A Deep Convolution Generative Adversarial Networks Based Fuzzing Framework For Industry Control Protocols

Wanyou Lv[1] · **Jiawen Xiong**[1] · **Jianqi Shi**[1,*] · **Yanhong Huang**[1] · **Shengchao Qin**[2,3]

Received: date / Accepted: date

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 the ICP is a common way to discover whether the ICP itself is designed and implemented with flaws and network security vulnerability. Traditional fuzzing methods promote the safety and security testing of ICPs, and many of them have practical applications. However, most traditional fuzzing methods rely heavily on the specification of ICPs, which makes the test process a costly, time-consuming, troublesome and boring task. And the task is hard to repeat if the specification does not exist. In this study, we propose a smart and automated protocol fuzzing methodology based on improved Deep Convolution Generative Adversarial Network (DCGAN) and give a series of performance metrics. An automated and intelligent fuzzing framework BLSTM-DCNNFuzz for application is designed. Several typical ICPs, including Modbus and EtherCAT, are applied to test the effectiveness and efficiency of our framework. Experiment results show that our methodology outperforms the existing ones like General Purpose Fuzzer(GPF) and other deep learning based fuzzing methods in convenience, effectiveness, and efficiency.

Keywords Fuzz testing · Industrial control protocol · Quality control · Deep adversarial learning · Convolution neural networks · Long short-term memory · Industry 4.0

Introduction

Industry 4.0 and Smart Manufacturing, as a national plan for many countries, are promoting a new global round of industrial prosperity. In manufacturing, there are many safety-critical control systems, and ensuring their safety based on IEC 61508 (Bell et al. 2006) and security based on IEC 62442 (Piggin et al. 2013) has been an important issue in industry and academia. The safety and security of the entire ICS can be considered in many ways. Some perform formal verification of embedded programs (Carpanzano et al. 2014) and others perform penetration testing (Zhang et al. 2017) to find system vulnerabilities. These efforts indeed improve system safety and security. However, intelligent manufacturing requires the increasing interconnectivity of ICSs. ICPs, as the bridge of communication among various parts of ICSs, have promoted the construction of industry informatization and improved the production and management efficiency, but they also expose ICSs to many potential risks.

A considerable part of attacks exploit vulnerabilities of the safety and security of ICPs. On one hand, there are some inherent flaws of ICPs which are likely caused at the design and application stage. Safety flaws are introduced at this time. On the other hand, ICPs have common characteristics, such as real-time, functional code abuse and unencrypted messages, which causes increasing frequency and sophisti-

✉ Jianqi Shi
jqshi@sei.ecnu.edu.cn

[1] National Trusted Embedded Software Engineering Center, East China Normal University, No.3663 Zhongshanbei Road, Shanghai, China

[2] School of Computing, Engineering, &Digital Technologies, University of Teesside, Tees Valley, UK

[3] College of Computer Science & Software Engineering, Shenzhen University, Shenzhen, China

cation of cyber-threats towards ICPs and influences the security of ICPs. It is urgent to take high-performance measures to mine vulnerabilities of ICPs, regardless of a safety flaw or a security flaw.

Fuzzing plays a vital role in finding these vulnerabilities (Bocaniala et al. 2005). Its effectiveness has been proven in the network protocol testing field by previous work (Hsiang et al. 2012; Voyiatzis et al. 2015). In the traditional fuzzing methods, testing data is generated according to the defined specifications. However, this brings some limitations: such methods do not work if they encounter unknown specifications; furthermore, the manual-based design for a specific protocol is not only demanding but also time-consuming. With powerful learning ability, some studies have incorporated deep learning into fuzzing (Godefroid et al. 2017; Böttinger et al. 2018) to avoid learning the message format of a specific ICP artificially. Nonetheless, these state-of-the-art deep learning based fuzzing methods use prior knowledge to deal with the time-step dimension or the feature vector dimension of protocol messages independently, which ignores that they are not mutually independent of each other.

Inspired by this, we integrate the characteristics of LSTM and Generative Adversarial Networks (GANs) (Goodfellow et al. 2014) to propose a deep convolution generative adversarial networks based fuzzing methodology and design an automated and intelligent fuzzing framework based on it, named BLSTM-DCNNFuzz, which covers the time-step dimension and the feature vector dimension of messages of ICPs and can generate massive test protocol messages in a short time. The model can not only break the limitations above but also be applied to both public and proprietary ICPs. The contributions are summarized as follows:

(1) We propose a methodology combined by Bi-directional LSTM (BLSTM) and DCGAN (Radford et al. 2015) to deal with fuzzing data generation, which can intelligently learn to generate fake but plausible testing data by itself.
(2) On top of the approach, we build a universal fuzzing framework, the BLSTM-DCNNFuzz, which can deal with the fuzz testing of most ICPs.
(3) To evaluate its effectiveness, we propose a series of performance metrics. The experimental results of fuzzing several ICPs var these metrics in Section IV reveal that our method is more efficient and more effective than GPF and other deep learning based fuzzing methods.

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.

Related work

Fuzzing (Miller et al. 1990; 1995), originally known as random testing, is used to find system vulnerabilities and detect abnormal behavior of the system by inputting a large number of unexpected inputs into the target system, such as software program crashes or performing unexpected operations. It has developed for decades, and practice has proven its effectiveness. The input of fuzzing has the characteristics of clutter and wide coverage. The skill of fuzzing is illogical, and it is likely to raise certain exceptions from a logical point of view.

The Origin of Fuzzing

In 1988, professor Miller developed a fuzzing tool to test Unix programs' robustness. At that time, fuzzing was simply feeding a program with random inputs. But their results were striking: 25%∼33% of the programs are crashed under test, underscoring the powerful potential of fuzzy testing. Inspired by the work of Miller et al., more and more researchers have joined in the research of fuzzy testing technology and made the methodology applied to more fields.

Traditional Fuzzing Works

The goal of the fuzzing tool is not to evaluate the safety of systems, but to evaluate the code quality and reliability of systems. Subsequently, some researchers proposed various methods to improve fuzz testing.

1) Researches for Traditional Fuzzing Works

In general, there are two kinds of strategy methods to generate fuzzy test data. (i) Model-based fuzzing (Utting et al. 2012; Peroli et al. 2018; Lunkeit and Schieferdecker 2018) models the input data based on a specific model. (ii) Grammar-based fuzzing (Guo et al. 2013; Hodován et al. 2018; Pratama et al. 2019) utilizes the input data grammar to guide the test data generation.

Because of its effectiveness, fuzzing has been studied in the network protocol testing field. Aitel et al. (2002) developed a block-based approach by divide the network packet into several blocks. Greg Banks et al. (2006) proposed a fuzzy test tool called SNOOZE for stateful network protocols such as SIP, TCP/IP, etc. Devarajan and Ganesh (2007) released a fuzzy test module based on Sully tool for Modbus, DNP3 and other industrial control protocols at the Black Hat Conference, which is mainly used to detect denial of service attacks, unauthorized command execution and other problems. Voyiatzis et al. (2015) designed a Modbus-TCP fuzzy test tool called MTF which builds the test model by Modbus official instructions.

2) A Summary of Traditional Fuzzing Works on Industrial Control Protocols

Since the principle of fuzzing is simple and it can be automated, fuzzing has been favored by the majority of researchers when proposed. Many researchers have conducted in-depth research and exploration on fuzzing. These constant efforts make fuzz testing more and more mature.

Fuzzing technology has also been applied to network protocol testing, and many tools have been developed. Their works promote the development of network protocol testing technology, improve the security of network protocol implementation to a certain extent, and enhance the reliability of the computer network. Some of these works are for ICPs and have made a certain contribution to the improvement of the safety and security of ICPs.

3) Limitations of Traditional Fuzzing Works

Although the above works have promoted the development of the testing technology of ICPs to a certain extent, most of these research results are extended based on the original Fuzzing framework, and there are some problems as follows.

a. These existing network protocol fuzzy test tools are not targeted at the field of industrial control. So these tools are not only lacking in targeted functional design but also lack of consideration for the characteristics of ICPs.

b. There are many kinds of industrial control protocols. And the number of ICPs supported by the existing Fuzzing test tools is extremely limited (usually supporting only Modbus, DNP3 and a few others), which cannot meet the testing requirements of most ICPs.

c. There are a large number of private protocols in ICPs, such as the protocol adopted by Conitel-2020 equipment. These protocol specifications are usually not disclosed and cannot be obtained directly, so the message format and session process of the protocol cannot be understood. Because the traditional network protocol fuzzing methods have a strong dependence on protocol specification, it is difficult to carry out fuzzing without protocol official instructions.

d. Many tools contain general description methods of the ICPs and there are general frameworks for common protocol representations. But in many cases, the network protocols which need to be tested are proprietary protocols and only work on a small scale. The formats of proprietary protocols may not conform to common formats and traditional fuzzing tools will fail in these cases.

e. Some fuzzing tools are to realize the test of the protocol by encapsulating the network protocol specification. Each time a new protocol is tested, its specification needs to be analyzed. Implementing the details of the specification is costly and error-prone.

Deep Learning Works for Fuzzing

Nowadays, with strong learning ability, deep learning is being applied to various fields. As was expected, some studies have incorporated deep learning into fuzzing.

The fuzzing algorithm of ICPs based on deep learning is to utilize deep learning technology to solve the problem of fuzzy test case generation. Network traces are used as the training set, generation models learn formats of protocol from the network traces and the statistical properties. Through full training, we can get the generation model that can generate the pseudo-message with very high similarity to the real message. Then the generation models are used to guide the generation of pseudo-messages, and pseudo-messages are taken as test cases and input into the network systems under test to realize the test of ICPs.

1) Researches of Deep Learning Works for Fuzzing

Godefroid et al. (2008) proposed a sequence-to-sequence model to learn the input grammar of PDF objects to help produce fuzzing data for PDF parser. Chockalingam et al. (2016) utilized an LSTM model to do intrusion detection about CAN bus protocol. Rajpal et al. (2017) applied a sequence-to-sequence neural network model to enhance the AFL fuzzer. It uses RNN as an assistive technology to improve the AFL's performance toward stand-alone programs.

2) Contributions of Deep Learning Works for Fuzzing

Compared with traditional fuzzing works, deep learning methods for fuzzing bypass the process of building protocol specifications and protocol automata, reducing the workload. Moreover, fuzzing processes based on deep learning algorithm does not have logic errors or human negligence caused by the misunderstanding of protocol specifications. Furthermore, by adjusting the parameters of the neural network, the structure of the generation model can be fine-tuned so that the "similarity" between the test cases and the legal input can be adjusted, which is able to change the effect of the test while changing the acceptance rate of the test case.

3) Superiorities of Our Method

These efforts contribute a lot to deep learning based fuzzing. In general, most of them use RNN models and prior knowledge to deal with their fuzzing problems, which ignore the partial structure of samples. To solve the above limitations deep learning based fuzzing, the model based on deep learning has to learn the time-step dimension and the feature vector dimension of messages of ICPs.

BLSTM layers contain two sub-networks for the forward and backward sequence context respectively which can

not only cover the time-step dimension like LSTM but also combining the forward and backward pass outputs. DCGAN combines the advantages of CNN in learning spatial structure features and the idea of deep adversarial learning. According to the characteristics of the above two models, we integrate the BLSTM based model and DCGAN as a core technique to deal with ICP fuzzing problems. Different than other existing deep learning methods in theory which just consider the time-step dimension or the spatial structure feature, our framework not only learns the mapping between different bytes of messages of ICPs but also learns the impact of the detailed protocol specification on the bytes from spatial structure feature dimension.

The feature representation with the time-step dimension learned by BLSTM is as the training input for DCGAN to process multidimensional features and generate fake but more plausible data. The results show our framework performs better than traditional fuzzing methods and current mainstream deep learning based models.

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 detecting vulnerabilities of ICPs.

Long Short-Term memory

Hochreiter and Schmidhuber first proposed LSTM to overcome the gradient vanishing problem of RNN (Recurrent Neural Networks) in 1997. It is a special RNN that introduces an adaptive gating mechanism which can decide the degree to keep the previous state and avoid the problem of long-tern dependencies. Given a sequence $S = x_1, x_2, \ldots, x_l$, where l is its length, LSTM processes it character by character. 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.

Generative Adversarial Network

Generative Adversarial Network, proposed by Goodfellow et al., is a promising framework to guide the training of generative models. Specifically, in GAN, a discriminative network D (discriminator) learns to distinguish the reality of a given data instance, and a generative network G (generator) learns to confuse discriminator by generating fake but plausible data. In this study, we utilize this feature to generate plausible sequence messages.

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. 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 layers, the output layer, etc.

Recently, NLP communities pay more and more attention to CNN and have achieved favorable results in various text classification tasks (Zhang et al. 2015; Peng et al. 2018). Different from RNNs accomplished in time-related problems, CNN is good at learning spatial structure features. Actually, most messages of ICPs have the following features: concise format, limited length and compact structure. This makes CNN a better way to solve this kind of problem if location features are added in the input.

DCGAN is proposed by Alec Radford et al. (2015) to bridge the gap between the success of CNNs for supervised learning and unsupervised learning. This innovation combines the advantages of CNN in processing multidimensional features and the idea of deep adversarial learning. Due to certain architectural constraints listed in Table 1, deep convolution generative adversarial networks (DCGAN) largely overcomes the problem of unstable training of GANs, such as non-convergence, vanishing gradient and mode collapse. We design our model architecture based on these architectural constraints.

Industrial Control Protocols

ICPs refer to the communication protocol deployed in ICSs. As composite systems, ICSs have various characteristics, such as requiring high real-time performance and providing kinds of specific functions. Correspondingly, message formats of ICPs tend to be concise. ICSs consist of master

Table 1 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 Normalization 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.

stations and slave stations. The data transmission and operation control between them are realized through the ICP. Currently, various ICPs are deployed in a wide variety of ICSs around the world. Therefore, it is important to maintain the safety and security of ICPs.

Fuzz Testing

Fuzz testing is a quick and cost-effective method for finding severe security defects in software. Traditionally, fuzzing tools apply random mutations to well-formed inputs of a program and test the resulting values. As ICSs become more and more interconnected, flaws at the design stages and implementations of ICPs could allow malicious parties to attack vulnerable systems remotely on the internet. To avoid this, we utilize fuzzing to discover the flaws in advance to guide the improvement of ICPs. The workflow is shown in Fig. 1.

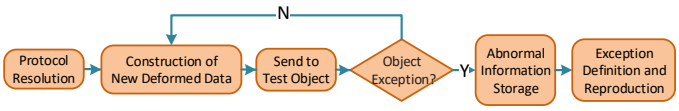

Fig. 1 General workflow of fuzz testing

DCNNFuzz Framework

In this section, we describe our methodology and the main aspects of the BLSTM-DCNNFuzz framework in detail. 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 tasks.

Preprocessing of ICP Communication Data

The current mainstream ICPs include Modbus, EtherCAT, Powerlink, Porfinet, Ethernet/IP, TSN (Time Sensitive Networks) (Wollschlaeger et al. 2017), and so on. There are various ways to capture data packets from different ICPs. The most direct way is to apply appropriate terminal capture tools to capture the data packets generated by the ICSs from the real industrial control network environment as training data. Training data in deep learning significantly influences model training. Thus after obtaining the raw data, we construct Data Preprocessing Module (DPM) and adopt appropriate strategies to preprocess the training data. DPM is divided into three steps and 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. Since sequences which have the same length always tend to share the same message type, frame length clustering is to cluster sequences based on their length. K-means, using Euclidean distance as a similarity benchmark, is also applied in the clustering of data frames based on function codes of data frames. Under these strategies, data augmentation is executed 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. We truncate the IP header (Including IP source address, destination address and additional information for some other delivery requests) and retain a file that holds the IP header for further packet injection attacks. Then we insert STA (start) as the sequence start flag at the beginning of the packet body, END (end) as the sequence 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, learning with the address is performed, and STA and END are added directly at the beginning and end of the entire captured data. 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. Finally, the processed data is stored in the training dataset.

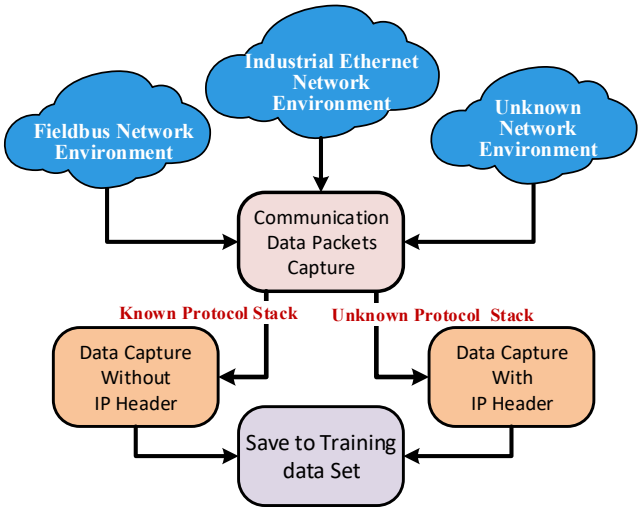

Fig. 2 Process of capturing data

3) Data Feature Conversion

The captured sequence 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 appropriate types. Prior to learning character embeddings (Zhang et al. 2015) about data frames, we use a character level preprocessing, inspired by Levy and Goldberg (2014). We transfer the protocol messages in the ICS into a hexadecimal sequence based on an alphabet of n characters in the first place. 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 with length l is encoded into a matrix $X \in R^{l\times n}$, as the input of character embedding.

Model Architecture and Training Strategies

In this section, we give the model architecture details of Model Training and MSG(message) Generating Module (MT-

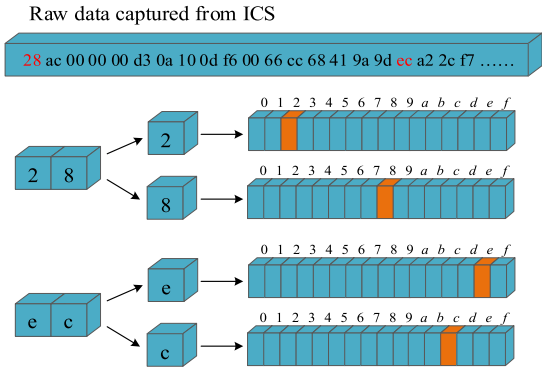

Fig. 3 Process of capturing data

MGM), which is illustrated in Fig. 6. After a while, training strategies about the model in this study is presented.

1) Model Architecture

There are two components in the model architecture, 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 a simplified architecture of DCGAN which are given in Fig. 6 (b). In general, this framework corresponds to a minimax two-player game and the adversarial relationship between the G and the D can be expressed as follows:

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

where $D(x)$ indicates that the probability that D distinguishes the real data x to be real, $G(z)$ represents that the probability distribution of the noise data z, P_x is the distribution of x and P_z is the distribution of z.

a. Generator. Fig. 6 (a) depicts the network structure of the generator. The generator applys a deconvolution (Sun et al. 2015) neural network architecture and consists of multiple deconvolution layers. Specifically, it replaces pooling layers with deconvolution layers, which is different from the traditional convolutional network. Deconvolutions, also called transposed convolutions or fractional-stride convolution, work through swapping the forward and backward passes of convolutions. Based on Zero padding and non-unit strides, the following formula formalizes the output size of the deconvolutions of the G in this study.

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

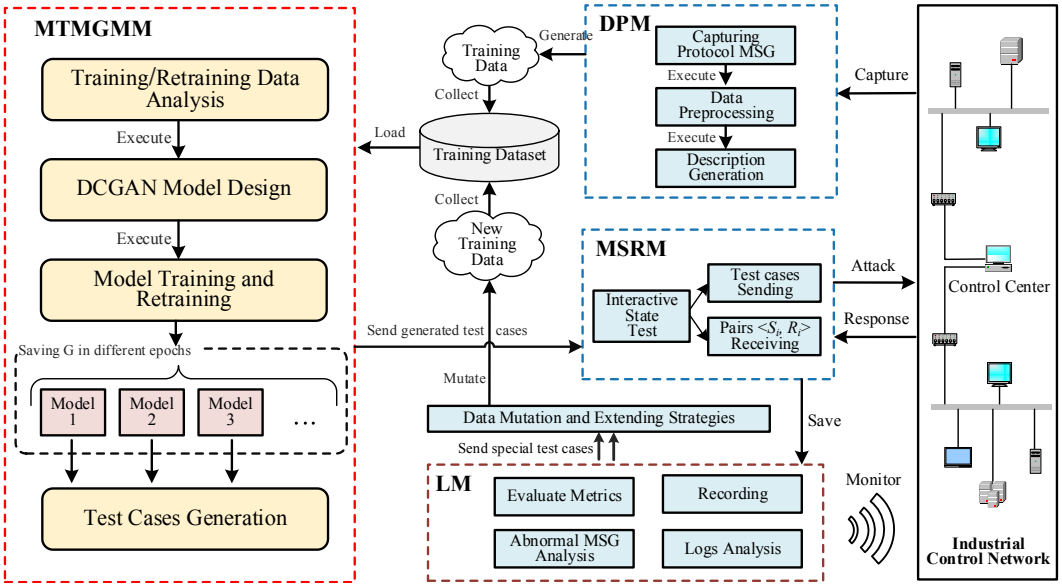

Fig. 4 BLSTM-DCNNFuzz framework

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

where o' ($o_1 = o_2 = o$) represents the square output size, i' ($i'_1 = i'_2 = i'$) represents square input size, k ($k_1 = k_2 = k$) represents square kernel size, s represents same strides along both axes, p represents same padding along both axes, i ($i_1 = i_2 = i$) 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. Here we focus on the simplified setting, but pay attention to that the formulas here also generalize to the N-D and non-square cases.

In addition, we adopt batch normalization (BN) right after each deconvolution and before activation, following Ioffe and Szegedy (2015). Except for the last layer, BN and ReLU (Rectified Linear Units) are selected as the activation in rest layers. The last layer applies Tanh as the activation. Notably, no pooling layers or fully connected layers are used in the G. The generator takes noise data z from the uniform noise distribution P_z as input, and output a 2D matrix which will be an input to the discriminator model. The 2D matrix can be seen as a sequence that converts by a character embedding layer and each row represents a character of the protocol messages. Here it is considered that an equivalence relation between the generated matrix of the generator and the generated intermediate semantics vector of the BLSTM network which is illustrated in Fig. 5. It decodes the output of G into generated test cases. Hypothetical characters are output through linear transformation and softmax layer via feeding the generated matrix of the G as the input to the output part of the BLSTM network. The training and optimization of the BLSTM network are mentioned below. 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 G. One-hot representation converts distributed representations of bytes of each real protocol message into vectors. 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 BLSTM and CNN so that the discriminator can hold not only the time-step dimension but also the feature vector dimension information. The one-hot vector is a sequence with a fixed length and dimension and can be regarded as a matrix. It is converted into another vector by a character embedding layer as the input of the BLSTM network from above. The BLSTM layers contain two sub-networks for the forward and backward sequence context respectively. As shown in the left part of Fig. 6 (c), the second BLSTM layer of the BLSTM network, as an encoder, generates an intermediate semantics vector as the input of the CNN (Zhou et al. 2016). The computation of the i_{th} character is shown in the following equation, combining the forward and backward pass outputs:

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

The intermediate semantics vector $H = \{h_1, h_2, \ldots, h_{l\_max}\}$, $H \in R^{l\_max \times d}$, can also be viewed as a matrix. l_max is

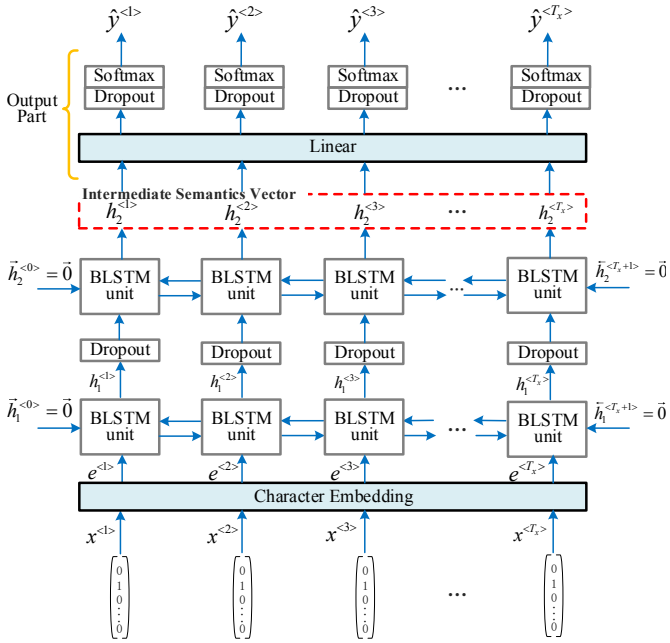

Fig. 5 BLSTM network structure

the length of maximum frame length of the ICP and d is the size of character embedding of BLSTM. The l_max of different ICPs is not the same, so the calculation steps on the discriminators of different ICPs are different. In order to simplify the operation here, we make $d = l\_max$ to get a square input size which contains sequential information. For the BLSTM network, a cross entropy function is utilized as the loss function, which is defined as:

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

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

Dropout and L2 regularization term can reduce the complexity of the model and reduce the risk of overfitting. Considering the above mentions, the BLSTM network utilizes both dropout and L2. Since the output of one BLSTM unit is a two-class, the loss function of a BLSTM unit is obtained:

$$L^{<t>}_{BLSTM}(y^{<t>},\hat{y}^{<t>},\omega_1) = -\sum_{j=1}^{C} y_j^{<t>}\log\hat{y}_j^{<t>} + \lambda_1\left\|\omega_1\right\|_2^2 \tag{10}$$

Where C is the size of charcter embedding, y is one-hot unit of the real character; $\hat{y}$ is the probability of each class from the softmax layer; λ_1 is the weight of the L2 regularization; ω_1 is the weight of the BLSTM layers and the output layer.

And the loss function of the BLSTM nettwork for the sequence S is:

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

Where T_x indicates the length of the input sequence.

Since BLSTM layers have access to the forward and backward context, CNN is utilized to explore more significative information, such as a hierarchy of representations and so on. After getting the intermediate semantics vector via the BLSTM network 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 (Yue et al. 2018), which is conducive to grasping the format features of the sequence data in ICSs. Different from the G, the D applies BN and Leaky ReLU as the activation instead of ReLU to avoid sample oscillation and model instability except for not applying BN in the input layer (Radford et al. 2015). 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 the D based on the simplified setting:

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

where the formula can also generalize to the N-D and non-square cases.

Taking an example of one convolution operation in Fig. 6 (c), it involves a 2D filter $K \in R^{k_r \times k_c}$, which is applied to a window of k_r characters 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 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 intermediate semantics vector 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 second BLSTM layer and the reshaped output of strided convolution layers is appied as the input of fully-connected layers. Sigmoid is utilized in the output layer of the D to convert the output to a 1x1 discrimination probability. The specific layout of the CNN network can be seen from the right part of Fig. 6 (c). The discriminator takes real-world processed matrix from the BLSTM layers or the output matrix from the generator as its input. The loss function of the D is:

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

Notably, when applying deep adversarial learning to fuzz testing ICPs, we expect the generated data to have correct

message formats but with various message content so that the code coverage and testing depth can be guaranteed. We draw lessons from the idea of the gradient penalty in WGAN-GP (Gulrajani et al. 2017) to make the samples more diverse. The penalty of the loss function of DCGAN is:

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

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 samples and the generated samples are minimized.

2) Model Training Strategies

To get a well-trained model, appropriate training strategies should be taken. Compared with the BLSTM network, 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. According to Dai and Le (2015), we pre-train the discriminator for several epochs, getting parameters of the D which help form a gradient to guide the generator firstly. Second, Adam optimizer (Kingma et al. 2014) with tuned hyperparameters are chosen for both BLSTM and DCGAN, which takes advantage of adaptive learning rates and momentum. All models are trained with mini-batch stochastic gradient descent (SGD) (Sutskever et al. 2013) with a fairish mini-batch size. And all weights are 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 anomalies. One factor that affects the effectiveness of fuzz testing is test data diversity. Rich test data tends to find more anomalies. In addition to mutation and data augmentation (Salamon and Bello 2017), 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.

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 (Heusel et al. 2017; Lucic et al. 2018; Pourjavad et al. 2019) have emerged only recently assessing GAN on image generation. There is no unified validation metric and benchmark in this field. Therefore, in accordance with our research purpose and specific requirements, we proposed the following metrics. Among them, $TCRR$ and DGD serve as the fuzzing effectiveness metrics and ATE serves as the fuzzing efficiency metric.

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 ICPs, we consider the test case is correct in format and necessary information if the target can recognize and respond the test case. The higher $TCRR$ indicates more generated test cases are similar to the real-world sequences in format, which shows the testing depth is guaranteed. Conversely, the lower $TCRR$ means more test cases are dropped directly by the target, which indicates that the model needs to be adjusted or retrained. 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. Anomalies Triggered Efficiency (ATE). On the one hand, ATE refers to the specific anomalies found. On the other hand, it reflects the number and time of anomalies triggered after sending a fixed number of test cases. It is an important indicator of the efficiency of models. Since the ultimate goal is to find as many vulnerabilities as possible in a short time, we consider not only whether anomalies can be found in the testing but also the testing efficiency. It should be noted that the number of anomalies found is also related to the test target. Weak targets will highlight method efficiency. However, in this study, we only focus on the efficiency of the method. The specific formula is as follows:

$$ATE = \frac{nAnomalies}{nAllCases} \times 100\% + \frac{\partial}{\sum_{k=1}^{M} t_i} \tag{19}$$

where $nAnomalies$ indicates the number of anomalies found, and the denominator $nAllCases$ is the number of all the test cases sent, $t_{anomalies}$ records the interval from the last normal request initiation to the next abnormal feedback (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} anomaly, $t_{anomalies} = [t_1, t_2, \ldots, t_m]$ and ∂ is the weight of the reciprocal of the sum of time intervals. The larger value indicates the stronger ability to trigger anomalies.

c. Diversity of Generated Data (DGD). DGD refers to the ability to maintain the diversity of the training data. More

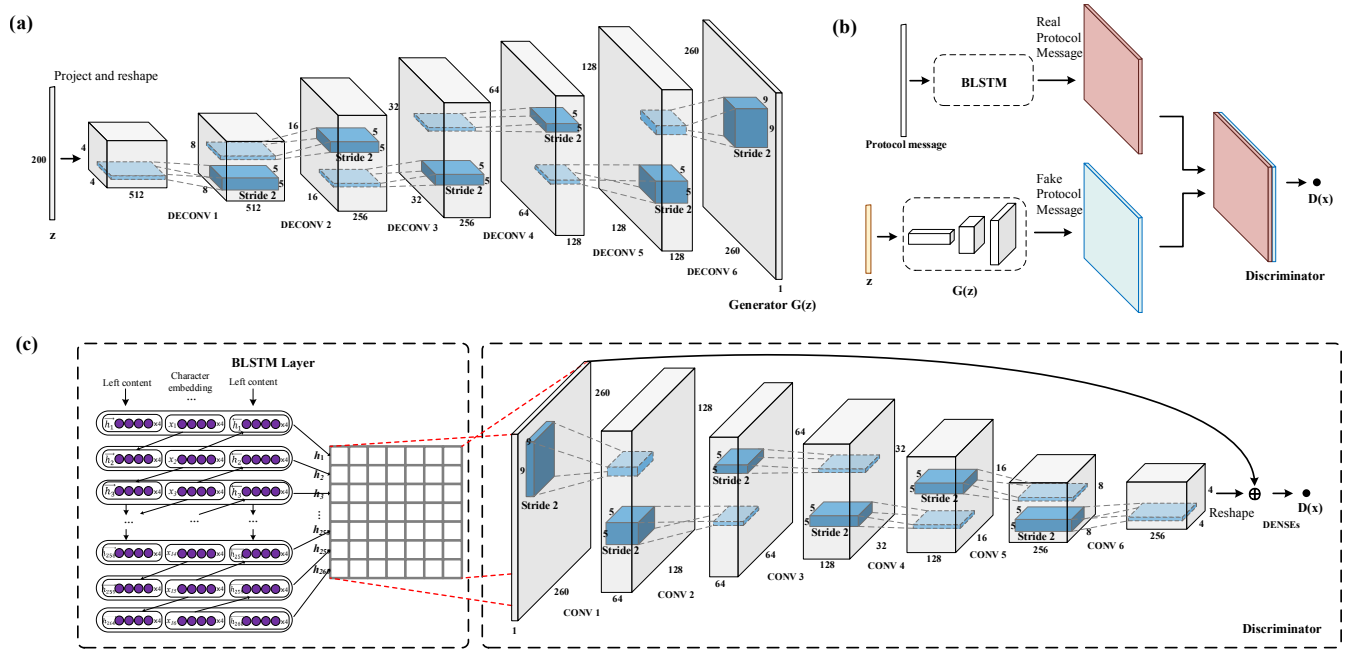

Fig. 6 The architecture of DCGAN: (a) generator network, (b) architecture of DCGAN, (c) and discriminator network. The first part of (c) is a BLSTM network for the 260 characters input sequence, one-hot has size 16, character embedding has size 260 and one BLSTM layer has 260 hidden units.

diversity of generated test data frames is 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 generated data frames, and $nACategory$ is the total number of message categories in the training set.

2) Logging and Evaluation

LM is constructed to record feedbacks from the ICP. The main function of LM is to deal with logs. It consists of two parts: one is to collect system logging of the master and slaves themselves; the other part is to record the feedback logs of the sent/received data frames 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.

Experiment

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

Modbus and EtherCAT

1) Modbus-TCP

Modbus protocols have many variants, including Modbus-TCP, Modbus-UDP and so on. In this study, we use Modbus-TCP as one of the fuzzing protocols. The message format of Modbus-TCP is illustrated in Fig. 7.

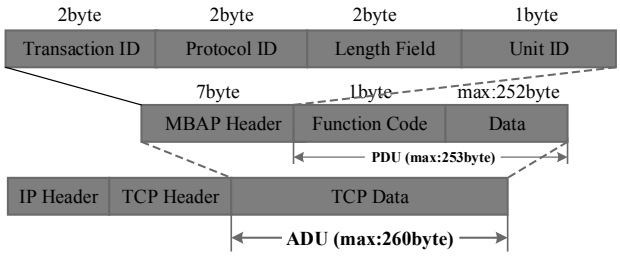

Fig. 7 Message format of Modbus-TCP

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. Furthermore, to better demonstrate the effectiveness of our approach, we deploy the serial communication mode between MCU (Roberts Jr et al. 1972) and PC, and adopt RS485 bus (Feng and Yu 2012) for signal transmission to build the real Modbus-TCP network envi-

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

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 as well as Modbus. 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.

Training Data

In the experiment, training data about the two industrial control protocols are collected separately.

1) Modbus-TCP

We use Pymodbus (Collins 2013), a python package that implements the Modbus protocol, to generate the training data frames. Through it, enough different types of data frames can be generated quickly and conveniently. Specifically, 300,000 pieces of data, including various types, are used as the training data after data preprocessing. The dataset 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 network environment based ICS as illustrated in Fig. 8. The master station is a Beckhoff (Mack et al. 2012) industrial PC, and the slave stations include EK1100, EL1004 and EL2004. ET2000 is used as the online Listener between the master and the slaves. Wireshark (Orebaugh et al. 2006), running on a computer, fetches and displays the massive communication data messages from the listener. After the processing of DPM, these messages serve as the training data for the BLSTM-DCNNFuzz framework.

Evaluation Setup

1) Experimental Environment

We adopt 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).

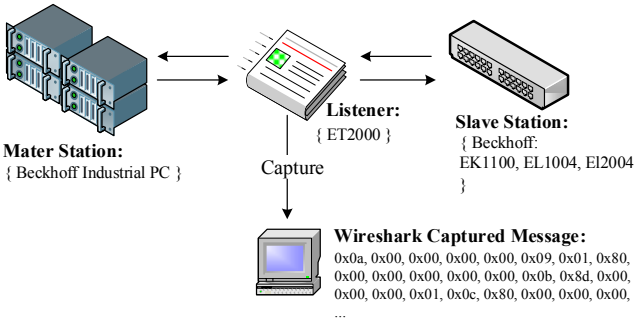

Fig. 8 EtherCAT enviroment

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 $keep\_prob$ hyperparameter of dropout is set to 0.8. The learning rate is set to 0.0001 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.

Experiment Results

We divide this section into three parts to make clear the experimental results. We first present and analyze the statistical result of fuzzing Modbus implementations to evaluate the effectiveness and efficiency of these aforementioned models. Then we reveal some special anomalies that occurred in the process and find out bugs among these anomalies afterward. Lastly, the experimental results of fuzzing EtherCAT protocol are shown.

1) Statistical Analysis And Results

In this study, we choose the widely used GPF, GAN-based model and LSTM-based seq2seq mode as fuzzers in the control group. The targets to be tested are 3 Modbus simulation softwares mentioned above and the real Modbus network environment we have built. In order to better evaluate the overall effect of the model on the protocol, we combined the experimental results of the four systems and the weights of the data obtained in these four targets are 20%, 20%, 20%, 40% in the holistic data. After fuzzers in the experimental group and control group are fully trained, fuzz testing is conducted by sending a total of 270,000 generated test cases to Modbus implementations through the TCP/502 port.

According to the three aforementioned evaluation indicators, the effectiveness and efficiency of our fuzzing framework BLSTM-DCNNFuzz and fuzzers in the control group

are evaluated and represented graphically. Details are as follows.

a. TCRR.

The experimental results of TCRR are shown in Fig. 9. Due to not involving a continuously learning process, the performance of GPF has no changing trend on the targets so that it is represented via a horizontal line when compared to the other three fuzzing models based on depth learning. From the perspective of the four models, the performance of the TCRR indicators is: GPF $\approx$ LSTM-based model $<$ GAN-based model $<$ BLSTM-DCNNFuzz.

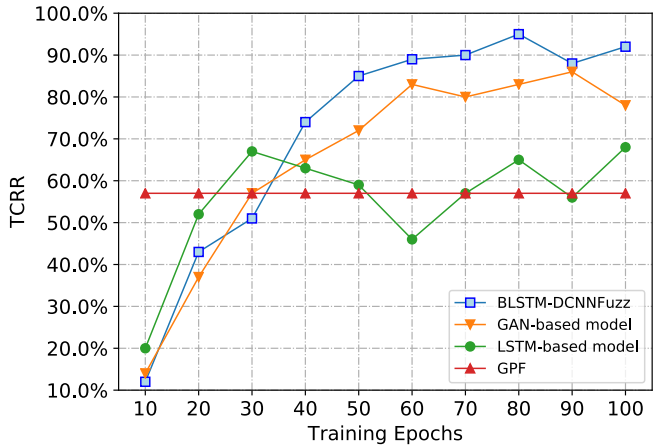

Fig. 9 TCRR changes with the training epochs

After training more than 30 epochs, the TCRR rates of generative adversarial algorithms obviously exceed the GPF algorithm. And the rising trends of rates slow down significantly after 60 epochs. The average TCRR rate of GPF is 58. 5% and the TCRR rates for generative adversarial algorithms are 75% to 90%. The final TCRR rate of the LSTM-based model algorithm is significantly lower than the other two groups for adversarial learning, which may be caused by the inability to learn a hierarchy of representations of the data effectively. Because the seq2seq model based LSTM is trained in an unsupervised learning environment, it may result in generating a large number of malformed protocol message sequences, leading to normal throwing and generating a rapid increase in the proportion of illegal messages which we speculate there is a good chance to be the main cause of the large fluctuation. The function codes and parameters range of the protocol messages are limited, which increases the possibility of abnormal recognition 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 the GAN-based model, which indirectly indicates that BLSTM-DCNNFuzz is more applicable to test cases generation for ICPs.

b. ATE.

When testing Modbus implementations, we record triggered anomalies and triggered frequency. We compare the four models combined with the following statistical or calculated data to interpret and assess the ATE indicator better: categories of Triggered anomalies (CTA), number of Triggered anomalies (NTA), average time interval of Triggered anomalies (ATITA), true negative rate (TNR) and true positive rate (TPR) values.

The result comparison shows our proposed methodology can trigger more vulnerabilities in a higher frequency than other models, which represents the testing efficiency of BLSTM-DCNNFuzz. While we expect performance gains through the integration of the BLSTM network and DCGAN, we are surprised at the magnitude of the gains. BLSTM-DCNNFuzz beats all baselines on Modbus RSSim v8.20, Modbus Slave v6.0.2, xMasterSlave v.156 and the real environment from the GAN-based model. As for the TPR metric, BLSTM-DCNNFuzz gets a second higher accuracy. Some of the previous techniques only work on whether an anomaly can be triggered, but not the efficiency of triggering anomalies. Our question becomes whether it is possible for our model when fuzzing targets to consider not only whether anomalies can be found in the testing but also the testing efficiency.

For that purpose, we test the four models on the four fuzzing targets by sending 270000 pieces of fake data generated by each of the four models. Compared with the other three models, BLSTM- DCNNFuzz achieves a comparable testing result as shown in Table 2.

• ***CTA:*** Compared with the GAN-based model, the proposed model does not depend on external language-specific features because it captures features from the output of BLSTM layer, which has already extracted features from the original input text. The comparison of columns of CTA indicates that the customized BLSTM- DCNNFuzz model for the diversity of generating test cases is more capable than the other three models when dealing with the same protocol.

• ***NTA & ATITA:*** NTA represents the total number of anomalies triggered by the models during test time and ATITA is the quotient of NTA and test time. The two metrics are able to measure the effectiveness of models and the values are powerful arguments for the effectiveness of our framework.

• ***TNR & TPR:*** BLSTM- DCNNFuzz is an extension of DCGAN, and the results show that BLSTM-2DCNN can capture more dependencies in protocol messages. Our method achieves the best results on the test set, where a false-positive rate of 5.95

• ***ATE:*** ATE represents the overall score of the efficiency of models that integrates CTA, ATITA and TNR. Results indicate that our model gets the highest average ATE scores on four fuzzing targets; the model does best on the

Table 2 Experimental Data Comparison

Test Model	Case Amount	Training Time/h	Test Time/h	Targets	CTA	NTA	ATITA	TNR(%)	TPR(%)	ATE Score
BLSTM-DCNNFuzz	270,000	12.13	9.67	Modbus RSSim v8.20	7	**265**	**0.54**	**94.05**	91.17	**1.39**
				Modbus Slave v6.0.2	5	76	1.88			1.06
				xMasterSlave v.156	**8**	109	1.31			1.13
				Real Environment	6	59	2.42			1.04
GAN-based model	270,000	8.57	9.52	Modbus RSSim v8.20	5	86	1.66	92.54	92.39	1.09
				Modbus Slave v6.0.2	6	57	2.50			0.81
				xMasterSlave v.156	5	62	2.30			0.94
				Real Environment	4	23	6.21			0.72
LSTM-based model	270,000	**5.18**	9.51	Modbus RSSim v8.20	4	38	3.76	87.39	**93.25**	0.76
				Modbus Slave v6.0.2	5	21	6.78			0.63
				xMasterSlave v.156	4	18	7.93			0.52
				Real Environment	3	14	10.20			0.47
GPF	270,000	29.05	**9.47**	Modbus RSSim v8.20	2	23	6.21	- -	- -	0.16
				Modbus Slave v6.0.2	2	8	17.84			0.12
				xMasterSlave v.156	3	12	11.89			0.19
				Real Environment	2	9	15.86			0.08

Modbus RSSim v8.20 among the four targets which may mean Modbus RSSim v8.20 is a weak target by comparison.

The concrete manifestation of BLSTM-DCNNFuzz is shown in Table 3, where NTP represents the number of triggered targets. According to the characteristics of the Modbus/TCP protocol data field, the following protocol abnormal feature descriptions which are triggered by our method can be obtained in the first column of Table 3.

The efficiency of the fuzzing test focuses on the vulnerability discovery and the testing execution time. The test efficiency is represented as the number of vulnerabilities divided by the testing execution time. As can be seen from Table 3, compared with the results of other deep learning methods in Table 2, our fuzzing test method has higher frequencies to discover more vulnerabilities. NTP which represents the number of triggered targets can be used as a metric for the code coverage of fuzz testing.

In short, the experimental results in Table 3 present specific experimental data of BLSTM-DCNNFuzz for the four fuzzing targets. It also highlights that BLSTM-DCNNFuzz can not only guarantee the capability of discovering vulnerabilities but also increase the frequency of anomalies triggering and improves test efficiency.

c. DGD. The message categories learned by GPF is constant and the coverages of different GPF are different, so it is not discussed in this part. The testing depth has increased as illustrated in Fig. 9 at the expense of reducing the code coverage of fuzz testing. Therefore, we need to

Table 3 Triggered Anomalies and Triggered Frequency of BLSTM-DCNNFuzz

Triggered Anomalies	Frequency (Times)	ATITA (Mins)	NTP
Slave crash	27	21.11	4
Station ID xx off-line	142	4.01	3
Using abnormal function code	108	5.28	4
Automatically closes the window	53	10.75	4
Data length unmatched	126	4.52	4
Abnormal address	43	13.26	3
Integer overflow	5	114.00	2
File not found	3	190.00	1
Debugger memory overflow	2	285.00	1

maintain high test case diversity on the premise of attaining high TCRR rates. A total of 13 categories of Modbus data frames are prepared in the raw data. When the training epochs is 10, the diversity of the three deep learning based models maintains the best. After training, some message categories are generally lost, as presented in Fig. 10. BLSTM-DCNNFuzz and LSTM-based model have a 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 the BLSTM model containing two sub-networks for the forward and backward sequence context respectively, it is able for the BLSTM-

DCNNFuzz 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 achieve, and the stronger ability a model has to detect anomalies. Thus as presented in Table 3, our trained model can effectively detect kinds of anomalies.

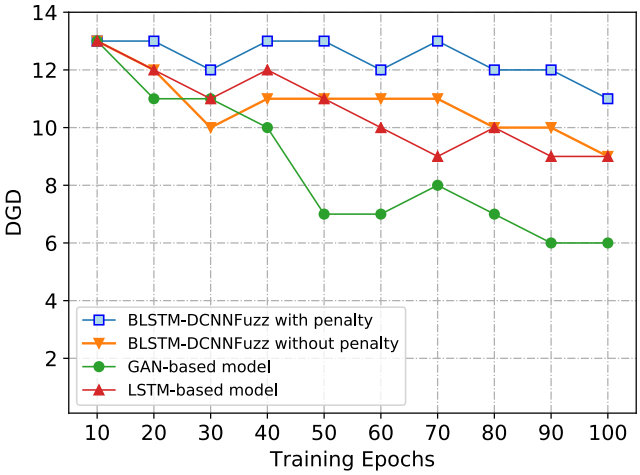

Fig. 10 Data diversity retention

2) Bugs Founded

The result is exciting that we have successfully triggered anomalies. Some anomalies have been defined and described in the detailed protocol specification so we are not surprised by their triggers. But there are some weird anomalies which are not declared before and cause some unexpected problems. The following describes in detail some representative anomalies among these which are probably bugs.

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 send about 3500 data frames, indicating that the program has crashed. We send data frames range from 3450th to 3500th to the other two Modbus implementations, Modbus Slave and xMasterSlave, with no anomaly occurring. This comparison indicates 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. The phenomenon makes us believe that there is an implementation flaw with the slave state judgment.

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 anomalies and all abnormal feedbacks from the three softwares and the real Modbus-TCP network 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.

3) Applying The Method to Another ICP-EtherCAT

As shown in Table 4, 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 EtherCat via the new trained BLSTM-DCNNFuzz. In the experiment, we send the generated data messages S_i to the slave stations and record the corresponding received message R_i. Massive message pairs $< S_i, R_i >$ are obtained. According to the abnormal protocol characterization above, we analyze and compare the specified field values and obtained the following statistical results. Experiments on the EtherCAT protocol prove that our method has great versatility.

Table 4 Potential Vulnerabilities and occurrences times in EtherCAT

Potential Vulnerabilities	NTA	Sent Number
Packet injection attack	118 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	197 times	30,000

Discussion and conclusion

In this study, we propose an effective fuzzing methodology to generate fake but plausible fuzzing protocol messages of ICPs. This methodology utilizes BLSTM and DCGAN to learn the sequential structure of real-world messages and generate similar data frames without knowing the detailed protocol specification in advance. We ultimately evaluate this method by fuzzing two safety-critical ICPs, including Modbus-TCP and EtherCAT. The results indicate that our

methodology has good expansibility and can apply to a series of ICPs.

In future studies, we expect to build a smarter and more automated network protocol fuzzing system deployed in embedded devices. The lightweight 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 further exploration of other architectures to enhance our methodology, such as WGAN-GP and Transformer-XL (Dai et al. 2019). Second, more intelligent log analysis methods and result analysis strategies are required to raise the efficiency of statistical analysis. The ultimate goal of our study is 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.

Acknowledgements This work was supported in part by the Shanghai Science and Technology Committee Rising-Star Program under Grant 18QB1402000, and in part by the National Natural Science Foundation of China under Grant No.61772347 and No.61602178.

References

Aitel D (2002) The advantages of block-based protocol analysis for security testing. Immunity Inc., February 105:106

Banks G, Cova M, Felmetsger V, Almeroth K, Kemmerer R, Vigna G (2006) Snooze: toward a stateful network protocol fuzzer. In: International Conference on Information Security, Springer, pp 343–358

Bell R (2006) Introduction to iec 61508. In: Proceedings of the 10th Australian workshop on Safety critical systems and software-Volume 55, Australian Computer Society, Inc., pp 3–12

Bocaniala CD, Da Costa JS, Palade V (2005) Fuzzy-based refinement of the fault diagnosis task in industrial devices. Journal of Intelligent Manufacturing 16(6):599–614

Böttinger K, Godefroid P, Singh R (2018) Deep reinforcement fuzzing. In: 2018 IEEE Security and Privacy Workshops (SPW), IEEE, pp 116–122

Carpanzano E, Ferrucci L, Mandrioli D, Mazzolini M, Morzenti A, Rossi M (2014) Automated formal verification for flexible manufacturing systems. Journal of Intelligent Manufacturing 25(5):1181–1195

Chockalingam V, Larson I, Lin D, Nofzinger S (2016) Detecting attacks on the can protocol with machine learning. In: 8th Annual EECS 588 Security Symposium

Collins G (2013) Pymodbus documentation

Dai AM, Le QV (2015) Semi-supervised sequence learning. In: Advances in neural information processing systems, pp 3079–3087

Dai Z, Yang Z, Yang Y, Carbonell J, Le QV, Salakhutdinov R (2019) Transformer-xl: Attentive language models beyond a fixed-length context. arXiv preprint arXiv:1901.02860

Devarajan G (2007) Unraveling scada protocols: Using sulley fuzzer. In: Defon 15 Hacking Conference

Feng ZL, Yu JX (2012) Design and implementation of rs485 bus communication protocol [j]. Computer Engineering 20

Godefroid P, Kiezun A, Levin MY (2008) Grammar-based whitebox fuzzing. In: Proceedings of the 29th ACM SIGPLAN Conference on Programming Language Design and Implementation, pp 206–215

Godefroid P, Peleg H, Singh R (2017) Learn&fuzz: Machine learning for input fuzzing. In: 2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE), IEEE, pp 50–59

Goodfellow I, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozair S, Courville A, Bengio Y (2014) Generative adversarial nets. In: Advances in neural information processing systems, pp 2672–2680

Gulrajani I, Ahmed F, Arjovsky M, Dumoulin V, Courville AC (2017) Improved training of wasserstein gans. In: Advances in neural information processing systems, pp 5767–5777

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

Heusel M, Ramsauer H, Unterthiner T, Nessler B, Hochreiter S (2017) Gans trained by a two time-scale update rule converge to a local nash equilibrium. In: Advances in Neural Information Processing Systems, pp 6626–6637

Hodován R, Kiss Á, Gyimóthy T (2018) 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, pp 45–48

Hsiang SH, Lin YW, Lai JW (2012) Application of fuzzy-based taguchi method to the optimization of extrusion of magnesium alloy bicycle carriers. Journal of Intelligent Manufacturing 23(3):629–638

Ioffe S, Szegedy C (2015) Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167

Kingma DP, Ba J (2014) Adam: a method for stochastic optimization. corr abs/1412.6980 (2014)

Levy O, Goldberg Y (2014) Neural word embedding as implicit matrix factorization. In: Advances in neural information processing systems, pp 2177–2185

Lucic M, Kurach K, Michalski M, Gelly S, Bousquet O (2018) Are gans created equal? a large-scale study. In: Advances in neural information processing systems, pp 700–709

Lunkeit A, Schieferdecker I (2018) 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, pp 244–251

Mack MA, Anspach CLA, Lostoski DA, Daily G, Klein PJ, Bliss RE (2012) Industrial operator interface terminal. US Patent App. 29/393,523

Miller BP, Fredriksen L, So B (1990) An empirical study of the reliability of unix utilities. Communications of the ACM 33(12):32–44

Miller BP, Koski D, Lee CP, Maganty V, Murthy R, Natarajan A, Steidl J (1995) Fuzz revisited: A re-examination of the reliability of unix utilities and services. Tech. rep., Technical Report CS-TR-1995-1268, University of Wisconsin

Orebaugh A, Ramirez G, Beale J (2006) Wireshark & Ethereal network protocol analyzer toolkit. Elsevier

Peng H, Li J, He Y, Liu Y, Bao M, Wang L, Song Y, Yang Q (2018) Large-scale hierarchical text classification with recursively regularized deep graph-cnn. In: Proceedings of the 2018 World Wide Web Conference, pp 1063–1072

Peroli M, De Meo F, Viganò L, Guardini D (2018) Mobster: A model-based security testing framework for web applications. Software Testing, Verification and Reliability 28(8):e1685

Piggin R (2013) 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, pp 1–6

Pourjavad E, Mayorga RV (2019) A comparative study and measuring performance of manufacturing systems with mamdani fuzzy inference system. Journal of Intelligent Manufacturing 30(3):1085–1097

Pratama M, Dimla E, Lai CY, Lughofer E (2019) Metacognitive learning approach for online tool condition monitoring. Journal of Intelligent Manufacturing 30(4):1717–1737

Radford A, Metz L, Chintala S (2015) Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint arXiv:1511.06434

Rajpal M, Blum W, Singh R (2017) Not all bytes are equal: Neural byte sieve for fuzzing. arXiv preprint arXiv:1711.04596

Roberts Jr JD, Ihnat J, Smith Jr W (1972) Microprogrammed control unit (mcu) programming reference manual. ACM Sigmicro Newsletter 3(3):18–57

Salamon J, Bello JP (2017) Deep convolutional neural networks and data augmentation for environmental sound classification. IEEE Signal Processing Letters 24(3):279–283

Sun HP, Huang Y, Wang XF, Zhang Y, Shen HB (2015) Improving accuracy of protein contact prediction using balanced network deconvolution. Proteins: Structure, Function, and Bioinformatics 83(3):485–496

Sutskever I, Martens J, Dahl G, Hinton G (2013) On the importance of initialization and momentum in deep learning. In: International conference on machine learning, pp 1139–1147

Utting M, Pretschner A, Legeard B (2012) A taxonomy of model-based testing approaches. Software Testing, Verification and Reliability 22(5):297–312

Voyiatzis AG, Katsigiannis K, Koubias S (2015) A modbus/tcp fuzzer for testing internetworked industrial systems. In: Emerging Technologies & Factory Automation (ETFA), 2015 IEEE 20th Conference on, IEEE, pp 1–6

Wollschlaeger M, Sauter T, Jasperneite J (2017) The future of industrial communication: Automation networks in the era of the internet of things and industry 4.0. IEEE industrial electronics magazine 11(1):17–27

Yue G, Ping G, Lanxin L (2018) An end-to-end model based on cnn-lstm for industrial fault diagnosis and prognosis. In: 2018 International Conference on Network Infrastructure and Digital Content (IC-NIDC), IEEE, pp 274–278

Zhang X, Zhao J, LeCun Y (2015) Character-level convolutional networks for text classification. In: Advances in neural information processing systems, pp 649–657

Zhang Z, Chen S (2017) Real-time seam penetration identification in arc welding based on fusion of sound, voltage and spectrum signals. Journal of Intelligent Manufacturing 28(1):207–218

Zhou P, Qi Z, Zheng S, Xu J, Bao H, Xu B (2016) Text classification improved by integrating bidirectional lstm with two-dimensional max pooling. arXiv preprint arXiv:1611.06639