IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received March 24, 2019, accepted April 7, 2019, date of publication April 15, 2019, date of current version April 24, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2911121

An Intelligent Fuzzing Data Generation Method Based on Deep Adversarial Learning

ZHIHUI LI[1], HUI ZHAO[1], JIANQI SHI[1,2], YANHONG HUANG[1,3], AND JIAWEN XIONG[1]

[1]National Trusted Embedded Software Engineering Technology Research Center, East China Normal University, Shanghai 20062, China
[2]Hardware/Software Co-Design Technology and Application Engineering Research Center, East China Normal University, Shanghai 20062, China
[3]Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, Shanghai 20062, China

Corresponding author: Yanhong Huang (yhhuang@sei.ecnu.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61602178, and in part by the Shanghai Science and Technology Committee Rising-Star Program under Grant 18QB1402000.

ABSTRACT Fuzzing (Fuzz testing) can effectively identify security vulnerabilities in software by providing a large amount of unexpected input to the target program. An important part of fuzzing test is the fuzzing data generation. Numerous traditional methods to generate fuzzing data have been developed, such as model-based fuzzing data generation and random fuzzing data generation. These techniques require the specification of the input data format or analyze the input data format by manual reverse engineering. In this paper, we introduce an approach using Wasserstein generative adversarial networks (WGANs), a deep adversarial learning method, to generate fuzzing data. This method does not require defining the input data format. To the best of our knowledge, this study is the first to use a WGAN-based method to generate fuzzing data. Industrial security has been an important and pressing issue globally. Network protocol fuzzing plays a significant role in ensuring the safety and reliability of industrial control systems (ICSs). Thus, the proposed method is significant for ICS testing. In the experiment, we use an industrial control protocol such as the Modbus-TCP protocol and EtherCAT protocol as our test target. Results indicate that this approach is more intelligent and capable than the methods used in previous studies. In addition, owing to its design, this model can be trained within a short time, which is computationally light and practical.

INDEX TERMS Automated vulnerability mining, deep adversarial learning, fuzzing, security testing, industrial security, industrial control protocol, protocol format learning.

I. INTRODUCTION

Industrial security [1] has been a concerning issue in various countries. It involves the vital infrastructure and a large number of manufacturing industries in the country, affecting national stability and the survival of its people. For instance, the notorious Stuxnet virus attacked a nuclear power plant in Iran, causing substantial damage. Thus, industrial safety has to be maintained. The industrial control industry has various supervisory control and data acquisition (SCADA) systems. To ensure the safety of industrial control systems, we must ensure the safety of industrial control protocols in SCADA systems. As one of the effective methods of identifying vulnerabilities, fuzzing [2], [3] plays a vital role in vulnerability discovery. Most software bugs published by well-known organizations were discovered by fuzz testing. The main idea of fuzzing is to use malicious inputs to stress-test the target program in order to cause abnormal behaviors, such as crashes or exceptions. Fuzzing can also be used to discover vulnerabilities in industrial control protocols. Traditional fuzzing usually builds test cases based on the specifications of the software or the protocol that needs to be tested. However, this method has several limitations. On the one hand, the specification and implementation of the software may not be entirely consistent. Thus, if we test it according to the specification, some details might be disregarded, leaving the details vulnerable to attack. On the other hand, current fuzzing data generation methods rely too much on manual analysis, which is tedious and time-consuming. Accordingly, a fuzzing method that can fully consider actual implementation and reduce test costs needs to be developed.

To address the problems encountered during fuzz testing, we use a WGAN-based [4] approach to generate fuzzing data. WGAN is well known for generating simulated

The associate editor coordinating the review of this manuscript and approving it for publication was Siddhartha Bhattacharyya.

pictures [5], [6] in particular. We can innovate by using WGAN to generate sequence data such as data frames in the ICS. Thus, there is no need to know the protocol specification [7]. The WGAN-based model can learn the spatial structure of the data frames without manual analysis and subsequently generate test data with the same structure as the real ones but has random differences in other aspects such as the variable values. This approach presents many advantages. On the one hand, it requires less effort than usual, which significantly simplifies the fuzzing process, and does not require that the tester be an expert in the field. On the other hand, it is highly adaptable and can be conveniently applied to test other similar protocols. The reasons for choosing WGAN as the basic architecture include the following: (i) WGAN is more stable during the model training process; (ii) The generated data are more diverse and thus exhibits a more extensive test coverage; and (iii) It provides an indicator of the training progress, which can indicate the training degree of the model. We use the method to test the Modbus-TCP [8], one of the widely used industry network protocols. The experimental results are satisfactory. It has a high test case pass rate, requires short training time, and can effectively identify vulnerabilities. In addition, to prove its generality and protocol independence, we also apply it to test EtherCAT protocol [9], another widely used industrial control protocol. The critical contributions of this study include three aspects:

1. We propose a technique based on WGAN for fuzz testing and analyze its application potential. During implementation of the method, in accordance with Occam's razor, we elaborately design the architectural model that would provide lightweight computing on the premise of achieving its goal.
2. Given the similarities of industrial control protocols, we propose a general fuzz testing process that can be used in most industrial control protocols.
3. We demonstrate the superiority of the proposed method by experimental comparison and successfully discover bugs.

The remainder of this paper is organized as follows: Section II discusses the related works. Section III details the analysis of WGAN and its specific application. Section IV presents the specific architectural design based on WGAN for testing industrial control protocols. Section V shows the evaluation results. Section VI concludes this paper.

II. RELATED WORKS

Fuzzing has more than 20 years of development, and practice has proven its effectiveness. Vulnerabilities in Word and Excel programs have been identified using this technology. Despite many contributions to this area over the years, research is ongoing.

In 1983, Vos and Aho [10] formally investigated the effectiveness of feeding a program with random inputs. The result shows that this stochastic input strategy is low-cost and effective. Their study is regarded as the earliest research on fuzz testing. Professor Barton Miller at the University of Wisconsin was the first to propose the term ''fuzzing'' in 1988. Miller *et al.* developed a fuzzing tool to test the robustness of Unix programs [2]. These are the original developments of this technology.

Subsequently, researchers proposed an increasing number of fuzzing techniques. Using these strategies, they developed a large number of fuzzing test suites. The PROTOS [11] test suite proposed by researchers at the University of Oulu uses protocol specifications to produce more structured test data. Aitel improved the SPIKE [12] by using the protocol specifications. The specification describes the format of the data blocks. They generate fuzzing data by filing those blocks with randomly generated data. Other researchers [13], [14] also attempted to establish a specific model to guide data generation. These methods improve the effectiveness and automation of the fuzzing test. However, these methods operate under the condition that the internal operation of the test target has to be understood. From this perspective, researchers design models based on the specification to guide the test data generation. This approach has two shortcomings. First, in practice, understanding the protocol requires time and manual effort. Second, during the implementation of the specification, some engineering details may be added, causing inconsistencies between implementation and specification. Therefore, the methods tend to ignore some details in the fuzzing process.

In a recent study, Rajpal *et al.* [15] from Microsoft applied the sequence-to-sequence neural network [16] model to enhance the AFL [17] (American Fuzzy Lop) fuzzer in which the model attempts to learn the optimal mutation locations in the input files. It helps the AFL fuzzer improve the test effectiveness toward stand-alone programs. They combined deep learning techniques with traditional fuzz testing tools using deep learning as accessorial technology. However, we use it as a core method and apply it to the industrial network protocol fuzzing test. More recently, Lv *et al.* [18] use machine learning to generate high valued binary seed files. Böttinger *et al.* [19] use the Q-learning algorithm to determine which mutation action to perform on a given input. Godefroid *et al.* [20] studied how to leverage neural-network-based learning techniques to learn the grammar for non-binary PDF data objects. These studies enhanced fuzzing with machine learning from different perspectives.

In general, the state-of-the-art method [21] of fuzzing is the model-based fuzzing, which leverages the model extracted from the grammar or format of the regular input to guide the fuzzing data generation. Model extraction usually requires human intelligence to analyze. However, our method does fuzz testing from another perspective, which is different from the state-of-the-art method in methodology. It hardly needs manual analysis to extract models. Therefore, it has some corresponding advantages. To show it more clearly, we tabulate the advantages of the proposed method compared to the model-based method. The details are shown in Table 1.

TABLE 1. Advantages of the proposed method.

#	Advantage
1	Can work under the situations where the target specification is unknown.
2	Consider the actual operating details of the test target.
3	Generate test cases without manual analysis.
4	Save testing time.
5	Simple operation and low requirements for testers.

III. BACKGROUND

As the foundation of both GAN and WGAN, the background of deep learning [22] is presented. Other preliminary knowledge of GAN and WGAN are then introduced.

A. DEEP LEARNING

Deep learning, a branch of machine learning, developed slowly in the early stages. Its rapid growth started in 2006 when Hinton and Salakhutdinov published an article [23] providing a solution to key problems. Krizhevsky *et al.* also proposed ImageNet [24], a neural network structure for image classification, in 2012, which marked the revolutionary development of deep learning.

Deep learning is currently drawing interest from the technology community, exhibiting great prospects for application and even surpassing their human counterparts in some areas. Most industries use this technology in intrusion detection [25], [26], self-driving cars [27], [28], robots [29], speech recognition [30], [31], machine translation [32], [33], and so on. With its critical role in the era of intelligence, it tends to liberate human brains. Its rapid development benefits from advances in recent hardware equipment such as Graphics Processing Unit (GPU) [34]. These pieces of equipment have markedly improved computing capacity. The massive empirical data brought about by the development of information technology also accelerated its development. The neural network is the base of deep learning. As presented in Fig. 1, a simple neural network includes one input layer, one or several hidden layers, and one output layer. Complex networks such as the widely known Deep Residual Nets [35] reach 152 layers deep. The neural network [36] learns from a large amount of data and forms a model that serves a specific purpose. With regard to the image classification problem, the trained model outputs the image category on the basis of an input image. The amount of data significantly affects the formation of an effective neural network model. The larger the data, the stronger the ability of the trained model to generalize. In the current study, we use the deep neural network as the basic model architecture.

B. GENERATIVE ADVERSARIAL NETS

In 2014, Goodfellow *et al.* proposed generative adversarial nets [37]. This innovation is considered a controversial subject in deep learning and has been widely applied in numerous applications, such as image generation and super-resolution [38]. With regard to image generation, the trained

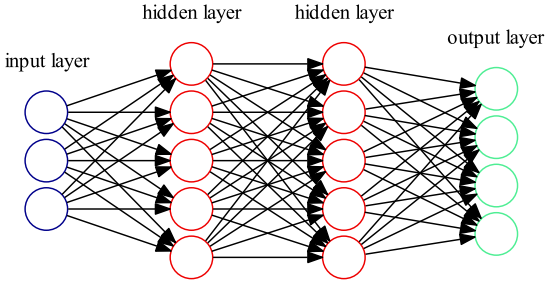


FIGURE 1. A simple multilayer perceptron.

GAN model uses random noise as its input and then outputs simulated pictures. However, simulated pictures vary from real pictures. Thus, we use this feature to generate simulated sequence data frames. By training the model with a large number of data frames, the model can grasp the format of the data frames. A GAN framework consists of the generative model and the discriminative model. The generative model generates data, sharing the same characteristics with the real data. The discriminative model distinguishes between the real data and fake data. After constant game and adjustment between the two models, a Nash equilibrium [39] is finally reached between the two models, and the discriminator does not distinguish between the real data and the generated data. When we train a single neural network, a loss function [40] indicates how well the model is trained. The training process continuously optimizes the loss function. In this process, we determine the global optimum only for one loss function. However, in training the GAN model, we need to dynamically optimize two loss functions at the same time. The following formula formalizes the objective function that needs to be optimized for the generator and the discriminator:

$$\min_G \max_D V(D,G) = E_{x\sim p_{data}}(x)\left[\log D(x)\right] + E_{z\sim p_z}(z)\left[\log(1-D(G(z)))\right] \quad (1)$$

where D represents the discriminator, and G represents the generator. $D(x)$ denotes the probability that x is real data, and $G(z)$ is the probability distribution of the sample data z. The overall optimization goal is to play a minmax game with the value function $V(D, G)$, which needs to maximize the probability of making correct discrimination as well as minimize the probability that the generated samples are discriminated.

Despite its wide use, the application of GAN has several disadvantages, such as non-convergence, vanishing gradient [41], and mode collapse [42]. To avoid these problems, a rational architectural design of the neural network model is needed. In addition, various model training strategies have to be adopted. In the aforementioned problems, the mode collapse indicates that the generated data lose some categories compared with the raw data. This problem needs more attention because, in fuzz testing, the diversity of generated data affects the test coverage. Improving the test coverage helps discover more bugs.

C. WASSERSTEIN GAN

Studies [43] attempting to solve the aforementioned problems have been conducted. Arjovsky *et al.* [4] proposed Wasserstein GAN, which solved this problem. Compared with GAN, Wasserstein GAN has the following characteristics:

First, Wasserstein GAN uses the Earth Mover's Distance (EMD) to measure the difference between two distributions. The definition is as follows:

$$W(P_r, P_g) = \inf_{\gamma \in \prod(P_r, P_g)} E_{(x,y)\sim\gamma} [\| x - y \|] \tag{2}$$

where P_r and P_g represent two distributions, γ represents the joint distributions of the two. We need to calculate the expectation value for the minimum distance between the two samples under the joint distribution γ. This expectation value is the Wasserstein distance. The Wasserstein metric is an efficient indicator to guide the training process. The smaller the value, the smaller the Wasserstein distance between the true distribution and the generated distribution, indicating that the two models are better trained. In Wasserstein GAN, the loss function of the discriminator is

$$-E_{x\sim P_r}\left[\log D(x)\right] - E_{x\sim P_g}\left[1 - \log D(x)\right] \tag{3}$$

and the loss function of the generator is

$$E_{x\sim P_g}\left[1 - \log D(x)\right] \tag{4}$$

During training, the losses of the generator and the discriminator are continuously minimized. These losses eventually become optimal.

Second, weight clipping is used to satisfy the Lipschitz continuity, which can accelerate the model convergence and stabilize training. Generally, after each parameter is updated, weight clipping truncates the weight parameters to no more than a fixed constant C.

Third, an optimization method [44] may be selected from a wide range of techniques to optimize the loss function. Different methods have different characteristics. Momentum-based optimization algorithms, such as Momentum SGD [45] and Adam [46], cannot keep the training stable in all situations; thus, they are not used in selecting the optimization method.

IV. FUZZING SYSTEM DESIGN

In this section, we first present an overview of the general fuzzing method based on WGAN and then elaborate on the main aspects. The overall architecture of the entire running process is illustrated in Figs. 2 and 3.

A. OVERVIEW

Our method aims to intelligently learn the format from raw network packets fetched from given industrial control networks. After learning, we can obtain a concrete generation model that can generate well-formed data frames similar to the real ones. We then send the generated data frames into the target system and monitor the entire system in time to record abnormal behaviors. This processing approach can be used to test most current industrial control protocols. The various stages of this method are described as follows:

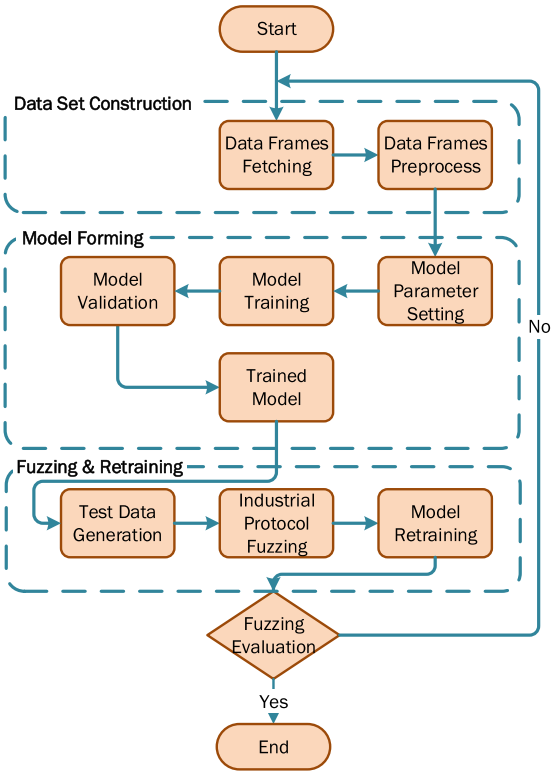


FIGURE 2. General fuzz testing process towards industrial control protocols.

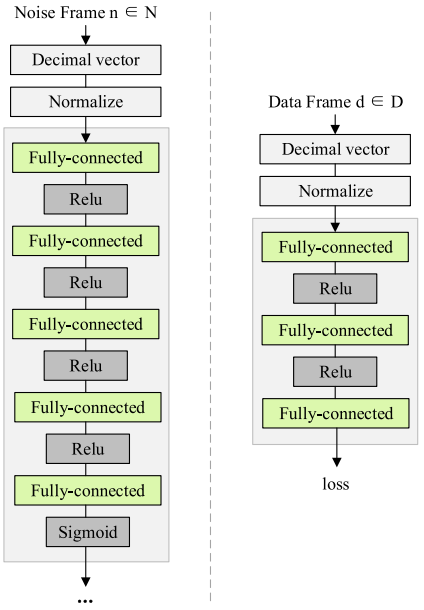


FIGURE 3. Architecture design of generator (left) and discriminator (right).

1) DATA FRAMES PREPROCESSING

The raw network data packets are in a hexadecimal form which cannot be directly inputted into the neural network for

processing. Preprocessing the raw data requires two steps. First, we transform the raw network data packets into digital vectors. Second, we use clustering strategies as well as data augmentation strategies to process the data packets, which can help improve the diversity of the final generated data.

2) ADVERSARIAL TRAINING

We formalize this learning problem into a process of solving a mathematical function. Our initial goal in training a generator model based on WGAN is to obtain a function that can generate fake data sharing the same data format with the real world data. Ultimately, we aim to conduct attack testing and discover the vulnerability of the target system. To achieve this goal, we need a specially designed neural network architecture. Thus, the final trained model does not only generate more aggressive testing data but also calculates lightweight.

3) FUZZ TESTING AND RETRAINING

When the generator model is achieved as planned, we can obtain any number of test cases by running the model. We then use these test cases to attack the target system. After this attack, we retrain the model with test cases that cause anomalies.

We elaborate on the aforementioned three stages in the following sections.

B. DATA FRAMES PREPROCESSING

Prior to model training, numerous real-world data frames are first obtained. The method of performing preliminary processing on these data can affect the capability of the final model to discover vulnerabilities. Ultimately, we aim to obtain desirable fuzzing results. Therefore, data preprocessing should also help achieve this goal. A common approach is to improve the test width and depth, which can help achieve enhanced fuzzing results. A useful technique to increase the test depth and width is by increasing the diversity of the test cases. We adopted the following methods to preprocess the data frames. These methods can not only make the model learn the data format but also maintain the diversity of data.

- Data Frame Clustering. For the model to enhance its learning of the message format, we use several clustering methods, such as frame length clustering and K-means clustering, to cluster the data. These techniques can classify data frames in similar formats. Frame length clustering is effective because data frames with the same length always tend to share the same frame type. K-means clustering is more inclined to group frames with the same function. Adopting these clustering methods during preprocessing can contribute to the establishment of a model that can elucidate the data frame structure.
- Data Frame Augmentation. In training deep learning models, data augmentation [47] methods are often used to prevent overfitting. In this study, we use this strategy to maintain the generated data diversity. In real-life scenarios, collected data are not evenly distributed-some data frames are very few and can even be ignored. For more diverse generated data, we deliberately increase the proportion of these data frames. Maintaining data diversity can help increase test depth and test breadth.

We performed these methods during the experiment.

C. ADVERSARIAL TRAINING

In this section, we focus on the adversarial learning process. We first abstract the mathematical expression of this data frame learning problem. We then detail the specific structure of the designed WGAN-based model. Finally, we present the training process.

1) PROBLEM ABSTRACTION

The raw data frames we obtain from the ICS is in the form of a sequence. We can naturally extract the mathematical expression $S_{1:n} = (e_1, e_2, \ldots, e_x, \ldots, e_n)$, $e_x \in E$ and $S_{1:n} \in S^*$, where E represents a collection of hexadecimal numbers and S^* is the sequence set, including $S_{1:n}$. Most elements are English letters and numbers. We aim to find a function f_T that captures the distribution of these real-world data frames and can generate data frames with the same characteristics.

The WGAN model includes the generator model and the discriminator model. We formalize the generator model as a function f_G which uses Gaussian noise as its input and then outputs a sequence $S_{g|1:n}$. We view the discriminator as a function f_D, which distinguishes between real data S_r and generated data S_g. The EMD, as described in Equation (2), can indicate the training degree. The purpose of the training process is to continuously optimize the loss function of the discriminator and the loss function of the generator. Finally, the function f_G almost shares the same distribution as the function f_T.

The aforementioned formal description is the modeling of the adversarial learning problem in this study.

2) MODEL DESIGN

This subsection details the structural design of the two models. One of our design philosophies is to simplify the model as much as possible on the premise of attaining its effect. Owing to its ability to reduce the consumption of computing resources, the simplified model can be conveniently deployed to embedded devices in the future. Fig. 3 illustrates the architecture of the discriminator and generator.

The generator in this experiment consists of an input layer, three hidden layers, and an output layer. Regarding the neural network structure of the discriminator, in each layer, we use a fully-connected neural network structure as described in Fig. 3. This structure can make the generator model much lighter in calculations. Regarding the size of the data input to the neural network, we set it according to the max frame length in the ICS. We will align frames of

different lengths to the same length. Thus, the neural network can uniformly process the data. In the middle layers, we select rectified linear units (ReLU) as the activation function [48]. The reason for choosing ReLU as the activation function lies in these aspects. First, it can avoid vanishing gradient problem in the backpropagation of model training. Second, it can make the output of some neurons become zero, which can help avoid overfitting. Third, it can reduce computational cost. In the output layer, we select the sigmoid function [49] as the activation function because its output is between 0 and 1. The input for the generator is Gaussian distributed noise, which is a common practice. We initialize the weight matrix according to a specific standard deviation. We remove the last log operation in accordance with the theory of WGAN. We choose the Root Mean Square Prop algorithm to optimize the loss function. The algorithm can perform jitter reduction during parameter adjustment and accelerate the training.

The discriminator includes one input layer, one hidden layer, and one output layer. Regarding the neural network structure of the discriminator, we also use a fully connected neural network. The entire discriminator contains only three layers. This relatively shallow network can save computing resources. The input data size of the discriminator is the same as the output size of the generator. The output layer of the discriminator removes the sigmoid function in accordance with the theory of WGAN, which varies from most discriminator models. In each training epoch, after updating the weight matrix, we clip all weight values to a fixed range to satisfy the Lipschitz continuity.

One of the purposes of this architecture design is to reduce the computational cost [50] of the model. In this study, two factors are closely related to computational cost. (i) One is the model architecture complexity [51] such as the network structure and the parameter amount. (ii) The other is the data set complexity. Complex data sets contain more features to learn. Here, we focus on the model architecture complexity. First, since deeper networks increase the amount of basic multiply-accumulate operations, the above designing adopts a shallower network structure, which can contribute to reducing the computational cost. Second, if we encode the one-dimensional ICP data frame into a two-dimensional matrix, the matrix will be very sparse. Dealing it with convolution network will increase the computational cost. Moreover, the convolutional operation involves a large number of parameters, which takes up much memory. Therefore, without convolutional operations, our model design can reduce the computational cost. Regarding the quantitative analysis of the computational cost, we show the time consumed under the same computing hardware and the same computing target in the experiment.

3) MODEL TRAINING

Once the architecture of the model is designed, we begin to train the model. Adversarial training [52] is conducted to continuously optimize the loss function of the discriminator and the loss function of the generator. In this optimization process, the values in the weight matrix are adjusted continuously, leading to a stable model that can learn the structure of real-world data frames. Owing to the proper design of the architecture based on WGAN, both models can be trained simultaneously without elaborately arranging the training order. We divide the training data into equal-sized batches as inputs until all training data are exhausted. This process runs for several epochs until highly realistic data can be generated.

Generally, we want the generated data to be similar to the real data as much as possible. However, our ultimate goal is to achieve effective fuzzing results and identify as many bugs as possible. To obtain improved fuzzing results, we need to maintain the difference between the generated data and the real data. Therefore, we deliberately save the generator model for each 10 training epochs and not only the final best-trained model. Using this strategy, we can generate test data with different degrees of similarity. In addition, we cannot only influence the data to conform to the protocol-specific data packet structure but also improve the diversity in content of the test data. Thus, we can increase the fuzz testing width and depth [53].

4) MODEL VALIDATION

In the formation of the final model, another important step is model validation. This step is carried out at the end of the model training. We use it to assess the model performance and adjust the hyper-parameters. Model validation validates the model with validation data set which shares the same distribution with the training data set. On the one hand, it prevents the model from over-fitting the training data. On the other hand, it helps us determine the optimal hyper-parameter such as the learning rate and the boundary value of weight clipping in this study.

After the model training and validation, we obtain the generator model that can generate data frames sharing the same format with the real ones. Simultaneously, a specific difference in variable values exist. Formally, the function f_G that represents the trained generator almost exhibit the same distribution as that of the target function f_T.

D. FUZZ TESTING AND RETRAINING

At this point, we can generate any number of test data frames by using the trained generator model. These data frames vary in type similar to the real-world data frames. We send test data frames to the test target and record any possible responses. During this sending and receiving process, one of the program modules outputs the log files recording all communication processes and abnormal behaviors. This log file provides the basis for experimental analysis. We also manually analyze the entire process, in addition to program monitoring, to discover potential anomalies. During fuzzing, data frame sequences that cause system anomalies are found and then recorded separately for model retraining.

After the first round of fuzz testing, we retrain the model with error-triggering sequences. The logical basis for this operation is the principle of temporal locality and spatial

locality in computer science. To conduct the retraining, we perform two steps on the recorded sequences. First, we apply the mutation operation to the sequence, including single-point mutation and multi-point mutation. In detail, the mutation operation refers to the random modification of the sequence data on one or more random positions. Second, we use data augmentation to increase the effect of these particular sequences because a small data ratio exerts no substantial effect on retraining the model. After these steps, we train the model again with the updated training data set.

V. EXPERIMENT

In this section, we evaluate the effectiveness of the proposed method by experimentation. In a previous study [54], we trained a GAN-based fuzzing data generation model. Therefore, the subsequent analysis compares the WGAN-based approach with the GAN-based approach. To analyze the experiment results from various dimensions, different Modbus-TCP implementations, including MOD_RSSim v8.20, Modbus Slave v6.0.2, and Diasalve v2.12 are used as fuzzing targets in our experiment. These programs are designed in accordance with the Modbus protocol specification. Lastly, to show the aforementioned adaptability of the method, we employ it to test the EtherCAT protocol, another industrial control protocol.

A. EVALUATION METRICS

In this subsection, we introduce the performance metric about the training and the fuzzing comparison. The evaluation [55] of deep adversarial learning, especially the GAN, is not an easy task [56]. Some quantitative criteria [57]–[59] 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 flowing metrics. Among them, *TIAR* and *DGD* serve as the training performance metrics, and the rest serve as the fuzzing effectiveness metrics.

1) TEST INPUTS ACCEPTANCE RATE (TIAR)

TIAR refers to the percentage of test cases accepted by the test target. It reflects the efficiency of test data generation. A higher *TIAR* indicates a higher similarity of the generated test case to the real-world data frames with respect to format. Conversely, a lower *TIAR* indicates a lower quality of the generated data, requiring adjustment of the model architecture or retraining the model. Therefore, we select it as one of the performance metrics in model training and validation. We defined *TIAR* as follows:

$$TIAR = \frac{nAccept}{nSent} \times 100\% \tag{5}$$

where *nSent* is the total number of test cases sent, and *nAccept* is the total number of test cases accepted. In the experiment, we adjust two hyperparameters to obtain a higher test pass rate. These two factors are the number of training epochs and the value of the loss function. Moreover, during the model training or validation, we use it as an indicator to adjust other hyperparameters such as the learning rate and the boundary value of weight clipping.

2) ABILITY OF VULNERABILITY DETECTION (AVD)

AVD [54] refers to the ability to find vulnerabilities, which is the most straightforward measure of the effectiveness of the method. We counted not only the number of vulnerabilities found in the fuzzing test but also the number of test cases used identify these vulnerabilities. It is the average number of test cases required to reveal a bug. The discovered vulnerabilities are also closely related to the testing target. If the target has more vulnerabilities, this value may increase accordingly. To facilitate a comparative analysis, we only focus on the effectiveness of the method itself in the experiment. The specific formula is as follows:

$$AVD = \frac{nBugs}{nCases} \times 100\% \tag{6}$$

where *nBugs* indicates the number of bugs found, and the denominator *nCases* is the number of test cases used. It can be informally regarded as the number of bugs found per hundred test cases. The larger the value, the greater the ability to discover vulnerabilities; the lower the value, the less the ability to discover vulnerabilities.

In addition, not all the fuzz testing techniques can find all the vulnerabilities in a target system. The proposed approach does not guarantee the discovery of all vulnerabilities in the testing target. Therefore, this metric just count on the occurred vulnerabilities.

3) DIVERSITY OF GENERATED DATA (DGD)

DGD refers to the ability to maintain the diversity of the generated data. More diverse generated data frames are likely to cause an exception. This indicator focuses on the number of data types in the generated data. When the number of the generated data types is significantly smaller than that of the training data types, it means that the model performs poorly and needs to be adjusted. Therefore, we select it as another model performance metric. Furthermore, studies [60], [61] consider that diversity-based approach is a useful test case selection criterion. Thus, it is also an important indicator of the method effectiveness in this study. *Code coverage* is also a good test case selection criterion. However, the test target in this study has no source code. Therefore, we use the *DGD* to evaluate the method.

4) TEST CASE GENERATION PER HOUR (TCGPH)

TCGPH directly indicates the test cases that the model can generate per hour.

$$TCGPH = \frac{generatCases}{hoursSpent} \tag{7}$$

where *generatCases* indicates the number of test cases generated by the model, and *hoursSpent* indicates the time it takes to generate these test cases. This equation reflects the speed of test case generation.

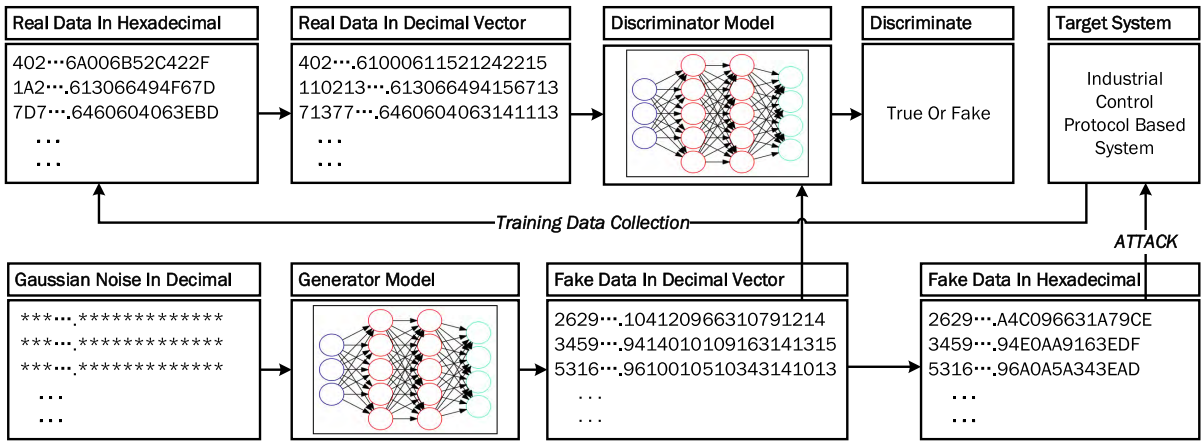


FIGURE 4. The architecture of the entire fuzzing system.

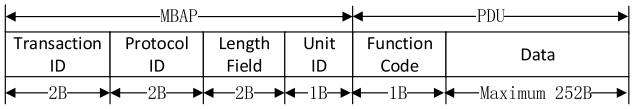


FIGURE 5. Message format of modbus-TCP.

5) TRAINING TIME (TT)

TT refers to the time consumed to establish the final model. A shorter training time can improve the test efficiency. Different neural network structures, different training data volumes, different computing hardware, and different calculation algorithms typically influence the training time. In this study, we only focus on the effect of different model architectures on training time.

B. MODBUS-TCP AND ETHERCAT

1) MODBUS-TCP

Modbus is a serial communications protocol initially published by Modicon in 1979 for use with its programmable logic controllers. This protocol has become a de facto standard communication protocol and is currently the available means of connecting industrial electronic devices. Its safety is of great significance to the security of the industrial control industry. Modbus protocols have many variants, including Modbus TCP and Modbus UDP. In this study, we use Modbus-TCP as the fuzzing target. To elucidate this study, we briefly introduce the Modbus-TCP frame structure. As illustrated in Fig. 5, the data have a fixed format. In the communication process, if the data format is incorrect, the receiver does not accept the data. The Modbus-TCP protocol is implemented by calling the TCP protocol. The message with the format is transmitted as the data component of the TCP data frame. This component is what the model needs to learn and generate.

2) ETHERCAT

EtherCAT is a protocol offering high real-time performance and provides a master-slave communication mode among the industry devices. It has grown to be a market winning technology due to its high real-time performance and openness. A typical EtherCAT network consists of one master and several slaves. The master generates telegrams and sends them to the whole network. These telegrams are reflected at the end of each network segment and sent back to the master. Here, we apply our fuzzing method to learn and generate these telegrams.

C. TRAINING DATA AND VALIDATION DATA

Training data and validation data in deep learning significantly influence the model forming. Thus, we need to accurately collect and preprocess these data. In the experiment, we separately collected data from two industrial control protocol communication environments. One is based on Modbus-TCP protocol, and the other is based on EtherCAT protocol. In the division of these data, we select 80% as the training data, and the remaining 20% as the validation data.

1) MODBUS-TCP

In order to obtain Modbus-TCP communication data, we built a Modbus-TCP-based communication environment with Modbus Poll 4.3.4, Modbus Slave 4.3.1 and VPSD. Modbus Poll works as the Modbus-TCP master station, and Modbus Slave works as the slave station. When the master station communicates with the slave, we use Wireshark [62], a widely-used network protocol analyzer, to capture the communication data. Specifically, we captured 1000,000 pieces of data, including various types, 80% as the training data and 20% as the validation data.

2) ETHERCAT

In order to capture the EtherCAT communication data, we prepare an EtherCAT based industrial system as illustrated in Fig. 6. The master station is a Beckhoff [63] industrial PC and the slave station includes EK1100 [64], EL1004 [65] and EL2004 [66]. We use ET2000 [67] as the online listener between the master and the slaves. The Wireshark, running on

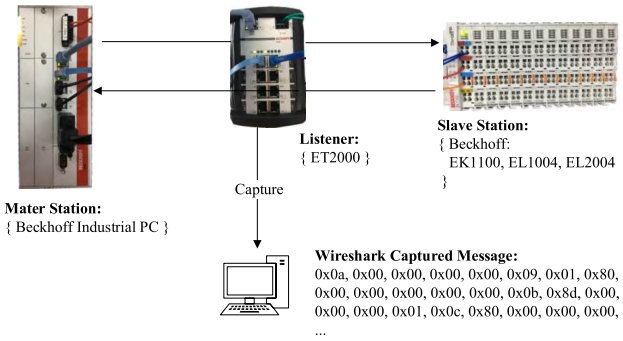


FIGURE 6. EtherCAT protocol communication environment.

a computer, can fetch and display the massive communication data messages from the listener. After processing, massive data messages will serve as the training data for the EtherCAT protocol.

D. EVALUATION SETUP

1) EXPERIMENTAL ENVIRONMENT

In the deep learning problems, the model training often heavily consumes computational resources. Thus, we introduce the experimental environment to elucidate the proposed technique. The model is trained on a machine with 8 processors (Intel(R) Core(TM) i7-6700K CPU@4.00GHz) 16.0GB memory(RAM) Nvidia GeForce GTX 1080 Ti(11GB), and 64-bit Microsoft Windows 10 Professional Operating System, x64-based processor. 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) and 64-bit Microsoft Windows 10 Professional Operating System, x64-based processor.

2) MODEL TRAINING SETTING

The adversarial networks contain two neural network models, which require different data inputs. Both the real-world data messages and the generated data messages are input into the discriminator model. Gaussian noise [0, 1] is input into the generated model. The learning rate is set to 0.001 for a total of 40 training epochs. After each parameter update, we clip the weight value to $[-0.01, 0.01]$. We save the generator model for every 10 epochs. These models can generate more test cases. Using the CPU to train the model requires 10.7 h of training time for 20 training epochs, with low requirements for hardware resources and time resources. Using the GPU requires only 90 min of training time for 20 training epochs, which is highly efficient.

E. MODEL VALIDATION RESULT

At the end of model training, we use validation set to validate the model performance. Thus, we can determine whether the model is overfitting on the training set as well as select the model hyperparameter. According to our research purpose and the actual situation, we select the *TIAR* and the *DGD* as the validation metric.

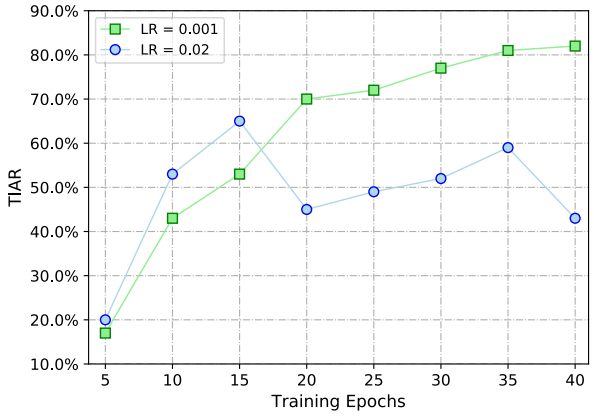


FIGURE 7. Model validation under different learning rates.

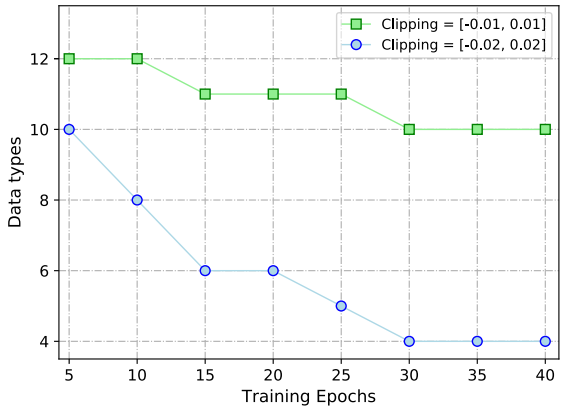


FIGURE 8. Model validation under different weight clipping values.

TABLE 2. Triggered exceptions and triggered frequency in Modbus-TCP. A comparison between GAN-based model and WGAN-based model. The fixed variable is the amount of test cases sent and the same test target, and the indicator is the number of abnormalities triggered.

Exception Item	GAN Based	WGAN Based	Sent Number
Station ID XX Off-Line	7 times	73 times	30,000
Slave Crash	5 times	21 times	30,000
Automatically Close The Window	0 times	23 times	30,000
Software Caused Connection Abort	0 times	31 times	30,000

In detail, we run the model on the validation set and count the *TIAR* metrics under different learning rates such as 0.02 and 0.001. Results in Fig. 7 shows that the *TIAR* index can be steadily improved when the learning rate is 0.001. Therefore, it is appropriate to set the learning rate to 0.001.

We also run the model on the validation set under different weight clipping values. In detail, we set the weight clipping values to [-0.01, 0.01] and [-0.02, 0.02] for the validation. Fig. 8 shows the statistics of the *DGD* indicators. It shows that the model is better at maintaining the *DGD* when the value is set to [-0.01, 0.01].

F. EXPERIMENT RESULTS

In this section, we elaborate on the experimental results in two aspects to prove the effectiveness of the method. We first present the exceptions in the experiment. We then reveal some statistical results and its analysis. In these two aspects, a comparison with our previous study is conducted. We ultimately compare our proposed method with the traditional method. In addition, we show the testing results of the EtherCAT protocol, another industrial protocol.

1) EXCEPTION FOUNDED

In this study, we have two kinds of model: the GAN-based model and the WGAN-based model. For comparison, we send the generated data frames to the same three types of Modbus slaves. We successfully triggered exception by using both models. The WGAN-based model exhibits enhanced capability and speed in finding errors, as described in Table 2. We sent 30,000 test cases generated by the two models. The WGAN-based model causes more errors, compared with the GAN-based model. The following describes some of these errors in detail.

Attacking the Modbus_Rssim causes it to crash. We send about 700 data frames, and a pop-up window prompt box pops up, indicating that the program has crashed. To identify the cause, we pack the 700 data frames and then send them to the Modbus Slave. However, no abnormality occurs. This comparison indicates that the Modbus_Rssim has a defect in its implementation.

In further attacks, we find that the Modbus_Rssim prompts abnormal information displaying the message. ''Station ID XX off-line, no response sent.'' Such an anomaly occurs several times within a short period

In fuzz testing the Modbus Slave, we find another exception that after sending about 1000 data frames, the target program automatically closes off the window itself. We identify memory overflow as the cause, which suggests that the programmer may not consider the extreme situation when implementing the simulator.

Other anomalies such as ''file not found'' and ''debugger memory overflow'' are found when attacking the Diaslave. These anomalies occurred only once or twice and thus are not discussed in detail. It should be noted that different abnormal behavior may be caused by the same cause. In the study, since the test target has no source code, we do not further distinguish between the essential causes behind each anomaly.

In general, the WGAN-based model has more triggered exceptions and more triggered frequencies, compared with the GAN-based model.

2) TIAR

After counting the *TIAR* in the whole process, we can see that *TIAR* rises with increasing training epochs, as depicted in Fig. 7. This behavior indicates that an increasing volume of generated data have the correct message format. These data will be accepted by the fuzzing target. We choose Modbus Slave as the target. If it accepts the data normally, the data format is potentially correct. Compared with the GAN-based model, the WGAN-based model can reach a higher *TIAR* in the final stages, which reveals that the customized WGAN-based model exhibits higher format accuracy compared with the GAN-based model. The highest point of *TIAR* increases to about 85%. Part of the data format remains incorrect in the experiment. Initially, TIAR increases significantly; with further training, it tends to increase slowly and eventually flattens, as shown in Fig. 9.

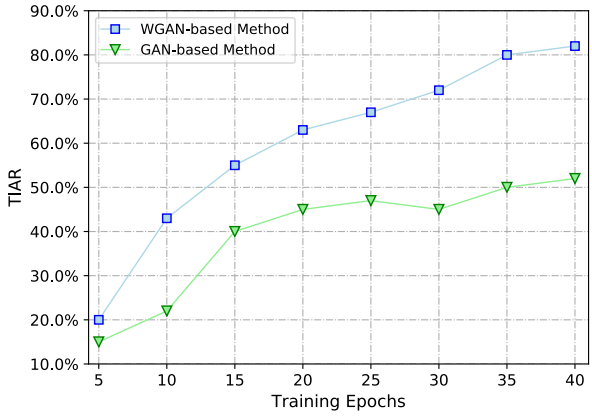


FIGURE 9. WGAN-based method vs GAN-based method on TIAR.

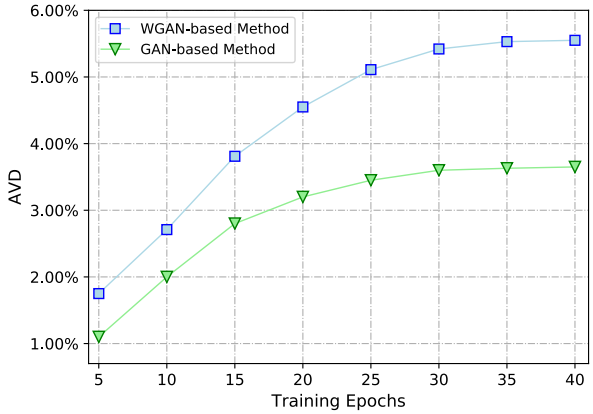


FIGURE 10. WGAN-based method vs GAN-based method on AVD.

3) AVD

AVD increases with the increase in training time, which indicates that an increasing number of errors are discovered. It ultimately reaches a stable flat stage. In practice, the level reached is not only related to the experimental method but to the test objectives as well. If the target to be tested includes numerous bugs, the peak of the line in the graph will be higher. In the experiment, Modbus Slave is chosen as the test target. Fig. 10 indicates that the customized WGAN-based model for finding bugs is more capable than the GAN-based model when dealing with the same protocol. This inference verifies the effectiveness and potential of our approach.

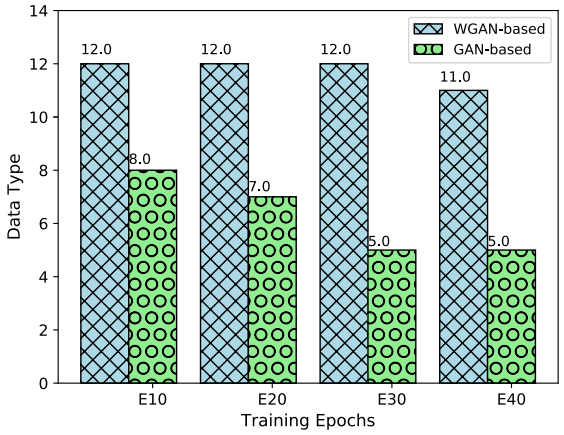


FIGURE 11. WGAN-based method vs GAN-based method on DGD.

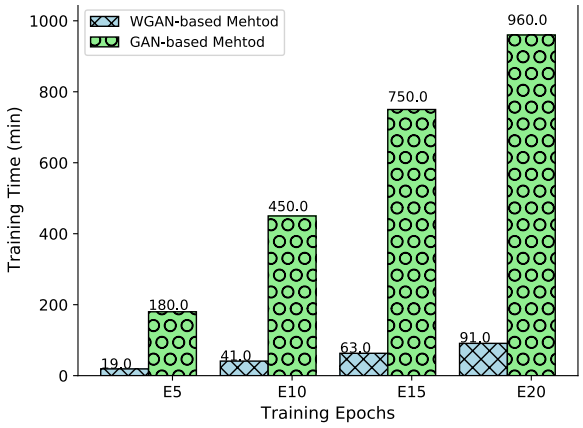


FIGURE 12. WGAN-based method vs GAN-based method on TT.

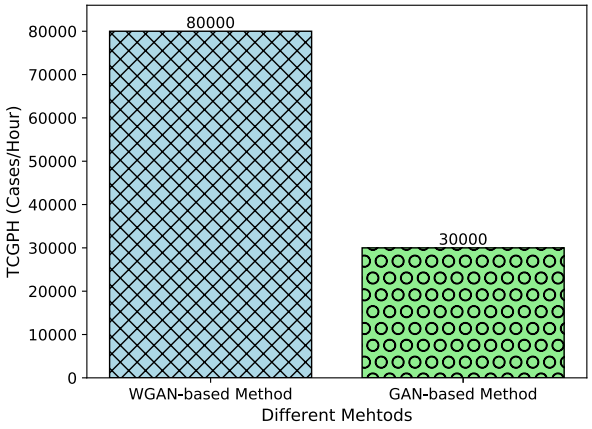


FIGURE 13. WGAN-based method vs GAN-based method on TCGPH.

4) DGD

To compare with GAN-based model, a total of 13 types of data frames are prepared in the original training data. Fig. 11 shows that the WGAN-based model maintains the diversity of the original data. After training, the WGAN-based model still generates 12 types of messages. However, the GAN-based model does not maintain the diversity as in the original data. Therefore, the WGAN-based model has an advantage over the GAN-based model in maintaining the diversity of data. Usually, the richer the data type, the stronger the ability to detect anomalies. Thus, the WGAN-based model can detect more bugs as presented in Table 2.

5) TT

We use the same amount of training data and the same GPU to train the two different models. Finally, we compare the length of time consumed for the training. The results are shown in Fig. 12. The GAN-based model requires longer training time, compared with the WGAN-based model. This difference is mainly attributed to the better design of the proposed model compared with that of the GAN-based model. Furthermore, this comparison proves that our targeted design, to reduce the computational cost, has already achieved a corresponding effect.

6) TCGPH

We use the same computing hardware to generate test cases. We finally compare the TCGPH of different models. The results are shown in Fig. 13. The proposed model shows increased efficiency in generating test cases. Fast generation suggests that we can obtain many test cases within a short time, allowing the quick testing of the test targets. This capability can improve the testing efficiency.

7) COMPUTATIONAL COST

To quantify the computational cost, we make a comparison between the two models on time consumption under the same computing hardware and the same computing target.

TABLE 3. Computational Cost under the same computing hardware.

Computing Hardware	WGAN Model	GAN Model
CPU (2.30GHZ 16GB)	15.3 h	57 h
GPU (1080Ti 11GB)	132 min	490 min

The model consumes a short time in the calculation, indicating that it can reduce the computational cost. In the comparison, we show the time spent on training the model for 30 training epochs. Regarding the computing hardware, we use CPU and GPU separately. The specific time consumption is shown in Table 3. It shows that our neural network structure uses less computation time under the same computational goals. Therefore, the model we designed can reduce the computational cost.

8) COMPARISON WITH THE TRADITIONAL METHODS

Traditional methods rely on professionals to design the test cases based on the already known message format. The entire process may take several days, whereas our proposed method only takes several hours. Thus, the proposed method is more automatic and faster than the manual method. In contrast to the traditional method, the proposed method mainly

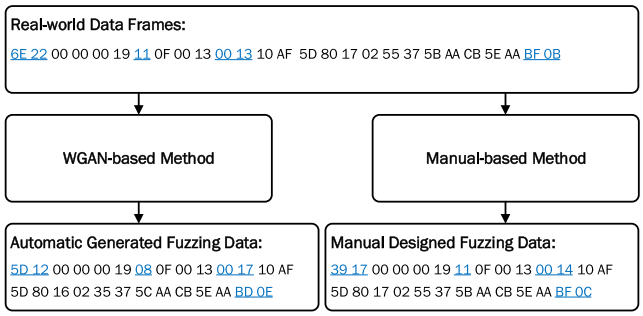


FIGURE 14. WGAN-based method vs traditional method on generating Modbus-TCP data frame.

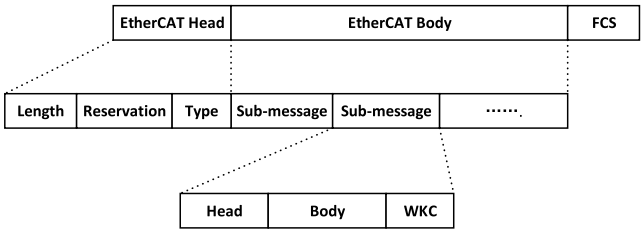


FIGURE 15. Frame structure of EtherCAT. A frame includes several sub-messages.

compares the format and content of the test data generated by the proposed method with those of manual methods. Fig. 14 shows that the test data frames generated are similar to the artificially designed data frames. The blue mark in the sequence data shows some variations on the original data. Although not all the generated data are as precise as those generated by the manual design, they can compensate for this deficiency in quantity. Therefore, our proposed approach exhibits great potential to replace traditional methods.

9) APPLY THE METHOD TO ANOTHER INDUSTRIAL PROTOCOL-EtherCAT

To demonstrate the versatility of the method, we retrained the model with massive EtherCAT data frames for 20 Epochs. With the newly trained model, we can generate massive test cases to attack the EtherCAT protocol. The following will introduce the relevant details.

a: COMMUNICATION BETWEEN THE MASTER AND THE SLAVE IN EtherCAT COMMUNICATION SYSTEM

The master station sends a message S_i containing multiple sub-messages. The slave completes the communication by reading or modifying the data of the corresponding sub-message. Then, the updated message S_i will return to the master as the message R_i. Each communication will correspond to a message pair $< S_i, R_i >$. As depicted in Fig. 15, a frame includes several sub-messages.

b: RECORD THE TEST CASE SENDING AND RECEIVING PROCESS

In order to record the data sent and received, we developed a test EtherCAT master station based on the SOEM (Simple Open EtherCAT Master) [68], an open source EtherCAT library. It can record the communication process for further analysis.

TABLE 4. Potential vulnerabilities and occurrences times in EtherCAT.

Potential Vulnerabilities	WGAN-Based Method	Sent Number
Packet Injection Attack	137 times	30,000
Man In The Middle Attack	352 times	30,000
Working Counter Attack	31 times	30,000

c: DETECTED POTENTIAL VULNERABILITIES

We discovered and counted these potential vulnerabilities including packet injection attack, man-in-the-middle (MITM) attack, working counter (WKC) attack. The packet injection means that the generated data message has changed the *value* of sub-message, but the *length* field in the frame header keeps unchanged. The MITM refers that a third-party malicious station changed the data in transit, but the master and the slave do not notice the change. In the experiment, if the generated message only changes the *data* field without change the *address* field and the slave still normally accept it, we regard this situation as the MITM attack. The WKC attack denotes that if the WKC value does not change as the *data* field changes. 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 aforementioned rules, we analyzed and compared the specified field values and obtained the following statistical results. Experiments on the EtherCAT protocol prove that our method can be conveniently applied to other industrial control protocols without knowing the protocol specifications or message format. Thus, it is effective for security testing of other industrial protocol systems.

VI. CONCLUSIONS AND FUTURE WORKS

In this study, we propose an effective fuzzing method based on Wasserstein GAN to generate fuzzing data about industrial control protocols. This method can learn the structure and distribution of real-world data frames and generate similar data frames without knowing the detailed protocol specification. Allowing the neural networks to learn the message format can save effort and reduce time. In this manner, when testing other network protocols, we do not need to understand their specifications, which is convenient. Our careful architectural design renders the model training more efficient. We ultimately evaluate this method by comparison with previous methods. The results obtained indicate the application potential of the proposed method. Also, we proved its versatility by fuzzing EtherCAT protocol.

In future studies, considerable research has to be conducted toward creating a more intelligent and more automated fuzzing system. Considering the current situation, we intend to perform the study in the following aspects. First, we will use our method to test a series of industrial control protocols, such as Profibus, Powerlink, and future TSN (Time Sensitive

Networks) [69]. These protocols constitute an essential part of most current industrial control systems. Second, we intend to integrate each processing module to form a complete software system, which can deal with most network protocols. Finally, we intend to implement the model with FPGA (Field-Programmable Gate Array) [70] for faster updating when new data become available.

REFERENCES

[1] E. D. Knapp and J. T. Langill, *Industrial Network Security: Securing Critical Infrastructure Networks for Smart Grid, SCADA, and Other Industrial Control Systems*. Amsterdam, The Netherlands: Syngress, 2014.

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

[3] B. P. Miller *et al.*, "Fuzz revisited: A re-examination of the reliability of unix utilities and services," University of Wisconsin, Madison, WI, USA, Tech. Rep. CS-TR-1995-1268, 1995.

[4] M. Arjovsky, S. Chintala, and L. Bottou. (2017). "Wasserstein gan." [Online]. Available: https://arxiv.org/abs/1701.07875

[5] J. Bao, D. Chen, F. Wen, H. Li, and G. Hua, "CVAE-GAN: Fine-grained image generation through asymmetric training," in *Proc. IEEE Int. Conf. Comput. Vis.*, Oct. 2017, pp. 2745–2754.

[6] J. Yang, A. Kannan, D. Batra, and D. Parikh. (2017). "Lr-gan: Layered recursive generative adversarial networks for image generation." [Online]. Available: https://arxiv.org/abs/1703.01560

[7] P. M. Comparetti, G. Wondracek, C. Kruegel, and E. Kirda, "Prospex: Protocol specification extraction," in *Proc. 30th IEEE Symp. Secur. Privacy*, May 2009, pp. 110–125.

[8] A. Swales *et al.*, *Open Modbus/TCP Specification*, vol. 29. Rueil-Malmaison, France: Schneider Electric, 1999.

[9] G. Cena, I. C. Bertolotti, S. Scanzio, A. Valenzano, and C. Zunino, "Evaluation of ethercat distributed clock performance," *IEEE Trans. Ind. Inform.*, vol. 8, no. 1, pp. 20–29, Feb. 2012.

[10] T. E. Vos and P. Aho, "Searching for the best test," in *Proc. IEEE/ACM 10th Int. Workshop Search-Based Softw. Test.*, May 2017, pp. 3–4.

[11] R. Kaksonen, M. Laakso, and A. Takanen, "Software security assessment through specification mutations and fault injection," in *Communications and Multimedia Security Issues of the New Century*. New York, NY, USA: Springer, 2001, pp. 173–183.

[12] D. Aitel, *The Advantages of Block-Based Protocol Analysis for Security Testing*. Miami, FL, USA: Immunity Inc., Feb. 2002.

[13] H. J. Abdelnur *et al.*, "Kif: A stateful sip fuzzer," in *Proc. 1st Int. Conf. Principles, Syst. Appl. IP Telecommun.*, May 2007, pp. 47–56.

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

[15] M. Rajpal, W. Blum, and R. Singh. (2017). "Not all bytes are equal: Neural byte sieve for fuzzing." [Online]. Available: https://arxiv.org/abs/1711.04596

[16] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 3104–3112.

[17] MichalZalewski. *Americanfuzzylop*. [Online]. Available: http://lcamtuf.c .cx/afl/

[18] C. Lv *et al.* (2018). "Smartseed: Smart seed generation for efficient fuzzing." [Online]. Available: https://arxiv.org/abs/1807.02606

[19] K. Böttinger, P. Godefroid, and R. Singh, "Deep reinforcement fuzzing," in *Proc. IEEE Secur. Privacy Workshops (SPW)*, May 2018, pp. 116–122.

[20] P. Godefroid, R. Singh, and H. Peleg, "Machine learning for input fuzzing," U.S. Patent 15 638 938, Oct. 4 2018.

[21] R. McNally, K. Yiu, D. Grove, and D. Gerhardy, "Fuzzing: The state of the art," DST Group Edinburgh, Edinburgh, Australia, Tech. Rep., 2012.

[22] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, p. 436, 2015.

[23] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[24] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, May 2012, pp. 1097–1105.

[25] S. Mukkamala, G. Janoski, and A. Sung, "Intrusion detection using neural networks and support vector machines," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, vol. 2, May 2002, pp. 1702–1707.

[26] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *Proc. IEEE Symp. Secur. Privacy*, May 2010, pp. 305–316.

[27] F. Falcini, G. Lami, and A. M. Costanza, "Deep learning in automotive software," *IEEE Softw.*, vol. 34, no. 3, pp. 56–63, Feb. 2017.

[28] M. Bojarski *et al.* (2016). "End to end learning for self-driving cars." [Online]. Available: https://arxiv.org/abs/1604.07316

[29] I. Lenz, H. Lee, and A. Saxena, "Deep learning for detecting robotic grasps," *Int. J. Robot. Res.*, vol. 34, nos. 4–5, pp. 705–724, 2015.

[30] G. Hinton *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 82–97, Nov. 2012.

[31] G. E. Dahl, D. Yu, L. Deng, and A. Acero, "Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition," *IEEE Trans. Audio, Speech, Lang. Process.*, vol. 20, no. 1, pp. 30–42, Jan. 2012.

[32] D. Bahdanau, K. Cho, and Y. Bengio. (2014). "Neural machine translation by jointly learning to align and translate." [Online]. Available: https://arxiv.org/abs/1409.0473

[33] A. V. M. Barone, J. Helcl, R. Sennrich, B. Haddow, and A. Birch. (2017). "Deep architectures for neural machine translation." [Online]. Available: https://arxiv.org/abs/1707.07631

[34] E. Nurvitadhi, J. Sim, D. Sheffield, A. Mishra, S. Krishnan, and D. Marr, "Accelerating recurrent neural networks in analytics servers: Comparison of FPGA, CPU, GPU, and ASIC," in *Proc. 26th Int. Conf. Field Program. Logic Appl.*, Aug. 2016, pp. 1–4.

[35] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. pattern Recognit.*, Jun. 2016, pp. 770–778.

[36] H. A. Rowley, S. Baluja, and T. Kanade, "Neural network-based face detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 20, no. 1, pp. 23–38, Jan. 1998.

[37] I. Goodfellow *et al.*, "Generative adversarial nets," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 2672–2680.

[38] C. Ledig *et al.*, "Photo-realistic single image super-resolution using a generative adversarial network," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2017, pp. 4681–4690.

[39] P. J. Reny, "Nash equilibrium in discontinuous games," *Econ. Theory*, vol. 61, no. 3, pp. 553–569, 2016.

[40] C. H. Sudre, W. Li, T. Vercauteren, S. Ourselin, and M. J. Cardoso, "Generalised dice overlap as a deep learning loss function for highly unbalanced segmentations," in *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support*. New York, NY, USA: Springer, 2017, pp. 240–248.

[41] B. Hanin, "Which neural net architectures give rise to exploding and vanishing gradients?," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 580–589.

[42] A. Srivastava, L. Valkov, C. Russell, M. U. Gutmann, and C. Sutton, "Veegan: Reducing mode collapse in gans using implicit variational learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 3308–3318.

[43] A. Radford, L. Metz, and S. Chintala. (2015). "Unsupervised representation learning with deep convolutional generative adversarial networks." [Online]. Available: https://arxiv.org/abs/1511.06434

[44] Q. V. Le, J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, and A. Y. Ng, "On optimization methods for deep learning," in *Proc. 28th Int. Conf. Int. Conf. Mach. Learn.*, 2011, pp. 265–272.

[45] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *Proc. Int. Conf. Mach. Learn.*, 2013, pp. 1139–1147.

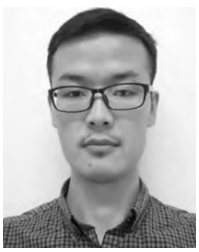
[46] D. P. Kingma and J. Ba. (2014). "Adam: A method for stochastic optimization." [Online]. Available: https://arxiv.org/abs/1412.6980

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

[48] B. Xu, N. Wang, T. Chen, and M. Li. (2015). "Empirical evaluation of rectified activations in convolutional network." [Online]. Available: https://arxiv.org/abs/1505.00853

[49] H. K. Kwan, "Simple sigmoid-like activation function suitable for digital hardware implementation," *Electron. Lett.*, vol. 28, no. 15, pp. 1379–1380, Jul. 1992.

[50] K. He and J. Sun, "Convolutional neural networks at constrained time cost," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Aug. 2015, pp. 5353–5360.
[51] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 1135–1143.
[52] L. K. Jones, "A simple lemma on greedy approximation in hilbert space and convergence rates for projection pursuit regression and neural network training," *Ann. Statist.*, vol. 21, no. 1, pp. 608–613, 1992.
[53] R. Mohammadi, R. Javidan, and A. Jalili, "Fuzzy depth based routing protocol for underwater acoustic wireless sensor networks," *J. Telecommun., Electron. Comput. Eng.*, vol. 7, no. 1, pp. 81–86, 2015.
[54] Z. Hu, J. Shi, Y. Huang, J. Xiong, and X. Bu, "Ganfuzz: A Gan-based industrial network protocol fuzzing framework," in *Proc. 15th ACM Int. Conf. Comput. Frontiers*, 2018, pp. 138–145.
[55] K. Shmelkov, C. Schmid, and K. Alahari, "How good is my gan?" in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, May 2018, pp. 213–229.
[56] L. Theis, A. V. D. Oord, and M. Bethge. (2015). "A note on the evaluation of generative models." [Online]. Available: https://arxiv.org/abs/1511.01844
[57] 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 *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 6626–6637.
[58] T. Karras, T. Aila, S. Laine, and J. Lehtinen. (2017). "Progressive growing of GANS for improved quality, stability, and variation." [Online]. Available: https://arxiv.org/abs/1710.10196
[59] M. Lucic, K. Kurach, M. Michalski, S. Gelly, and O. Bousquet, "Are GANS created equal? a large-scale study," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 698–707.
[60] D. Mondal, H. Hemmati, and S. Durocher, "Exploring test suite diversification and code coverage in multi-objective test case selection," in *Proc. IEEE 8th Int. Conf. Softw. Test., Verification Validation (ICST)*, Sep. 2015, pp. 1–10.
[61] H. Hemmati, A. Arcuri, and L. Briand, "Achieving scalable model-based testing through test case diversity," *ACM Trans. Softw. Eng. Methodol.*, vol. 22, no. 1, p. 6, 2013.
[62] Wireshark. (2018). *Wireshark*. [Online].Available: https://www.wireshark.org/
[63] Beckhoff. (2015). *Beckhoff*. [Online]. Available: https://www.beckhoff.com/
[64] Beckhoff-EK1100. (2019). *Ek1100*. [Online]. Available: http://www.beckhoff.com.cn/cn/default.htm?Ethercat/ek1100.htm/
[65] Beckhoff-EL1004. (2000). *El1004*. [Online]. Available: http://www.beckhoff.com.cn/cn/default.htm?Ethercat/el1004.htm/
[66] Beckhoff-EL2004. (2015). *El2004*. [Online]. Available:http://www.beckhoff.com.cn/cn/default.htm?Ethercat/el2004.htm/
[67] Beckhoff-ET2000. (2015). *Et2000*. [Online]. Available: https://www.beckhoff.com/english.asp?ethercat/et2000.htm/
[68] SOEM. (2015). *Soem*. [Online]. Available: https://openethercatsociety.github.io/
[69] M. Wollschlaeger, T. Sauter, and J. Jasperneite, "The future of industrial communication: Automation networks in the era of the Internet of Things and industry 4.0," *IEEE Ind. Electron. Mag.*, vol. 11, no. 1, pp. 17–27, Mar. 2017.
[70] E. Nurvitadhi *et al.*, "Can FPGAS beat GPUS in accelerating next-generation deep neural networks?" in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Aug. 2017, pp. 5–14.



ZHIHUI LI was born in Luoyang, Henan, China, in 1990. He received the B.S. degree in information management and information system from Henan Agricultural University, Zhengzhou, China, in 2013. He is currently pursuing the M.S. degree in software engineering with East China Normal University, Shanghai, China. Since 2016, he has been a Research Assistant with the National Trusted Embedded Software Engineering Technology Research Center. His research interests include industrial safety, deep learning, and model checking.

HUI ZHAO was born in Xuancheng, Anhui, China, in 1994. She is currently pursuing the M.S. degree in software engineering with East China Normal University, Shanghai, China. Since 2016, she has been a Research Assistant with the National Trusted Embedded Software Engineering Technology Research Center. Her research interests include model checking, simulation tools development, runtime verification, and machine learning.

JIANQI SHI was born in Tianjin, China, in 1984. He received the B.S. degree in software engineering and the Ph.D. degree in computer science from East China Normal University, Shanghai, China, in 2007 and 2012, respectively.

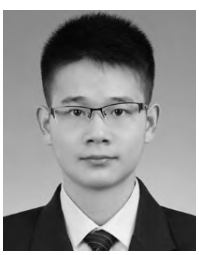
From 2012 to 2014, he was a Research Fellow with the National University of Singapore. In 2014, he joined the Temasek Laboratory under the Ministry of Defense of Singapore, as a Research Scientist. He is currently an Associate Researcher with the School of Computer Science and Software Engineering, East China Normal University. His research interests include formal method, formal modeling and verification of real-time or control systems, and IEC 61508 and IEC 61131 standards.

Dr. Shi's awards and honors include the Shanghai Science and Technology Committee Rising-Star Program, in 2018, and the ACM&CCF nomination of Excellent Doctor in Shanghai, in 2014.

YANHONG HUANG was born in Neijiang, Sichuan, China, in 1986. She received the B.S. degree in software engineering and the Ph.D. degree in computer science from East China Normal University, Shanghai, China, in 2009 and 2014, respectively.

In 2012, she joined Teesside University, U.K., as a Research Student. Since 2015, she has been an Assistant Researcher with the School of Computer Science and Software Engineering, East China Normal University. Her research interests include formal method, semantics theory, analysis and verification of embedded systems, and industry software.

Dr. Huang's awards and honors include the National Scholarship, in 2013, the IBM China excellent students, in 2013, and the Shanghai excellent graduates, in 2009 and 2014.



JIAWEN XIONG was born in Hangzhou, Zhejiang, China, in 1994. He received the B.S. degree in software engineering from Donghua University, Shanghai, China, in 2015. He is currently pursuing the Ph.D. degree in formal methods with East China Normal University, Shanghai. His research interests include model-driven development, model checking, runtime verification, program analysis, and related applications for safety-critical software systems.