A Deep Adversarial Networks based Industrial Control Protocol Fuzzing Framework from A Self-Attention Perspective

Wanyou Lv[1], Bohao Wang[1], Jianqi Shi[2⋆], Yanhong Huang[3], and Zhihui Li[4]

National Trusted Embedded Software Engineering Technology Research Center,
East China Normal University
Shanghai, China
{wanyou.lv, bohao.wang, zhihui.li}@ntesec.ecnu.edu.cn
{jqshi, yhhuang}@sei.ecnu.edu.cn

Abstract. The Industrial Control Protocol (ICP) is the cornerstone of the communication of the Industrial Control System (ICS). However, the industrial control environment applicable to ICPs has a strong diversity, which is difficult for testers to formulate a series of universal security rules. Therefore, fuzz testing (fuzzing) has already become the main method of detecting vulnerabilities in ICPs. It is noticed that the process of fuzzing relies heavily on specifications of ICPs. And it will take a lot of time and manual engineering to analyze and understand specifications. In this paper, we propose a new simple and smart sequence generation neural network framework based on Improved Wasserstein GANs (WGAN-GP), called HexGANFuzzer, to solve problems. Moreover, we put forward a series of performance metrics to evaluate different models in the field of fuzzing. We evaluate its performance by testing several typical ICPs, including MQTT and Modbus. Extensive experiments demonstrate significant improvements of HexGANFuzzer on test effectiveness and efficiency: the accuracy that test case packages are generated correctly is 86.08% and 92.71% in datasets of different data scales, F-measure reaches 82.08% and 85.02%, and the test effectiveness and efficiency are 8.03% and 5.26% higher than the WGAN-based model.

Keywords: Deep adversarial learning · Self-attention · Fuzz testing · Industrial control protocol

1 Introduction

With the arrival of Industry 4.0 [11] and Internet+, there occurred some new trends in the development of manufacturing. Strategic plans have been put forward to build the next generation of manufacturing. The ICS, as the basic component of the automated production in the national economy and the people's livelihood, is a key part of the industrial community. With the rapid development

⋆ Jianqi Shi@sei.ecnu.edu.cn

of industrial informatization, the interaction between different subsystems in the ICP becomes more frequent, leading to the fact that ICSs are facing increasing external security threats. So it is necessary to discover potential protocol vulnerabilities in time and prevent them in advance when applying the ICP into actual production.

Currently, applying traditional fuzz testing techniques to detect loopholes in ICPs is an effective method. However, there are some limitations: (i) High demand for the testers. The tester is required to design appropriate test cases according to the communication protocol specifications running in the ICS. (ii) Lengthy testing cycle. The entire testing cycle will last a long time. It is impossible to fulfil the test task efficiently when it is in urgent need. (iii) No universality. Traditional methods design specific test cases based on specific test objectives, which is not universal.

Compared with traditional fuzzing works, deep learning methods for fuzzing bypass the process of building protocol specifications and protocol automata, reduce the workload, and break the border of different protocols to achieve the universality. However, poor machine learning algorithms not only consume a lot of computing resources during model training but also tend to generate a large number of ill-formed protocol message sequences. And the generated ill-formed protocol message frames will result in normal crashes and error messages, which are quickly rejected by the server and prevent further testing.

In the process of fuzzing, there are a lot of crashes and error messages. But it is a challenge in distinguishing what of them are potential vulnerabilities and how to find real vulnerabilities from these crashes. In order to explore the balance between normal and abnormal anomalies, we propose a fuzzy test case generation methodology based on the ideal of deep adversarial learning in this paper. The contributions are summarized as follows:

(1) We propose a methodology based on GAN to deal with fuzzy data generation, in which it can intelligently learn to generate testing data by itself. We apply Wasserstein distance[1] to solve GAN's limitations of discrete sampling for sequences, and introduce a penalty term to ensure the diversity of generated test cases.
(2) On the premise of ensuring the lightweight of the model and saving computing resources, we introduce the self-attention mechanism which dispenses with recurrence and convolutions entirely and allows significantly more parallelization.
(3) On top of the approach, we build a universal fuzzing framework based on improved WGAN [1], called HexGANFuzzer, which can not only deal with the fuzzing of most ICPs but also show the superiority over other existing deep learning methods in theory and application.
(4) Since there are no corresponding evaluation standards in fuzzing based on deep adversarial learning, we propose a series of metrics to evaluate the performance of our framework from the evaluation of the performance of the machine learning model and the evaluation of the vulnerability detection capability

The remainder of this paper is organized as follows. Section II discusses the related work. Section III details the optimized improved SAGAN algorithm and the entire methodology design. Section IV presents the performance metrics that we propose. Section V presents the experiment and evaluation results. Section VI concludes the paper and discusses some ideas about future work.

2 Related Works

Fuzzing is a stress test by inputting a large number of unexpected abnormal inputs into the test target so as to trigger abnormal behavior of the target system, and analyze the system according to abnormal behavior [10]. Duran and Ntafos are pioneers in this field [5]. Since then, a variety of different techniques have been proposed to improve the efficiency of fuzzing. These techniques include static analysis [20], dynamic analysis [2]. Because of its effectiveness, fuzzing has been studied in the network protocol testing field to enhance the reliability of the computer network [3] [17]. And some of these works are for ICPs and have made a certain contribution to the improvement of the safety and security of ICPs.

However, fuzzing for ICPs still faces many challenges, such as how to mutate seed inputs, how to increase code coverage, and how to effectively bypass verification [12]. With the advancement of machine learning in the field of cybersecurity, it has also been adopted by many studies for vulnerability detection [22] [23] , including the applications in fuzzing [6] [15]. ICPs have many features in common such as short-data frames and no encryption. They are designed to satisfy the real-time requirements of ICSs. As expected, some studies have incorporated fuzzing algorithms of ICPs based on deep learning into the fuzzing process of ICPs [14]. These efforts all contribute to fuzzing based on deep learning.

There are still some limitations of these aforementioned fuzzing algorithms based on deep learning, such as unbalanced training samples, lack of feature classification ability of industrial communication behavior, and difficult to extract the characteristics related to vulnerabilities. We integrate the characteristics of self-attention and GAN to propose a deep convolution generative adversarial networks based fuzzing methodology and design an automated and intelligent fuzzing framework based on it, named HexGANFuzzer. And with using Wasserstein distance and a gradient penalty (WGAN-GP) [7], our framework can not only solve the problem of poor performance for discrete data but also ensure the diversity of generated test cases.

3 HexGANFuzzer Framework

In this section, we preprocess the communication data captured from the ICS as the training data of the deep adversarial learning model established. Secondly, a generative model and a discriminant model are designed to generate the adversarial network.

3.1 Preprocessing of ICP Message Frames

In order to get desirable fuzzing results, data preprocessing is a necessary part. There are two steps in preprocessing: **Message Frame Clustering** and **Data Conversion**.

Message Frame Clustering After message frames collection, there are many different length message frames of the ICP. We adopted the Length Clustering method to preprocess the data frames. First, we gather the data frames which have the same length. Groups whose quantity is less than a threshold (e.g. 5000) will be moved to the longer adjacent group, while group contains more data than the threshold are left unchanged. After clustering, the max length of message frames of every group will be the standard length and message frames whose length is less than max length will be added special token 'u' at the end of them.

Data Conversion The raw data frame of most protocols is hexadecimal, which can not be fed into the model directly. So after clustering, the hexadecimal data frames will be mapped into the digital vectors. The vocabulary we use is as followed:

$$\boldsymbol{0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ a\ b\ c\ d\ e\ f\ u} \tag{1}$$

Based on the vocabulary, message frames $x \in R^{seq\_size \times 1}$ were converted into one hot vector $X \in R^{seq\_size \times 17}$(seq_size is the max length of the current group), and one-hot vectors of real message frame will be the input of the critic.

3.2 Model Design

In this part, we describe our test case generation model in detail. Our model consists of a generator and a critic, which is the same as a WGAN model. The generator is born to generate the fake message frames and the critic will evaluate the Wasserstein distance. The Wasserstein distance is the minimum cost of transporting mass in converting the data distribution q to the data distribution p. During the training, the Wasserstein distance between real message frame distribution and fake message frame distribution will be shortened and our model grasps the knowledge of the protocol grammar.

Generator The structure of the generator is as followed in Fig. 1. There are four parts, noise block, conversion block, transformer block, and output block. In order to generate different message frame data every time, the noise block will output some random noise data $z \in \mathbb{R}^{1 \times zd}$ where zd is the initial dimension of the noise. And it is a common practice to set the noise data to Gaussian distributed noise.

The conversion block contains a fully connected layer and a reshape layer. This block will convert the noise data z into Noise Conversion Representation

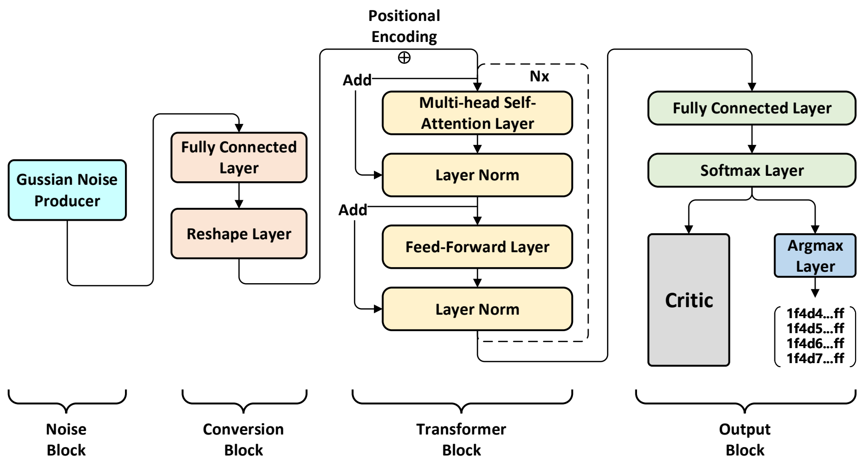

Fig. 1: Generator Network

(NCR) $\tilde{z} \in \mathbb{R}^{ss \times dm}$ where ss is the max sequence length of the current group and dm is the number of feature dimension.

The transformer block contains a positional encoding layer and a sub-block. The sub-block is the same as the encoder part of the Transformer [21]. It is formed by a multi-head self-attention layer and a feed-forward layer. Besides, there is a residual connection around each of the two layers, followed by layer normalization. And the input and output of the transformer block are the same, so this part can repeat as many times as we want.

The output block is a fully connected layer and a softmax layer. After the output block, the generator will not output the message frame data directly, instead, it outputs a probability vector $\tilde{x} \in \mathbb{R}^{ss \times vs}$. Where vs is the size of vocabulary, which is 17 in our situation. The probability vector is an intermediate result which is used to feed to the critic. To obtain the message frame which can be sent, the probability vector will be applied $argmax$ function and be translated into hexadecimal data according to the vocabulary mentioned before.

The sequence information is normally one-dimensional vector, but during the computation in the NLP model, the sequence information is represented by two-dimensional vectors including word embedding or one-hot vector for the sake of a better representation of the sequence. In our model, the two-dimensional vector is NCR. The reason we use NCR is that the output of the noise block is a random noise vector, which is hard to get the corresponding word embedding. In the NLP tasks, discrete data is usually regarded as the result of continuous sampling from the categorical distribution. If we force to find the word embedding with some tricks, it may cause the whole procedure non-derivable, which will make the backpropagation algorithm unavailable. The solution of most GAN text generation model is using the one-hot vector to replace the word embedding in the model. In this paper, our solution is NCR. Define the fully connected layer as f, processed by fully connected layer, ReLU activation and reshape operation, noise data z will become $\tilde{z} = Reshape(f(z))$ and $\tilde{z} \in \mathbb{R}^{ss \times dm}$. And it is the input of the transformer block. NCR looks like word embedding, but it is fake.

The transformer block is a feature extractor. Compared with the commonly used feature extractor, Recurrent Neural Network (RNN), the transformer block is better in semantic feature extraction and long-range feature extraction. Moreover, the self-attention layer in the transformer block supports parallel training, which can accelerate the training process. The loss function of the generator is

$$L_g = - \mathop{\mathbb{E}}_{\tilde{x} \sim \mathbb{P}_g} [D(\tilde{x})] \tag{2}$$

where D is a 1-Lipschitz function, $\tilde{x} \in \mathbb{R}^{ss \times vs}$ is the output of the generator, $\mathbb{P}_g$ is the model distribution implicitly defined by $\tilde{x} = G(z)$, and z is the noise data.

Critic It looks like discriminator in normal GAN model, but it is not a classifier, we call it critic here, which is the same as WGAN [1]. The structure of the critic is shown in Fig. 2. The critic includes five conv1d layers, five self-attention layers, and one fully connected layer. For the critic, there are two types of inputs. One is the one-hot vector from the real message frames, and another is the output of the generator which represents the fake message frames. The second dimension of the output of the generator means the probability of each word in the vocabulary, and one-hot vector can also be understood as a special case of it, that is, the probability of one word is 1 and the probability of other words is 0.

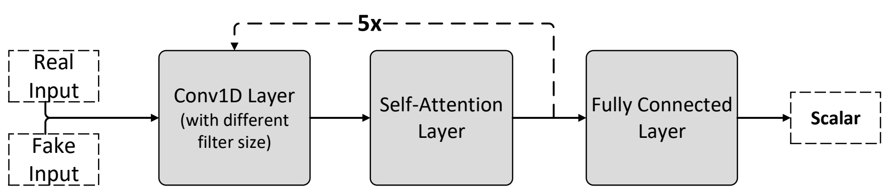

Fig. 2: Critic Network

It is noted that the first conv1d layer changes the second dimension of inputs from the probability of word to the feature representation, which is the same as the dimension of NCR. And other conv1d layers keep the second dimension as dm. Every conv1d layer followed by a self-attention layer but the kernel size of conv1d is not the same. The kernel size of ith conv1d layer $ks_i = i$ where $i \in \{1, 2, 3, 4, 5\}$.

Protocols have message headers, which specify the content of the message body in the message frame. More simply, the front part of the message frame affects the back part. It is hard to distinguish the boundary of two parts without the knowledge of the protocol grammar. If the model has learned which part is the message header, it can pay more attention to the message header part. Due to different protocols have different lengths of the message header, the conv1d layer with different kernel size can capture the information better. As for the content in the message header, several flags in the message header affect different parts of the content in the message body. self-attention layers can capture how flags in the message header affect the part of the message body.

The self-attention layer considers the attention weight of every position of the input and is good at learning long-range dependencies. With the boundary information leaned by conv1d layers, self-attention layers will pay more attention to the correlation between the content in the message header and body. After repeated self-attention and conv1d layers five times, the fully connected layer will output a scalar score. The loss of the critic is:

$$L_c = \underset{\tilde{x}\sim\mathbb{P}_g}{\mathbb{E}}[D(\tilde{x})] - \underset{x\sim\mathbb{P}_r}{\mathbb{E}}[D(x)] + \lambda \underset{\hat{x}\sim\mathbb{P}_{\hat{x}}}{\mathbb{E}}\left[(\|\nabla_{\hat{x}}D(\hat{x})\|_2 - 1)^2\right] \tag{3}$$

where $\mathbb{P}_{\tilde{x}}$ define sampling uniformly along straight lines between pairs of points sampled from the real data distribution $\mathbb{P}_r$ and the generator distribution $\mathbb{P}_g$. In order to satisfy the 1-Lipschitz constraint, the solution of the WGAN-GP model was adopted. We use gradient penalty instead of the weight clipping to enforce the 1-Lipschitz constraint. It leads to a more stable training process and the model is easier to train.

Training strategy Due to the properties of the Wasserstein distance, we train the critic better first and then narrow the Wasserstein distance between fake message frame distribution and real message frame distribution, which is the process of improving generator. In every epoch, we train generator once and then train the critic c_iters (5 in our model) times. In order to judge the convergence of the model, the following equation, which is the approximate value of the Wasserstein distance, is an indicator. If the value of the equation trends to be stable, we consider the model is convergent.

$$W\_distance = \underset{x\sim\mathbb{P}_r}{\mathbb{E}}[D(x)] - \underset{\tilde{x}\sim\mathbb{P}_g}{\mathbb{E}}[D(\tilde{x})] \tag{4}$$

The model is saved every 10 training epoch deliberately. In such a training strategy, the goal of the high code coverage and deeper testing depth can be achieved.

4 Performance Metrics

Some quantitative criteria [8] [16] have emerged recently for evaluating the performance of GAN on image generation. However, there is no corresponding evaluation metrics in fuzzing based on deep adversarial learning, and the lack of these indicators includes two aspects: the evaluation of the performance of the machine learning model [9] in the field of fuzzing based on deep learning [19] and the evaluation of the vulnerability detection capability. Therefore, in accordance with our research purpose and specific situation, the following evaluation metrics are proposed in this study.

4.1 F-measure

In order to further demonstrate the performance of our model in this paper, we compare the performance of different models on test data generation of ICPs with several indexs such as *Sensitivity*, *Specificity*, *Accuracy* and *F-measure*. Among them, Sensitivity represents the percentage of function codes for correctly generated test case packages versus function codes for real test case packages, Specificity means to the percentage of non-essential hexadecimal characters for correctly generated test case packages versus non-essential hexadecimal characters for real test case packages, and Accuracy is the percentage of the entire test case that correctly generates the format and message content of the test case package. The specific formula is as follows:

$$Accuracy = \frac{TP + TN}{TP + FN + TN + FP} \tag{5}$$

$$Sensitivity = \frac{TP}{TP + FN} \tag{6}$$

$$Specificity = \frac{TN}{TN + FP} \tag{7}$$

$$Precision = \frac{TP}{TP + FP} \tag{8}$$

$$F - measure = 2 \times Precision \times \frac{Sensitivity}{Precision + Sensitivity} \tag{9}$$

wherein, TP/FP is the number of correctly/wrongly generated hexadecimal characters that constitute the function codes of ICPs' packages, and TN/FN is the number of correctly/wrongly generated other non-essential hexadecimal characters including message content of ICPs' packages.

4.2 Effectiveness of Vulnerability Detection (EVD)

EVD refers to the ability to trigger anomalies on basis of the test target accepting the test data. The effectiveness of fuzzing depends predominantly on the testing depth and high code coverage. Therefore, we propose the effectiveness evaluation metric EVD for fuzzing models from the above two aspects.

When the generated data can be accepted by the test target, it indicates that the generated data is similar to real data in terms of the format. This phenomenon reflects a certain depth of testing. But our ultimate goal is to trigger as many anomalies as possible to the ICS to find vulnerabilities, so two sub-metrics are taken into account when we finally calculate the testing depth for EVD: *Acceptance Rate of Test Cases (ARTC)* and *Ability of Vulnerability Detected (AVD)*. ***ARTC*** is the quotient of the total number of test cases accepted and the total number of test cases sent. It refers to the acceptance rate at which the generated test data is received by the test target. ***AVD*** is the quotient of the total number of test cases which trigger anomalies and the total number of test cases sent. It refers to the ability of triggering anomalies for the test target.

Diversity of Generated Cases (DGC) is a sub-metric proposed as the extent of code coverage for fuzzing in EVD. It refers to the ability to maintain the diversity of the generated data [18]. We define DGC as the quotient of the total number of message categories in the generated data frames and the total number of message categories in the training dataset.

In order to balance the different weights of sub-metrics on model effectiveness, dimensionless quantity method is applied to eliminate the influence of different scales between sub-metrics so that each sub-metric can be converted into a value that can be directly added or subtracted. We normalize aforementioned sub-metrics and add them up to get the comprehensive scores for EVD. We define $Norm$ as follows:

$$Norm(sm) = \frac{sm_i - sm_{min}}{sm_{max} - sm_{min}} \tag{10}$$

where sm is a sub-metric, sm_i is the sub-metric value of the ith model, sm_{max} and sm_{min} are respectively the maximum and minimum values of the corresponding sub-metric. And we define $EVDS = \{ARTC, AVD, DGC\}$, and:

$$EVD = \frac{1}{m}\sum_{j}^{m} Norm(EVDS_j) \times 100\% \tag{11}$$

4.3 Efficiency of Vulnerability Detection (EFVD)

EFVD includes two sub-metrics, which are *Training Time (TT)* and *Time of Anomalies Triggered (TAT)*. ***TT*** refers to the time required to train a model. Short TT correspondingly improves the efficiency of testing. ***TAT*** refers to the time interval from the first request time (t_1) to the time when the third anomaly (such as error code 3, no response from the server, etc.) is triggered ($t_{3_{\mathrm{th\_anomaly}}}$) .

It should be noted that the number of anomalies found is also related to the test target. Weak target will highlight the method effectiveness. However, because our ultimate goal is not to find the differences between the various test target, we only focus on the efficiency of the method in this study. We define $EFVDS = \{TT, TAT\}$. The specific formula is as follows:

$$EFVD = \frac{1}{m}\sum_{j}^{m}(1 - Norm(EFVDS_j)) \times 100\% \tag{12}$$

5 Experiment and Result Analysis

In this section, MQTT and Modbus are chosen as our target protocols from a variety of ICPs in the experiment. Test cases generated by HexGANFuzzer and control group are used to stress test the target and trigger anomalies of the ICP.

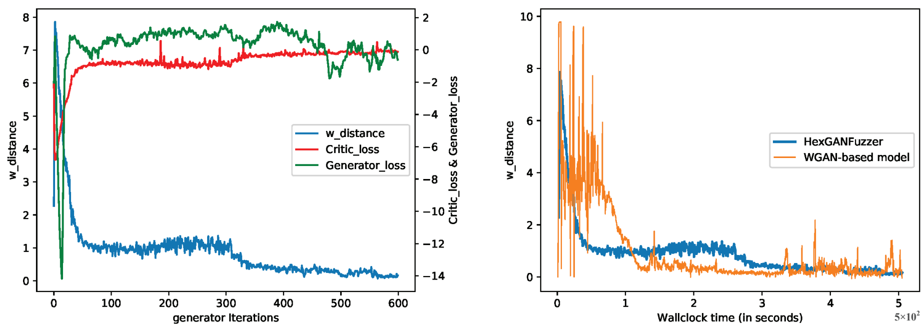

Fig. 3: Dataset(500,000 test cases) W-distance over generator iterations (left) or wall-clock time (right) for HexGANFuzzer

5.1 Evaluation Setup

Tensorflow is adopted to implement the model architecture. To improve the training efficiency, we train the model and launch an attack 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 ubuntu16.04, x64-based processor.

As for the parameter setting, the mini-batch size is set to 64 in all models based on a large amount of training data. The *keep_prob* hyperparameter of dropout is set to 0.7. The learning rate of Generator and Critic is set to 0.0001 and 0.0004 in the Adam optimizer. λ in the loss equation of Critic is 10. The dimension of the initial noise zd is 10 and feature dimension dm is 50. The repeat times of the transfomer block is 4 and the number of heads is 2. We train the models for 100 epochs and save the generator model for every 10 epochs to get plentiful test cases.

5.2 Fuzz Testing The Target ICP

One advantage of our method over GANs is largely overcoming the problem of unstable training of GANs, such as non-convergence, vanishing gradient and mode collapse. To demonstrate this, we train HexGANFuzzer with Wasserstein distance and gradient penalty on a dataset with 500,000 test cases and plot Wasserstein distance and losses of the generator and the discriminator in Fig. 3 (left). For comparision, we train another WGAN-based model with the same hyperparameter setting. The result in Fig. 3 (right) represents that our method converges faster and is more stable at convergence.

With the trained model, we generate test cases. Then the generate test cases are input to the system under test, and we implement the fuzzy test process according to the standard fuzzy test process. In the process of inputting test cases into ICSs under test, we monitor the system in real time. If anomalies occur in the process, such as the output clobber problems and interface crash,

Table 1: Performance Metrics Comparison

Dataset (number of training cases)	**Methods**	**Year**	**F-measure**	**SE**	**SP**	**AC**
10,000	GPF	2007	–	0.5252	0.5798	0.6474
	CNN-1D model	2014	0.6701	0.8437	0.9743	**0.8626**
	LSTM-based model	2018	0.6300	**0.8563**	0.9007	0.8254
	WGAN-based model	2019	0.7960	0.7696	0.9115	0.8165
	HexGANFuzzer	2020	**0.8208**	0.8274	**0.9775**	0.8608
500,000	GPF	2007	–	0.5720	0.5730	0.6510
	CNN-1D model	2014	0.8373	**0.8670**	0.9842	0.8690
	LSTM-based model	2018	0.8388	0.8203	**0.9856**	0.8700
	WGAN-based model	2019	0.8396	0.8108	0.9371	0.8906
	HexGANFuzzer	2020	**0.8502**	0.8538	0.9478	**0.9271**

we record and restart the test process at the current point. At the end of the entire test, we analyze the causes of antecedent anomalies and the logs of server systems to find the running status of potential vulnerabilities.

5.3 Result Analysis

In this part, we show the experimental results in three aspects. We first reveal statistical results and their analysis. Then we present a special anomaly that occurred in fuzzing the MQTT implementations. Lastly, to show the methodology's protocol independence in fuzz testing of ICPs, the result of testing Modbus-TCP is presented.

Results And Statistical Analysis The widely used General Purpose Fuzzer (GPF) [4], CNN-1D model, LSTM-based seq2seq model, and WGAN-based model are chosen as fuzzers in the control group in this study. After fuzzers in the experimental group and control group are fully trained, fuzz testing is conducted by sending a total of 100,000 generated test cases to MQTT implementations through the TCP/1883 port. According to the three aforementioned evaluation metrics, the details of different models are as follows.

a. F-measure. Table 1 shows the accuracy of each method in generating legitimate test cases on a dataset of 10,000 and 500,000 training data. It can be observed from the table that the algorithm in this paper has Higher scores

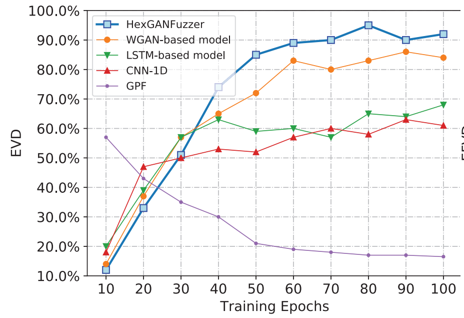

Fig. 4: EVD changes with the training epochs

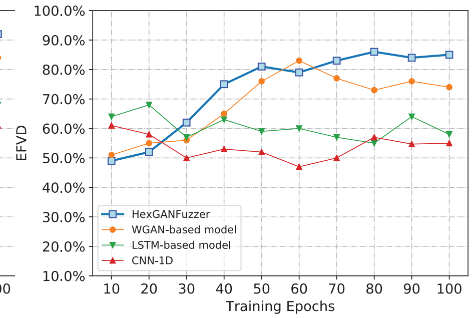

Fig. 5: EFVD changes with the training epochs

of Sensitivity, Accuracy, F-measure than CNN-1D model, LSTM-based seq2seq model and WGAN-based model in datasets of different sizes. Although the Sensitivity of the CNN-1D model and LSTM-based seq2seq model is higher than our model, the types of test cases generated are relatively single, and the algorithm in this paper has the highest F-measure. For the dataset of 10,000 training data, the F-measure of generating legitimate test cases reaches 82.08%, which is 2.48% higher than the WGAN-based model, and the Sensitivity of this method is 5.78% higher than the WGAN-based model. On the dataset of 500,000 training data, our method is 1.06% higher in F-measure and 4.30% higher in Sensitivity than the WGAN-based model.

Therefore, we can draw a conclusion from the evaluation metrics of the performance of machine learning models in the table: in terms of datasets with different data sizes, the model in this paper is superior to other fuzzy test case methods in each evaluation metric of generating legitimate test cases.

b. EVD. In order to map the EVD values of each model to a relatively small scope for diagrammatic comparison, we normalize all EVD values under the same iteration through Z-score standardization. The experimental results of EVD are shown in Fig. 4. Due to not involving a continuously learning process, the performance of GPF has a downward trend on the targets when compared to the other four fuzzing models based on depth learning. From the perspective of the five models, the performance of the EVD indicators is: GPF $<$ CNN-1D model $\approx$ LSTM-based model $<$ WGAN-based model $<$ HexGANFuzzer.

After training more than 30 epochs, the EVD rates of deep learning algorithms obviously exceed the GPF algorithm. With the rising trends of rates slow down significantly after 50 epochs, the average EVD rates of CNN-1D model and LSTM-based model are 60% to 70% compared with 75% to 90% of generative adversarial algorithms, which may be caused by the inability to learn a hierarchy of representations of the data effectively. For the generative adversarial algorithms from 50 to 100 epochs, the average EVD rate of HexGANFuzzer is approximately 8% higher than the WGAN-based model, which indirectly indicates that HexGANFuzzer is more applicable to test cases generation for ICPs.

It's worth going into detail that our trained model can effectively detect kinds of anomalies, which shows we are likely to achieve the higher code coverage and the stronger ability of our model has to detect anomalies.

c. EFVD. There is a huge difference in the training time of different GPF, so it is not discussed in this part. It can be seen from Fig. 5 that when doing less training epochs, models with relatively short training time can achieve higher EFVD scores. And with the deepening of training, TAT values of different models become more and more different, so models with higher potential efficiency of vulnerability detection will get higher EFVD scores. Looking at the overall trend, the average EFVD score of our model is 19.45% higher than that in the CNN-1D model, 15.14% higher than the LSTM-based model and 5.26% higher than the WGAN-based model. And the fluctuation of our model is relatively small, which indicates that our model can maintain relatively stable efficiency on the basis of balancing TT and TAT.

Potential Vulnerabilities Found Due to the limited space, we mainly talk about one exception that we analyzed, which is ***Buffer Exception***. The broker truncates the message frames according to the reamining length in the MQTT protocol only. The part of the data that exceeds the remaining length will be left in the buffer, which will cause the problem of no answering of the following message frames.

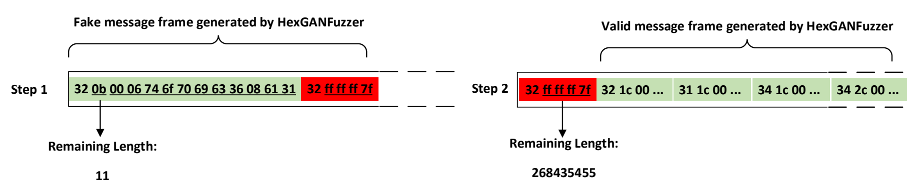

Fig. 6: Potential vulnerabilities analysis

As the Fig. 6 shows, in *step*1, we send the generated message frames formed by green and red box part to the broker. The red part is left in the buffer. In this situation, the broker has treated **0x32ffffff7f** as the part of the message header of the next message frame. In *step*2, when real next message frames arrived at the buffer, the broker has considered 268 million as the remaining length of this message frame. This will make the communication invalid between the client and the broker for a period of time unless the buffer is full or the length of the message frames reach 268 million.

Applying The Method to Modbus-TCP As shown in Table 2, we detect these potential vulnerabilities, including slave crash, station off-line, using abnormal function code and so on, in Modbus-TCP via the new trained Hex-

GANFuzzer. Experiments on the Modbus-TCP protocol prove that our method has great versatility.

Table 2: Potential Vulnerabilities and in Modbus-TCP

Triggered Anomalies	**NTA**	**ATITA (Mins)**
Slave crash	29 times	21.11
Station ID xx off-line	164 times	0.79
Using abnormal function code	207 times	0.52
Automatically closes window	13 times	28.47
Data length unmatched	190 times	0.63
Unknown attack	197 times	0.59

The number of Triggered anomalies(NTA) represents the total number of anomalies triggered by the models during test time and average time interval of triggered anomalies (ATITA) is the quotient of the number of triggered anomalies and test time.

6 Conclusions and Future works

This paper applies deep adversarial learning from a self-attention perspective to generate fake but plausible fuzzing protocol messages of ICPs. We combine Wasserstein distance with self-attention to the model, making the model training more stable and computing hidden relationship between any two hexadecimal characters in parallel for all the input and output. And gradient penalty is applied to enforce the 1-Lipschitz constraint, which maintains the diversity of the generated data. The performance of our method is verified on datasets of different data sizes: the accuracy of the datasets containing 10,000 and 500,000 training data are 86.08% and 92.71% respectively, and F-measure are 82.08% and 85.02% respectively.

Our work motivates many avenues for future research, such as: GAN Compression [13], online learning with being deployed in embedded devices and anti-random strategies to improve the probability of anomalies triggering.

References

1. Arjovsky, M., Chintala, S., Bottou, L.: Wasserstein gan. arXiv preprint arXiv:1701.07875 (2017)
2. Bastani, O., Sharma, R., Aiken, A., Liang, P.: Synthesizing program input grammars. ACM SIGPLAN Notices **52**(6), 95–110 (2017)
3. Chen, J., Diao, W., Zhao, Q., Zuo, C., Lin, Z., Wang, X., Lau, W.C., Sun, M., Yang, R., Zhang, K.: Iotfuzzer: Discovering memory corruptions in iot through app-based fuzzing. In: NDSS (2018)

4. DeMott, J., Enbody, R., Punch, W.F.: Revolutionizing the field of grey-box attack surface testing with evolutionary fuzzing. BlackHat and Defcon (2007)
5. Duran, J.W., Ntafos, S.C.: An evaluation of random testing. IEEE transactions on Software Engineering (4), 438–444 (1984)
6. Godefroid, P., Peleg, H., Singh, R.: Learn&fuzz: Machine learning for input fuzzing. In: 2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE). pp. 50–59. IEEE (2017)
7. Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., Courville, A.C.: Improved training of wasserstein gans. In: Advances in neural information processing systems. pp. 5767–5777 (2017)
8. Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., Hochreiter, S.: 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 (2017)
9. Karras, T., Aila, T., Laine, S., Lehtinen, J.: Progressive growing of gans for improved quality, stability, and variation. arXiv preprint arXiv:1710.10196 (2017)
10. Kaschner, K., Lohmann, N.: Automatic test case generation for interacting services, service-oriented computing—icsoc 2008 workshops: Icsoc 2008 international workshops, sydney, australia, december 1st, 2008, revised selected papers (2009)
11. Lasi, H., Fettke, P., Kemper, H.G., Feld, T., Hoffmann, M.: Industry 4.0. Business & information systems engineering **6**(4), 239–242 (2014)
12. Li, J., Zhao, B., Zhang, C.: Fuzzing: a survey. Cybersecurity **1**(1), 6 (2018)
13. Li, M., Lin, J., Ding, Y., Liu, Z., Zhu, J.Y., Han, S.: Gan compression: Efficient architectures for interactive conditional gans. arXiv preprint arXiv:2003.08936 (2020)
14. Li, Z., Zhao, H., Shi, J., Huang, Y., Xiong, J.: An intelligent fuzzing data generation method based on deep adversarial learning. IEEE Access **7**, 49327–49340 (2019)
15. Liu, X., Prajapati, R., Li, X., Wu, D.: Reinforcement compiler fuzzing (2019)
16. Lucic, M., Kurach, K., Michalski, M., Gelly, S., Bousquet, O.: Are gans created equal? a large-scale study. In: Advances in neural information processing systems. pp. 700–709 (2018)
17. Martin-Lopez, A., Segura, S., Ruiz-Cortés, A.: A catalogue of inter-parameter dependencies in restful web apis. In: International Conference on Service-Oriented Computing. pp. 399–414. Springer (2019)
18. Mondal, D., Hemmati, H., Durocher, S.: Exploring test suite diversification and code coverage in multi-objective test case selection. In: 2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST). pp. 1–10. IEEE (2015)
19. Shmelkov, K., Schmid, C., Alahari, K.: How good is my gan? In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 213–229 (2018)
20. Sparks, S., Embleton, S., Cunningham, R., Zou, C.: Automated vulnerability analysis: Leveraging control flow for evolutionary input crafting. In: Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007). pp. 477–486. IEEE (2007)
21. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. In: Advances in neural information processing systems. pp. 5998–6008 (2017)
22. Wu, F., Wang, J., Liu, J., Wang, W.: Vulnerability detection with deep learning. In: 2017 3rd IEEE International Conference on Computer and Communications (ICCC). pp. 1298–1302. IEEE (2017)
23. Xia, H., Liu, C.: Remote sensing image deblurring algorithm based on wgan. In: International Conference on Service-Oriented Computing. pp. 113–125. Springer (2018)