BLSTM-DCNNFuzz: A Deep Convolution Generative Adversarial Networks Based Framework For Industry Control Protocols Fuzzing

***Abstract*—A growing awareness is brought that the safety and security of Industrial Control Systems (ICS) cannot be dealt with in isolation and the safety and security of industrial control protocols (ICPs) should be considered jointly. Fuzz testing (fuzzing) for ICP is a common way to discover whether the ICP itself is designed and implemented with flaws and network security vulnerability. The traditional fuzzing test methods promote ICP safety and security testing and many of them have practical applications. However, they rely heavily on the specification of ICPs protocol, which causes the test process is a costly, time-consuming, troublesome, and boring task and hard to be repeated if the specification does not exist. In this study, a smart and automatical protocol fuzzing methodology based on Improved Deep Convolution Generative Adversarial Networks (DCGAN) and a series of performance metrics are proposed. An automatically and intelligent fuzzing framework BLSTM-DCNNFuzz for application is designed. Several typical ICPs, including Modbus and EtherCAT, are applied to test its effectiveness and efficiency. Experiment results show that, compared with General Purpose Fuzzer(GPF) and other fuzzing test methods based on deep learning, our methodology is more convenient, more effective and more efficient.**

***Index Terms*—convolution neural networks, long short-term memory, deep adversarial learning, fuzz testing, industrial control protocol**

I. Introduction

Industry 4.0 and Smart Manufacturing as a national plan for many countries are promoting a new round of industrial prosperity globally. In manufacturing, there are many safety-critical control systems, and ensuring its safety (Based on IEC 61508 [1]) and security (Based on IEC 62442 [2]) has been an important issue in academic and industry group. The entire system's safety and security can be considered in many ways. Some perform formal verification [3] of embedded programs in the system. Others perform penetration testing [4] to find system vulnerabilities. These efforts indeed improved system safety and security. However, intelligent manufacturing requires the increasing interconnectivity of ICSs. This reality exposes ICSs to more diverse and unexpected threats from outside, which has raised the potential risk to the security of ICSs. ICPs, as the bridge of communication between various parts of ICSs, have promoted the construction of industry informatization and improved the production and management efficiency, but it also laid many potential risks for ICSs. With the inherent flaws of ICPs which are likely caused in the design and application phase and increasing frequency and sophistication of cyber-threats towards ICPs, it is urgent to take high-performance measures to mine vulnerabilities of ICPs.

A considerable part of attacks exploits the vulnerabilities of safety and security in ICPs. First, when these protocols are designed and applied in ICSs, there inevitably exist defects of design and differences between the implementation and the specifications. Some safety flaws were introduced at this time. Second, ICPs have common characteristics, such as real-time, functional code abuse and unencrypted, which can be exploited by malicious parties to launch attacks. Whether it is a safety flaw or a security flaw, we all need to find it out first, and then make up for it. Fuzzing [5, 6] plays a vital role in finding these vulnerabilities. Its effectiveness has been proven by previous work [7, 8]. When performing the fuzz testing, we need to design and generate testing data according to the defined specifications, which brings some limitations. First, it does not work if it encounters an unknown specification. Second, the manual- based design for a specific protocol is not only demanding but time-consuming. Therefore, we attempt to find ways to improve the current situation.

Benefiting from its recurrent structure, Long Short-Term Memory Network (LSTM) [9], as an alternative type of neural network, shows great power in the precise timing of sequence data [10]. And Generative Adversarial Networks (GANs) [11] is particularly famous for generating highly simulated images [12]. Inspired by these, we attempted to integrate the characteristics of two networks to propose a combined model, replacing engineers, to generate massive fake but plausible test protocol messages. The model can not only be applied to both public and proprietary ICPs but also break the limitations above. In conclusion, we propose and design a fuzz testing methodology based on DCGAN (Deep Convolution Generative Adversarial Networks) [13] in this study. The contributions are summarized as follows:

(1) We propose a methodology based on Bi-directional LSTM (BLSTM) and DCGAN to deal with fuzzing data generation, in which it can intelligently learn to generate testing data by itself.
(2) On top of the approach, we build a universal fuzzing framework, the BLSTM-DCNNFuzz, which can deal with most ICPs' fuzz testing. Also, in data processing, character- level data conversion is implemented.
(3) To evaluate its effectiveness, we apply it to fuzzing several ICPs. The results reveal that our method has good performance.

The remainder of this paper is organized as follows. Section

II presents preliminary knowledge. Section III details optimized DCGAN algorithm and the entire methodology design. Section IV presents the evaluation results. Section V discusses the related work. Section VI concludes the paper and discusses some ideas about future work.

II. Preliminary

In this section, we introduce some preliminary knowledge. First, the basis of LSTM, GAN and DCGAN is presented. We then introduce the preliminary knowledge of ICPs and their common features. Lastly, we give an overview of fuzz testing and its application in finding ICPs' vulnerabilities.

A. Long Short-Term memory

Hochreiter and Schmidhuber (1997) first proposed LSTM to overcome the gradient vanishing problem of RNN (Recurrent Neural Networks). It is a special RNN that introduces an adaptive gating mechanism which can decides the degree to keep the previous state and avoid the problem of long-tern dependencies. Given a sequence $S = x1, x2, \ldots, xl$, where l is the length of input text, LSTM processes it word by word. At time-step t, the memory c_t and the hidden state h_t are updated with the following equations:

$$\begin{bmatrix} \Gamma_u^{<t>} \\ \Gamma_f^{<t>} \\ \Gamma_o^{<t>} \\ \hat{c}_t \end{bmatrix} = \begin{bmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{bmatrix} W \cdot [h_{t-1}, x_t] \tag{1}$$

$$c_t = \Gamma_f^{<t>} \odot c_{t-1} + \Gamma_u^{<t>} \odot \hat{c}_t \tag{2}$$

$$h_t = \Gamma_o^{<t>} \odot \tanh(c_t) \tag{3}$$

where x_t is the input at the current time-step, Γ_u, Γ_f and Γ_o is the update gate activation, forget gate activation and output gate activation respectively, $\hat{c}$ is the current cell state, σ denotes the logistic sigmoid function and $\odot$ denotes element-wise multiplication.

B. Generative Adversarial Networks

Generative Adversarial Networks, proposed by Goodfellow and others, is a promising framework to guide the training of generative models. Specifically, in GAN, a discriminative network D (discriminator) learns to distinguish whether a given data instance is real or not, and a generative network G (generator) learns to confuse discriminator by generating fake but plausible data.

In this adversarial network framework, the goal of the training generation model is to obtain a probability distribution of P_z, which approximates the distribution P_x of the real data x. In order to achieve this goal, a known distribution (such as Gaussian distribution, uniform distribution) is used to sample and get the "noise data " (represented by z) in the first place. Then, these "noise data" are input into the generation model, and the training of the generation model is carried out. After the training of the generator is finished, simulated data is output.

In this study, we utilize this feature to generate simulated sequence messages. Notably, when applying deep adversarial learning to fuzz testing ICPs, we expect the generated data to have a correct message format but with various message content so that the code coverage and testing depth can be guaranteed.

C. Deep Convolution Generative Adversarial Networks

Prior to introducing DCGAN, it is necessary to briefly introduce CNNs (Convolution Neural Networks), which have recently enjoyed great success in image and video recognition. Its success is mainly due to the large public image repositories, such as ImageNet, and high-performance computing systems, such as GPUs (Graphics Processing Units) or large-scale distributed clusters. Because the pooling layer has no weights, no parameters and only a few hyperparameters, one convention comprises one convolutional layer part and one pooling layer part in general. A typical CNN consists of many layers, including the input layer, conventions, the fully-connected layer, the output layer, etc.

Recently, NLP communities pay more and more attention to CNN and have achieved favorable results in various text classification tasks [14, 15]. Different from RNNs accomplished in time-related problems, CNN is good at learning spatial structure features. Actually, most ICPs' messages have the following features: concise format, limited length and compact structure. This makes CNN a better way to solve this kind of problem [14]. After proper preprocessing via Bi-LSTM which adds the location feature in the input, each filter in CNN can be regarded as a detector that detects whether a functional unit in the data frame is correct [16], which is conducive to grasping the format features of the sequence data in ICS. Hence in this study, CNN serves as the generator and the discriminator in DCGAN accordingly.

DCGAN is proposed by Alec Radford et al. [13] to bridge the gap between the success of CNNs for supervised learning and unsupervised learning. They extend GAN to the CNN domain and invent a structure called deep convolution generative adversarial networks (DCGAN) that have certain architectural constraints. This innovation combines the advantages of CNN in processing multidimensional features and the idea of deep adversarial learning. Due to the constraints, DCGAN largely overcomes the problem of unstable training of GANs, such as non-convergence, vanishing gradient and mode collapse. These constraints are listed in Table I. We designed our model architecture based on the constraints.

D. Industrial Control Protocols

ICPs refers to the communication protocol deployed in ICSs. As a class of systems, ICS has its characteristics, such as requiring high real-time performance and just providing several specific functions. Correspondingly, ICP's message format tends to be concise. ICS consist of master stations and slave stations. The data transmission and operation control between them are realized through the ICP in it. Currently, various ICPs operate in a wide variety of ICSs around the

TABLE I
ARCHITECTURE GUIDELINES FOR STABLE DEEP CONVOLUTIONAL GENERATIVE ADVERSARIAL NETWORKS

#	Architecture constraints
1	Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
2	Use batch normal in both the generator and the discriminator.
3	Remove fully connected hidden layers for deeper architectures.
4	Use ReLU activation in the generator for all layers except for the output, which uses Tanh.
5	Use LeakyReLU activation in the discriminator for all layers.

world. Therefore, it is important to maintain ICPs' safety and security. Except for the popular ICPs, Some ICPs are modified from the existing protocols or solely designed. These ICPs may have no clear specifications. Thus, the manual-based fuzz testing method will suffer from this.

E. Fuzz Testing

Fuzz testing is a quick and cost-effective method for finding severe security defects in software. Traditionally, fuzz testing tools apply random mutations to well-formed inputs of a program and test the resulting values. Besides, fuzzing is a brute force vulnerability method, in which it uses a large amount of malicious input to have stress tests on the target. As industrial control networks become more and more interconnected, flaws in the implementations of ICP could allow a malicious party to attack vulnerable systems remotely over the internet. To avoid this, we use fuzzing to discover the flaws in advance. The workflow is shown in Fig. 1.

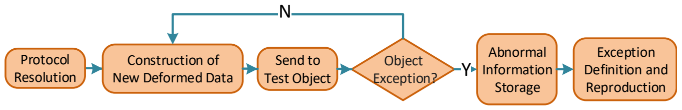

Protocol Resolution
Construction of New Deformed Data
Send to Test Object
Object Exception?
N
Y
Abnormal Information Storage
Exception Definition and Reproduction

Fig. 1. General workflow of fuzzing test

III. DCNNFUZZ FRAMEWORK

In this section, we first present an overview of the BLSTM-DCNNFuzz and then describe the main aspects in detail. As seen in Fig. 4, the overall fuzzing framework includes Data Preprocessing Module(DPM), Model Training and MSG(message) Generating Module(MTMGM), MSG Sending and Receiving Module(MSRM) and Logging Module(LM). These modules collaborate with each other in completing fuzzing test tasks.

A. Framework Overview

The purpose of this framework is to conquer the aforementioned limitations of traditional fuzz testing methods. We make it meet these requirements: (i) Highly intelligent. It can design and generate test cases by itself. (ii) Protocol independence. It can deal with most ICPs without knowing their protocol specifications. (iii) Very efficient. The entire testing can be completed in just a few days. As for the general process, first, the framework fetches network traffic data from the target ICS. Then, it learns from the traffic data and generates various testing data. Finally, the data is injected into the target system for testing. The following describes the main steps.

1) Preprocessing of ICP Communication Data: Data preprocessing has a significant impact on model training. We take two steps to process the raw data. First, we cluster the raw data according to specific criteria. Then, we augment the test cases with a small percentage to increase their influence on forming the model. Lastly, we encode the raw data into a fully digital matrix form.

2) Model Architecture and Training Strategies: We use DCGAN as the underlying architecture. Our purpose is to generate aggressive fuzz testing data to attack the target ICP for triggering more bugs. To achieve this purpose, we carefully design the network structure of the generator and the discriminator. As to the model training, we take measures to handle the unstable training problem such as mode collapse and non-convergence. Lastly, we use the trained model to generate enough fuzzing data.

3) Fuzz Testing The Target ICP: In this step, we use the generated fuzzing data to attack a specific ICP. In the testing, we deliberately collect test cases that caused exceptions. They can enrich the original training data and help improve model performance. The following gives more detailed information about these steps.

B. Preprocessing of ICP Communication Data

The current mainstream ICPs include Modbus, EtherCAT, Powerlink, Porfinet, Ethernet/IP, TSN (Time Sensitive Networks) [16], and so on. There are various ways to capture data packets from different ICPs. The most direct way is to use the appropriate terminal capture tool from the real industrial control network environment to capture the data packets generated by the ICSs as training data. After obtaining the raw data, Data Preprocessing Module (**DPM**) is divided into three steps to preprocess data. The details are as follows:

1) Data Frame Clustering: The effect of fuzz testing depends predominantly on the testing depth and high code coverage. Protocol messages captured from the ICP vary in length and message type. The better our model comprehends the differences between protocol messages, the better testing depth we can achieve. We leverage a variety of clustering strategies to improve the classification scores, such as frame length clustering, K-means clustering and so on. Frame length clustering is to cluster the sequences based on their length. Since sequences which have the same length always tend to share the same message type. K-means, using Euclidean distance as a similarity benchmark, is also applied in the clustering of data frames based on the function code of data frames. Under these strategies, we can do data augmentation to a class of messages with a small percentage. This makes the generated data more diverse, which can help improve code coverage.

2) Adding Special Symbols: The other step is to add special symbols to guide DPM to obtain higher quality training data.

The process of capturing data by DPM can be divided into two categories, as shown in Fig. 2. First, for a known protocol stack, such as the TCP/IP protocol stack which Modbus-TCP is running on, the IP header can be used as a demarcation point for capturing. We truncate the IP header (Including IP source address, destination address and additional information for some other delivery requests) and retain the file that holds the IP header for a further packet injection attack. Then we insert STA (start) as the sequences start flag at the beginning of the packet body, END (end) as the sequences end flag. The operation eliminates the influence of irrelevant information on the model and improves the quality of the captured data. Second, if the protocol is unknown, the learning with the address is performed, and STA and END are added directly at the beginning and end of the entire captured data. Finally, the processed data is stored in the training data set.

Moreover, pad the short sequence with uniform character PAD(pad) to the maximum frame length. It helps unify the sequence length to standardize the training.

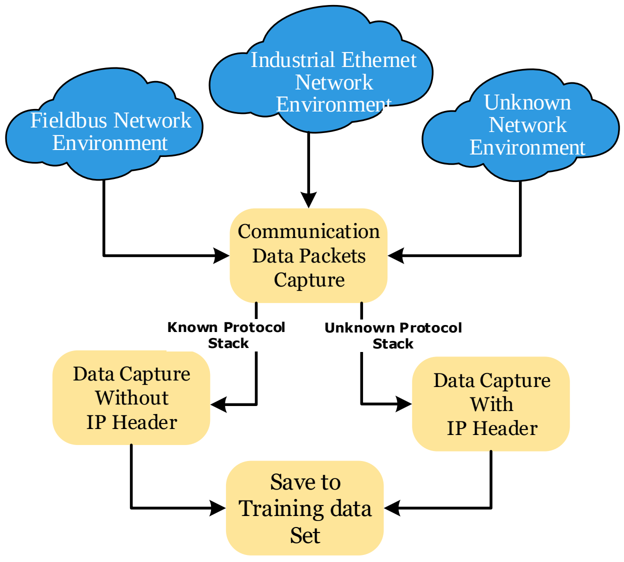

Industrial Ethernet Network Environment
Fieldbus Network Environment
Unknown Network Environment
Communication Data Packets Capture
Known Protocol Stack
Unknown Protocol Stack
Data Capture Without IP Header
Data Capture With IP Header
Save to Training data Set

Fig. 2. Process of capturing data

3) Data Feature Conversion: The captured sequence data is in the native features which cannot be processed directly. In order for the model to perceive these features, raw digital protocol messages need to be converted to an appropriate pattern. Due to few ways to learn word embeddings [17] about data frames, we transfer the network traffic in the ICS into a hexadecimal sequence based on an alphabet of n characters.

In addition to the sequence length alignment, we use a character level preprocessing, inspired by [15], to deal with the converted data feature in this work. In the alphabet of data frames, there are 10 digit characters and 6 English letter characters, as shown in the following:

0 1 2 3 4 5 6 7 8 9 a b c d e f

According to the alphabet, each character in the sequence is encoded as a one-hot vector of the n dimension $x \in R^{1\times n}$. As depicted in Fig. 3, the position where the character locates in the alphabet is one, and the rest are zeros. Thus, a sequence data with length l_0 is encoded into a matrix $X \in R^{l_0 \times n}$, as a stack of character embedding.

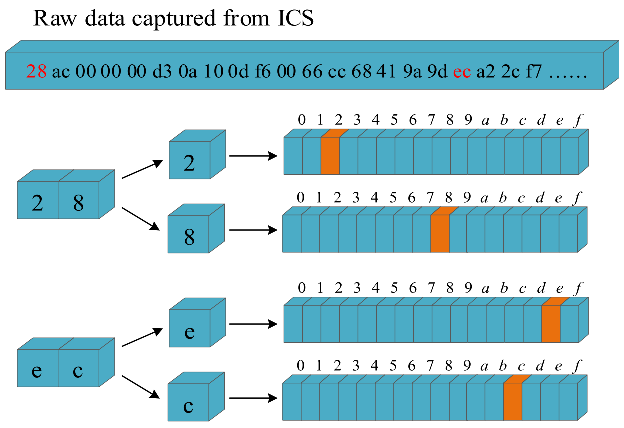

Raw data captured from ICS
28 ac 00 00 00 d3 0a 10 0d f6 00 66 cc 68 41 9a 9d ec a2 2c f7 ……
2 8
2
8
e c
e
c
0 1 2 3 4 5 6 7 8 9 a b c d e f

Fig. 3. A sample of raw ICS data and its feature quantization

C. Model Architecture and Training Strategies

In this section, we give the model architecture details of Model Training and MSG(message) Generating Module (**MTMGM**) as illustrated in Fig. 6. Later, we present the training strategies about the model in this study.

1) Model Architecture: There are two components, namely the generator model and the discriminator model. One of our design philosophies is to design lightweight models based on attaining its effect. In virtue of the distinguishing feature of reducing the consumption of computing resources, it is convenient to deploy to embedded devices and lays the groundwork for continuous online learning in the future. Referring to the aforementioned DCGAN architecture constraints and our requirements, we reasonably designed the architecture of DCGAN which are given in Fig. 6 (b). In general, the adversarial relationship between the G and the D can be expressed as follows:

$$\begin{aligned} \min_G \max_D V(D,G) = E_{x\sim P_x}[\log(D(x))] \\ +E_{z\sim P_z}[-\log(D(G(z)))] \end{aligned} \tag{4}$$

where $D(x)$ indicates that the probability of x is real, and $G(z)$ indicates that the probability distribution of the sample data z.

a. Generator. The generator uses a deconvolution [18] neural network architecture. We adopt batch normalization (BN) right after each convolution and before activation, following [19]. In addition, The generator consists of multiple fractional convolutional layers. Specifically, it replaces the pooling layer with fractional convolution layers, which is different from the traditional convolutional network.

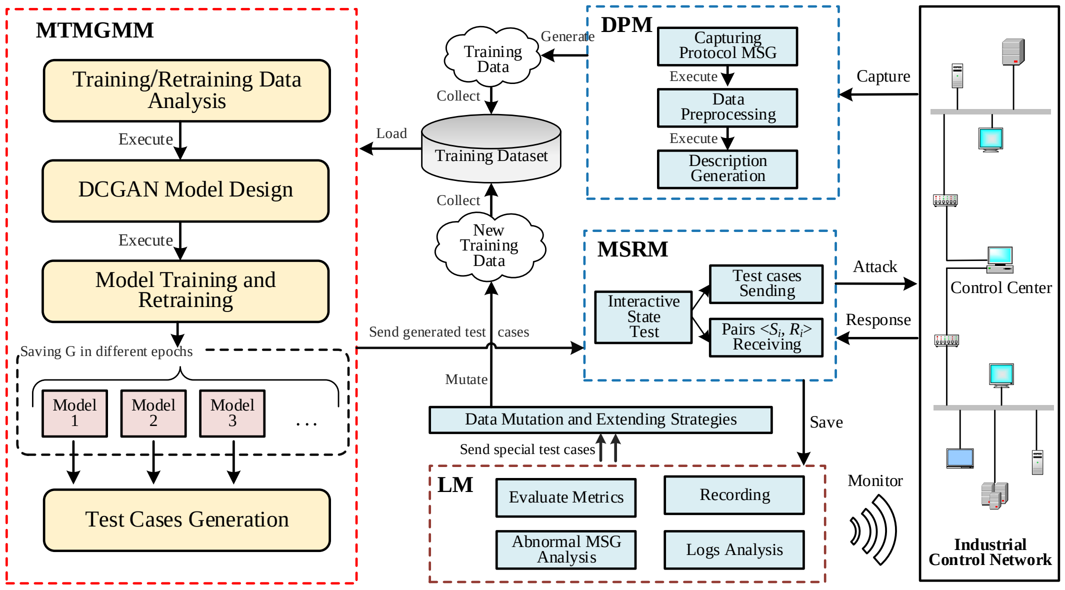


Fig. 4. BLSTM-DCNNFuzz Framework

Deconvolutions, also called transposed convolutions or fractional-stride convolution [18], work by swapping the forward and backward passes of a convolution. Based on Zero padding and non-unit strides in G, the following formula formalizes the output size of the deconvolutions in Fig. 6 (a).

$$a = (i + 2p - k)\, \%s \tag{5}$$

$$o' = s\,(i' - 1) + a + k - 2p \tag{6}$$

where o' represents the square output size, i' represents square input size, k represents square kernel size, s represents same strides along both axes, p represents same padding along both axes, i represent the input size of the next convolution layer, and a represents the number of zeros added to the bottom and right edges of the input.

Fig. 6 (a) depicts the network structure of the generator. Except for the last layer, we select BN and ReLU (Rectified Linear Units) as the activation in rest layers. The last layer applies Tanh as the activation. The generator takes noise data z from the uniform noise distribution P_z as input, and output a matrix which will be an input to the discriminator model. The 2D matrix can be seen as a sequence which converts by an embedding layer and each row represents a byte of the protocol messages. A LSTM layer, which inputs the 2D matrix and output sequences, is applied for decoding matrixes into sequences. It decodes the output of G into generated test cases. Notably, no pooling layers or fully connected layers are used. The loss function of the generator is:

$$E_{z\sim P_z}\left[-\log(D(G(z)))\right] \tag{7}$$

b. Discriminator. In adversarial training, the discriminator model is mainly designed to guide the training of the generator. One hot representation converts distributed representations of bytes of each protocol message into vectors, which form a matrix. The matrix is converted into a vector by an embedding layer as the input of the encoder. The vector includes two dimensions: the time-step dimension and the feature vector dimension. Most existing models just take notice of the time-step dimension of texts to obtain a fixed-length vector, ignoring the spatial structure features. However, the time-step dimension and the feature vector dimension are not mutually independent of each other.

To integrate the features on both dimensions of the matrix, we propose a combined model BLSTM-DCGAN based on the BLSTM and CNN so that the discriminator can hold not only the time-step dimension but also the feature vector dimension information. As shown in Fig. 6 (c), the BLSTM layer, as an encoder, contains two sub-networks for the forward and backward sequence context respectively and output a semantics vector. The output of the i_{th} byte is shown in the following equation, combining the forward and backward pass outputs:

$$h_i = \left[\vec{h}_i \oplus \overleftarrow{h}_i\right] \tag{8}$$

A matrix $H = h_1, h_2, \ldots, h_{l\_max}$, $H \in R^{l\_max \times d}$, is obtained from the encoder, where l_max is the length of maximum frame length of the ICP and d is the size of word embedding of BLSTM. The l_max of different ICPs is not the same, so the calculation steps on the discriminators are different. In this study, we make $d = l\_max$ to get a square input size which contains sequential information [20]. For the BLSTM layer, a cross entropy function is utilized as the loss function, which is defined as:

$$H(p,q) = -\sum\nolimits_x p(x)\log q(x) \tag{9}$$

where $p(x)$ is the real distribution of the sample and $q(x)$ is the probability of the model output.

L1 regularization term can make the model get more sparse solution, which facilitates the selection of features. L2 regularization term can reduce the complexity of the model and reduce the risk of overfitting. Considering the above, the BLSTM layer utilizes both L1 and L2. Since the output of the D is a two-class, the loss function of a BLSTM unit is obtained:

$$L_{BLSTM}^{<t>}(y^{<t>}, \hat{y}^{<t>}, \omega_1, \omega_2) = -y^{<t>} \log \hat{y}^{<t>} - (1 - y^{<t>}) \log(1 - \hat{y}^{<t>}) + \lambda_1 \|\omega_1\|_1 + \lambda_2 \|\omega_2\|_2^2 \tag{10}$$

Where y is the label; $y_h at$ is the probability of each class of the D output; λ_1 is the weight of the L1 regularization; λ_2 is the weight of the L2 regularization; ω_1 is the weight of the input fully connected layer; ω_2 is the weight of the BLSTM layer and the output layer.

And the loss function of the BLSTM layer for S is:

$$L_{BLSTM}(y, \hat{y}, \omega_1, \omega_2) = \sum_{t=1}^{T_x} L_{BLSTM}^{<t>}(y^{<t>}, \hat{y}^{<t>}, \omega_1, \omega_2) \tag{11}$$

Since BLSTM has access to the forward and backward context, CNN is utilized to explore more significative information. The discriminator applies strided convolution layers to replace any pooling layers instead of deconvolution layers. And z-score is utilized to normalize H. As depicted in Fig. 6 (c), the following formula formalizes the output size of the convolutions of D:

$$o' = \left\lfloor \frac{i + 2p - k}{s} \right\rfloor + 1 \tag{12}$$

Different from the generator, discriminator does not apply BN in the input layer to avoid sample oscillation and model instability [13]. It applies BN and Leaky ReLU [21] as an activation instead of ReLU. Taking an example of a process of one convolution operation in Fig. 6 (c), which involves a 2D filter $K \in R^{k_r \times k_c}$ which is applied to a window of k_r bytes and k_c feature vectors. For example, a feature $o_{m,n}$ is generated from a window of vectors $H_{m:m+k_r-1, n:n+k_c-1}$ by:

$$o_{m,n} = f(K \cdot H_{m:m+k_r-1, n:n+k_c-1} + b) \tag{13}$$

where $k_r = k_c$ in this study, m ranges from 1 to $(l\_\max - k_r + 1)$, n ranges from 1 to $(d - k_c + 1)$, $\cdot$ represents dot product, $b \in R$ is a bias and f is Leaky ReLU function. This filter is applied to each possible window of the matrix H to produce a feature map O:

$$O = [o_{1,1}, o_{1,2}, \cdots, o_{l\_\max - k_r + 1, d - k_c + 1}] \tag{14}$$

with $O \in R^{(l\_\max - k_r + 1) \times (d - k_c + 1)}$.

At the end of the network, the sum of the output matrix of the BLSTM layer and the reshaped output of strided convolution layers is appied as the input of fully-connected layers. Sigmoid is used in the output layers to make the output convert to a 1x1 discrimination probability. The detail layout of CNN can be seen from the right part of Fig. 6 (c). The discriminator takes real-world processed matrix from BLSTM or the output matrix from the generator as its input. The loss function of the discriminator is:

$$-E_{x \sim P_x}[\log(D(x))] - E_{z \sim P_z}[-\log(D(G(z)))] \tag{15}$$

What's more, in order to make the samples more diverse, we utilize the idea of the gradient penalty in WGAN-GP. The penalty of the loss function of DCGAN is:

$$\Omega(G) = E_{x \sim P_x}[\|x - G(z)\|_1] \tag{16}$$

The total loss function of DCGAN is:

$$L_{DCGABN}(D, G) = \min_G \max_D V(D, G) + \lambda \Omega(G) \tag{17}$$

where λ is the weight of the penalty. Our target is to minimize $L_{BLSTM}(y, \hat{y}, \omega_1, \omega_2)$ and $L_{DCGABN}(D, G)$ so that the difference between the real sample and the generated sample are minimized.

2) Model Training Strategies: To get a well-trained model, appropriate training strategies should be taken. DCGAN Model training is difficult because the two models need to be trained synchronously. We have adopted a reasonable strategy to avoid the emergence of the aforementioned problems such as model collapse and non-convergence. Dai et al. [22] found that setting parameters by pre-training is better than random initialization for deep learning network models, which can significantly stabilize the training. Therefore, we pre-train the discriminator for several epochs, getting parameters of D which helps form a gradient to guide the generator firstly. Second, we choose Adam optimizer [23] with tuned hyperparameters for both BLSTM and DCGAN, which takes advantage of adaptive learning rates and momentum. All models were trained with mini-batch stochastic gradient descent (SGD) [24] with a fairish mini-batch size. All weights initialized from a normal distribution. These tactics help reduce training oscillation and instability.

Our ultimate goal is to leverage the generated fake data to test the target and trigger more bugs. One factor that affects the effectiveness of fuzz testing is test data diversity. Rich test data tends to find more bugs. In addition to data augmentation [25], we save the generator model for several training epochs. Data generated in different epochs can enrich fuzzing data diversity. There exists a tradeoff between the correct data format and data diversity. We intend to not only make the generated data formats correct but also make the generated data more diverse in data content.

D. Fuzz Testing The Target ICP

With the trained model, we can generate as much test data as we want. When conducting a fuzz testing, MSG Sending and Receiving Module (**MSRM**) is in charge of monitoring interactive states, sending the test data to the target and receiving the feedback. Besides recording the relevant logs during the fuzzing process, the Logging Module(**LM**) is applied for abnormal MSGs and logs analysis based on the following performance metrics.

1) Performance Metrics: Some quantitative criteria [26–28] have emerged only recently assessing GAN on image generation. There is no unified validation metrics and benchmarks in this field. Therefore, in accordance with our research purpose and specific situation, we proposed the following metrics. Among them, $TCRR$ and DGD serve as the fuzzing effectiveness metrics., and BTE serves as the fuzzing efficiency metrics.

a. Test Case Recognition Rate (TCRR). $TCRR$ refers to the percentage of test cases recognized by the test target. It indicates the proportion of valid test cases. In the fuzz testing of ICP, if the target can recognize the test case, we consider the test case is correct in format and necessary information. The higher $TCRR$ indicates more generated test cases are similar to the real-world traffic sequence in format, which indicates testing depth is guaranteed. Conversely, the lower $TCRR$ means more test cases are dropped directly by the target, which needs to adjust or retrain the model. The formula is shown below:

$$TCRR = \frac{nRecognized}{nSent} \times 100\% \tag{18}$$

where $nRecognized$ is the total number of test cases recognized and $nSent$ is the total number of test cases sent.

b. Bugs Triggered Efficiency (BTE). On the one hand, BTE refers to the specific bugs found. On the other hand, it reflects the number and time of bugs triggered after sending a fixed number of test cases. It is an important indicator of model efficiency. Since the ultimate goal is to find as many vulnerabilities as possible, we consider not only whether bugs can be found in the testing but also the testing efficiency. It should be noted that the number of errors found is also related to the test target. Weak targets will highlight the method efficiency. However, in this study, we only focus on the efficiency of the method. The specific formula is as follows:

$$BTE = \frac{nBugs}{nAllCases} + \frac{\partial}{\sum\limits_{k=1}^{M} t_i} \tag{19}$$

where $nBugs$ indicates the number of bugs found, and the denominator $nAllCases$ is the number of all the test cases sent, $t_{abnormal}$ records the interval from the last normal request initiation to the next check-out of bug (five maximum values and five minimum values are discarded), M is the total number of time intervals, t_i is the interval of discovering the i_{th} bug, $t_{abnormal} = t_1, t_2, \ldots, t_m$ and $a = 1/e^8$ (empirical value). The larger value indicates the stronger bug trigger ability.

c. Diversity of Generated Data (DGD). DGD refers to the ability to maintain the diversity of the training data. More diverse generated test data frames are likely to cause more exceptions, which presents high code coverage. This indicator focuses on the number of message types in the generated data. It is also an important indicator of the method effectiveness.

$$DGD = \frac{nGCategory}{nACategory} \times 100\% \tag{20}$$

where $nGCategory$ is the total number of message categories in the generated data frame, and $nACategory$ is the total number of message categories in the training set.

2) Logging and Evaluation: LM is constructed to record the feedback from the ICP. The main function of LM is how to deal with logs. As shown in Fig. 5, it consists of two parts: one is collecting system logging of the master and slaves themselves; the other part is recording the feedback logs of the sent/received data to the log file. In the communication process, normal communication data and occurred anomalies will be logged into a log file by LM through collaboration with MSRM.

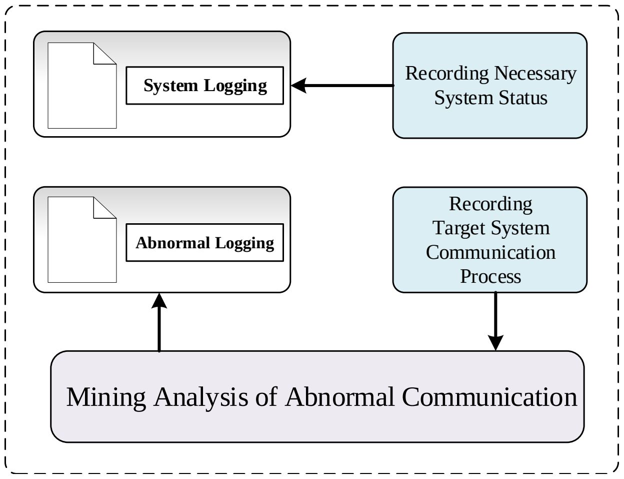

System Logging
Recording Necessary System Status
Abnormal Logging
Recording Target System Communication Process
Mining Analysis of Abnormal Communication

Fig. 5. Procedure of collecting logs

IV. Experiment

In this section, we evaluate the effectiveness and efficiency of the proposed method by experimentation. To show its superiority, we apply it to test Modbus, one of the widely used ICPs. To indicate the versatility of our method, another ICP, EtherCAT, will also be used to test.

A. Modbus and EtherCAT

We choose Modbus and EtherCAT as our test targets from a variety of ICPs in the experiment. ICPs have much in common features such as a short-data frame, no encryption. They are designed to meet the real-time requirements of the control system.

1) Modbus-TCP: Modbus protocols have many variants, including Modbus-TCP and Modbus-UDP. Here, we use Modbus-TCP as one of the fuzzing targets, as illustrated in Fig. 7. It uses master-slave communication mode, in which the master communicates with the slave by sending and receiving data frames. In the experiment, different Modbus-TCP implementations, including Modbus RSSim v8.20, Modbus Slave v6.0.2, and xMasterSlave v.156, are applied as the fuzzing targets. Finally, to better demonstrate the effectiveness of our approach, we use the serial communication mode between MCU [29] and PC, and adopt RS485 bus [30] for signal transmission to build the real Modbus network environment.

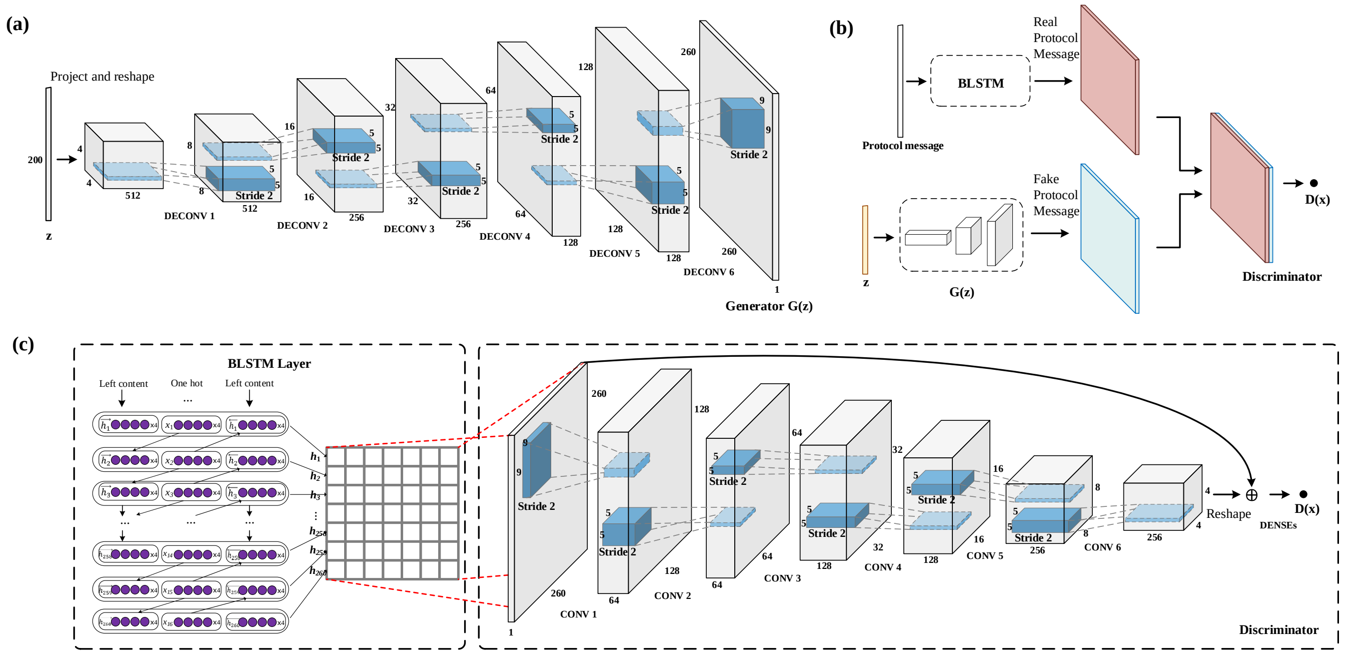


Fig. 6. The architecture of DCGAN: (a) generator network, (b) architecture of DCGAN, (c) and discriminator network. The first part of (c) is a BLSTM2DCNN for the 260 byte input sequence, one hot have size 16, word embedding have size 260 and BLSTM has 260 hidden units.

The generated test cases are sent in the real environment to test the effects in practical applications.

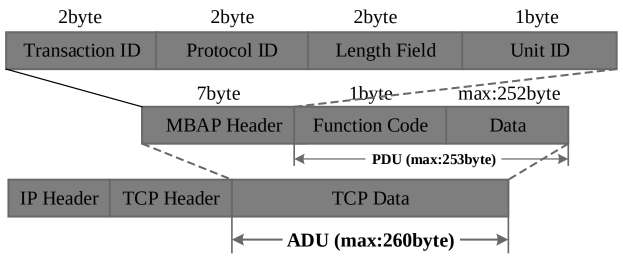


Fig. 7. Message format of Modbus-TCP

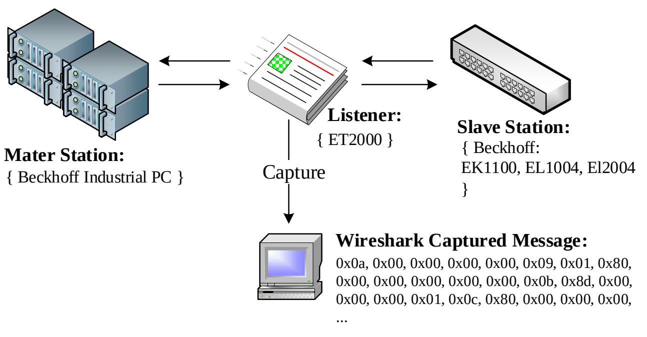
2) EtherCAT: EtherCAT offers high real-time performance and provides a master-slave communication mode among the industry devices. A typical EtherCAT network consists of one master and several slaves. The master generates datagrams and sends them to the loop network. These datagrams are reflected at the end of each network segment and sent back to the master. We test EtherCAT to prove the versatility of our method.

B. Training Data

Training data in deep learning significantly influence model training. Thus, we accurately collect and preprocess the training data. In the experiment, training data about the two industrial control protocols is collected separately.

1) Modbus-TCP: We use the Pymodbus [31], a python package that implements the Modbus protocol, to generate the training data frames. Through it, we can quickly generate enough different types of data frames conveniently. Specifically, 300,000 pieces of data, including various type, are used as the training data. The data set is divided into a training set, a verification set, and a test set according to the proportion of 10-fold cross-validation.

2) EtherCAT: In order to capture the EtherCAT communication data, we prepare an EtherCAT based ICS. The master station is a Beckhoff [32] industrial PC, and the slave station includes EK1100 [33], EL1004 [34] and EL2004 [35]. ET2000 [36] is used as the online Listener between the master and the slaves. The Wireshark [37], running on a computer, can fetch and display the massive communication data messages from the listener. After processing, these messages will serve as the training data for the EtherCAT protocol.



0x0a, 0x00, 0x00, 0x00, 0x00, 0x09, 0x01, 0x80,
0x00, 0x00, 0x00, 0x00, 0x00, 0x0b, 0x8d, 0x00,
0x00, 0x00, 0x01, 0x0c, 0x80, 0x00, 0x00, 0x00,
...

Fig. 8. EtherCAT enviroment

C. Evaluation Setup

1) Experimental Environment: We adopted Tensorflow, one of the popular deep learning framework, to implement the model architecture. To improve the training efficiency, we train the model on a Windows machine with 8 processors (Intel (R) Core (TM) i7-6700K CPU@4.00GHz) 16.0GB memory (RAM) Nvidia GeForce GTX 1080 Ti (11GB). When launching an attack, the simulators run on another machine with 4 processors (Intel (R) Core (TM) i5-5300U CPU@2.30GHz) 8.00GB memory (RAM).

2) Model Training Setting: As for the parameter setting, we initialize all weights from zero-centered Normal distribution with a standard deviation of 0.02. The mini-batch size is set to 256 in all models. The learning rate is set to 0.0002 in the Adam optimizer. As to the Leaky ReLU function in the discriminator model, the slope of the leak is set to 0.2. We train the models for 100 epochs and save the generator model for every ten epochs to get plentiful test cases.

D. Experiment Results

In this section, we show the experimental results in three aspects. We first present the bugs that occurred in the process of fuzzing the Modbus implementations. Then we reveal statistical results and its analysis. Lastly, to show protocol independence of our methodology in ICP's fuzz testing, we apply it to test the EtherCAT protocol.

1) Exception Founded: We send the generated data frames to the aforementioned Modbus implementations which serve as Modbus slave stations. A total of 270,000 test cases generated was sent to each Modbus implementations. The effect is exciting that we successfully triggered bugs. The following describes these bugs in detail.

Much abnormal information is displayed at the console of the simulation software when the Modbus Rssim is attacked by the generated data frames. For a while, it goes crash. In detail, the software pop-ups windows prompt box after we sent about 3500 data frames, indicating that the program has crashed. We send data frames range from 3450th to 3500th to the other two simulation softwares, Modbus Slave and xMasterSlave; no abnormality occurs. This comparison shows that Modbus Rssim has some errors in the emulating Modbus-TCP protocol.

Another exception worth discussing is "Station ID xx off-line, no response sent" in Modbus Slave. The log indicates that "Station ID 32 off-line, no response sent" after sending about 6540 data frames. But we observe that the station 32 is still online. This phenomenon makes us believe that there is an implementation flaw with the slave state judgment of Modbus Slave.

In fuzz testing the xMasterSlave, we find that the software automatically closes the window itself at times. Through the analysis of the system log, we conclude that memory overflow is the cause of the software crash, which suggests that the programmer may not consider the exception of populating with data boundary values when implementing the simulator.

In further attacks of fuzz testing the three simulation softwares and the real environment, anomalies such as "Using Abnormal Function Code", "Data length Unmatched", "Integer Overflow" and "Abnormal Address" occur on a regular basis. We record the test cases that cause these abnormalities. All the abnormal feedbacks from the three softwares and slaves in the real environment are counted for further analysis. Other anomalies, such as "File not Found" and "Debugger Memory Overflow" and so on, are found only once or twice and thus are not discussed in detail.

2) Statistical Analysis And Results: In the study, we choose the widely used GPF [38], GAN-based model and LSTM-based seq2seq mode as fuzzers in the control group. The systems to be tested are 3 Modbus simulation softwares, namely Modbus Slave, xMasterSlave, Pymodbus and Modbus RSSIM, and the real Modbus network environment we put up. In order to better evaluate the overall effect of the model on the protocol, we combined the experimental results of the four systems. The weights of the data obtained in these four experiments are 20%, 20%, 20%, 40% in the holistic data.

After fuzzers in the experimental group and control group are fully trained, fuzzing test is conducted by sending generated test cases through the TCP/502 port.

According to the three evaluation indicators mentioned above, we evaluate the effectiveness and efficiency of our fuzzing framework BLSTM-DCNNFuzz. Details are as follows.

a. TCRR.

GPF is compared with the other three fuzzing models based on depth learning, and the experimental results about $TCRR$ are shown in Fig. 9. The horizontal line represents the performance of GPF on the systems. Due to not involving the learning process, there is no changing trend of GPF.

From the perspective of the four experiments, the performance of the TCRR indicators is: GPF $\approx$ LSTM-based model $<$ GAN-based model $<$ BLSTM-DCNNFuzz. After training more than 30 epochs, the TCRR rates of generative adversarial algorithms obviously exceed GPF algorithm, and the rising trend of rate slows down significantly after 60 epochs. The average TCRR rate of the GPF is 58. 5%, and the TCRRs rate for generative adversarial algorithms is 75% to 90%. The final TCRR rate of the LSTM-based model algorithm is significantly lower than the other two groups for deep learning, which may be caused by the inability to learn the spatial characteristics of the data effectively. The function codes and parameters range of the protocol message are limited, which increases the possibility of abnormal exception in the random generation. For the generative adversarial algorithms from 50 to 100 epochs, the average TCRR rate of BLSTM-DCNNFuzz is approximately 8% higher than GAN, which indicates that BLSTM-DCNNFuzz is more applicable to test cases generation for ICPs indirectly.

b. BTE.

When testing the Modbus implementations, we recorded triggered bugs and triggered frequency. The fuzzing test results of the four models are as follows: Table II. The result comparison shows our proposed methodology can trigger more vulnerabilities in higher frequency than other models, which shows the testing efficiency of BLSTM-DCNNFuzz.

The concrete manifestation of BLSTM-DCNNFuzz is shown in Table III.

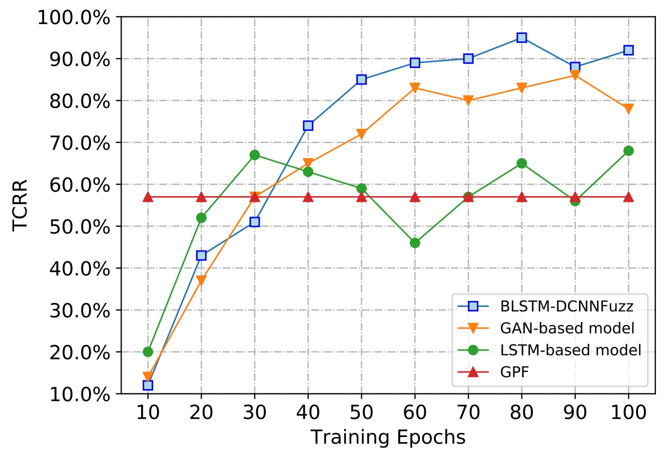


Fig. 9. TCRR changes with the training epochs

TABLE II
EXPERIMENTAL DATA COMPARISON

Test Model	Case Amount	Test time/h	Bugs triggered	BTE Score
BLSTM-DCNNFuzz	18,000	16	674	1.37
GAN-based model	38,785	3	101	0.98
LSTM-based model	128,711	9	283	0.86
GPF	120,000	28	49	0.15

c. DGD. The message categories learned by GPF is constant and the coverages of different GPF are different, so it is not discussed here. The testing depth has increased as illustrated in Fig.9 at the expense of reducing the code coverage of fuzzing test. Therefore, we need to maintain high test case diversity on the premise of attaining high TCRR rates. A total of 13 types of Modbus data frames are prepared in the original training data. When the training epochs is 10, the diversity of 3 models maintains the best. After training, some message categories are generally lost, as presented in Fig. 10. BLSTM-DCNNFuzz and LSTM-based model have good performance on maintaining basically the test case diversity, which illustrates the two models can learn the time-step dimension of protocol messages. And owing to BLSTM-DCNNFuzz containing two sub-networks for the forward and backward sequence context respectively in the BLSTM layer of BLSTM-DCNNFuzz, it is able to exploit information from both the past and the future, which performs better than LSTM-based model. Usually, the richer the data type, the higher code coverage we are likely to reach, and the stronger ability a model has to detect anomalies. Thus as presented in Table III, our trained model can effectively detect kinds of bugs.

TABLE III
TRIGGERED BUGS AND TRIGGERED FREQUENCY OF BLSTM-DCNNFUZZ

Triggered bugs	Frequency (Times)	Sent number
Slave crash	57	30,000
Station ID xx off-line	177	30,000
Using Abnormal Function Code	248	30,000
Automatically closes the window	73	30,000
Data length Unmatched	26	30,000
Abnormal Address	83	30,000
Integer Overflow	5	30,000
File not found	3	30,000
Debugger memory overflow	2	30,000

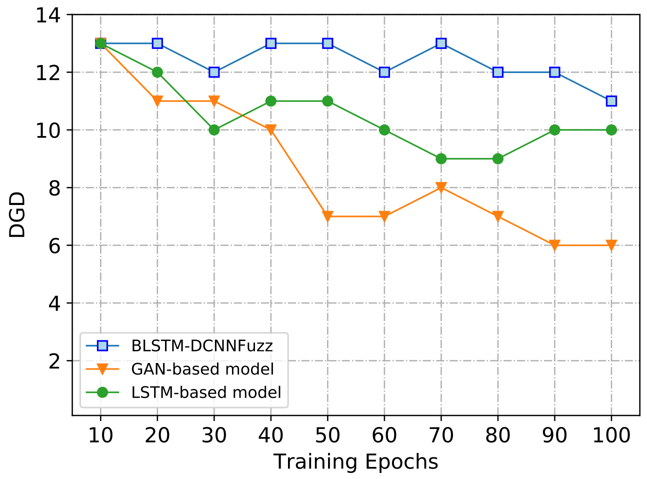


Fig. 10. Data diversity retention

3) Applying The Method to Another ICP-EtherCAT: To demonstrate our methodology is not just akin to a particular ICP, we apply BLSTM-DCNNFuzz to test EtherCAT, another widely used ICP. We retrain the model with captured EtherCAT data frames. With the newly trained model, massive test cases are generated expediently to detect the potential vulnerabilities of the EtherCAT protocol. The following presents the details.

a. Communication details of EtherCAT. EtherCAT uses a dedicated Ethernet data frame type definition to transport EtherCAT packets of Ethernet data frames. The master station initiates the communication process and controls the working status of the slave stations via process data communication. The EtherCAT slave stations extract the control information and commands from the protocol messages sent by the master station, and insert the relevant information and the collected data of the local industrial field device associated with itself. Then this Ethernet data message is transferred to the next EtherCAT slave stations. The operation repeats until all slave stations are traversed.

b. Detected potential vulnerabilities. As shown in Table IV, we detected these potential vulnerabilities, including man-in-the-middle (MITM), MAC address spoofing, slave address attack, packet injection, working counter (WKC) attack and so on. In the experiment, we send the generated data messages S_i to the slave stations and record the corresponding received message R_i. We get massive message pairs $< S_i, R_i >$. According to the abnormal protocol characterization above, we analyzed and compared the specified field values and obtained the following statistical results. Experiments on the EtherCAT protocol prove that our method has great versatility.

TABLE IV
POTENTIAL VULNERABILITIES AND OCCURRENCES TIMES IN ETHERCAT

Potential Vulnerabilities	Occurrences times	Sent Number
Packet Injection Attack	178 times	30,000
Man In The Middle Attack	26 times	30,000
Working Counter Attack	209 times	30,000
MAC Address Spoofing	41 times	30,000
Slave Address Attack	13 times	30,000
Unknown Attack	597 times	30,000

V. RELATED WORKS

Fuzzing has developed for decades, and practice has proven its effectiveness. In 1988 Professor Miller et al. [6] developed a fuzzing tool to test Unix programs' robustness. The goal of the tool is not to evaluate the safety of the system, but to evaluate the code quality and reliability of the system. At that time, fuzzing was simply feeding a program with random inputs. Subsequently, some researchers proposed various methods to improve fuzz testing. (i) Model-based fuzzing [39–41] models the input data based on a specific model. (ii) Grammar-based fuzzing [42–44] utilizes the input data grammar to guide the test data generation. Because of the effectiveness, fuzzing has been studied in the network protocol testing field. Aitel et al. [45] developed a block-based approach by divide the network packet into several blocks. Y. Hsu et al. [46] conducted the testing by abstracting a behavioral model from target protocols. These constant efforts make fuzz testing more and more mature.

Nowadays, with strong learning ability, deep learning is being applied to various fields. Without exception, some studies have incorporated deep learning into fuzzing. P. Godefroid et al. [47] proposed a sequence-to-sequence model to learn the input grammar of PDF objects to help produce fuzzing data for PDF parser. William Blum et al. [48] also applied a sequence-to-sequence neural network model to enhance the AFL (American Fuzzy Lop) [49] fuzzer in which the model attempts to learn the optimal mutation locations in the input files. It uses RNN as an assistive technology to improve the AFL's performance toward stand-alone programs. Chockalingam [50] uses a LSTM model to do intrusion detection about CAN bus protocol. These efforts all contribute a lot to deep learning based fuzzing. In general, most of them use RNN models and prior knowledge to deal with their fuzzing problem. However, in this study, we use the CNN based model as a core technique and attempt to deal with ICP fuzzing problems without knowing the prior knowledge.

VI. CONCLUSIONS AND FUTURE WORKS

In this study, we propose an effective fuzzing methodology based on DCGAN to generate fake but plausible fuzzing data about ICPs. This methodology utilizes CNN to learn the spatial structure and distribution of real-world messages and generate similar data frames without knowing the detailed protocol specification. Allowing the convolution neural networks to learn message formats can save human effort and reduce time. In this manner, when testing other network protocols, we do not need to understand their specifications, which is convenient. We ultimately evaluate this method by fuzzing two safety-critical ICPs, including Modbus-TCP and EtherCAT. The results indicate that the proposed method has application potential to test a series of ICPs.

In future studies, we expect to create a more intelligent and more automated network protocol fuzzing system deployed to embedded devices. The system can apply the manner of online learning to learn protocol specifications or message formats of different protocols automatically. Considering the current situation, we intend to perform the study in the following aspects. First, we want to do a further exploration of other architectures to enhance our approach. Second, we will use our method to test other stateful ICPs, such as Profibus, Powerlink and future TSN. These protocols constitute an important part of most current ICPs. Finally, we intend to integrate each excellent architecture and processing module to form a hybrid model and a complete software system, which can deal with most network protocols, including stateful protocols and non-stateful protocols.

REFERENCES

[1] R. Bell, "Introduction to iec 61508," in *Proceedings of the 10th Australian workshop on Safety critical systems and software-Volume 55*. Australian Computer Society, Inc., 2006, pp. 3–12.

[2] R. Piggin, "Development of industrial cyber security standards: Iec 62443 for scada and industrial control system security," in *IET Conference on Control and Automation 2013: Uniting Problems and Solutions*. IET, 2013, pp. 1–6.

[3] T. Braibant, "Coquet: a coq library for verifying hardware," in *International Conference on Certified Programs and Proofs*. Springer, 2011, pp. 330–345.

[4] J. P. McDermott, "Attack net penetration testing," in *Proceedings of the 2000 workshop on New security paradigms*. ACM, 2001, pp. 15–21.

[5] B. P. Miller, L. Fredriksen, and B. So, "An empirical study of the reliability of unix utilities," *Communications of the ACM*, vol. 33, no. 12, pp. 32–44, 1990.

[6] B. P. Miller, D. Koski, C. P. Lee, V. Maganty, R. Murthy, A. Natarajan, and J. Steidl, "Fuzz revisited: A re-examination of the reliability of unix utilities and services," Technical Report CS-TR-1995-1268, University of Wisconsin, Tech. Rep., 1995.

[7] T. Wang, Q. Xiong, H. Gao, Y. Peng, Z. Dai, and S. Yi, "Design and implementation of fuzzing technology for opc protocol," in *Intelligent Information Hiding and Multimedia Signal Processing, 2013 Ninth International Conference on*. IEEE, 2013, pp. 424–428.

[8] A. G. Voyiatzis, K. Katsigiannis, and S. Koubias, "A modbus/tcp fuzzer for testing internetworked industrial systems," in *Emerging Technologies & Factory Automation (ETFA), 2015 IEEE 20th Conference on*. IEEE, 2015, pp. 1–6.

[9] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[10] F. A. Gers, N. N. Schraudolph, and J. Schmidhuber, "Learning precise timing with lstm recurrent networks," *Journal of machine learning research*, vol. 3, no. Aug, pp. 115–143, 2002.

[11] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[12] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks," *arXiv preprint arXiv:1812.04948*, 2018.

[13] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434*, 2015.

[14] Y. Kim, "Convolutional neural networks for sentence classification," *arXiv preprint arXiv:1408.5882*, 2014.

[15] X. Zhang, J. Zhao, and Y. LeCun, "Character-level convolutional networks for text classification," in *Advances in neural information processing systems*, 2015, pp. 649–657.

[16] H. Adel and H. Schütze, "Exploring different dimensions of attention for uncertainty detection," *arXiv preprint arXiv:1612.06549*, 2016.

[17] O. Levy and Y. Goldberg, "Neural word embedding as implicit matrix factorization," in *Advances in neural information processing systems*, 2014, pp. 2177–2185.

[18] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," *arXiv preprint arXiv:1603.07285*, 2016.

[19] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[20] P. Zhou, Z. Qi, S. Zheng, J. Xu, H. Bao, and B. Xu, "Text classification improved by integrating bidirectional lstm with two-dimensional max pooling," *arXiv preprint arXiv:1611.06639*, 2016.

[21] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolutional network," *arXiv preprint arXiv:1505.00853*, 2015.

[22] A. M. Dai and Q. V. Le, "Semi-supervised sequence learning," in *Advances in neural information processing systems*, 2015, pp. 3079–3087.

[23] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[24] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *International conference on machine learning*, 2013, pp. 1139–1147.

[25] J. Salamon and J. P. Bello, "Deep convolutional neural networks and data augmentation for environmental sound classification," *IEEE Signal Processing Letters*, vol. 24, no. 3, pp. 279–283, 2017.

[26] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "Gans trained by a two time-scale update rule converge to a local nash equilibrium," in *Advances in Neural Information Processing Systems*, 2017, pp. 6626–6637.

[27] T. Karras, T. Aila, S. Laine, and J. Lehtinen, "Progressive growing of gans for improved quality, stability, and variation," *arXiv preprint arXiv:1710.10196*, 2017.

[28] M. Lucic, K. Kurach, M. Michalski, S. Gelly, and O. Bousquet, "Are gans created equal? a large-scale study," in *Advances in neural information processing systems*, 2018, pp. 700–709.

[29] J. D. Roberts Jr, J. Ihnat, and W. Smith Jr, "Microprogrammed control unit (mcu) programming reference manual," *ACM Sigmicro Newsletter*, vol. 3, no. 3, pp. 18–57, 1972.

[30] Z.-L. Feng and J.-X. Yu, "Design and implementation of rs485 bus communication protocol [j]," *Computer Engineering*, vol. 20, 2012.

[31] G. Collins, "Pymodbus pakage," https://pypi.org/project/pymodbus/.

[32] Beckhoff, "Beckhoff," https://www.beckhoff.com/.

[33] Beckhoff-EK1100, "Ek1100," http://www.beckhoff.com.cn/cn/default.htm?Ethercat/ek1100.htm/.

[34] Beckhoff-EL1004, "El1004," http://www.beckhoff.com.cn/cn/default.htm?Ethercat/el1004.htm/.

[35] Beckhoff-EL2004, "El2004," http://www.beckhoff.com.cn/cn/default.htm?Ethercat/el2004.htm/.

[36] Beckhoff-ET2000, "Et2000," https://www.beckhoff.com/english.asp?ethercat/et2000.htm/.

[37] Wireshark, "Wireshark," https://www.wireshark.org/.

[38] J. DeMott, R. Enbody, and W. F. Punch, "Revolutionizing the field of grey-box attack surface testing with evolutionary fuzzing," *BlackHat and Defcon*, 2007.

[39] M. Peroli, F. De Meo, L. Viganò, and D. Guardini, "Mobster: A model-based security testing framework for web applications," *Software Testing, Verification and Reliability*, vol. 28, no. 8, p. e1685, 2018.

[40] M. Utting, A. Pretschner, and B. Legeard, "A taxonomy of model-based testing approaches," *Software Testing, Verification and Reliability*, vol. 22, no. 5, pp. 297–312, 2012.

[41] A. Lunkeit and I. Schieferdecker, "Model-based security testing-deriving test models from artefacts of security engineering," in *2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, 2018, pp. 244–251.

[42] R. Hodován, Á. Kiss, and T. Gyimóthy, "Grammarinator: a grammar-based open source fuzzer," in *Proceedings of the 9th ACM SIGSOFT International Workshop on Automating TEST Case Design, Selection, and Evaluation*. ACM, 2018, pp. 45–48.

[43] S. Jero, M. L. Pacheco, D. Goldwasser, and C. Nita-Rotaru, "Leveraging textual specifications for grammar-based fuzzing of network protocols," *arXiv preprint arXiv:1810.04755*, 2018.

[44] T. Guo, P. Zhang, X. Wang, and Q. Wei, "Gramfuzz: Fuzzing testing of web browsers based on grammar analysis and structural mutation," in *Informatics and Applications (ICIA), 2013 Second International Conference on*. IEEE, 2013, pp. 212–215.

[45] D. Aitel, "The advantages of block-based protocol analysis for security testing," *Immunity Inc., February*, vol. 105, p. 106, 2002.

[46] Y. Hsu, G. Shu, and D. Lee, "A model-based approach to security flaw detection of network protocol implementations," in *Network Protocols, 2008. ICNP 2008. IEEE International Conference on*. IEEE, 2008, pp. 114–123.

[47] P. Godefroid, A. Kiezun, and M. Y. Levin, "Grammar-based whitebox fuzzing," in *ACM Sigplan Notices*, vol. 43, no. 6. ACM, 2008, pp. 206–215.

[48] M. Rajpal, W. Blum, and R. Singh, "Not all bytes are equal: Neural byte sieve for fuzzing," *arXiv preprint arXiv:1711.04596*, 2017.

[49] MichałZalewski, "American fuzzy lop," http://lcamtuf.coredump.cx/afl/.

[50] V. Chockalingam, I. Larson, D. Lin, and S. Nofzinger, "Detecting attacks on the can protocol with machine learning."