[1*] Ya Ma
[1] Chunqiang Li
[1] Yongfeng Wang
[1] Yu Wang

An Analysis of Network Protocol Vulnerability Mining Using Fuzz Testing Combined with Deep Learning Models

Journal of Electrical Systems

Abstract: - The protection of industrialized management systems and related network protocols is guaranteed by vulnerability mining innovation. The inadequate receiving efficiency and insufficient vulnerability mining capacity of vulnerability mining strategies are their drawbacks. So, this study analyzes the network protocol vulnerability mining using fuzz testing combined with deep learning (DL). In this study, Modbus TCP is employed as a network protocol regarding vulnerability mining. This paper presents a unique threshold-sample-driven deep neural network (T-s DNN) framework. Based on the TSDNN, we construct a fuzzing framework (T-s DNN Fuzzer) for Modbus TCP protocols. The DNN algorithm is first trained to understand the meaning of the protocol's data unit using this framework. The likelihood distribution of every value in the information is quantified using the softmax mechanism. The technique then examines the highest likelihood and the random variable's threshold in deciding whether to use the information value with the optimal likelihood in place of the existing information value. The MBAP header has been finished by the protocol standard. Fuzz tests demonstrate that in addition to increasing sample receipt levels and exploitability, fuzzing devices can identify protocol vulnerabilities rapidly. Experiments conducted with the T-s DNN fuzzer demonstrate that it can detect industrial control protocol vulnerabilities greater in addition to increasing test case reception scores and exploitability.

Keywords: Network protocol, TCP, vulnerability mining, deep learning (DL), fuzz test, threshold-sample driven deep neural network (T-s DNN)

I. INTRODUCTION

Modern communication systems are based on network protocols, which allow devices to transmit data across networks with ease [1]. But as networks develop more intricate and associated, they also open themselves up to other security risks. One of the most crucial parts of information systems and information products is software. Software vulnerabilities are increasingly the factors that directly impact information system security [2]. It has been demonstrated that a majority of information security events are started by attackers who take advantage of software vulnerabilities. These kinds of instances have been worse over the past few years. As a result, identifying network components that are susceptible and require more investigation to determine their degree of risk and if required, rapidly assigning suitable patches was a growing and significant problem for software programmers [3].

A. Detection of Vulnerability

A vulnerability is characterized as an issue that might be exploited by a threat source in an information system, system security protocols, internal controls, or implementation, whereas a fault or bug refers to a systemic imperfection that might (or might not) result in a vulnerability [4]. As a result, software faults that can be utilized maliciously are categorized as vulnerabilities. Because vulnerabilities are often ignored by users or programmers during regular system operation, vulnerabilities need a whole different strategy to identification than flaws, which can be found more easily and naturally [5]. Compared to usual faults, these make tackling vulnerabilities considerably more difficult.

B. Comparison of Static and Dynamic Evaluation

The two conventional methods for identifying vulnerabilities are (a) static evaluation and (b) dynamic evaluation [6]. Static analysis examined for vulnerabilities in the code without executing it. Consequently, during evaluation, the possible influence of the executable environment that was, the hardware and operating system was not considered [7]. To test whether the network will function in a run-time context, dynamic evaluation involves executing code. However, they can only make assumptions about the observable implementation routes, not all potential programming pathways [8]. Therefore, on their own, static and dynamic code evaluations both have certain issues.

[1] Zunyi Cigarette Factory of China Tobacco Guizhou Industrial Co., Ltd. Zunyi, 63000, China
Corresponding author: Ya Ma, maya_zunyi@163.com

C. Challenges for Detection of Vulnerability

The constant threat scenery, resource-intensive nature, absence of thorough specifications and complexity of network protocols make mining vulnerabilities in them difficult [9]. Complicated relationships and lacking requirements can enable such vulnerabilities to pass undetected. New hazards are brought forth by a shifting cyber threat landscape, which makes vulnerability mining strategies more dynamic and demands constant monitoring. Accessibility was restricted for academics or lesser numbers due to the resource-intensive characteristics. These issues highlight the significance of constant improvement and teamwork to boost network security [10].

D. Contribution of the Study

The following are the article's primary contributions:

1. The article addresses the shortcomings of conventional vulnerability mining systems, which contain low capacity for vulnerability mining and low receiving efficiency.
2. We proposed a threshold-sample-driven deep neural network (T-s DNN) architecture that is designed specifically for evaluating vulnerabilities in network protocols, with an emphasis on Modbus TCP.
3. The algorithm evaluates a threshold for a random parameter along with its chances of the maximum probability. Its evaluation assists in determining whether to replace existing information values with ones that offer the highest probability, thus boosting vulnerability mining efficacy.
4. Results from experiments show that the Ts DNN fuzzer increases test case variation scores, running duration and number of bugs while improving industrial control protocol detection of vulnerabilities significantly.

The rest of the article is arranged as follows. To provide a brief overview of many relevant vulnerability assessment and management strategies in section 2. The background information for the suggested method is provided in section 3. Section 4, provides a series of comparative experimental evaluations and our vulnerability prediction methodology. Section 5 concludes with an assessment of the network's future.

II. LITERATURE REVIEW

The article [11] evaluated the effectiveness of Machine learning (ML) and Data Mining (DM) algorithms in assessing the accuracy of vulnerability prediction in computer security, specifically in identifying and mitigating computer software flaws. Potential tactics were identified via the literature assessment, but it also brought attention to problems including database size, misconceptions and changing assault vectors. The detection of phishing with ML techniques; the results showed that neural networks performed best in [12]. The IP address inclusiveness, URL width and encryption key validity were found to be important aspects. However, to counteract phishing assaults as they evolve, modifications to models regularly and user knowledge were crucial. The protocol analysis developed [13] a variety of scenario tests and created a fuzzy testing vulnerability mining approach to improve network security. The investigation [14] investigated using neural systems and DL techniques for software vulnerability detection, emphasizing the latter were capacity to comprehend code semantics and spot susceptible patterns. As this field developed, research was yet needed to address issues like the complexity of models and data scarcity; even with encouraging gains. The DL approach to software vulnerability identification was examined in [15], which discovered that the obstacles faced by traditional approaches include data duplication and erroneous class distribution. Considerable increases in recall and precision were demonstrated by suggested improvements based on practical scenarios, underscoring the necessity of more ethical data gathering and model building in DL-based vulnerability forecasting research. Table 1 presents previous researcher's studies on network protocol vulnerability.

Table 1: Previous Researcher's Studies for Network Protocol Vulnerability

Reference	Objective	Findings	Limitation
[16]	Decawatt Package (D-PACK) especially targets susceptible Internet of Things (IoT) devices, such as IP cameras, to strengthen defensive systems versus large-scale DDoS assaults by effectively filtering aberrant data at an early stage.	By examining only, the initial two messages of each flow, D-PACK proves effective in achieving approximately the highest accuracy with a low false-positive rate, based on the experimental data. The strategy lowers the processing load and makes it possible to stop harmful flows in real-time.	D-PACK's dependence on pre-defined characteristics that are taken from the original packets presents a drawback since it might miss minute irregularities or change attack tactics.
[17]	The article efficiently detected defects in binary applications by employing an adaptive fuzzing framework known as V-Fuzz. The framework prioritizes code	An evolutionary technique was used by V-Fuzz to target locations of software problems once a vulnerability detection strategy has identified them. Its efficacy was shown by the fact that it	The precision of the vulnerability forecasting algorithm and the intricacy of the two-dimensional applications under test

	parts that are susceptible using a flaw assessment algorithm.	effectively identified three new and 10 common Vulnerabilities and Exposures (CVEs).	could have an impact on the efficacy of V-Fuzz performs.
[18]	The study addressed the deficiency of organized knowledge concerning the utilization of DL in C/C++ software vulnerability identification. The goal of Syntax-based, Semantics-based and Vector Representations (SySeVR) was to offer a structured approach for acquiring program representations that are capable of efficiently detecting vulnerabilities.	Four software product tests show the benefit of the framework, fifteen vulnerabilities that had not been disclosed before were found, seven of which were unidentified and disclosed to vendors.	One potential drawback of SySeVR might be its reliance on the caliber and variety of the training data, which could impair the algorithm's ability to effectively generalize to new vulnerabilities.
[19]	Bidirectional Graph Neural Network for Vulnerability Detecting (BGNN4VD), a vulnerability recognition technique that builds a Bidirectional Graph Network (BGNN) to overcome the shortcomings of the DL-based vulnerability detection techniques presently in operation.	Based on experiment data, BGNN4VD improved F1-measure, accuracy and precision by 4.9%, 11.0% and 8.4%, respectively, when compared to these baselines.	The use of vectorized source code visualizations, which could not adequately capture subtle weaknesses in intricate code structures, was BGNN4VD's primary drawback.
[20]	They create Formal Fuzzer, an emulation-based hybrids system that integrates fuzz testing and formal validation to effectively identify known and new vulnerabilities in SoCs, overcoming the drawbacks of conventional verification methods.	Formal Fuzzer reduces the space for fuzzing tests by using template-based assertion generation and formal-verification-based pre-processing. By selecting mutation algorithms using feedback from security-oriented cost functions, it effectively finds vulnerabilities in SoCs.	The precision of the vulnerabilities samples and requirements utilized for determining the cost function could have an impact on the efficacy of Formal Fuzzer performs.
[21]	They build an ML-based approach that can anticipate software vulnerabilities from the source code before they become accessible, to improve software security and avert possible system damages.	They developed a way for expressing source code that uses ML to intelligently interpret Abstract Syntax Tree (AST) representations of source code. Tests conducted on a publicly accessible set of function-level program segments show the efficacy it can be in comparison to other solutions.	The possible bias or restrictions in the data set with labels might be one of the research's shortcomings, impacting the extent to which the findings can be applied.
[22]	A thorough analysis of SS7 assaults, they included information on attack techniques, sites of entry within the SS7 core system, and recommended defenses.	The study presented an anomaly detection technique for SS7 systems that was based on ML and suggested defenses against these threats. The outcomes demonstrate that when it involves anomaly detection and enhanced network security, the ML structure works better than rule-based filtering.	The intricacy and dynamic character of SS7 assaults could be a study constraint, as it could provide difficulties in creating all-encompassing defenses and detection techniques.
[23]	To increase the effectiveness of bug identification, the research developed Suzzer, a vulnerability-guided fuzzer, with an emphasis on examining code blocks that are more likely to have vulnerabilities.	Suzzer was a lightweight static the detector that effectively targets susceptible code blocks and enhances bug discovery performance by extracting Abstract Control Flow Graph (ACFG) channels from target applications in 64.5% less time than VUzzer.	The complexity and scale of the target software could have an impact on Suzzer's effectiveness.

A. *Research Gap*

It is difficult for ML, DM, and DL techniques to anticipate computer security vulnerabilities with sufficient accuracy, particularly when attack vectors change frequently and available data sets are limited. While conventional techniques involving vulnerability tools for evaluation, fuzz evaluation and official verification have demonstrated potential, they also have drawbacks, including false positives, dependence on pre-defined features and scaling problems. The article develops and implement innovative techniques like T-s DNN fuzzer that have been identified in recent research to address these issues. These solutions increase the accuracy of vulnerability identification, lower false positives and strengthen system safety in the face of dynamic threats by utilizing advanced algorithms, hybrid structures and improved data representations.

III. PROPOSED METHODOLOGY

Modbus TCP/IP, often known as Modbus TCP (Transmission Control Protocol), is essentially the Modbus RTU (Remote Terminal Unit) protocol with an Ethernet-based TCP interface. The software protocol that establishes standards for organizing and interpreting data regardless of the data transmission channel is called the Modbus message structure. The Internet Protocol and Transmission Control Protocol, or IP/TCP, serve as the Modbus TCP/IP messaging's communication channel. TCP/IP facilitates the transfer of binary data blocks between computers. It is an international standard that forms the basis of the World Wide Web. Separate inputs, input registers, Coils and holding registers are the four memories that make up the Modbus slave framework's approach. A pattern of reading and writing of these memories, either via the physical procedures themselves or by remote requests sent by the Modbus master, can be used to mimic the control loops and reporting. The seven-byte header of the Modbus TCP/IP Application Unit (ADU) includes the protocol and transaction identifiers, protocol data unit, length field and unit identifier which is made up of function data and code. The ADU is transmitted to system port 502 using TCP by embedding it within the data area of a typical TCP frame. All clients and servers that use Modbus receive and attend for information over this port, which is only meant for use with Modbus applications. The Modbus Application Protocol (MBAP) consists of a one-byte unit identification (set to $0xFF$, which is similar to the slave tackle in the serial model of Modbus), a 2-byte transaction identifier, a two-byte protocol recognition (set to $0x0000$ for Modbus) and a two-byte length field that indicates the number of subsequent bytes. A slave could be corresponding with several masters in Modbus/TCP, and a master can have several pending transactions with slaves. Figure 1 shows the Modbus frames in TCP segments.

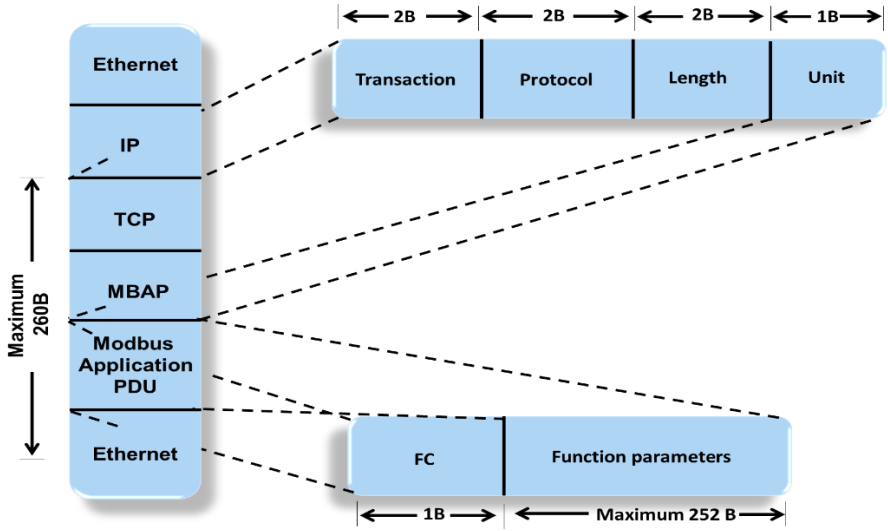

Figure 1: Modbus frames in TCP segments

A. *Distribution of Probabilities using deep neural network (DNN)*

The deep neural network classifier is built in this stage. A deep neural network (DNN), is a type of neural network that descended from the traditional artificial neural network. The architecture of a DNN is shown in Figure 2. The three layers that make up a DNN are the output, input, hidden and SoftMax layers.

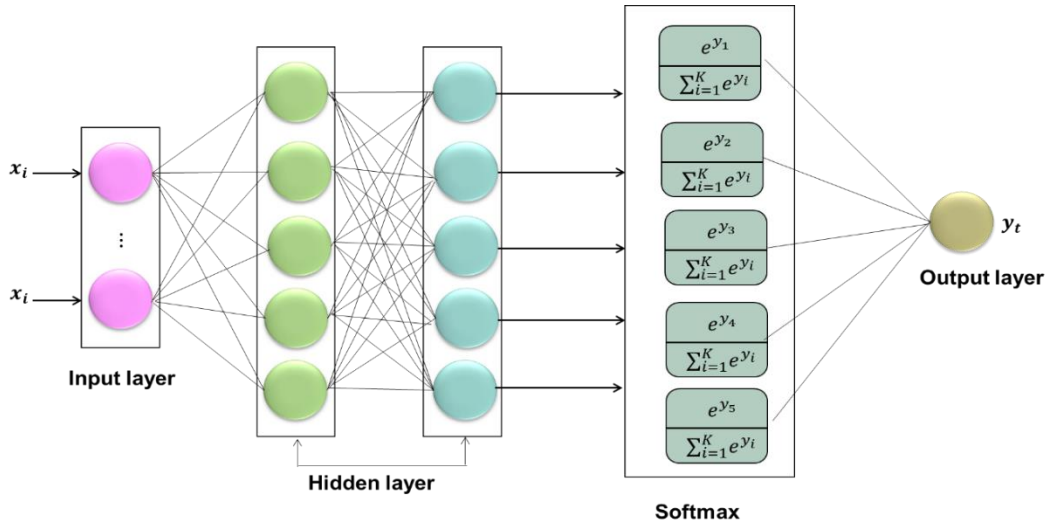

Figure 2: Structure of DNN

The pre-processed input information is fed to the network via the input layer. The quantity of the Information collected by a network of neurons is equal to the number of characters being input in the sample. Equation (1) presents the expression for the input layer that has C inputs.

$$W = [w_1, w_2, \ldots., w_C] \tag{1}$$

Since the DL network supports the addition of various hidden layers, the hidden layer comes next. The input layer's input W is mapped by the hidden layer using bias (a_i) and random weights (w_j). Hence, inputs from the hidden layer are represented in equation (2).

$$g_i = \sum_j x_j w_j + a_i \tag{2}$$

Where$i$-the total number of DNN's hidden units$i = 1, 2, 3, \ldots, L$.There is an associated nonlinear activation function with every hidden layer. The neurons perform better than Softmax and hyperbolic tangent neurons, despite their high nonlinear dynamics and Discontinuity at 0. As a finding, producing real zeros in sparse representations yields excellent outcomes suitable for data that is limited. For 100 epochs of binary categorization both the sigmoid and tanh activation functions exhibit respectable detection rates compared to Softmax. Softmax beats out sigmoid and tanh activation operates when the investigation is run for 500 epochs. We thus select to use Softmax activation functions in the implementation of our suggested paradigm. Equation (3) expresses the hidden layer's outcomes.

$$g = e(g_i) \tag{3}$$

The hidden layer's inputs are processed by the output layer's stimulation function, which generates the outputs of DNNs. The output layer's nonlinear activation function, Softmax, which adapts inputs towards a class of likelihoods, is used for vulnerability detection, $\sigma(W)_i$. The DNN's output is represented in equation (4).

$$\sigma(W)_i = \frac{f^{W_i}}{\sum_{l=1}^{l} f^{W_l}} \#(4)$$

Where i is the total amount of output units, $i = 1, 2, \ldots, L$, and w represents a vector of input to the output layer. Using this DNN authentication, the inputs to each class output, example for normal 1 and abnormal 0 are used to execute the network training. To decrease training errors, the large training sample is employed to train DNN and each input connection's weight is adjusted repeatedly in Vulnerability detection. For the network to be trained with greater speed and effectiveness, the DNN's model parameters are adjusted. During training with a learning method, these tuning parameters also referred to hyper parameters are utilized to regulate softmax optimization techniques and approach selection. These hyperparameters determine whether the strategy overfits throughout the learning phase.

SoftMax function is one potential DNN method that approaches the issue as a multiclass vulnerability estimation issue in which the user's inquiry is the input. The model transfers the final layer's $\varphi(w)$ output to a probability distribution $\hat{P} = g(\varphi(w)U^S)$ via a SoftMax layer.

$G: \mathbb{Q}^n \to \mathbb{Q}^n$ - threshold Softmax function is represented as $G(z)_j = \frac{f^{z_j}}{\sum_i f^{z_i}}$, $U \in \mathbb{Q}^{m*c}$ - threshold SoftMax layer's weight matrix.

A vector of scores, commonly referred as logits, $Z \in \mathbb{Q}^m$ is mapped to a probability distribution using the threshold layer known as threshold soft-max, is represented in Figure 3.

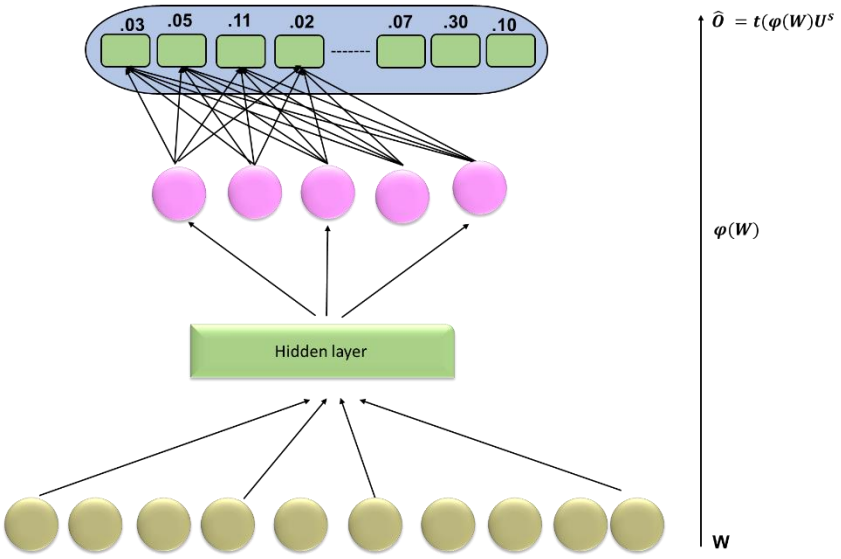

Figure 3: Predicted Probability Distribution

A loss function that contrasts the subsequent threshold values. $\hat{P}$, which is a probability distribution reflecting the Softmax layer's output and P which is the ground truth and represents the objects the user has interacted with (e.g., selected or watched videos). A probability vector, or a normalized multi-hot distribution, can be used to illustrate in equation (5).

$$\hat{P} = T(\varphi(w)U^S) \tag{5}$$

The probability that item i shows in $\hat{P}_i = \frac{\exp(<\varphi(w)U_i>)}{Y}$, where Y is an independent normalization constant based on i. In another way, $log(\hat{P}_i) =< \varphi(w), U_i > -\log(Y)$, meaning that the proportional likelihood of item i is the dot product of two d-dimensional vectors, which are equivalent to queries and item-embedded data, up to an additive constant.

$\varphi(w) \in \mathbb{Q}^c$- greatest hidden layer's output. It refers to the query w's embedding. $U_i \in \mathbb{Q}^c$- weight vector that links output$i$ to the final hidden layer. This is referred as the item i's embedding. Equation (6) is used to determine the probability output P_{u_j} (provided by equation (6)) of the DNN for every value of the information u_j in trajectory u. More probability indicates more accurate semantic training and a larger likelihood of the target's output result. Equation (7) demonstrates the distribution of probability guidelines for information values and all probabilities make up the output distribution of probability measures O at a period of s.

$$0 \le P_{u_j} = P(\hat{O}_s = u_j) = P_{u_j}(\hat{O}_s | \{W_1, \hat{O}_1, \dots, \hat{O}_{s-1}\}| \le 1 \tag{6}$$

$$O = (P_0, P_1, \dots, P_{255}, P_{256}) \tag{7}$$

B. Threshold-sample Approach for Industrial Protocol Vulnerability Detection

The threshold-sample approach-based test instance generation technique for Modbus TCP maximizes vulnerability identification by utilizing random threshold variables and distribution of probability measures. The threshold sample space Ω^s(equation (8)) is made with all potential output outcomes of the resultant value $\hat{O}_s$at period t, where wit comprises an essential occurrence. Based on Equation (9), the output data value while the fundamental event occurs is $\hat{O}_{s,j}$. The possibility that the data value $\hat{O}_{s,j}$equals $\hat{O}_s$ is denoted by$P(\hat{O}_s = e(x_j^s)$.

$$\Omega^s = \{x_0^s, \cdots, x_j^s, \cdots, x_{256}^s\} \tag{8}$$

$$\hat{O}_{s,j} = e(x_j^s) = j \tag{9}$$

The Softmax layer's output result is the chance that the projected threshold value vulnerability is within the range of 257 information values. The highest likelihood is used to indicate the greatest level of DNN learning semantics of message correctness and the statistics value that corresponds to the highest likelihood is employed to forecast the input message's information value. By representing the constraints of data values in industrial automation instrument records using a distribution of probabilities, the threshold sample approach generates the test cases by using the information cost with the smallest likelihood as the output cost, thus exploiting protocol vulnerability. Initially, DNN employs a Softmax layer to produce the supply of probabilities within the present period based on the input data that is currently available, and the information is hidden and included in the message

that was learned. The probability of the learned message's semantic correctness is expressed by the Softmax layer. Subsequently, the technique produces a random threshold variable that is not constant for every probability matrix produced by DNN. Finally, a comparison is made among the maximum probability and the threshold variable. The information's value was formed by standard message semantics at that precise instant based on the comparison result, which is to increase the probability that an anomalous event would occur in the industrial hierarchy of authority. The random threshold variable is between 0 and 1 while all the possible values in the distribution of probability are also among 0 and 1. The generation strategy which generates test cases using the threshold sample approach is simplified.

The threshold data is the Modbus TCP mining vulnerability fuzzer built with the threshold sample approach. In the probability matrix, P_i stands for the greatest and minimum probabilities. The random threshold variable is DP_{fuzz}, the distribution of probability measures of the resultζ is O, and ζ is the DNN strategy in equations (10) to (12).

$$P_i = P\left(\hat{O}_{s,j} = e\left(x_j^s\right)\right) = \max_{\xi}(O) \tag{10}$$

$$P_i = P\left(\hat{O}_{s,j} = e\left(x_j^s\right)\right) = \max_{\xi}(O) \tag{11}$$

$$DP_{fuzz} = random(0,1) \tag{12}$$

When P_i is larger than or equal to DP_{fuzz} in the threshold sample-generating method, it means that the information values accessible by the register have defined uniformity and often exist in a specific location. Performance in the exact opposite manner as a result at this moment, a message is generated by substituting the data value of the smallest possibility for the information value of the highest likelihood. Although these test scenarios are rarely employed in the industrial control apparatus, such as an industrial management analyzer communication that has the potential to seriously compromise protocol security. For instance, to read the moisture value in the industrial management analyzer, access the register at location $0x002B$. This moisture value is represented by the information value T_s^+, demonstrated in Equation (13). If T_s^- (as given in equation (14) is chosen as the moisture element at this point, the protocol can become vulnerable even though like test scenarios are seldom employed in vulnerability detection.

$$T_s^+ = e\left(x_j^s\right) = arg\max_{\xi}(O) \tag{13}$$

$$T_s^- = e\left(x_j^s\right) = arg\max_{\xi}(O) \tag{14}$$

It doesn't need to alter the register's access mode when P_i is smaller than DP_{fuzz}since this suggests that the information threshold value employed by the register lacks some clear authority. For instance, to recite the point of dew threshold value in the industrial management analyzer, access the register at location $0x002C$. Usually, the data threshold elementT_s^+ is the dew point threshold value. In these situations, using T_s^+as the dew point threshold value for vulnerability detection outcome in anomalous or nonexistent responses from industrial mechanisms is for control.

A complete test scenario requires both MBAP and PDU due to the relationships between the threshold variables in the protocol message. Since the message created by the strategy for generation needs to be used to include the header message portion of MBAP, it is important to create a standard test case and then monitor the status of the object that is tested. The Modbus TCP message's PDU component is created using the model's generation strategy, but the MBAP message header must be generated at random using the Modbus TCP protocol. The PDU and MBAP header are combined in this manner to create a comprehensive validation scenario for the vulnerability detection framework for the industrial management protocol.

C. Framework for Test Case Generation in Industrial Control Protocols

The learning phase and the case generation phase make up the two sections of the framework used in this study to generate test cases. The learning step is used to ascertain the DNN's weight parameters and learn the training dataset's message. Protocol evaluation and generation approach comprise the case generation phase. Considering that TCP packet learning guidelines, DNN can ascertain the probability distribution connection of the subsequent produced field when a new field is provided in protocol analysis. For generating the subsequent protocol field, the generation method must modify the data associated with its probabilistic connection. Figure 4 depicts the protocol for industrial control.

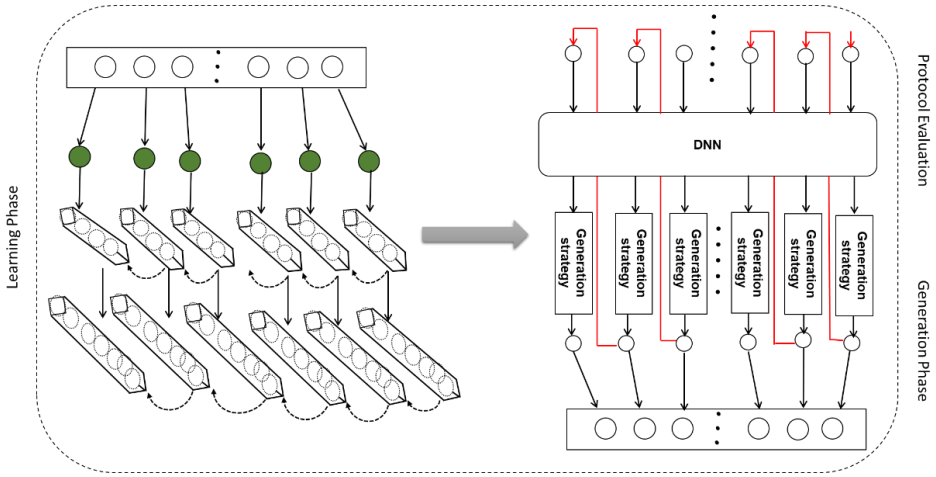

Figure 4: Protocol for Industrial Control

1) Learning Protocol Message Generation

The Function Data and Code of every protocol were chosen for every Modbus TCP message. Depending on the low-byte and high-byte data, the 2-byte communication areas are separated between low-byte and high-byte segments. Each byte's hexadecimal information points are transformed into decimal points. A function that translates hexadecimal to decimal is represented by D. Since a byte's hexadecimal data range is $0x00$ to $0xFF$, each data value's value range in decimal is $0-255$ and all of them are integers after the byte is converted. Assume that there are no messages of TCP and that the total length of all messages is m. Initially, the message is retrieved using this format and each message terminates with the data value 256 for the termination flag. Each data item has a value range of $0-256$, which is an integer. Therefore, $m - num \cdot 7 + num \cdot 1$ is the total acquired message length, with 7, where 1 is the length of the enhanced termination signal value and the MBAP message's header byte. The sequence of the fields in each message is preserved throughout training when combining every message to generate the huge matrix T since the DNN model's input data matrix has the same size. During network learning, each interval of m information amounts, beginning with the initial information amount, symbolizes a message input of T. Similarly, every interval of m information represents an output of the message through the training phase, beginning with the second data value. Consequently, $\{T_j, \ldots, T_{j+n}$is the model's j^{th} input message., and the message that appears on the j^{th} output is $\{T_{j+1}, \ldots, T_{j+n+1}\}$, where there are $\lfloor (n - num \cdot 7 + num \cdot 1 - 1)/n \rfloor$ messages overall and each message has a length of n. The j^{th} message is represented by the subscript j. Based on equations (15) and (16), the matrix made up of the set of input messages in the TCP instruction package is w, where w_j stands for the j^{th}input message.

$$W = \left[W_0, W_1, \ldots, W_j, \ldots, W_{\left\lfloor num - \frac{1}{n} \right\rfloor}\right] \tag{15}$$

$$W_j = \left[T_j, T_{j+1}, \ldots, T_{j+n-1},\right] \tag{16}$$

Equations (17) and (18) illustrate that the intended output message is represented by the matrix Z. Z_j stands for the i^{th} message that should be produced.

$$\mathrm{Z} = \left[\mathrm{Z}_0, \mathrm{Z}_1, \ldots, \mathrm{Z}_j, \ldots, \mathrm{Z}_{\left\lfloor num - \frac{1}{n} \right\rfloor}\right] \tag{17}$$

$$\mathrm{Z}_j = \left\{T_{j+1}, T_{j+2}, \ldots, T_{j+1+n-1}\right\} \tag{18}$$

During the phase of learning, the supply strategy for determining the protocol information threshold point is designed using a two-layer DNN. The message of input at every period is w_j, the predicted message output is Z and the message input is W based on the time series expansion.

During the learning phase of this study, Z, the predicted output, is transformed into $\mathrm{Z}_{[j]}$, the polynomial supply matrix of single-hot encoding, representing the distribution of probability θ of the actual result $\hat{\mathrm{Z}}_j$. The distance among them is then described by a value and the total duration of every message is obtained to form the learned loss function, which is provided in equation (19).

$$loss(\mathrm{Z}, \hat{\mathrm{Z}}) = -\frac{\sum_{j=0}^{\left\lfloor (n-num-7+num1-\frac{1}{n} \right\rfloor} \mathrm{Z}_{[j]} \times log \mathrm{Z}_{[j]}}{\left\lfloor (n - num.7 + num1 - \frac{1}{n} \right\rfloor} \tag{19}$$

2) Protocol Evaluation with DNN Distribution of Probability

The primary components of the generation model, the case generation strategy and protocol evaluation, are included in the case generation stage. The DNN model's internal weighted parameters for the network are established following the learning phase. The information value of the j^{th}test case at time s is represented by the superscript (s) when the j^{th}test case is generated. The threshold softmax function generates the distribution of probability connection measures for each period in the protocol evaluation. $w_j^{[0]}$ is the model's starting input. Every time, the framework generates an information value that is used in the following iteration.

There is a probability output matrix for every time. The output value, θ_i, represents the probability of each of the 257 data values that are recorded in the matrix; the total probabilities added together equals one. Equations (20) to (22) reflect these probabilities, with ξ denoting a two-layer DNN network. Equation (20) is satisfied by the real output probability measures θ of ξ; Equation (21) is satisfied by the information value associated to the highest probability in the matrix, $\hat{Z}_j^{(s)+}$; and Equation (22) is satisfied by the information associated to the smaller probability in the measures $\hat{Z}_j^{(s)-}$.

$$(\theta_0, \theta_1, \cdots, \theta_{256}) = \begin{pmatrix} P\left(\hat{Z}_j^{(s)+} = 0 | \left\{W_j^{(0)}, \hat{Z}_j^{(0)}, \hat{Z}_j^{(1)-}, \cdots \hat{Z}_j^{(s-1)}\right\}\right) \\ P\left(\hat{Z}_j^{(s)+} = 1 | \left\{W_j^{(0)}, \hat{Z}_j^{(0)}, \hat{Z}_j^{(1)}, \cdots \hat{Z}_j^{(s-1)}\right\}\right) \\ \vdots \\ P\left(\hat{Z}_j^{(s)+} = 256 | \left\{W_j^{(0)}, \hat{Z}_j^{(0)}, \hat{Z}_j^{(1)-}, \cdots \hat{Z}_j^{(s-1)}\right\}\right) \end{pmatrix}^s \tag{20}$$

$$\hat{Z}_j^{(s)+} = arg \max_{\xi}(\theta) \tag{21}$$

$$\hat{Z}_j^{(s)-} = arg \max_{\xi}(\theta) \tag{22}$$

D. Threshold-Sample Approach for Test Case Generation

The phases that are specifically involved in generating the information value $\hat{Z}_j^{(i)} (i \in [0, s-1])$ every time using the threshold-sample generation approach are as follows.

1. Based on the current input ($w_j^{(0)}$or $\hat{Z}_j^{(i-1)}$and the learned semantics, the protocol analysis part's distribution of probabilities relation matrix θ is produced.

2. Every probability matrix has a threshold DP_{fuzz} that is randomly determined during generation. Equation (23) provides the highest probability θ_i in the measures, which is the result of the constructed strategy framework. If θ_i exceeds or is equivalent toDP_{fuzz}, $\hat{Z}_j^{(S)-}$is chosen as the value to be output. Alternatively,$\hat{Z}_j^{(S)+}$remains chosen to be the variable of output.

$$\theta_j = \max_{\xi}(\theta) \tag{23}$$

3. The termination signal information value of 256 is compared to the information created by the present period. To repeat the phase (1) and (2), If the process has finished, it shouldn't be the termination signal information value.

The Function data and code portions of the validation cases are created according to the dependent connection in TCP and then generate the entire test cases. Initially, a random generator creates the protocol ID (2 bytes), unit ID (1 byte), and transaction ID. Length is then determined by dividing the total amount of bytes by Unit ID, Data Function and Code into 2 hex numbers. As demonstrated by Algorithm 1, the threshold-sample algorithm obtains as input the double-layer DNN network ξ and the produced message $[\hat{Z}_j^{(0)}, \hat{Z}_j^{(1)}, \ldots, \hat{Z}_j^{(s-1)}]$ and outputs the resulting PDU message $\widehat{Z}_j$.

Algorithm 1: *Threshold-sample Approach*

$\boldsymbol{input}$: $\zeta \leftarrow the\ double-layer\ DNN\ strategy$;
$[\hat{Z}_j^{(0)}, \hat{Z}_j^{(1)}, \ldots, \hat{Z}_j^{(s-1)}] \leftarrow generated\ message$;
$\boldsymbol{output}$: $\hat{Y}_j \leftarrow the\ generated\ PDU\ message$
∇ *based on the generated message*
$[\hat{Z}_j^{(0)}, \hat{Z}_j^{(1)}, \ldots, \hat{Z}_j^{(s-1)}]$ *strategy* ζ *outputs its distribution of probability measures* θ.

1. $\boldsymbol{do} DP_{fuzz} \leftarrow random(0,1)\theta_i \leftarrow \max_{\zeta}(\theta)$
2. $\hat{Z}_j^{(s)+} \leftarrow \arg\max_{\xi}(\theta), \hat{Z}_j^{(s)-} arg\max_{\xi}(\theta)$

3. $If\ \theta_i \geq DP_{fuzz}$
4. $then\ \hat{Z}_j^{(s)} \leftarrow \hat{Z}_j^{(s)+}$
5. $else\ \hat{Z}_j^{(s)} \leftarrow \hat{Z}_j^{(s)+}$
6. $\hat{Z}_j \leftarrow \hat{Z}_j + \hat{Z}_j^{(s)}$
7. $s \leftarrow s + 1$
8. **do while** $Z_j^{(s)} != 256\ \nabla\ The\ end\ flag\ stops\ the\ loop$
9. **end while**

IV. Experimental Findings

To identify the network vulnerability in the control system equipment, this article uses the TCP as the validation target. Test cases are sent to the industrial control system as part of the Modbus TCP vulnerability detection process and the system either responds or returns an abnormal state. After that, the apparatus sends out fresh test cases and it is monitored for indications if the initial experiment resulted in an unusual reaction. The system discards the case, suggesting a protocol vulnerability, if it reacts to the test cases normally.

A. Dataset Acquisitions

This section assesses the suggested T-s DNN Fuzzer algorithm performs versus random Industrial Control Systems (ICS) models using two separate ICS datasets. Secure Water Treatment (SWaT) and Gas Pipeline (GP) in real and modeled industrial locations from which the SCADA training sample employed in this research was selected due to its variety and distinctive features. This dataset was collected from a gas pipeline system and includes a pre-processed Modbus validation frame in the Attribute Relation File Format (ARFF) to facilitate the usage of specific preprocessing methods. Eleven days of nonstop operation are included in this dataset; seven of those days were operated normally, while the other four days were attack scenarios. Attributes are gathered from communication between network ports, detectors and motors, comprise SWaT [24].

B. Experimental Setup

Secure Water Treatment (SWaT) of ICS, Siemens programmable logic controller SIMATIC $S7-300$, Modbus TCP $V2.6$ server and $Cisco\ WS-C2960-24TC-L$ switch was among the industrial control systems in this set. An $Nvidia\ GTX1050Ti\ GPU$, 64 −bit of RAM and an 8 GB operating system Windows 10 was used to install the industrial control tester. $Python$ 3.6.6, $TensorFlow-GPU$ 1.8.0, $CUDA$ 9.0 and CUDNN 6.0 served as the foundation for the learning model.

C. Test Case Diversity Analysis in T-s DNN Fuzzer

As the study's learning data, all of the messages that were in the data set with typical replies had to be filtered out and then the repetitive messages were eliminated. Eighty epochs were learned and 52,000 TCP messages in total were chosen. The generation strategy was stored every ten epochs, the training proportion was 0.001, the variable τ was 0.6 and the predicted size was 128 and 256. To maintain the capacity to generate test case diversity in this study, the initial phase was to determine the appropriate hyper parameters, τ is (0.4, 0.6, and 0.7). The validation then contrasted the instrument's reception proportion with a learning proportion (0.01, 0.001, 0.003 and 0.0003) and simulation size ((64 and 64), (128 and 256) and (256 and 256)). Figure 5 illustrated the comparison of reception rate contains 5(a) 0.4, 5(b) 0.6, and 5(c) 0.7). The reception proportion of experiments produced by various model sizes and learning proportion for ICS changed continuously and typically followed a trend of initially growing, then τ, as the parameter rose. The strategy size was 128 and 256, the learning proportion was 0.0001 and the reception proportion reached its extreme when the parameter was 0.6.

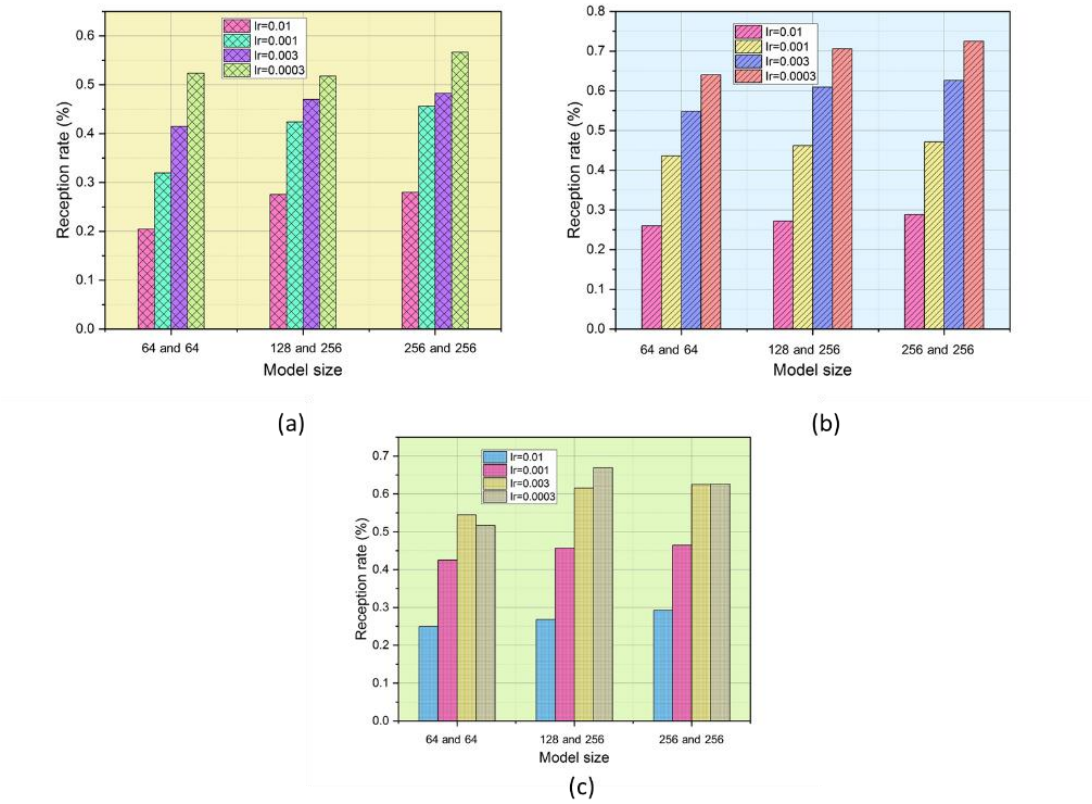

Figure 5: Comparison of reception rate under different values

Four hours of fuzzing tests were conducted throughout the experiment, as indicated in Figure 6, employing a learning epoch to create 16700 instances of testing for 40, 60, 70 and 80, respectively. The industrial control system was exposed to a test case every second to verify the effectiveness of the T-s DNN fuzzer. Every 20 minutes, the industrial control system's vulnerability number and reception rate were recorded. No manual intervention or deletion was used for the test instances produced by the T-s DNN fuzzer. The more training epochs the framework had, the more accurate it was at detecting message meanings. The T-s DNN fuzzer test system learned 80 epochs to provide sensor instances for vulnerability detection, with a learning proportion of 0.001 and strategy sizes of 128 and 256, resulting in 90% of the test instances receiving the actual message.

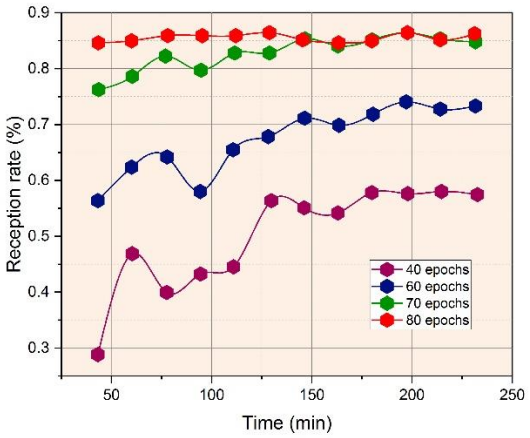

Figure 6: Learning Epochs' Impact on Reception Rate

D. Performance Evaluation

In this section, the proposed fuzzer is compared to traditional fuzzers such as APF Kitty [25], Peach Fuzzer [26], Radamsa [26] and EU Fuzzer [26] in terms of running time and number of bugs.

Running Time (s): This measure describes the amount of time that a fuzzing tool requires to execute through its test cases, create packets and communicate with the intended program or system. The proposed method is

compared to the existing fuzzers. Figure 7 represents the comparison outcome of the suggested method. The T-s DNN fuzzer method is 8.79 attain current methods [26] like Radamsa is 16.58, Peach Fuzzer is 11.60 and EU Fuzzeris 12.03. The T-s DNN fuzzer approach is very useful for the detection of vulnerability.

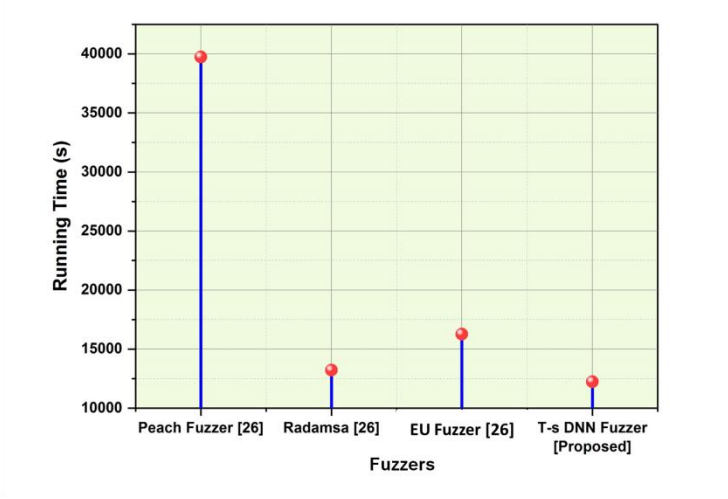

Figure 7: Comparison outcome of running time

Number of Bugs: This metric measures the flaws or vulnerabilities, the fuzzing tool found during the testing stages. The proposed method is compared to current methods, shown in Figure 8. The suggested T-s DNN fuzzer strategy (79.15) has lower numerical results compared to an existing method like Peach Fuzzer (93.53), Radamsa (63.25) and EU Fuzzer (68.25). The T-s DNN Fuzzer is significant for the detection of vulnerability.

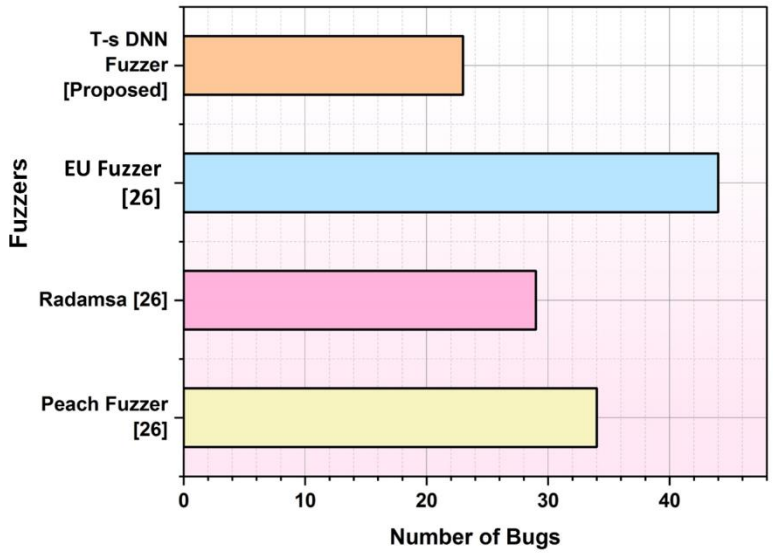

Figure 8: Result of Number of Bugs

E. Vulnerability Testing

In this section, each testing instance in the vulnerability process of mining has an identification number. Three different vulnerabilities (v) categories were discovered during the T-s DNN fuzzer experiment, along with the case number and vulnerability categories. The case code denotes the test instance number from the initial discovery of this vulnerability type.

V1 (Data Authentication - 3569): Vulnerabilities that make it possible for attackers to navigate around authentication restrictions in Modbus TCP-based platforms and access vital data or resources without authorization.

V2 (Data Tampering - 2588): Vulnerabilities that let intruders alter or manipulate data sent back and forth between Modbus TCP devices, potentially causing integrity of data difficulties or unauthorized device behavior modifications.

V3 (Data Injection - 2863): Vulnerabilities in Modbus TCP requests messages permit attackers to include execute arbitrary instructions, probably granting them access to or control over devices without authorization.

The T-s DNN fuzzer discovered the vulnerability, which was then examined by more conventional fuzzers including APF Kitty [25] and Peach fuzzer [26] for calculating VMA and TCVR.

Vulnerability mining ability (VMA): The VMA calculated both the total amount of weaknesses detected in the evaluation and the total amount of trail instances necessary to find these vulnerabilities, revealing an average amount of trial instances need to create an exemption. The three approaches' APF Kitty fuzzer [25], Peach fuzzer [26] and T-s DNN fuzzer respective VMAs were compared as the assessment indicator represented in Figure 9. The efficiency of three distinct fuzzers in identifying vulnerabilities is during a four-hour VMA and their execution times. Each fuzzer's initial vulnerability discovery time is displayed in the Fuzzer Period of Execution in Initial Vulnerability. The total amount of vulnerabilities found by every fuzzer throughout the assessment is shown in the Number of Vulnerabilities Identified in 4 h VMA. With a rate of detection of 0.09%, the APF Kitty Fuzzer spent 2689 seconds to identify 15 vulnerabilities. With a rate of detection of 0.06%, the Peach Fuzzer spent 4638 seconds to identify 12 vulnerabilities. With a rate of detection of 0.21%, the T-s DNN Fuzzer spent 2588 seconds to identify 27 vulnerabilities.

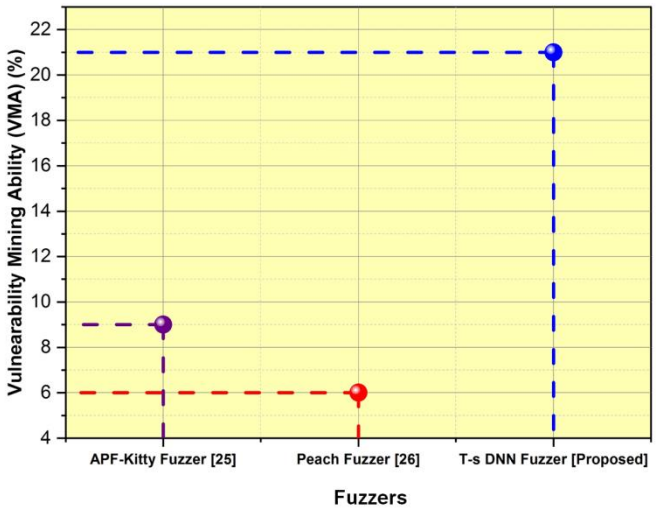

Figure 9: Outcome of VMA

Test case variation rate (TCVR) (%): The TCVR identified as the proportion of anomalous validation incidents to all test cases created, is another experimental evaluation indicator. For each test scenario, the TCVR using three fuzzers was assessed in this investigation. Figure 10 shows the outcome of TCVR. T-s DNN fuzzer (86.99%) was quite less than the APF Kitty fuzzer in contrast to all the test cases' variation rates, the Peach fuzzer (80.6 %) had the smallest, while the Kitty fuzzer (88.1 %) had the greatest TCVR while.

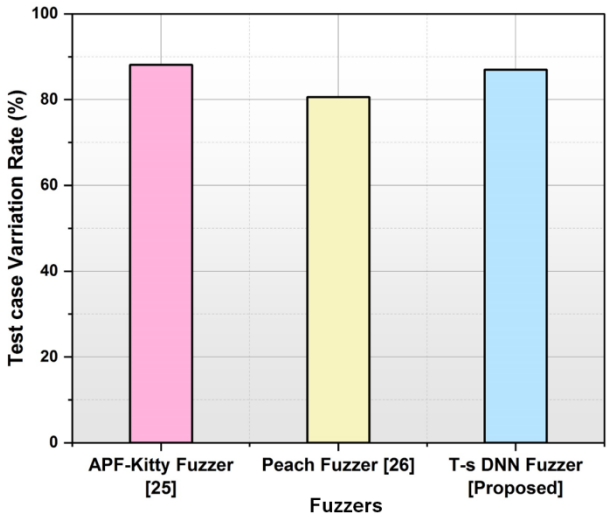

Figure 10: Outcome of TCVR

V. CONCLUSION

The study improved the inadequacies of low reception efficiency and low mining capacity by improving the vulnerability detection abilities of network protocols and industrial management systems. The approach integrated DL with fuzz testing, with a focus on Modbus TCP protocols. To examine and identify vulnerabilities, a special threshold-sample-driven deep neural network (T-s DNN) technology and fuzzing framework (T-s DNN Fuzzer) was created. Using thresholds and probabilities, the DNN algorithm was trained to read protocol information units, quantify probability variations via soft-max and select the most effective information values. The T-s DNN Fuzzer was utilized to conduct fuzz testing, which demonstrated enhanced exploitability, faster protocol vulnerability discovery and better sample receiving rates. The research findings underscore the efficacy of the T-s DNN methodology in augmenting vulnerability detection for protocols in networks. The computing resource needs of DL implementation in real-time applications could present issues. The T-s DNN technology should be improved for flexibility and its application to a larger range of networking protocols through more study.

ACKNOWLEDGMENT

This research received no external funding.

REFERENCES

[1] Bansal, S. and Kumar, D., 2020. IoT ecosystem: A survey on devices, gateways, operating systems, middleware, and communication. International Journal of Wireless Information Networks, 27(3), pp.340-364.

[2] Thota, M.K., Shajin, F.H. and Rajesh, P., 2020. Survey on software defect prediction techniques. International Journal of Applied Science and Engineering, 17(4), pp.331-344.

[3] Dissanayake, N., Jayatilaka, A., Zahedi, M. and Babar, M.A., 2022. Software security patch management systematic literature review of challenges, approaches, tools, and practices. Information and Software Technology, 144, p.106771.

[4] Tahmasbi, B., Haghshenas, H. and Birzhandi, S., 2021. Network vulnerability analysis is based on the overall inequity impacts of the distribution of the added travel time to the network users. European Journal of Transport and Infrastructure Research, 21(1), pp.94-114.

[5] Shrestha, P., Sathanur, A., Maharjan, S., Saldanha, E., Arendt, D. and Volkova, S., 2020. Multiple social platforms reveal actionable signals for software vulnerability awareness: A study of GitHub, Twitter, and Reddit. Plos one, 15(3), p.e0230250.

[6] Mateo Tudela, F., Bermejo Higuera, J.R., Bermejo Higuera, J., Sicilia Montalvo, J.A. and Argyros, M.I., 2020. On combining static, dynamic, and interactive analysis security testing tools to improve the top ten security vulnerability detection in web applications. Applied Sciences, 10(24), p.9119.

[7] Niu, W., Zhang, X., Du, X., Zhao, L., Cao, R. and Guizani, M., 2020. A deep learning-based static taint analysis approach for IoT software vulnerability location. Measurement, 152, p.107139.

[8] Chatterjee, S. and Thekdi, S., 2020. An iterative learning and inference approach to managing dynamic cyber vulnerabilities of complex systems. Reliability engineering & system safety, 193, p.106664.

[9] Pankajakshan, R., Biswal, S., Govindarajulu, Y. and Gressel, G., 2024. Mapping LLM Security Landscapes: A Comprehensive Stakeholder Risk Assessment Proposal. arXiv preprint arXiv:2403.13309.

[10] Hussain, S., Nadeem, M., Baber, J., Hamdi, M., Rajab, A., Al Reshan, M.S. and Shaikh, A., 2024. Vulnerability detection in Java source code using a quantum convolutional neural network with self-attentive pooling, deep sequence, and graph-based hybrid feature extraction. Scientific Reports, 14(1), p.7406.

[11] Shah, I.A., Rajper, S. and ZamanJhanjhi, N., 2021. Using ML and Data-Mining Techniques in Automatic Vulnerability Software Discovery. International Journal of Advanced Trends in Computer Science and Engineering, 10(3).

[12] Alloghani, M., Al-Jumeily, D., Hussain, A., Mustafina, J., Baker, T. and Aljaaf, A.J., 2020. Implementation of machine learning and data mining to improve cybersecurity and limit vulnerabilities to cyber-attacks. Nature-inspired computation in data mining and machine learning, pp.47-76.

[13] Liu, T., 2022. The application of machine learning models in network protocol vulnerability mining. Security and Communication Networks, 2022.

[14] Lin, G., Wen, S., Han, Q.L., Zhang, J. and Xiang, Y., 2020. Software vulnerability detection using deep neural networks: a survey. Proceedings of the IEEE, 108(10), pp.1825-1848.

[15] Chakraborty, S., Krishna, R., Ding, Y. and Ray, B., 2021. Deep learning-based vulnerability detection: Are we there yet? IEEE Transactions on Software Engineering, 48(9), pp.3280-3296.

[16] Hwang, R.H., Peng, M.C., Huang, C.W., Lin, P.C. and Nguyen, V.L., 2020. An unsupervised deep learning model for early network traffic anomaly detection. IEEE Access, 8, pp.30387-30399.

[17] Li, Y., Ji, S., Lyu, C., Chen, Y., Chen, J., Gu, Q., Wu, C. and Beyah, R., 2020. V-fuzz: Vulnerability prediction-assisted evolutionary fuzzing for binary programs. IEEE transactions on cybernetics, 52(5), pp.3745-3756.

[18] Li, Z., Zou, D., Xu, S., Jin, H., Zhu, Y. and Chen, Z., 2021. Sysevr: A framework for using deep learning to detect software vulnerabilities. IEEE Transactions on Dependable and Secure Computing, 19(4), pp.2244-2258.

[19] Cao, S., Sun, X., Bo, L., Wei, Y. and Li, B., 2021. Bgnn4vd: Constructing bidirectional graph neural network for vulnerability detection. Information and Software Technology, 136, p.106576.

[20] Dipu, N.F., Hossain, M.M., Azar, K.Z., Farahmandi, F. and Tehranipoor, M., 2024, January. FormalFuzzer: Formal Verification Assisted Fuzz Testing for SoC Vulnerability Detection. In 2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC) (pp. 355-361). IEEE.

[21] Bilgin, Z., Ersoy, M.A., Soykan, E.U., Tomur, E., Çomak, P. and Karaçay, L., 2020. Vulnerability prediction from source code using machine learning. IEEE Access, 8, pp.150672-150684.

[22] Ullah, K., Rashid, I., Afzal, H., Iqbal, M.M.W., Bangash, Y.A. and Abbas, H., 2020. SS7 vulnerabilities—a survey and implementation of machine learning vs rule-based filtering for detection of SS7 network attacks. IEEE Communications Surveys & Tutorials, 22(2), pp.1337-1371.

[23] Zhao, Y., Li, Y., Yang, T. and Xie, H., 2020. Suzzer: A vulnerability-guided fuzzer based on deep learning. In International Conference on Information Security and Cryptology (pp. 134-153). Springer, Cham.

[24] Al-Abassi, A., Karimipour, H., Dehghantanha, A. and Parizi, R.M., 2020. An ensemble deep learning-based cyber-attack detection in industrial control system. IEEE Access, 8, pp.83965-83973.

[25] Fu, G.Y., Liu, J.L., Cai, Y.N. and Li, H.L., 2017. APF-Kitty: A New Appropriate Protocol Fuzzy Testing Tool Based on Word Embedding. Sci. Technol. Eng, 17, pp.82-88.

[26] Men, J., Xu, G., Han, Z., Sun, Z., Zhou, X., Lian, W. and Cheng, X., 2019. Finding sands in the eyes: vulnerabilities discovery in IoT with EUFuzzer on human-machine interface. IEEE Access, 7, pp.103751-103759.