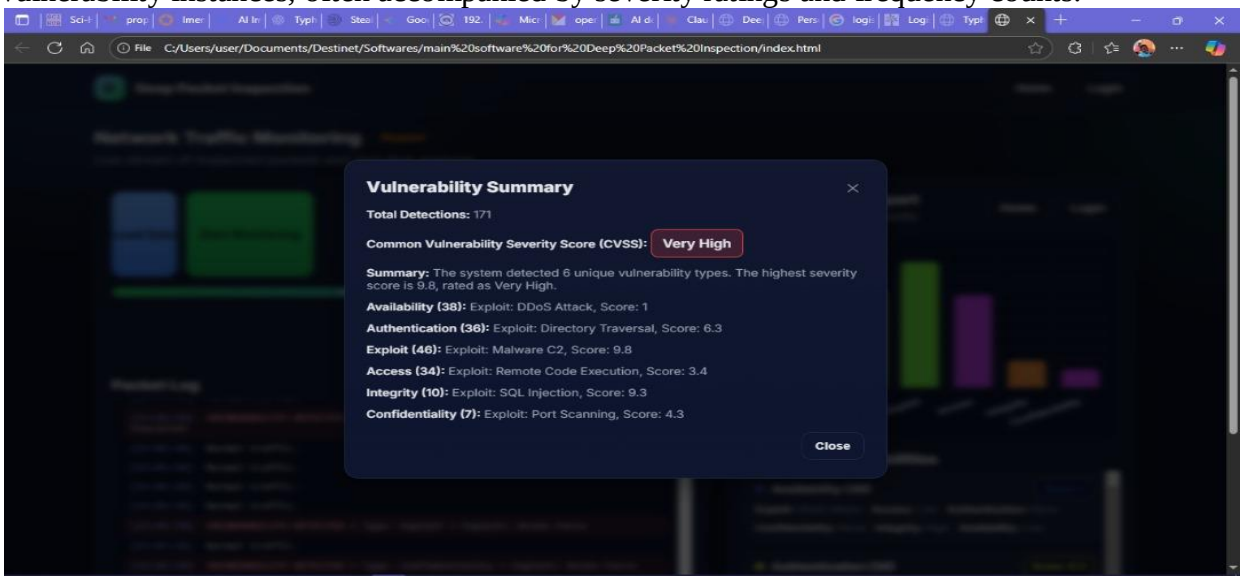

Figure 6: Summary of deep packet inspection results

Overall, Figure 6 demonstrates how the system moves from micro-level packet analysis to macro-level security evaluation, ensuring that the vulnerability management framework is not only precise in detection but also comprehensive in providing aclear operational snapshot of the network's risk exposure.

4. CONCLUSION

The research set out to design and implements a deep packet inspection model for vulnerability management in healthcare IoT infrastructures. The increasing digitization of healthcare delivery has created an urgent need for effective vulnerability detection, prioritization, and mitigation strategies, as healthcare IoT systems remain highly attractive targets for cyberattacks due to their interconnected nature and sensitivity of data.This work proposed and implemented a hybrid deep learning framework, the convolutional VulNet, which combined convolutional neural networks for packet-level feature extraction with ensemble neural networks for classification and prioritization of vulnerabilities. The integration of a decision support scoring mechanism ensured that detected vulnerabilities were not only identified but also evaluated in terms of their severity

and potential impact on the healthcare environment. This bridged the gap between detection and actionable decision-making, which has often been a limitation of existing solutions.

The experimental implementation of the study reported that through extensive training, testing, and validation experiments. The model achieved high accuracy (99.42%), low loss (0.0165), and consistent results across repeated trials as shown in Figures 5.11–5.16. The gauge repeatability test confirmed the stability and reproducibility of the system outputs. Comparisons with baseline models further validated the superiority of the ensemble CNN-ANN architecture. The developed software system, implemented with Python and Visual Studio .NET, demonstrated its capability to perform real-time vulnerability monitoring, deep packet inspection, and log analysis. Experimental results showed that the system consistently achieved high detection accuracy, reduced false positives, and maintained stability across multiple validation tests.

REFERENCES

Ahanger, T. A., Aljumah, A., &Atiquzzaman, M. (2022). State-of-the-art survey of artificial intelligent techniques for IoT security. Computer Networks, 206, Article 108771. https://doi.org/10.1016/j.comnet.2022.108771

Anand, P., Singh, Y., Selwal, A., Alazab, M., Tanwar, S., & Kumar, N. (2020). IoT vulnerability assessment for sustainable computing: Threats, current solutions, and open challenges. IEEE Access, 8, 168825–168853. https://doi.org/10.1109/ACCESS.2020.3022842

Arampatzis, A. (2022, December 29). Top 10 vulnerabilities that make IoT devices insecure. Venafi. https://venafi.com/blog/top-10-vulnerabilities-make-iot-devices-insecure/

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

Madanian, S., Chinbat, T., Subasinghage, M., Airehrour, D., Hassandoust, F., &Yongchareon, S. (2024). Health IoT threats: Survey of risks and vulnerabilities. Future Internet, 16(11), Article 389. https://doi.org/10.3390/fi16110389

Naeem, H., &Alalfi, M. H. (2020). Identifying vulnerable IoT applications using deep learning. In 2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER) (pp. 582–586). IEEE. https://doi.org/10.1109/SANER48275.2020.9054817

Neshenko, N., Bou-Harb, E., Crichigno, J., Kaddoum, G., & Ghani, N. (2019). Demystifying IoT security: An exhaustive survey on IoT vulnerabilities and a first empirical look on Internet-scale IoT exploitations. IEEE Communications Surveys & Tutorials, 21(3), 2702–2733. https://doi.org/10.1109/COMST.2019.2910750

Nikolaou, N., Papadakis, A., Psychogyios, K., & Zahariadis, T. (2023). Vulnerability identification and assessment for critical infrastructures in the energy sector. Electronics, 12(14), Article 3185. https://doi.org/10.3390/electronics12143185

Ntafloukas, K., McCrum, D. P., & Pasquale, L. (2022). A cyber-physical risk assessment approach for Internet of things enabled transportation infrastructure. Applied Sciences, 12(18), Article 9241. https://doi.org/10.3390/app12189241

Ray, P. P. (2018). A survey on Internet of Things architectures. Journal of King Saud University - Computer and Information Sciences, 30(3), 291–319. https://doi.org/10.1016/j.jksuci.2016.10.003

Wenguang, S., Mykola, B., Przystupa, K., Beshley, H., Kochan, O., Pryslupskyi, A., Pieniak, D., & Su, J. (2020). A software deep packet inspection system for network traffic analysis and anomaly detection. Sensors, 20(3), Article 642.

Yu, M., Zhuge, J., Cao, M., Shi, Z., & Jiang, L. (2020). A survey of security vulnerability analysis, discovery, detection, and mitigation on IoT devices. Future Internet, 12(2), Article 27. https://doi.org/10.3390/fi12020027